Bachelorarbeit

Steffen Bredemeier

Konzept für eine hochverfügbare Datenhaltung für Maschinen- und Anlagendaten im Umfeld Industrie 4.0

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Steffen Bredemeier

Konzept für eine hochverfügbare Datenhaltung für Maschinen- und Anlagendaten im Umfeld Industrie 4.0

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Bachelor of Science Technische Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Ulrike Steffens
Zweitgutachter: Prof. Dr. Klaus-Peter Kossakowski

Eingereicht am: 17.03.2016

Steffen Bredemeier

Thema der Arbeit
Konzept für eine hochverfügbare Datenhaltung für Maschinen- und Anlagendaten im Umfeld Industrie 4.0

Stichworte
Industrie 4.0, Datenhaltung

Kurzzusammenfassung
Dieses Dokument beschreibt die Wichtigkeit, mögliche Hauptschwerpunkte und Lösungsansätze für die Datenhaltung im Umfeld Industrie 4.0.
Steffen Bredemeier

Title of the paper
Concept for a high-availability data management for machine and plant data in the environment industry 4.0

Keywords
industry 4.0, data management

Abstract
This document describes the importance, possible main priorities and solutions for data management in the environment industry 4.0.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

1. Einleitung

1.1. Datenhaltung in der Industrie

Schon seit jeher ist die Erstellung von Daten zu Analysezwecken in der Industrie essentiell. Diese Informationen dienen meist dem optimalen Betrieb einer Produktionsanlage, der Überwachung von Systemprozessen oder auch der Protokollierung von Abläufen. So können Fehlermeldungen zeitlich aufgenommen werden. Dadurch kann zu einem späteren Zeitpunkt eine genaue Analyse Auskunft darüber geben, ob eine Anlage evtl. vorzeitig gewartet werden muss. Darüber hinaus gibt es noch eine Vielzahl an Möglichkeiten die Daten zu nutzen, häufig mit dem Ziel Prozesse zu optimieren und Kosten zu senken oder andere weitreichende Entscheidungen für die zukünftige Planung zu treffen. Damit wird klar, dass die Informationen einer gesonderten Datenhaltung bedürfen. Zum einen müssen diese auch in Fehlerfällen geschützt sein, damit sie nach der Fehlerbehebung wieder zur Verfügung stehen. Zum anderen sollten die Daten jederzeit abrufbar sein. Wird beispielsweise eine Wartung an einer Anlage durchgeführt, darf dies keinen Einfluss auf andere Anlagen haben. Die Daten der zu wartenden Anlage stehen dann immer noch für andere Anlagen zur Verfügung. Damit wird die Datenhaltung von dem Betrieb der Anlage entkoppelt.

1.2. Aktueller Stand in der Industrie

Konzepte zum redundanten Betrieb der IT-Infrastruktur gibt es vielerorts seit längerem. Dazu zählen redundante Switches, RAID-Systeme, Netzwerk-Ringleitungen usw. Häufig sind diese Konzepte aber nur unzureichend umgesetzt.

Die meisten Anlagenteile brauchen Parameter, damit die entsprechenden Produkte erzeugt werden. Als Beispiel könnte eine Mehrkopfwaage in einer Müsliverpackungsanlage dienen. Diese Waage benötigt Informationen welches Produkt – in diesem Fall sind es verschiedene Müslisorten – zusammengemischt werden soll. Es gibt viele Gründe, warum die Mehrkopfwaage keine Daten mehr aus dem übergeordneten System bekommen kann. Dazu zählen Netzwerkausfall, Wartungsarbeiten an dem übergeordneten System oder einen Stromausfall in

einem Bereich des Systems. In jedem Fall kann die Anlage nicht weiter produzieren und die Herstellung verzögert sich.

Unabhängig vom Speicherort müssen die Daten langfristig geschützt sein. Aktuell beschränken sich die Sicherheitsansätze meist auf RAID-Systeme. Damit sind die Daten weder bei einem Rechnerausfall verfügbar, noch ist der Schutz bei Unfällen, wie z.B. Bränden, gegeben.

1.3. Aufgabenstellung

Ziel dieser Arbeit soll es sein, die aktuelle Entwicklung in Richtung Industrie 4.0 mit Informationen zu unterstützen, damit es möglich wird, die meist unzureichende Datenhaltung von Produktionsanlagen in hohem Maße aufzuwerten. Darüber hinaus ist es notwendig, bestimmte Begriffe, die im Zusammenhang zu Industrie 4.0 auftreten, zu erläutern.

Diese Arbeit erhebt nicht den Anspruch, eine ideale Lösung für auftretende Speicherprobleme zu zeigen, da der Weg zu Industrie 4.0 noch lang ist und niemand das Ende der Entwicklung absehen kann. Vielmehr ist es ein erster Schritt um Verantwortliche, die solche Probleme zukünftig lösen müssen, für bestimmte Aspekte zu sensibilisieren.

1.4. Themenabgrenzung

Im Zuge dieser Ausarbeitung nahm die Erstellung einer Übersicht aktueller Produkte einen beträchtlichen Teil der Zeit in Anspruch. Nichtsdestotrotz ist diese Übersicht bei weitem nicht komplett. Vielmehr wurde versucht möglichst Produkte zu wählen, die sich unterschiedlicher Technologien bedienen.

Um eine hochverfügbare Datenhaltung zu erzielen, ist es erforderlich neben dem eigentlichen Speichermedium auch andere Aspekte zu berücksichtigen. Dazu zählen unter anderem die Netzwerkstruktur oder auch Wartungszyklen. Diese Herausforderungen wurden hier nicht berücksichtigt.

Neben der reinen Datenhaltung sind weitere Anforderungen für einen professionellen Einsatz in der Industrie erforderlich. Eine wichtige Aufgabe ist die IT-Sicherheit. Um das zu analysieren, wäre es notwendig gewesen, die Produkte dahingehend praktisch zu prüfen, da die Dokumentation meist unzureichend ist. Dies bedeutet einen beträchtlichen Aufwand und konnte deshalb nicht berücksichtigt werden.

Neben den technologischen Veränderungen die Industrie 4.0 mit sich bringt, wird es Neuerungen in anderen Bereichen der Produktionsindustrie geben. Dazu zählt beispielsweise die

flexiblere Gestaltung der Arbeitsaufgaben von Angestellten. Diese ”Brennpunkte” wurden bewusst nicht in diese Arbeit aufgenommen.

1.5. Aufbau der Arbeit

Das Kapitel Grundlagen dient dem Einstieg. Hierbei soll in das Thema Industrie 4.0 und die damit im Zusammenhang stehenden Begriffe erklärt werden. Darüber hinaus sind hier für die Funktionsweise der Produkte wichtige Grundlagen erklärt.

Die weiter ansteigende Bedeutung der Informationen in Anlagen, und der damit verbundene Aufwand diese Daten verfügbar zu machen, erfordert den Einsatz zeitgemäßer Technologien. Einen Überblick aktueller Produkte, die für den Einsatz in hochverfügbaren Systemen in Frage kommen könnten, gibt Kapitel 3. Der Schwerpunkt liegt bei dem Einblick in die jeweiligen Produkte und dem Versuch, die Vor- und Nachteile herauszukristallisieren.

Nachdem die funktionalen und nicht funktionalen Anforderungen in Kapitel 4 formuliert wurden, soll Kapitel 5 dazu dienen ein Produkt detaillierter zu betrachten. Hierbei stehen die Einrichtung sowie eine erste Performanceanalyse im Vordergrund.

Die Ergebnisse der Arbeit werden in der Zusammenfassung noch einmal reflektiert. In diesem letzten Kapitel wird zudem ein Ausblick gewährt, bei dem die Lambda-Architektur eingeführt wird.

2. Grundlagen

Industrie 4.0 bezieht sich in den meisten Fällen auf die produzierende Industrie. Aber was genau ist mit Industrie 4.0 gemeint? Was hängt damit zusammen? Warum ist die Datenspeicherung so immens wichtig in diesem Bereich? Um den Antworten etwas näher zu kommen, wird dieses Kapitel genutzt, um allgemeine Begrifflichkeiten zu erklären und Zusammenhänge darzustellen.

Im Laufe dieser Arbeit werden verschiedene Synchronisierungsprodukte vorgestellt. Um die Funktionsweise besser zu verstehen, dient dieses Kapitel der Beleuchtung der in den Produkten verwendeten Technologien und Strategien.

Warum werden konventionelle Methoden der Datenhaltung im Umfeld Industrie 4.0 scheitern?
Die Arbeit soll zeigen, dass sich neue Herausforderungen für die Datenhaltung in der Industrie 4.0 ergeben. Zum einen wächst sowohl die Menge als auch die Vielfalt der Daten. Es müssen nicht nur strukturierte Informationen abgespeichert werden, sondern auch unstrukturierte Daten (Bilder, Videos, Textdateien). Die heutzutage im Einsatz befindlichen Lösungen wie Raid-Systeme oder Datenbanken werden diesen Herausforderungen nicht mehr gewachsen sein. Zusätzlich müssen die Storage-Lösungen Hochverfügbarkeit gewährleisten damit im Fehlerfall der Betrieb der Anlagen sichergestellt ist.

2.1. Industrie 4.0

Nach der Dampfmaschine, der Massenabfertigung durch Fließbänder und der digitalen Revolution soll nun die vierte industrielle Revolution folgen. Diese unterscheidet sich zu den ersten drei dahingehend, dass sie eine Vision ist und bewusst herbeigeführt wird. 2011 hat die deutsche Bundesregierung den Arbeitskreis ”Industrie 4.0” ins Leben gerufen, um die Entwicklung zur vierten industriellen Revolution voran zu treiben [BHV14, S. 249]. Ziel dieses Arbeitskreises ist es, Deutschland eine Vorreiterrolle in diesem Gebiet und somit einen technologischen Vorsprung gegenüber anderen Ländern zu sichern. Die Verlagerung von Arbeitsplätzen in Niedriglohnländer soll verlangsamt, bestenfalls sogar umgekehrt werden. Technologische

Grundlage für diese Entwicklung sind ”Cyber-Physical Systems” (CPS) und das ”Internet of Things” (IoT).

Durch eine Veränderung der Produktionswelt – unter anderem durch den Einsatz von CPS – soll es möglich werden, dass ”…technische Systeme selbständig, autonom und auch via Internet Produktionsszenarien aufbauen, auflösen und neu konfigurieren können …” [BHV14, S. 249]. Eine einzelne Produktionslinie wird somit in der Lage sein, verschiedene Produkttypen herzustellen, ohne dass der Produktionsprozess grundlegend umstrukturiert werden muss. Eine wesentliche Voraussetzung, um das Ideal der ”*Losgröße* eins” zu erreichen.

2.2. Cyber-Physical System (CPS)

Die Deutsche Akademie der Technikwissenschaften definiert in dem Projekt ”Integrierte Forschungsagenda Cyber-Physical Systems” ein CPS unter anderem wie folgt: ”Cyber-Physical Systems,*[…]*, sind demzufolge keine in sich abgeschlossenen Einheiten. Es sind vielmehr offene *soziotechnische Systeme*, die durch die hochgradige Vernetzung der physikalischen, sozialen und virtuellen Welt sowie durch die intelligente Nutzung von Informations- und Kommunikationstechnologien entstehen.”[GB10, S. 17] Einfacher ausgedrückt könnte man sagen, dass in einem CPS die physikalische Welt – also die reale – mit der Welt der Informationstechnik verknüpft ist, um den Menschen in allen Bereichen des Lebens zu unterstützen.

***Beispiel**: Fahrer/innen von Autos der gehobenen Mittelklasse oder der Luxusklasse ist es heutzutage möglich, den sogenannten Stauassistenten zu nutzen. Das Auto kann bis zu einer bestimmten Geschwindigkeit selbst fahren. Nimmt der Fahrer/in die Hände ans Lenkrad, schaltet sich der Assistent ab. In diesem System arbeiten die Technologien ACC und Spurhalteassistent zusammen. Somit kann das Auto selbständig beschleunigen, bremsen und lenken. Verschiedene im Auto verbaute Technologien interagieren mit Hilfe von Sensoren sowohl untereinander als auch mit dem/der Fahrer/in und bilden somit ein Cyber Physisches System.*

Internet of Things (IoT)

Sinngemäß wurde der Begriff ”Internet of Things” wohl von Neil Gerschenfeld 1999 in seinem Buch ”Wenn die Dinge denken lernen” als erstes verwendet. Er schrieb: ”Es kommt mir so vor, als sei das rasante Wachstum des WWW nur der Zündfunke einer viel gewaltigeren Explosion gewesen. Sie wird losbrechen, sobald die Dinge das Internet nutzen.”[Mat10] Den Begriff hat

aber Kevin Ashton in einem Artikel im Forbes-Magazin[Sch02] 2002 verwendet und populär gemacht.

Heutzutage werden die Daten im Internet fast ausschließlich vom Menschen produziert. Im IoT geschieht dies aber durch Dinge. Sie kommunizieren mit der Umwelt, erfassen Daten und interagieren mit anderen Dingen und Menschen. Möglich wird dies durch den Einsatz von Sensoren, die in so gut wie allen Dingen des Lebens integriert sind.

Eng verwandt mit dem IoT ist der Begriff *Ubiquitous Computing*. Hiernach werden die Computer, wie sie heute existieren, verschwinden und durch intelligente Gegenstände ersetzt mit dem Ziel, den Menschen im Alltag zu unterstützen.

Zusammen mit den Cyber Physical Systems, soll das Internet of Things einen der Grundpfeiler für die Smart Factory bilden. Dadurch verändert sich die Fabrik, wie sie heute existiert, gravierend.

2.3. Smart Factory

Unternehmen, die international agieren, stehen gewissen Komplexitäten gegenüber. Zu der äußeren Komplexität gehört unter anderem die Lieferfähigkeit, die Verfügbarkeit, die Preiselastizität sowie die Verlässlichkeit von Produkten. Die äußere Komplexität beschreibt die Herausforderung, die von außen auf ein Unternehmen wirkt. Demgegenüber steht die innere Komplexität. Hierzu zählen unter anderem das Produktportfolio, Prozesstechnologien und IT-Systeme.

Um diese Komplexitäten zu bedienen und auch langfristig international konkurrenzfähig zu sein muss der Übergang von der konventionellen Fabrik hin zum cyber-physischen Produktionssystem gelingen. Durch die Integration von CPS in die Produktionsanlagen und Logistikkomponenten werden diese befähigt, das Internet und Internetdienste zu nutzen (IoT), untereinander zu kommunizieren und sich selbst zu organisieren. Damit entsteht die *Smart Factory* [BHV14, S. 16].

***Beispiel**: Die Inbetriebnahme von Maschinen ist ein sehr zeitaufwändiger und damit teurer Prozess. Dabei müssen unter anderem die Kennwerte der Maschinenteile manuell in die Maschinensteuerung eingegeben werden. Das benötigt Zeit und ist fehleranfällig. In der Smart Factory sind diese Informationen in den Anlagenteilen integriert (Selbstbeschreibung), so dass diese im Falle einer Inbetriebnahme automatisch ausgelesen und abgespeichert werden.*

Zitat: "Die digitale Vernetzung beschert eine nie dagewesene Fülle an Daten, die den Zustand von Maschinen und Anlagen beschreiben."[Len]

In der *Smart Factory* sind CPS allgegenwärtig. Anlagen produzieren große Mengen an Daten und benötigen entsprechende Speicherressourcen aber auch Möglichkeiten, diese großen Datenmengen, wenn nötig zeitnah, abzuspeichern, auszuwerten oder abzurufen. Beim Start eines Projektes ist es aber in den meisten Fällen schwierig, das Ausmaß der Daten langfristig einzuschätzen und damit einhergehend die Leistungsfähigkeit des Speichers bezüglich Dimensionierung und Performance zu planen. Hier zeigt sich schon ein wichtiges Kriterium für eine Storage-Lösung. Sie sollte skalierbar sein – idealerweise horizontale Skalierung – , um im späteren Verlauf den Ansprüchen gerecht zu werden.

Ab einer bestimmten Datenmenge wird es zunehmend schwieriger diese zeitnah auszuwerten. Die Daten werden zu einem Big-Data-Problem und erfordern geeignete Maßnahmen, um mit diesen Daten umzugehen.

2.4. Big Data

Heutige Storage-Cluster können durchaus mehrere Petabyte aufnehmen (z.B. Google oder Facebook). Die Performance zum Verarbeiten der Informationen macht eine Parallelisierung notwendig, um die Reaktionsfähigkeit des Systems sicher zu stellen. Anhand von relativ geringen Festplattengrößen lässt sich schon erahnen, dass dies notwendig ist. Aufgrund der günstigen Preise für Festplattenspeicher können Daten kostengünstig gespeichert werden. Im Gegensatz zu den Speicherkapazitäten haben sich die Zugriffszeiten und Transferraten in den letzten Jahren nur unwesentlich verbessert. In Tabelle 2.1 sind typische Festplatten aus vier verschiedenen Jahren aufgelistet. Die Kapazität ist zwischen 1988 und 2015 um den Faktor 500000 gestiegen und die Transferrate im Vergleich nur um 250. Mit dem Anstieg der Datenmenge folgt also auch eine Verlängerung der Datenbearbeitung.

Jahr	**HDD-Kapazität** GB	**Transferrate** MB/s	**Zugriffszeit** ms	**Zeit zum Auslesen** min
1988	0,020	0.625	65	0,5
2002	320	54	9	98
2009	2000	110	8,9	303
2015	10000	157	8,5	1000

Tabelle 2.1.: Auslesezeiten von Festplatten aus verschiedenen Jahren

Wird die Größe der Datenmenge, bzgl. der Performance für die Auswertung, zu einem Problem, spricht man von Big Data.

Was genau ist Big Data? Um den Begriff Big Data zu definieren werden häufig die drei V – **V**olume (Speicherbedarf), **V**elocity (Geschwindigkeit) und **V**ariety (Vielfalt) – verwendet.

Volume

Volume beschreibt dabei nicht unbedingt das Datenvolumen an sich. Vielmehr ist damit die Ausführungszeit in Abhängigkeit der Datengröße gemeint. Eine Datei, die sich auf einem leistungsstarken Rechner ohne Probleme zeitnah öffnen lässt, kann auf schwächerer Hardware eventuell erst nach einer längeren Zeit geöffnet werden. Damit wird die Datei auf schwächeren Rechnern durch die Verarbeitung zu einem Big-Data-Problem und nicht durch den benötigten Speicherbedarf.

Velocity

Velocity bedeutet Geschwindigkeit und ist auf die Verarbeitung und Aktualität der Daten bezogen.

Variety

Der Datenvielfalt wird mit Variety Rechnung getragen. Bei Big Data können die Daten nicht ohne weiteres in feste Strukturen überführt werden, um sie beispielsweise in Datenbanken zu organisieren. Stattdessen sind es häufig Dateien (z.B. Bilder, Videos, HTML-Dokumente oder Log-Dateien), die einen höheren Aufwand bei der Auswertung erfordern.

Veracity

Um die Richtigkeit und Echtheit von Daten zu beschreiben hat IBM ein viertes V eingeführt, das für **V**eracity (Richtigkeit) steht. Hierbei geht es darum, dass Inhalte häufig durch Werbung, fehlerhafte Übersetzungen, falsche bzw. veraltete Suchergebnisse oder gezielte Falschaussagen/Fehlinformationen verfälscht werden.

***Beispiel**: Ein Brite streute am 26. Februar 2012 das Gerücht vom Tod des Schauspielers Rowan Atkinson. Innerhalb von drei Stunden existierte bereits ein Eintrag auf Wikipedia mit dem Todestag von Atkinson.[Fre14] Dieses Ereignis soll zeigen, dass es im Internet keine Instanz gibt, die Informationen auf ihre Richtigkeit (Veracity) untersucht. Der vermeintliche Tod von Herrn Atkinson wird bis zum Dementi für die Internetgemeinschaft Realität.*

Es existieren auch Definitionen, die das Problem etwas weiter abstrahieren. Wenn die Daten selbst Teil des Problems werden und die Mengen zu umfangreich für aktuelle Verarbeitungsmethoden sind spricht man von Big-Data.

Die von CPS-Systemen – meist automatisch – generierten Daten in einer Smart Factory sollen, sowohl lang- als auch kurzfristig, verschiedenen Systemen[1] zur Verfügung stehen. Ausfallzeiten kosten Geld und sind nicht hinnehmbar. Deshalb sind Hardwarefehler oder Stillstandszeiten durch Wartungsarbeiten zu vermeiden. Daher ist es unerlässlich die Daten zu synchronisieren, um sie jederzeit verfügbar zu machen. Das nächste Kapitel soll deshalb den Begriff Synchronisierung einführen und erläutern.

[1] Das können sowohl übergeordnete (z.B. ERP, SCADA) als auch untergeordnete (z.B. Produktionsstraßen) Systeme sein.

2.5. Synchronisierung/Replikation

In dieser Arbeit ist Synchronisation im Sinne von Datenabgleich gemeint. Wird also eine Ressource an einem Punkt *A* im System verändert, wird sie auch an einem Punkt *B* im gleichen System geändert. Bei den meisten hier vorgestellten Technologien spielt es keine Rolle, ob Punkt *A* und Punkt *B* auf ein und dem selben Rechner vorhanden sind. In den meisten Fällen liegen die Ressourcen verteilt im Netzwerk. Man unterscheidet zwischen unidirektionaler- und bidirektionaler Synchronisation.

Werden die Daten ausschließlich von Gerät *A* zu Gerät *B* abgeglichen spricht man von unidirektionaler Synchronisation.

***Beispiel**: Bestimmte Daten eines PC sollen als Sicherung auf einem Backup-Medium gespeichert werden. Die Richtung des Datenaustausches ist hierbei immer von PC zu Backup-Medium. Werden die Daten auf dem Backup-Medium gelöscht, hat das keine Auswirkungen auf die Datenquelle, also den PC.*

Im Gegensatz zur unidirektionalen Synchronisierung werden bei der bidirektionalen Synchronisierung Daten in beide Richtungen gesendet.

***Beispiel**: Wenn man einen Eintrag in seinem Kalender am PC hinzufügt wird er auch auf dem Smartphone hinzugefügt. Wird der Eintrag am Smartphone eingefügt, so geschieht das auch am PC.*

Bei der one-to-one-Synchronisation werden die Daten zwischen einem Client und einem Server ausgetauscht.

***Beispiel**: Zur Sicherung der Daten einer Datenbank A sollen die Daten auf eine zweite Datenbank B repliziert werden. Solange der Datenfluss ausschließlich von A nach B geht und nicht umgekehrt handelt es sich um eine unidirektionale Synchronisierung. Sind nur die Datenbanken A und B an dieser Synchronisierung beteiligt können Konflikte ausgeschlossen werden.*

Tauschen die Clients ihre Informationen über genau einen Server aus, spricht man von einer Many-to-one-Kommunikation. Hierbei können die Clients nicht direkt untereinander kommunizieren.

***Beispiel**: Die Kalenderdaten auf einem PC sollen mit dem Smartphone und dem Tablet synchronisiert werden. Hierbei dient der PC als Server und die beiden Mobilgeräte als Clients.*

In dieser übersichtlichen Topologie können Konflikte ausschließlich auf dem Server auftreten und sind auch nur von diesem zu lösen. Auf der anderen Seite stellt der Server den SPoF (*Single Point of Failure*) im System dar. Das bedeutet sowohl eine höhere Ausfallwahrscheinlichkeit als auch eine geringere Verfügbarkeit, falls der Server an seine Leistungsgrenze stößt.

Wenn alle Clients ohne einen Server untereinander Daten austauschen können handelt es sich um eine Many-to-many-Kommunikation oder auch Peer-to-Peer-Kommunikation. Jeder Client hat die Daten der anderen Clients und kann diese als Server propagieren. Einen SPoF existiert in dieser Topologie nicht und die Peers können das System jederzeit beteten oder verlassen. Daraus resultiert eine höhere Komplexität der Peers, um den erhöhten Anforderungen Rechnung zu tragen.

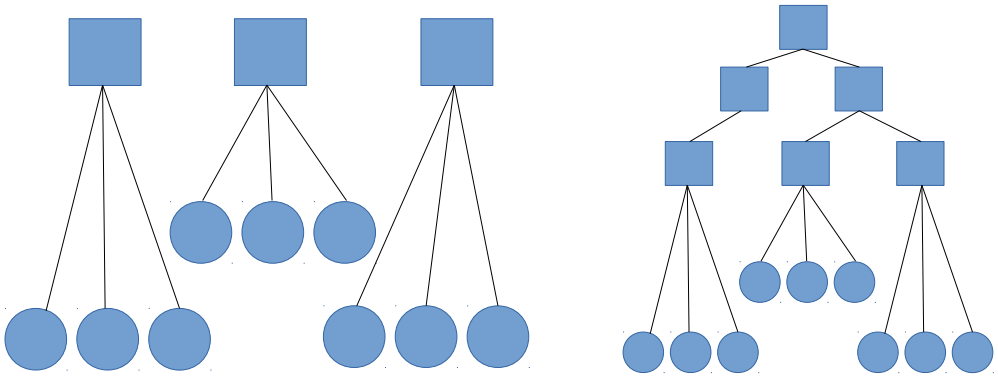
Um die Vorteile der many-to-one und many-to-many-Architekturen zu vereinen, kann man hybride Topologien (Abbildung 2.1) aufbauen. Hierbei gibt es zwei verschiedene Strategien. Bei der Cluster-Topologie besteht die obere Schicht aus Servern, welche Daten von den Clients sammeln und diese zur Datenreplikation untereinander austauschen. In der unteren Schicht befinden sich die Clients.

Bei der hierarchischen Struktur sind die Server wie die Struktur eines Unternehmens angeordnet. Damit wird jeder Server unter dem Rootserver zu einem Client, der die Daten an den darüber liegenden Server schickt.

Je nach Einsatzzweck können die Topologien genutzt werden, um Daten redundant zu halten. Fehlerszenarien helfen u.a. bei der Auswahl einer geeigneten Topologie. So kann ein System entworfen werden, das Hochverfügbarkeitsanforderungen erfüllt. Das folgende Kapitel soll als Einstieg dienen, um den Begriff Verfügbarkeit zu erläutern.

2.6. Verfügbarkeit eines Systems

Unter Verfügbarkeit versteht man den Anteil der Zeit, die ein System funktionsfähig ist und berechnet sich aus $\frac{verfügbare\ Zeit}{maximal\ mögliche\ Zeit}$.



(a) Cluster-Topologie (b) Hierarchy-Topologie

Abbildung 2.1.: Hybrid-Lösung

Quelle: [Han+13]

***Beispiel**: Für eine Verfügbarkeit von 99 % ist die Ausfallzeit eines Systems am Tag (*$24\ \mathrm{h/d} * 60\ \mathrm{min/h} * 60\ \mathrm{s/min} = 86400\ \mathrm{s}$*) nach der Rechnung (Dreisatz)*

$$86400\ \mathrm{s} - \frac{86400\ \mathrm{s} * 99\ \%}{100\ \%}$$

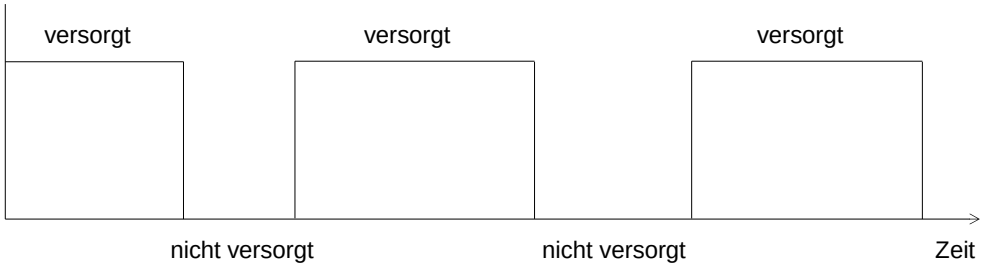
$864\ \mathrm{s}$. *Daraus folgt, dass das System* $85536\ \mathrm{s}$ *am Tag funktionieren muss um eine* $99\ \%$*ige Verfügbarkeit zu erreichen.*

Die Havard Research Group[Gro] hat fünf Klassen definiert,die die Anforderungen eines Systems an die Verfügbarkeit beschreiben.

Eine höhere Verfügbarkeit geht meistens auch mit höheren Kosten einher. Daher sollten die Anforderung in der Planungsphase genau spezifiziert werden.

In einem Rechnersystem sind neben dem Speichersystem auch andere Komponenten für die Verfügbarkeit verantwortlich. Dazu zählen die Daten, Applikationen, das Netzwerk und Wartungszeiten. Es sind also mehrere Bestandteile eines Systems für die Verfügbarkeit zuständig. Was nützt ein redundanter Cluster-Speicher, wenn das Netzwerk ausfällt oder falsche Wartungspläne zu Ausfallzeiten führen? Für jede Systemkomponente muss ein spezieller Verfügbarkeitsansatz entwickelt werden. In dieser Arbeit wird ausschließlich auf das Speichersystem eingegangen.

Im Zusammenhang mit der Verfügbarkeit eines Systems stehen die Begriffe MTBF (*Mean Time Between Failures*) und MTTR (*Mean Time To Repair*). Der Zusammenhang dieser Begrifflichkeiten ist in Abbildung 2.2 dargestellt.



$$Verfügbarkeit\ A = \frac{MTBF}{MTBF + MTTR}$$

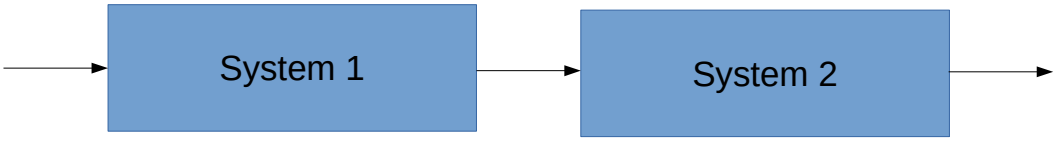
$$Unverfügbarkeit\ U = \frac{MTTR}{MTBF + MTTR}$$

Abbildung 2.2.: Zusammenhang zwischen Mean Time Between Failure, Mean Time To Repair und Verfügbarkeit
Quelle: [Spe]

Speichersysteme können häufig mit Hilfe von *Striping* zu höherer Leistung geführt werden. Dabei ist zu beachten, dass dies zu einer Verschlechterung der Verfügbarkeit führt. Im Gegensatz zu Striping lässt sich durch *Mirroring* die Zuverlässigkeit steigern, was aber keine Steigerung der Geschwindigkeit mit sich bringt, sondern eventuell noch Leistungseinbußen bedeuten könnte. Die Ausfallwahrscheinlichkeit von kombinierten Systemen kann man über die Wahrscheinlichkeitsrechnung ermitteln. Auf Abbildung 2.3 sind die Zusammenhänge verdeutlicht. Mirroring und Striping lassen sich auch kombinieren. Mit einem erhöhten Hardwareaufwand ist dann sowohl eine Steigerung der Sicherheit als auch der Performance möglich.

Ob nun Mirroring oder Striping, durch die Kombination von Subsystemen ist eine Performancesteigerung und/oder eine Steigerung der Gesamtkapazität möglich. Ein Gesamtsystem besteht häufig aus einer Kombination von Knoten, die die einzelnen Subsysteme bilden. Können während des Betriebes einzelne Knoten hinzugefügt oder entfernt werden spricht man von der Möglichkeit der horizontalen Skalierung. Um den Unterschied zwischen der horizontalen und der vertikalen Skalierung zu erklären, sind beide Begriffe im nächsten Kapitel erklärt.

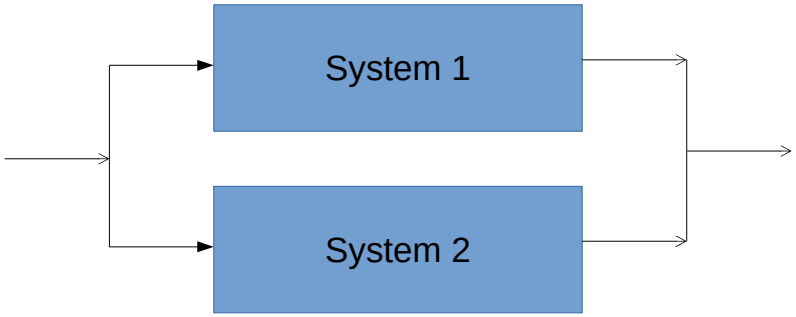
Multiplikationssatz der Wahrscheinlichkeit (UND)



$$A_{ges} = A_{Sys1} \wedge A_{Sys2}$$

$$Beispiel:$$
$$A_{Sys1} = 0.9$$
$$A_{Sys2} = 0.9$$
$$A_{ges} = A_{Sys1} \cdot A_{Sys2} = 0.81$$

Additionssatz der Wahrscheinlichkeit (ODER)



$$A_{ges} = A_{Sys1} \vee A_{Sys2}$$

$$Beispiel:$$
$$A_{Sys1} = 0.9$$
$$A_{Sys2} = 0.9$$
$$A_{ges} = 1 - U_{Sys1} \cdot U_{Sys2}$$
$$= 1 - (1 - 0.9) \cdot (1 - 0.9) = 0.99$$

Abbildung 2.3.: Über Wahrscheinlichkeitsrechnung lässt sich die Verfügbarkeit von Systemen bestimmen.

Quelle: [Spe]

***Hinweis**: Auch bei einer hochverfügbaren Konfiguration ist eine Backstrategie in den meisten Fällen sinnvoll und nötig. So können verloren gegangene Daten, die versehentlich oder durch kriminelle Energie bewusst gelöscht wurden, wiederhergestellt werden.*

2.7. Skalierung

Viele Systeme stoßen im Laufe der Zeit an einen Punkt, an dem sie die an ihnen gestellten Aufgaben nicht mehr ausreichend erfüllen können. Antworten von Servern kommen zu spät zurück oder eine Datenbank hat nicht mehr ausreichend Speicherplatz zur Verfügung, um neue Einträge zu speichern. Ein Upgrade muss dafür sorgen, dass das System wieder die Aufträge abarbeiten kann. Die konventionelle Art eines Upgrades ist der Tausch einer CPU gegen eine leistungsfähigere oder der Tausch der Festplatte gegen eine größere. Das bedeutet aber zwangsläufig, dass das System während des Upgrades nicht zur Verfügung steht. Um dies zu vermeiden wurden Systeme entwickelt, die mit anderen Systemen ein Gesamtsystem bereitstellen, bei dem neue Ressourcen in Form von neuen Rechnern hinzugefügt oder auch entfernt werden können.

Vertikale Skalierung (scale up)

Bei dieser Art der Skalierung werden einem Knoten weitere Ressourcen hinzugefügt, um dessen Leistung zu steigern. Dies kann durch Vergrößern des Speicherplatzes, dem Hinzufügen von einer CPU oder auch einer leistungsstärkeren Grafikkarte sein. Zu den Vorteilen gehört, dass die Erweiterung eine Leistungssteigerung keine Änderung an der implementierten Software benötigt. Sollte die beste Hardware bereits verbaut sein, stößt diese Art der Skalierung an ihre Grenzen.

Horizontale Skalierung (scale out)

Wird die Leistungssteigerung durch Hinzufügen von zusätzlichen Rechnern/Knoten erreicht handelt es sich um eine horizontale Skalierung. Hierbei sind theoretisch keine Grenzen gesetzt, aber die implementierte Software muss die Parallelisierung unterstützen.

Da in dieser Arbeit die Skalierung des Festplattenspeichers im Vordergrund steht, ist häufig die Rede von verteilten Dateisystemen. Diese sollen ortsunabhängige Speicherressourcen zu einem Gesamtspeicher zusammenfassen. Dies geschieht in der Regel über ein Netzwerk. Im nächsten Kapitel wird der Begriff Verteiltes Dateisystem näher definiert.

2.8. Verteilte Dateisysteme

Ein verteiltes Dateisystem (englisch distributed file system, DFS oder network file system) ist ein spezielles Dateisystem, mit dem der Zugriff auf Dateien über ein Rechnernetz erfolgt und das Zugriff und Datenspeicherung auf mehreren als Server eingesetzten Rechnern erlaubt. Anders als beim Netzwerk-Dateisystem werden in einem klassischen lokalen Dateisystem nur die angeschlossenen Massenspeicher verwaltet. Ein DFS sollte die Eigenschaften der Transparenz erfüllen. Hierbei muss es für den Benutzer egal sein, ob er auf lokale Dateien oder auf Dateien im Netzwerk zugreift.

Zurzeit sind ZFS und btrfs die fortschrittlichsten Dateisysteme. Sie bieten unter anderem Software-RAID. Beide benötigen vergleichsweise viele Systemressourcen. Da der Einsatz von einem verteilten Dateisystem auch auf älteren Systemen möglich sein soll, werden daher diese beiden Dateisysteme nicht näher in dieser Arbeit thematisiert.

Mit einem verteilten Dateisystem ist es möglich, riesige Datenmengen abzuspeichern. Hierfür können tausende Knoten zusammengeschlossen werden. Doch wie werden die Daten bei einem Zugriff lokalisiert? Die konventionelle Suche nach einer Datei, wie sie zum Beispiel auf einem normalen Windows-Rechner erfolgt, ist nicht durchführbar. Die Zugriffszeiten sind schon bei einer Festplatte zu hoch und wenig praktikabel. Das Problem wird bei einem Object Storage gelöst, indem die Daten nicht mehr als Dateien in einer Hierarchie abgelegt werden. Im nächsten Kapitel Object Storage werden die Eigenschaften von Object Storages aufgeführt.

2.9. Object Storage

Was sind Objekte im Object Storage? Objekte sind Daten und besitzen Metadaten wie normale Dateien. Sie haben keine Hierarchien wie Ordnerstrukturen. In den Metadaten eines Objektes ist ein Code gespeichert, der es dem Server erlaubt, das Objekt zu finden, ohne zu wissen wo es sich befindet.

***Analogie**: Den Code in den Metadaten könnte man mit einer Abrissmarke bei einem Opernbesuch vergleichen. Der Mantel wird an der Garderobe im Tausch mit einer Marke abgegeben. Der Besucher weiß nicht, wo sein Mantel abgelegt wird oder ob dieser während der Vorstellung den Ort in der Garderobe wechselt. Der Opernbesucher ist dabei der Nutzer, der seine Daten, also den Mantel, irgendwann wieder haben will.*

Im Gegensatz zu einem RAID-System können die Festplatten beim Object-Storage verteilt im Netz liegen. Der dadurch einhergehenden höheren Latenz für Datenzugriffe wird durch den

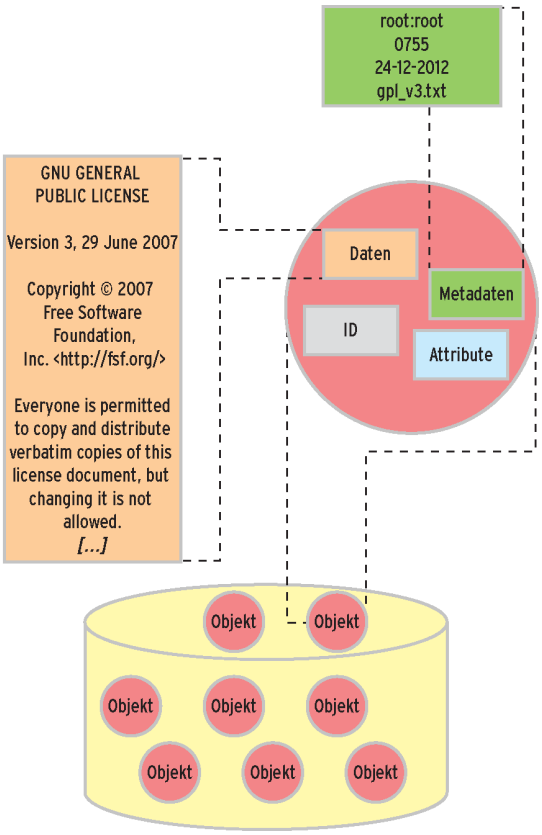


Abbildung 2.4.: Das DFS Ceph speichert Informationen als Objekte.
Quelle: [Sei]

Einsatz von lokalem Caching, Netzwerkkomprimierung und Lastausgleich entgegengewirkt. Mittels HTTP-REST-Schnittstelle kann der Zugriff auf die Daten beschleunigt werden.

Erasure Code erlaubt es, die Daten um zusätzliche Informationen anzureichern und verteilt abzuspeichern, um Bitfehler zu vermeiden. Damit sind die Daten auch bei Ausfall eines oder mehrerer Knoten lesbar. Durch die erhöhte Robustheit können kostengünstigere Festplattenlösungen zum Einsatz kommen, die evtl. weniger Strom- und Kühlungsbedarf haben.

3. Synchronisierungstechnologien im Überblick

3.1. rsync-Algorithmus

rsync ist ein Algorithmus zum unidirektionalen Abgleichen von Dateien zweier Computer.

Angenommen PC A möchte eine Datei a mit PC B abgleichen und B besitzt mit b eine ältere Version von a. Mit folgenden Schritten wird die Datei abgeglichen[Tri]:

1. A teilt a in gleichgroße Blöcke auf.
2. A berechnet die Checksumme für jeden Block.
3. A sendet Checksumme nach B
4. B vergleicht b mit den Checksummen von a und tauscht wenn nötig geänderte Blöcke

Der Algorithmus beschränkt den Netzwerkverkehr auf das Nötigste, indem nur die geänderten Blöcke übertragen werden. Je größer eine Datei ist, desto effektiver ist der Abgleich mit rsync.

Eine Implementierung von rsync ist das gleichnamige Programm für Unix-Systeme. Listing 3.1 zeigt einen möglichen Aufruf von rsync. Parameter *a* fasst mehrere Optionen zusammen.

```
1    rsync -a /boot/grub/ /data/backup
```

Listing 3.1: Der Inhalt von Ordner *grub* wird vollständig in den Ordner *backup* repliziert

Ein Ausfall des Rechners mit den Originaldaten bringt das System zum Stillstand. Die Backup-Daten müssen dann wieder zurück gespielt werden. rsync allein ist nicht geeignet, um Anwendungen im Bereich Industrie 4.0 zu bedienen. Wichtige Kriterien wie Verfügbarkeit, Skalierbarkeit oder Fehlertoleranz sind nicht möglich. Aber häufig wird der Algorithmus in Kombination mit anderen Strategien für Synchronisierungslösungen genutzt.

3.2. SyncML

SyncML ist sowohl ein plattform- als auch ein transportunabhängiger Standard, der die Datensynchronisation zwischen vielen verschiedenen netzwerkfähigen Geräten wie Smartphones und Computern erleichtern soll. Am häufigsten wird SyncML für den Datenabgleich von Adressbuch, Kalender und E-Mail in Mobilgeräten verwendet. Hierfür muss sich der Client mit einem gültigen Benutzernamen und Passwort mit einem Webserver verbinden. Das Ablaufschema auf Abbildung 3.1 zeigt einen Synchronisationsprozess mit folgenden SyncML-Teilnehmern:

- Der *Sync Client Agent* startet den Synchronisierungsvorgang und verwaltet die Übertragungsvorgänge auf Seiten des Clients.
- Der *Sync Server Agent* wartet auf Anfragen vom Client und verwaltet die serverseitigen Übertragungsvorgänge.
- Die *Sync Engine* analysiert die Nachrichten auf Veränderungen und aktualisiert entsprechend die Datenbank. Darüber hinaus muss die *Sync Engine* Konflikte entdecken und lösen.

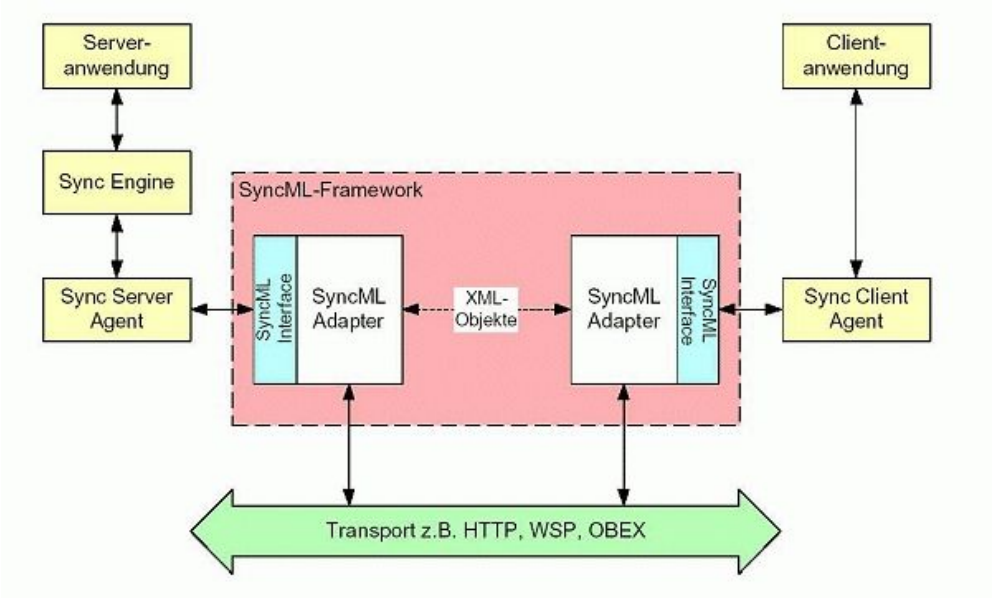


Abbildung 3.1.: Ablauf der Synchronisierung zwischen Server und Client
Quelle: [Man03]

Die SyncML-Syntax basiert auf den XML-Standard und unterstützt die Topologien *one-to-one*, *many-to-one*, *many-to-many* und Hybridlösungen aus *many-to-one* und *many-to-many*.

Zusammenfassung

SyncML wurde entwickelt, um mobile Geräte zu synchronisieren. Dabei liegt der Schwerpunkt auf Daten wie Kontakt- und Kalenderdaten, Aufgaben und Notizen und nicht für unstrukturierte Daten oder gar Big Data. Durch den Einsatz eines *Sync Servers* gilt hier die Gefahr des *Single Point of Failure*. Dadurch ist SyncML nicht für Industrie 4.0 geeignet.

3.3. Cloud-Dienst Seafile

Zitat: "There is NO CLOUD, just other people's Computer."[1]

Cloud-Dienste sind heutzutage in aller Munde. Ob nun Dropbox, iCloud, Telekom-Mediacenter, Onedrive und viele mehr. Diese Dienste speichern ihre Daten in der Cloud. Einen entscheidenden Nachteil haben die Cloud-Lösungen: Man hat theoretisch nicht mehr die alleinige Kontrolle über seine Daten. Whistleblower Edward Snowden warnt explizit vor Cloud-Diensten, da der Datenschutz für die Anbieter kein vorrangiges Ziel ist.[Kli] Ausnahmen sind Anbieter die das *Zero-Knowledge-Prinzip* garantieren. Aber was passiert mit den Daten, wenn der Anbieter Probleme hat? Sind die Daten wirklich verfügbar oder vielleicht sogar hochverfügbar? Wie viel Einfluss hat der Staat auf einen Cloud-Anbieter? Wie sicher sind die Daten in der Cloud? Diese Fragen sind für Unternehmen noch bedeutender als für Privatleute. Werden hohe Investitionen getätigt um den Schritt zur Smartfactory zu gehen, können Datendiebstahl und Datenmanipulation einen noch größeren Schaden anrichten.

Um die Unternehmensdaten selber zu verwalten, bietet sich die Möglichkeit der eigenen Cloud. Einer der bekanntesten Anbieter ist Owncloud. Hierfür wird ein Webserver, PHP und eine Datenbank benötigt. Die Einrichtung und der Betrieb ist, abgesehen von eventuell erforderlicher Hardware, kostenlos. In dieser Arbeit wird statt Owncloud die Software Seafile vorgestellt. Zum einen verwendet Seafile E2E-Verschlüsselung, was den Mitarbeitern erlaubt ihre Daten verschlüsselt auf dem Server abzulegen. Zum anderen ist die Software quelloffen und damit sind keine Hintertüren *Backdoor* zu erwarten.

Neben der Verschlüsselung und dem Open-Source-Konzept zeichnet sich Seafile durch einen einfachen Aufbau aus. Durch den Einsatz der Load Balancer Schicht (siehe Abbildung 3.2) wird, auch bei einer hohen Anzahl von Anfragen, eine erhöhte Verfügbarkeit geschaffen. Möchte man aber ein hochverfügbares System aufbauen muss der Backend-Storage ebenfalls entsprechende Eigenschaften aufweisen. Damit würde sich der Einsatz eines DFS als Speicherlösung anbieten. Seafile alleine reicht also nicht aus, um Hochverfügbarkeit zu garantieren.

[1] Free Software Foundation Europe

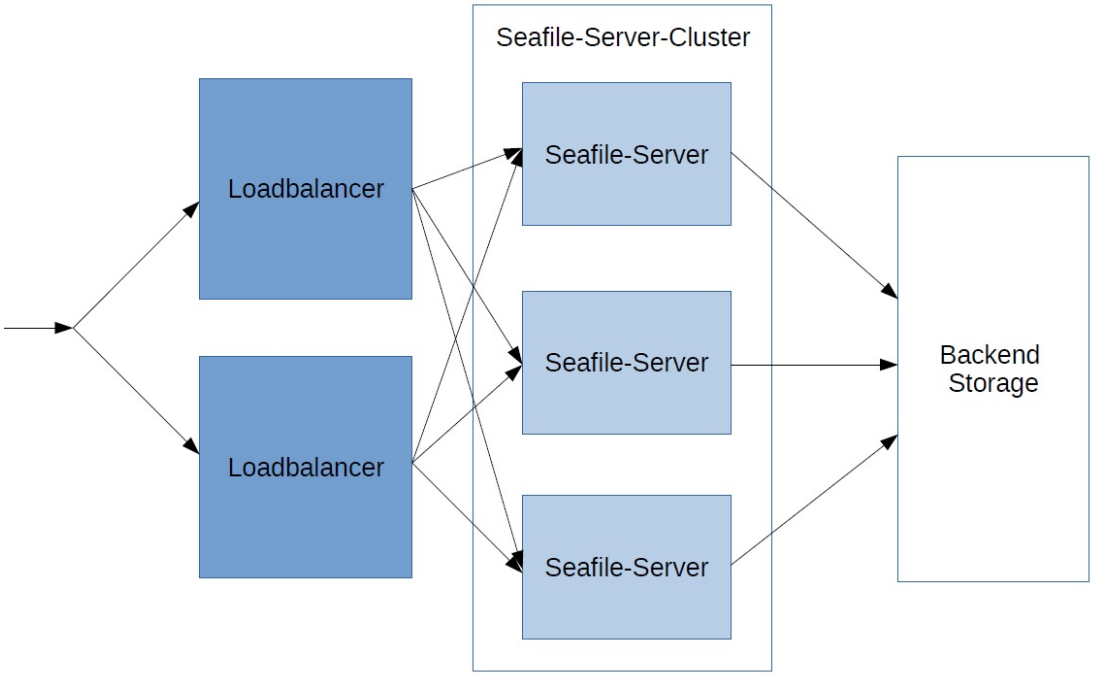


Abbildung 3.2.: Beispiel für einen 3-stufigen Aufbau eines Seafile-Server-Cluster

Zusammenfassung

Mit Seafile als Cluster-Lösung ist es möglich, eine hochverfügbare Synchronisierungslösung aufzubauen. Dafür ist ein redundantes Dateisystem, wie z.B. GlusterFS (Kapitel 3.5), als Grundlage für den Backend-Storage nötig und eine Load Balancer Schicht, um Anfragen auf die Server zu verteilen. So ist es möglich, mobile Endgeräte per Client (Android, Windows oder IOS) auf einfache Art und Weise in das System einzubinden.

3.4. Syncthing - Datensynchronisierung mit P2P-Übertragung

In der klassischen Client-Server-Struktur muss der Server alle Anfragen der Clients bearbeiten. Durch fehlerhafte Clients (sendet ununterbrochen) oder durch aktiv hervorgerufene Angriffe durch DoS-Attacken kann der Server soweit ausgelastet werden, dass er die Anfragen der Clients nicht mehr entgegennehmen kann. Auch im Normalfall kann der Server überlastet werden, wenn das System nicht skaliert. Zu viele Clients führen zu Verzögerungen der Anfragen. In diesem Fall müsste die Rechenleistung des Servers erhöht oder ein weiterer hinzugefügt werden, um die Verfügbarkeit zu gewährleisten. Darüber hinaus ist der Server in dieser Archi-

tektur der SPoF und ein Ausfall bedeutet das Komplettversagen des Systems. Hier geht das P2P-Netzwerk einen anderen Weg. Die einzelnen Peers übernehmen sowohl Client- als auch Serverfunktionalitäten und kommunizieren bilateral als gleichberechtigte Partner. Diese Art der Netze soll adaptiv und selbstkonfigurierend sein. Ziel ist es, die verteilten Ressourcen der Peers im Netz aufzufinden und zu nutzen.[Cht+] Vergleicht man die Client-Server- und die P2P-Architektur ergeben sich folgende Vorteile für die P2P-Kommunikation:

- symmetrisch und geringere Netzlast, da Peers direkt untereinander kommunizieren
- kein Server der Flaschenhals (eingeschränkte Skalierbarkeit) oder SPoF darstellt
- Peers können sich zu jeder Zeit am Netz anmelden und es auch wieder verlassen
- sehr gute Skalierungs- und Verfügbarkeitseigenschaften

Dadurch, dass die Daten im Netz verteilt sind und nicht auf einem Server lagern, reduziert sich die Gefahr, dass Daten verloren gehen[Cht+]. Allerdings kann es auch in P2P-Netzen Knoten geben, die eine besondere Rolle einnehmen, oder gar einen Server, der den Aufenthaltsort der jeweiligen Ressourcen verwaltet. Deswegen kann man die Netze unterteilen in:

- Zentrale P2P-Systeme: Ein Server hält eine Übersicht der Inhalte der Peers vor. Die Peers stellen Suchanfragen an den Lookup Server und tauschen dann die Daten direkt untereinander aus.
- Reine P2P-System: Sowohl Ressourcennutzung als auch die Organisation geschieht dezentral. Alle Knoten sind gleichgestellt. Das Auffinden von bestimmten Peers wird hier durch eine *Distributed Hash Table* (DHT) geregelt.
- Hybride P2P-Systeme: Bestimmte Knoten (Supernodes oder Superpeers), meist mit höherer Bandbreite und Rechenleistung, übernehmen dynamisch gesonderte Funktionalitäten.

Eine der Hauptaufgaben von P2P ist File-Sharing. In einem betrieblichen Umfeld kann durch den Einsatz von File-Sharing-Anwendungen auf kostenintensive und zentrale Massenspeicher-Lösungen verzichtet werden. Dadurch ergeben sich aber auch Nachteile, die P2P mitbringen. Sind die Daten in einem *heterogenen System* verteilt, sollten diese auch nur autorisierten Personen zugänglich sein. Um Peers untereinander kommunizieren zu lassen werden häufig die Sicherheitsmechanismen wie Firewalls umgangen und somit eine Schwachstelle geöffnet. P2P-Anwendungen müssen die Kriterien Authentifizierung, Autorisierung, Verfügbarkeit, Datenintegrität und Vertraulichkeit zur Verfügung stellen[WCb].

Daten-Sharing muss nicht der einzige Nutzen von P2P-Netzen sein. Andere Ressourcen wie CPU oder Arbeitsspeicher können von Rechnern mit leistungsfähigerer Hardware über das Netz zur Verfügung gestellt werden.

***Beispiel**: Ein leistungsschwacher Sensorknoten möchte eine aufwändige Berechnung seiner über einen gewissen Zeitraum aufgenommenen Daten durchführen, um das Ergebnis weiterzureichen. Dabei kann er die Berechnungsaufgabe an die anderen Teilnehmer im Netzwerk delegieren und erhält das Ergebnis in einer kürzeren Zeit zurück. Währenddessen kann der Knoten seine normale Arbeit fortführen.*

Eine Synchronisationssoftware, die das P2P-Netzwerk nutzt, ist Syncthing. Verzeichnisse lassen sich über verschiedene Rechner synchronisieren, wobei es keine Rolle spielt, ob die Verbindung über das Internet geschieht. Da es keinen zentraler Server gibt fällt das Problem Single Point of Failure weg. Syncthing bietet folgenden Eigenschaften:

- Einfacher Abgleich von Verzeichnissen über das Netzwerk
- Knoten können ohne Austausch von IP-Adressen über das Internet kommunizieren
- Synchronisation ohne zentralen Server
- Administration über Webinterface
- Kommunikation ist mit *Transport Layer Security* (TLS) und *Perfect Forward Secrecy* (PFS) gesichert
- basiert auf Blocks Extensible Exchange Protocol
- Open-Source/Quelloffen
- Unterstützt Linux – auch für ARMv5, ARMv6, ARMv7–, FreeBSD, Windows, Mac OS X
- *Universal Plug and Play* (UPnP) um aus dem Internet auf die Daten zu zugreifen

Zusammenfassung

Mit Syncthing ist es mit einfachen Mitteln möglich, eine Datenreplikation in einem System zu etablieren. Dadurch, dass alle Knoten an der Replikation beteiligt sind, sind die Daten auch entsprechend hochverfügbar. Übersteigen die Daten ein bestimmtes Volumen, müssen die Peers skalieren. Das geht aber nur mit einer vertikalen Skalierung und ist entsprechend

umständlich für alle Peers durchzuführen. Möchte man für zukünftige Herausforderungen gewappnet sein, bietet sich Syncthing deshalb nicht an. Wird beispielsweise mehr Speicherplatz benötigt, müssen alle Knoten mit neuen Festplatten erweitert werden. Syncthing ist deswegen nicht für den Einsatz im Bereich Industrie 4.0 geeignet.

3.5. Datencluster mit GlusterFS

Hardware

GlusterFS ist ein Open Source Distributed Filesystem und soll mit geringem Aufwand sowie einfacher Consumer Hardware eine hoch skalierbare und hochverfügbare Speicherlösung ermöglichen.

Skalierbarkeit

Sowohl der Speicherplatz als auch die Schreibperformance kann über das Hinzufügen bzw. das Entfernen von Knoten beeinflusst werden.

kein SPoF

Im Gegensatz zu vielen anderen skalierbaren Systemen existiert bei GlusterFS kein SPoF, der die Verfügbarkeit des Systems negativ beeinflussen kann.

Geo-replication

Mit Geo-Replication unterstützt GlusterFS ein Feature mit dem es möglich ist, verschiede geographisch getrennte Orte dem Speichercluster hinzuzufügen. Diese Standorte können über Local Area Networks (LANs), Wide Area Network (WANs) und über das Internet verbunden sein. Im Fehlerfall können so Daten immer noch gespeichert werden. Sobald der Fehler beseitigt ist, werden die Daten automatisch synchronisiert.[Glua]

heterogene Dateisysteme

Freigegebener Speicher kann aus verschiedenen Speicherorten mit unterschiedlichen Dateisystemen (Heterogenität) bestehen. Nach außen verhält sich der freigegebene Speicher wie ein normales Dateisystem. Zudem kann der Speicher für nahezu jedes Betriebssystem genutzt werden.

modulare Funktionen

Mit Hilfe von Modulen, den sogenannten Translatoren, können dem Cluster Zusatzfunktionen hinzugefügt oder das Verhalten geändert werden.

Architektur

Bei GlusterFS kommt kein zentraler Metadaten-Server zum Einsatz. Stattdessen werden die Daten mit Hilfe des Elastic Hash Algorithmus (Abbildung 3.3) verteilt. Die Daten können sowohl über *Infiniband* als auch Ethernet verteilt werden.

Freier Festplattenspeicher auf Linux-Rechnern, den sogenannten Bricks, bildet die Grundlage für einen Cluster. Das können RAID-Arrays, einzelne Festplatten oder auch Verzeichnisse sein. Die Bricks lassen sich in verschiedene Konfigurationen zusammenschließen.

Distributed

Wenn eine gute Skalierbarkeit gewünscht ist und die Datenredundanz keine Rolle spielt

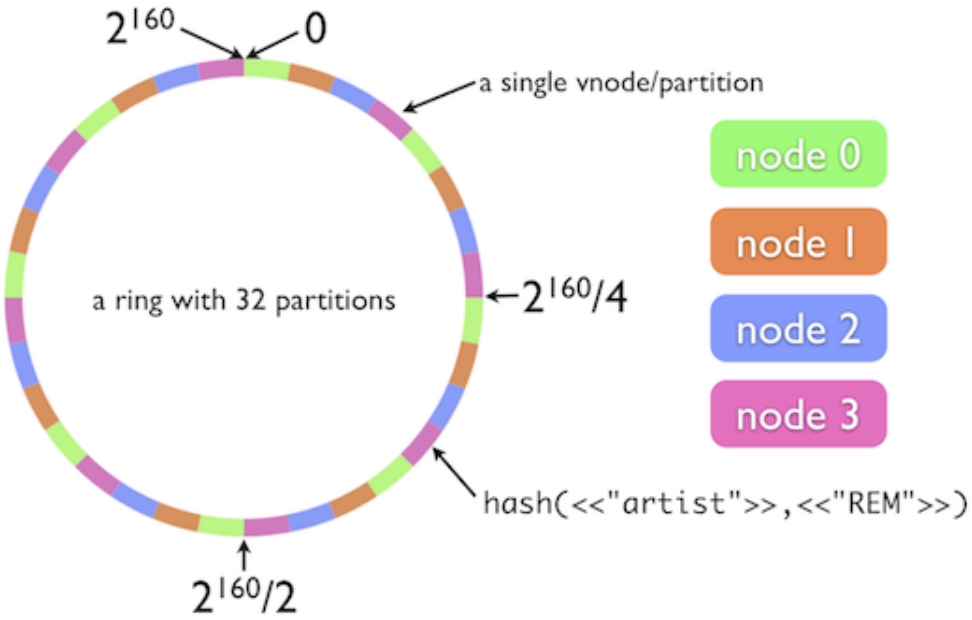


Abbildung 3.3.: Daten werden mit Hilfe einer Hash-Funktion auf die Nodes verteilt
Quelle: [Mey]

wird die Konfiguration *Distributed* eingesetzt. Hierbei werden die Dateien zufällig über die im Cluster enthaltenden Bricks gespeichert[Glua].

Replicated

Die zweite Möglichkeit Bricks zusammenzuschließen ist der Modus *Replicated*, der besonders in Umgebungen mit hohen Anforderungen an Verfügbarkeit und Zuverlässigkeit zum Einsatz kommt. Hier werden die Daten redundant auf mehrere Knoten gespeichert.

Striped

Kommt es auf Performance an, weil z.B. mehrere Clients gleichzeitig auf große Datenmengen zugreifen, ist der Einsatz vom *Striped*-Modus sinnvoll, da hier einzelne Dateien aufgeteilt und die einzelnen Teile auf die Bricks verteilt werden.

Um die Eigenschaften (Speicherskalierung, Geschwindigkeit, Redundanz) verschiedener Modi zu nutzen, kann man diese, wie bei dem Distributed-Striped- oder Distributed-Replicated-Modus, auch kombinieren.

Dispersed

Mit dem Dispersed-Modus ist noch eine weitere Konfiguration möglich. Hierbei handelt es sich um einen Modus, der sich den *Erasure Code* zu Nutze macht, um die Daten mit zusätzlichen Informationen anzureichern. Damit lassen sich beschädigte Daten rekonstruieren oder auch ausgefallenen Knoten kompensieren.

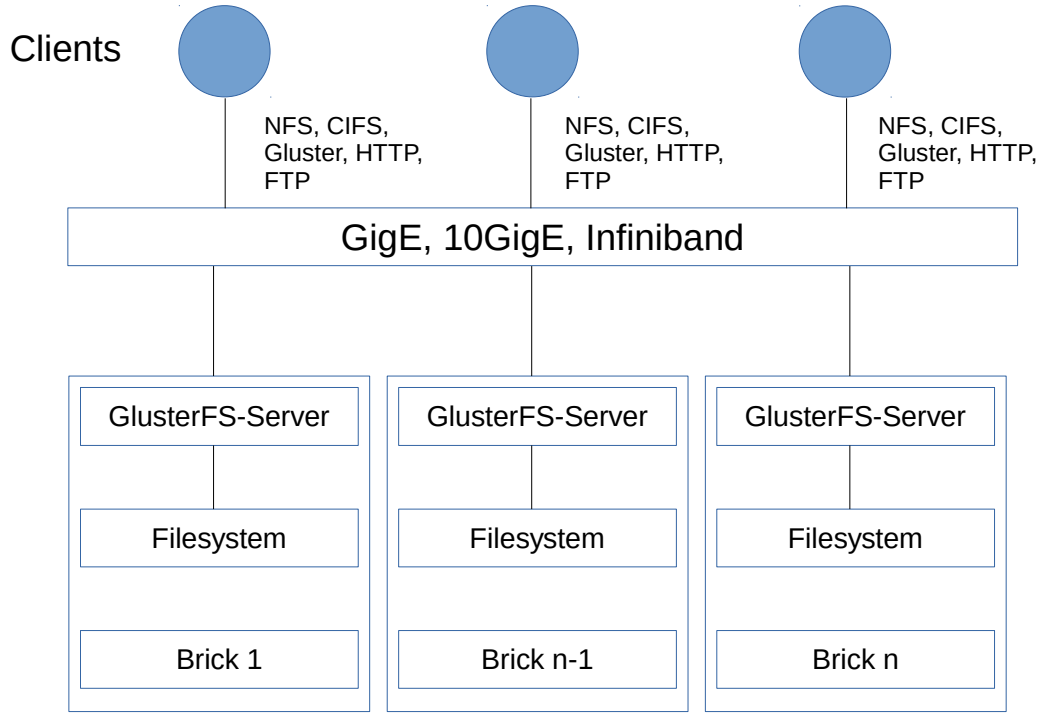


Abbildung 3.4.: Vereinfachte Architektur von GlusterFS
Quelle: [Dok]

***Beispiel**: In einem Cluster mit drei Knoten und einem Wert für die Redundanz von 1 stehen aufgrund der Zusatzinformationen lediglich* 66.7 %*[Glua] der eigentlichen Kapazität zur Verfügung. Das entspricht einer "1 aus 3"-Konfiguration (EC 1/3). Es kann ein Knoten ausfallen ohne das ein Datenverlust entsteht.*

Erasure Code ist von Nachteil, wenn Daten gelesen, bearbeitet und wieder zurück geschrieben werden müssen. Dieser Vorgang beinhaltet, die Fragmente von den Bricks zu sammeln, zusammenfügen, zu bearbeiten und wiederum in Blockgröße auf die Bricks zu verteilen. Das wirkt sich negativ auf die Latenz und die Performance aus.

Der Dispersed-Modus lässt sich zwecks Speicherplatzskalierung mit dem Distributed-Modus kombinieren.

Bei GlusterFS können Funktionalitäten, wie z.B. die Netzwerkkommunikation, Clustering oder Load Balancing, durch die so genannten Translatoren bereitgestellt werden. Mit Hilfe der

Translatoren ist es darüber hinaus möglich, die Bricks mit zusätzlichen Funktionen (z.B eine POSIX-Schnittstelle) auszustatten.

Der Zugriff auf GlusterFS-Storages kann über einen nativen Client (glusterfs), NFS, CIFS, SMB, HTTP/REST oder auch per Hadoop-Plugin erfolgen.

Zusammenfassung

Speichercluster mit Commidity-Hardware und ohne SPoF? Das ist mit GlusterFS möglich. Hinzu kommen Features wie Geo-replication, verschiedene Betriebsmodi und praktisch beliebige Skalierung des Speicherplatzes. Da eine GlusterFS-Speicherfreigabe als Dateisystem transparent erscheint, ist es zudem einfach zu handhaben.

3.6. Das Hadoop Framework

Hadoop ist ein Open Source Projekt (Hadoop Framework) von der Apache Software Foundation und umfasst mehrere Teilprojekte. Mit Hilfe dieses Framework soll es möglich sein, zwei der vier V von Big Data, nämlich Velocity und Volume, zu beherrschen. Dafür werden die Daten auf Commodity-Hardware-Cluster verteilt gespeichert und parallel verarbeitet. Dadurch, dass man Knoten im laufenden Betrieb zu dem System hinzufügen kann, wird eine horizontale Skalierung erreicht. Für diese Arbeit relevante Teilprojekte sind Map Reduce, HDFS (Kapitel 3.6.1) und HBase (Kapitel 3.6.2).

Ein Schwerpunkt von Hadoop ist das Konzept der Datenlokalität. Anders als bei vielen anderen Anwendungen wird bei einem Auftrag der Programmcode an die erforderlichen Cluster verteilt. Dadurch entfällt das Verschicken von großen Datenmengen über das Netzwerk. Die Knoten führen Berechnungen auf die ihnen zugewiesenen Daten aus.

Bei Map Reduce handelt es sich um einen Algorithmus und wird im nächstem Abschnitt betrachtet.

Map-Reduce-Algorithmus

2010 musste Facebook eine Datenmenge von 21 Petabyte verwalten. Ein Jahr später waren es 30 Petabyte.[HPS] Ohne eine hochgradige Parallelisierung der Daten wäre eine akzeptable Bearbeitungszeit nicht möglich. Der Map-Reduce-Algorithmus dient dem Speichern und Verwalten von Datenmengen im Petabyte-Bereich und wird in Apache Hadoop implementiert. Genau genommen handelt es sich bei MapReduce um ein Softwareframework, das entwickelt wurde, um großen Datenmengen, die unter anderem entstehen, wenn Daten automatisch erfasst werden, in akzeptabler Zeit zu verarbeiten. Dabei werden die Plattenspeicher mehrerer Rechner

zu einem *Cluster* logisch zusammengefasst. Eine Datei wird in einzelne Blöcke aufgeteilt, um sie dann, verteilt auf die Rechner im Cluster, zu speichern. Bei Zugriffsoperationen können die Blöcke parallel auf den jeweiligen Knoten ausgeführt werden. Damit kann eine deutliche Zeitreduzierung bei Zugriffsoperationen erreicht werden.

Bei MapReduce ist der Datenfluss in zwei Phasen unterteilt (siehe Abbildung 3.5). Als Beispiel soll eine Abfrage dienen, die Anzahl der vorkommenden Wörter einer Textdatei zählt. Die Datei ist auf vier Knoten verteilt.

1. **Map-Phase**: Der Mapper verarbeitet jeweils seine Eingabedaten. Das Ergebnis wird als Tupel an die Reduce-Phase weiter geleitet.

2. **Reduce-Phase**: Die von den Mappern gelieferten Zwischenergebnisse werden gruppiert. Das Ergebnis ist wiederum ein Tupel.

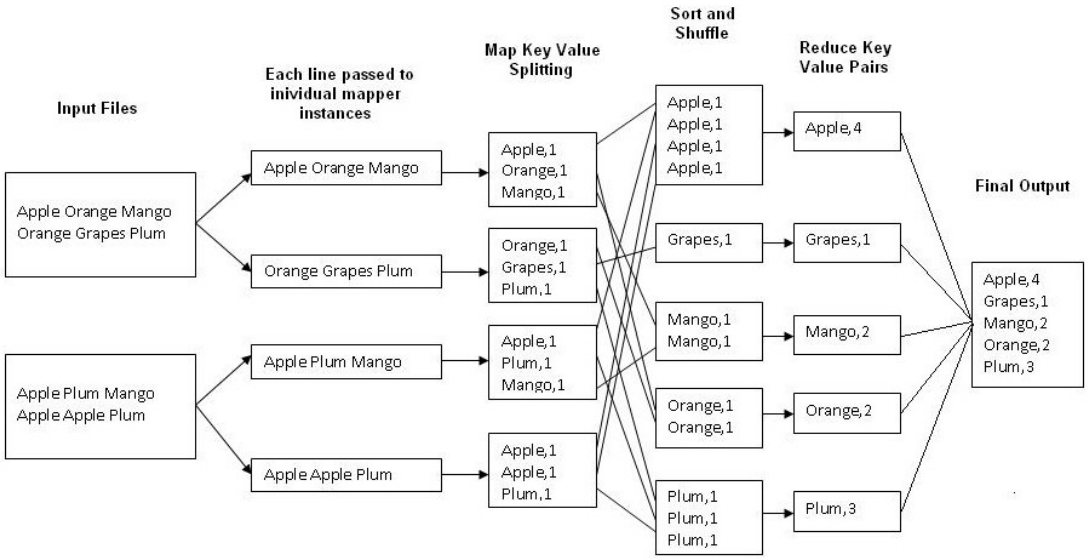


Abbildung 3.5.: WordCount-Beispiel: Bei MapReduce werden die Aufgaben parallel verarbeitet. Quelle: [KS]

MapReduce eignet sich, um in einem verteilten System Berechnungen auf unstrukturierte Daten (Dateien) aufzuteilen.

3.6.1. HDFS

Neben der Möglichkeit Daten in relationalen Datenbanken abzulegen lassen sich diese auch als Dateien (sogenannte Flat-Files) auf einem Dateisystem speichern. Derzeit sind zahlreiche

verteilte Dateisysteme verfügbar. GlusterFS ist eines davon und wird in Kapitel 3.5 vorgestellt. Neben GlusterFS wird im Rahmen dieser Arbeit das Dateisystem HDFS vorgestellt, da es als Grundlage für das verteilte Datenbanksystem HBase dient (Kapitel 3.6.2).

Ein HDFS-Cluster kann aus Hunderten oder Tausenden von Rechnern bestehen, die durchaus günstige Standard-PC-Hardware beinhalten können. Damit sind Hardwarefehler die Regel. HDFS ist darauf ausgerichtet diese Fehler zu erkennen und die Verfügbarkeit zu erhalten. Speziell für dieses Dateisystem entwickelte Anwendungen greifen auf Dateien sequentiell zu, erreichen einen hohen Datendurchsatz und sind in der Lage auf riesigen Datenmengen zu arbeiten. Daten, die einmal geschrieben wurden, können nicht mehr verändert werden.

HDFS ist kein normales Dateisystem, sondern für Map-Reduce-Anwendungen (Kapitel 3.6) optimiert.

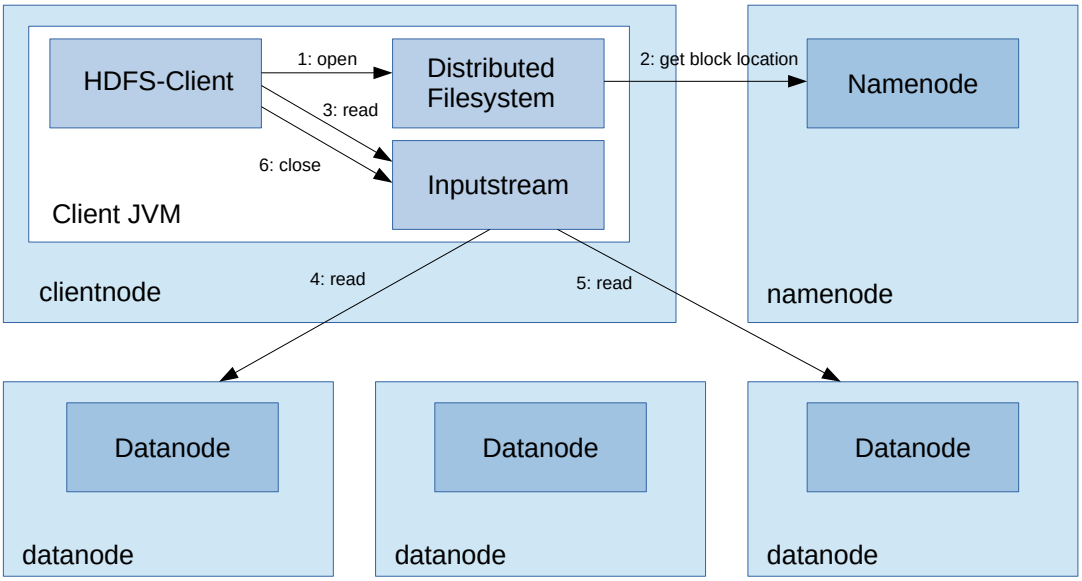


Abbildung 3.6.: Lesevorgang eines Clients in HDFS

Architektur von HDFS

HDFS verfügt über eine Master-Slave-Architektur, bei dem in einem Cluster der NameNode als Master fungiert. Dieser verwaltet das Dateisystem, den Namensraum, den Zugriff auf die Clients und führt Dateisystemoperationen wie z.B. Öffnen, Schließen oder Umbenennen aus. Ein oder mehrere Slaves/DataNodes können auf einem Knoten laufen und verwalten den am Knoten zur Verfügung stehenden Speicher. Eine Datei wird in mehrere Blöcke aufgeteilt und mit Hilfe des NameNode auf verschiedenen DataNodes verteilt (siehe MapReduce Kapitel 3.6).

Die DataNodes nehmen Anfragen des NameNode entgegen und führen Operationen auf den einzelnen Blöcken aus.

Der NameNode speichert seine Informationen im FSImage, das komplett im Hauptspeicher liegt. Alle Änderungen werden im EditLog des NameNode abgelegt. Startet der NameNode neu, liest er den letzten Stand des FSImage und führt alle Änderungen aus dem EditLog am FSImage aus. Die aktualisierte Version des FSImage wird dann abgespeichert.

Dateioperationen

Hadoop bietet Anwendungen an (meist über Kommandozeile), die dabei helfen Dateioperationen zwischen dem lokalen Dateisystem und HDFS auszuführen, Lese- und Schreibrechte der Dateien zu ändern oder neue Verzeichnisse im HDFS anzulegen bzw. zu löschen. Hierzu ein paar Beispiele von dem Programm *hdfs* aus [Fre14]:

```
hdfs dfs -copyFromLocal QUELLE ZIEL
hdfs dfs -copyFromLocal /usr/input/test.txt /hdfs/input/test.txt
```

Listing 3.2: Eine Datei vom lokalen Dateisystem zu HDFS kopieren

```
hdfs dfs -copyToLocal QUELLE ZIEL
hdfs dfs -copyToLocal /hdfs/input/test.txt /usr/input/test.txt
```

Listing 3.3: Eine Datei von HDFS zum lokalen Dateisystem kopieren

```
hdfs dfs -ls ORDNER
hdfs dfs -ls /
```

Listing 3.4: Inhalt eines Ordenrs im HDFS auslisten

```
hdfs dfs -cp QUELLE ZIEL
hdfs dfs -cp /hdfs/input/test.txt /hdfs/input/test2.txt
```

Listing 3.5: Datei innerhalb von HDFS kopieren

Fehlertoleranz

Jeder DataNode sendet periodisch eine *Heartbeat*-Nachricht an den NameNode. Knoten, die keine Heartbeat-Nachrichten schicken, werden vom NameNode als tot markiert und stehen HDFS nicht mehr zur Verfügung. Der NameNode leitet keine IO-Operationen an ausgefallene Knoten. Da der Replikationsfaktor durch den Ausfall sinkt, versucht der NameNode die betroffenen Blöcke wieder zu replizieren.

Die Checksummen-Prüfung stellt sicher, dass bei Speicher-, Netzwerk- oder Softwarefehlern beschädigte Blöcke erkannt werden. Wird eine Datei auf einem Client erstellt, wird die Checksumme der einzelnen Datenblöcke berechnet und in einer versteckten Datei im HDFS-Namespace abgelegt. Wenn ein Client eine Datei abruft berechnet er die Checksumme der empfangenen Datei. Stimmen die neu berechnete Checksumme und die im Namespace abgelegte Checksumme nicht überein, kann der Client die fehlerhaften Blöcke von anderen DataNodes abrufen.

Vor- und Nachteile

Big-Data

Durch das Aufteilen und verteilte Speichern der Daten dient das HDFS-Dateisystem als Grundlage für die Datenverarbeitung bis in den Petabyte-Bereich und erfüllt somit die Anforderungen von Big-Data.

Code-to-Data

Mit Hilfe des *Code-to-Data-Prinzip* müssen nicht große Mengen an Daten bei einer Auswertung über das Netzwerk übertragen werden, sondern nur das Ergebnis.

Fehlertolerant

Die für die Datenspeicherung zuständigen DataNodes können ausfallen, ohne dass die Funktionsweise des Systems beeinträchtigt wird.

Auslastung

Der NameNode bildet die zentrale Instanz im HDFS-Datesystem und sorgt dafür, dass die DataNodes gleichmäßig ausgelastet werden. Dieser NameNode stellt aber gleichzeitig eine Schwachstelle in diesem System dar. Fällt er aus ist das komplette System funktionsunfähig.

SPoF

Map-Reduce

Um Dateioperationen auszuführen muss der Map-Reduce-Algorithmus in Java implementiert werden. Das bedeutet einen erhöhten Aufwand, macht den Einstieg in HDFS komplizierter und erschwert die Integration in bestehende Systeme.

Overhead

Je kleiner die zu speichernde Datenmenge, desto höher ist der relative Anteil an Overhead durch z.B. Metadaten. Sollten also ausschliesslich kleine Dateien gespeichert werden, ist der Einsatz von HDFS zu überdenken.

Echtzeit

So geeignet HDFS für Big-Data ist, so ungeeignet ist es für Echtzeitanalysen. Dieser Umstand wiegt sehr schwer, da es in der Industrie unumgänglich ist Daten in Echtzeit auszuwerten.

Zusammenfassung

HDFS wurde für Big-Data-Anwendungen entworfen und bietet die Möglichkeit, große Datenmengen hochgradig parallel zu verarbeiten. Das Map-Reduce-Programmiermodell ist die Grundlage für das Code-to-Data-Prinzip. Dadurch kann das Ergebnis schneller berechnet und das Netzwerk entlastet werden, da nur das Ergebnis bei Berechnungen verschickt wird. Durch die Aufteilung der Daten auf die Knoten wird zudem das Netzwerk gleichmäßiger ausgelastet.

Durch den MapReduce-Algorithmus ist das HDFS-Filesystem für große und schnell wachsende Daten geeignet. Allerdings eignet sich das Framework nicht, um Daten in Echtzeit zu analysieren. Dafür ist die Performance nicht ausreichend. Bei der Auswahl für eine bestimmte Aufgabe ist das zu berücksichtigen.

3.6.2. Apache Hadoop Database (HBase)

HBase ist eine Open-Source-NoSQL-Datenbank und baut auf Hadoop Filesystem (HDFS siehe Kapitel 3.6.1) auf. Sie eignet sich gut für den wahlfreien Lese- und Schreibzugriff auf sehr große Datenbestände.

Wie von relationalen Datenbanken bekannt, speichert auch HBase die Daten in Tabellen (Tables), Zeilen (Rows) und Spalten (Columns). Jede Row besitzt ein beliebiges Byte-Array als Schlüssel (Row Key), welcher innerhalb der Tabelle eindeutig sein muss. Jede Spalte gehört zu einer Spaltenfamilie (Column Family) und besitzt einen Schlüssel (Column Qualifier), der innerhalb der Column Family eindeutig sein muss. Beim Einfügen von Spaltenwerten, wird zur Versionierung der Informationen automatisch ein Zeitstempel eingefügt.

Anders als bei relationalen Datenbanken wird bei HBase kein Datenbankschema benötigt. Tabelle 3.1 zeigt ein Beispiel, um Informationen für Personen zu speichern. Person1 hat keine Werte für die Spalte *Property* und für Person2 sind für verschiedene Zeitpunkte Werte in der Datenbank. Ein Datensatz wird mit Hilfe von *Key* und *Timestamp* identifiziert. Wird bei einer Abfrage keine Angabe zur Version gemacht, liefert HBase den aktuellsten Wert.

Row Key	**Timestamp**	**Column Family ID**	**Column Family Property**
row1	ts1	ID:Email ”chef@jofre.de”	
row2	ts2	ID:Name ”Rene K.”	Property:Job ”Consultant”
	ts3	ID:id ”123456”	
	ts4	ID:Name ”Jon”	Property:Hobby ”Schach”

Tabelle 3.1.: HBase-Tabelle ”Personen”

In der Tabelle 3.1 sind leere Felder. Bei HBase werden diese Felder nicht mitgespeichert und verbrauchen demnach auch keinen Speicherplatz.

Um mit normalen SQL-Statemants auf HBase zuzugreifen empfiehlt sich der Einsatz von Apache Phoenix, das als Datenbank-Layer direkt auf HBase aufbaut. Damit werden SQL-Statements in HBase-Scans übersetzt.

3.7. MySQL Cluster

MySQL Cluster ist ein skalierbares, echtzeitfähiges und ACID konformes Datenbanksystem, das eine Verfügbarkeit von 99.999 % gewährleisten soll. Durch verteilte Knoten wird ein single point of failure vermieden. Zudem skaliert MySQL Cluster horizontal mit Mainstream-Hardware. Folgende Eigenschaften werden bereitgestellt:

- Hochverfügbarkeit von 99.999 %
- Horizontale Skalierbarkeit
- Hohe Performance
- Niedrige Kosten durch Einsatz von Mainstream-Hardware
- Interfaces für SQL und NoSQL (Key-Value)
- Synchrone Replikation
- Auto-sync beim Hochfahren eines Datenknotens

Architektur

Das Datenbanksystem beinhaltet drei Typen von Knoten:

- **Storage Node**:
  - speichern Daten
  - synchrone Replikation zwischen Nodes einer Gruppe
  - nutzen *Shared-Nothing-Architektur*, mit Ausnahme der Datenpartitionierung
- **Management Server Node**: Dient zur Konfiguration und wird nur beim Starten und bei Änderungen der Konfiguration benötigt. Die Storage Nodes arbeiten ohne Management Server Node.
- **MySQL Server Node**: Anwendungen greifen mit Hilfe von MySQL Server auf die Storage Nodes zu. Die MySQL Server sind mit allen Storage Nodes im Cluster verbunden.

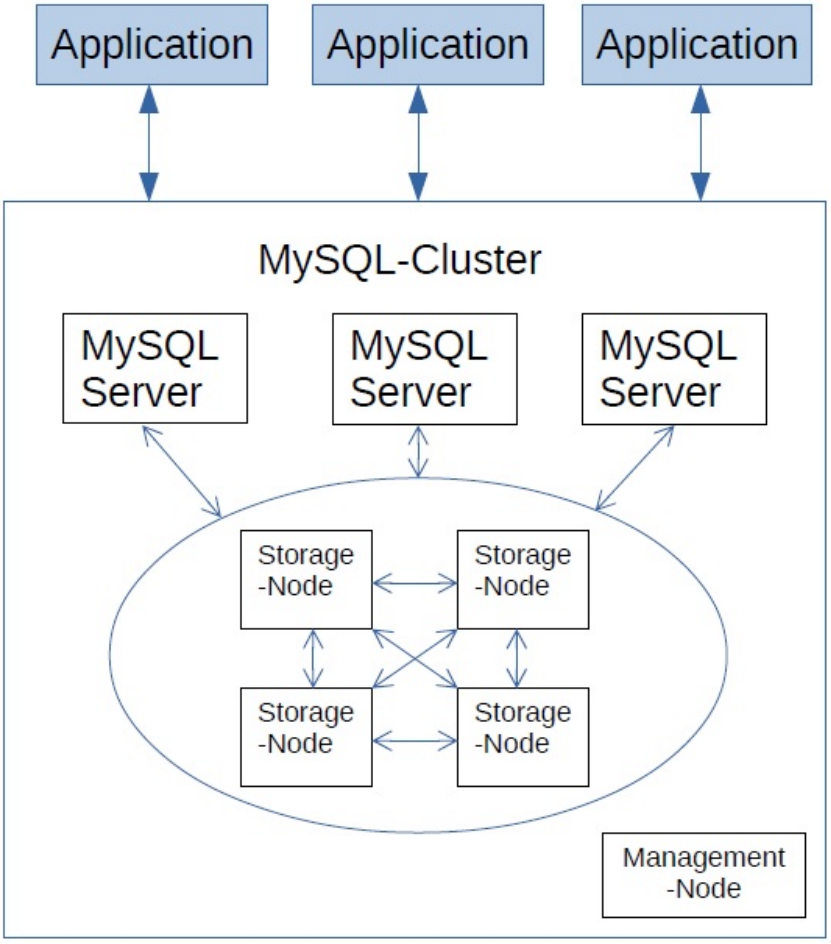


Abbildung 3.7.: Architekturbeispiel für ein MySQL Cluster

Fehlererkennung

Es gibt zwei mögliche Wege einen fehlerhaften Knoten zu erkennen. Sowohl bei einem Verbindungsabbruch als auch beim Ausbleiben der Heartbeat-Nachricht eines Knotens wird eine Nachricht an alle Storage Nodes geschickt und geprüft, ob noch genügend Knoten online sind, um den Cluster zu betreiben. Fehlerhafte Nodes werden automatisch neu gestartet. Für Komponenten, die nicht über die Kommunikation überwacht werden können, ist der Einsatz von Heartbeat-Nachrichten geeignet. Dies gilt für Festplatten-, Arbeitsspeicher- oder CPU-Fehler. Die in einem logischen Ring angeordneten Storage Nodes schicken jeweils ein Heartbeat-Signal an den Nachbarn.

Zusammenfassung

Mit MySQL-Cluster ist es möglich, bestehende MySQL-Lösungen um Replikation und Hochverfügbarkeit zu erweitern. Damit werden aber nicht alle Nachteile einer relationalen Datenbank behoben. Möchte man unstrukturierte Daten speichern ist dies mit einer relationalen Datenbank nicht möglich, da diese Daten keine SQL-Struktur aufweisen.

4. Anforderungen an das System

4.1. Funktionale Anforderungen (Funktionalitäten)

Replikation

Die wichtigste Eigenschaft für eine Synchronisierungslösung ist die automatische Replikation von Daten. Dabei sollte es keine Rolle spielen, ob es sich um strukturierte oder unstrukturierte Daten handelt. Beherrscht das System beide Formate ist es für die Zukunft gerüstet.

CAP

Synchronisierungskonflikte sind so weit wie möglich zu vermeiden. Wünschenswert sind dabei die ACID-Eigenschaften, die in verteilten Systemen aber nicht realisierbar sind. Dafür wurde das CAP-Theorem entwickelt, das besagt, dass in einem verteilten System zwei der drei folgenden Eigenschaften zeitgleich erfüllt werden können:

- **c**onsistency: In verteilten Systemen mit replizierten Daten muss sichergestellt sein, dass nach Abschluss einer Transaktion auch alle Replikate aktualisiert werden.
- **a**vailability: Alle Anfragen an das System werden stets beantwortet, bzw. alle Clients können jederzeit lesen und schreiben.
- **p**artition tolerance: Das System arbeitet auch bei Verlust von Nachrichten, einzelner Netzknoten oder Partition des Netzes weiter.

Die Verfügbarkeit hat einen hohen Stellenwert und sollte stets vom System eingehalten werden. Aber permanente Konsistenz und Netzwerkpartitionierung können nicht immer zeitgleich funktionieren. Tritt in einem Netzwerk ein Fehler auf und muss die absolute Konsistenz der Daten für die Clients gewährleistet werden, dann wird der Zugriff auf die Daten so lange gesperrt, bis der Fehler behoben ist. Das widerspricht jedoch dem Gedanken, dass alle Clients, auch im Falle einer Netzwerkpartitionierung, stets auf den Cluster lesen und schreiben können. Für diesen Fall lässt sich keine pauschale Aussage treffen, welche Eigenschaft zu priorisieren ist. Für die weitere Bewertung der Produkte in dieser Arbeit wird angenommen, dass eine Aufweichung der Konsistenz zu Gunsten der Fehlertoleranz sinnvoll ist. Eine Produktionsanlage kann so auch im Fehlerfall weiter produzieren und die durch die Sensoren generierten Daten in den Cluster schreiben. Eine Produktionsverzögerung kann so verhindert werden.

Authentifizierung
Autorisierung

Der Zugriff muss durch Autorisierung geregelt werden. Zusätzlich ist es erforderlich, dass

sich die Kommunikationspartner authentifizieren, idealerweise über ein Zertifikat.

Fehler

Knoten, die wegen eines Fehlers ausfallen, dürfen den Betrieb nicht gefährden. In so einem Fall müssen die Daten konsistent und verfügbar bleiben und der Fehler sollte auf geeignete Weise signalisiert werden, um eine Störungsbeseitigung zu beschleunigen.

Geo-Replikation

Um die Daten auch bei katastrophalen Ereignissen verfügbar zu machen, ist die Unterstützung einer Geo-Replikation wünschenswert. Damit kann auch eine Lokalität der Daten erreicht werden, um Firmenstandorten eine optimierte Zugriffszeit auf die Daten zu gewährleisten.

mobile Devices

Im Umfeld Industrie 4.0 könnten verstärkt mobile Geräte zum Einsatz kommen, weswegen eine Schnittstelle zu den mobile Devices zwingend nötig ist.

4.2. Nichtfunktionale Anforderungen (Qualität)

Einfachheit

Für eine reibungslose Integration in eine bestehende Anlage ist die Lösung möglichst einfach zu halten. Die Bedienung für Endanwender darf nicht kompliziert sein und die Integration sowie Administration muss unproblematisch sein.

Skalierung

Eine horizontale Skalierung ist einer vertikalen Skalierung vorzuziehen. Wird der Speicherplatz knapp, können neue Festplattenkapazitäten zu dem System hinzufügen werden, am besten ohne dass die Benutzer etwas von dieser Erweiterung mitbekommen. Hiermit ist gemeint, dass das System nicht herunter gefahren werden muss, um die Erweiterungen durchzuführen. Bei der horizontalen Skalierung von Speicher existieren zwei Kategorien. Im Kapitel 2.6 sind die Modi Mirroring und Striping erklärt. Der Vorteil beim Striping ist der Geschwindigkeitszuwachs. Mit einer Mischung aus beiden Modi lassen sich sowohl Kapazität als auch Performance eines Speicherpools verbessern.

Performance

Zwischen Synchronisierungsknoten muss der Datenabgleich so schnell wie möglich durchgeführt werden. Das setzt eine Hochgeschwindigkeitsverbindung voraus, über die die Knoten kommunizieren können.

Zitat: "Zeit ist Geld!"[Benjamin Franklin (1706-1790)]

Kosten

Ausfallzeiten sorgen in der Industrie für einen Gewinnausfall. In einer Smartfactory wird es möglich sein, eine Losgröße von eins einzurichten. Damit werden individuelle Produkte in kürzester Zeit produziert, um sie dann schnellstmöglich zum Kunden zu bringen. Ausfallzeiten spielen dadurch eine verstärkte Rolle, da sich die Wartezeit auf eine Bestellung für den Kunden verlängert, wenn die Produktion außerplanmäßig stoppt. Dennoch sollte die Verfügbarkeit nicht unnötig hoch gewählt werden, denn dadurch entstehen hohe Kosten für den Speicherpool. Commodity-Hardware ist kostengünstig, führt aber zu erhöhter Ausfallwahrscheinlichkeit. Wird der Striping-Modus verwendet muss die Wahrscheinlichkeit von

Fehlern neu berechnet werden. Zuallerletzt ist es empfehlenswert, bei einem SPoF im System eventuell gesonderte Maßnahmen zu ergreifen.

Hardware

Ein geringer Hardwareaufwand ist eine weitere nicht funktionale Anforderung. Damit ist gemeint, dass für eine akzeptable Performance ein möglichst geringer Hardwareeinsatz genügen muss. Dadurch werden die Kosten zusätzlich gedrückt und die Akzeptanz der Synchronisierungslösung steigt.

Netzwerklast

Die Netzwerklast wird in der Industrie und der Smart-Factory deutlich steigen. Unmengen an verteilten Sensoren liefern Echtzeitdaten, die archiviert und ausgewertet werden können. Eine Datensynchronisierung stellt eine zusätzliche Belastung für das Netzwerk dar. Um nicht an die Grenzen des Netzwerkes zu kommen, sind dedizierte Verbindungen zwischen den Knoten eines Clusters eventuell sinnvoll (Abbildung 4.1). Der Vorteil ist, dass dadurch eine schnelle Verbindung für verhältnismäßig geringe Kosten etabliert wird. Hierfür gibt es heute neben 100 Gb- und 40 Gb-Ethernet unter anderem das Infiniband. Alle anderen Netzwerkteilnehmer können über eine kostengünstigere, also langsamere Verbindung kommunizieren.

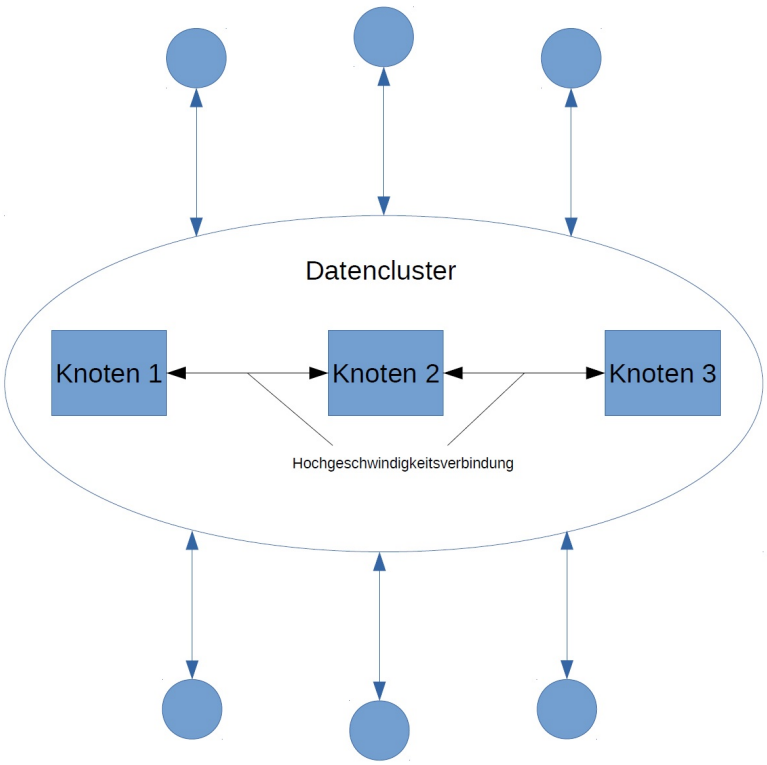


Abbildung 4.1.: Hochgeschwindigkeitsverbindung zwischen den Knoten im Cluster

Partitionierung

Neben der Verfügbarkeit von Daten, z.B. für Untersysteme, die auf diese Daten zugreifen

möchten, hat die Datensynchronisation den Vorteil, dass bei Netzwerkpartitionierung immer noch Daten auf den Datencluster geschrieben werden können. Daten von z.B. Feldgeräten gehen so nicht verloren. Ist die Netzwerkpartitionierung behoben, wird die automatische Synchronisierung weitergeführt. Auf Abbildung 4.2 ist zu sehen, dass durch einen Fehler im Netzwerk zwischen dem ersten und dem zweiten Knoten eine Unterbrechung besteht. Solange der Fehler nicht behoben ist können die Daten nicht vom ersten Knoten auf die anderen beiden Knoten abgeglichen werden. Dennoch können Geräte, die mit dem ersten Knoten verbunden sind, ihre Daten an diesen schicken. Das gleiche gilt für die anderen beiden Knoten. Nach der Fehlerbehebung wird die Synchronisation fortgeführt und es gehen keine Daten verloren.

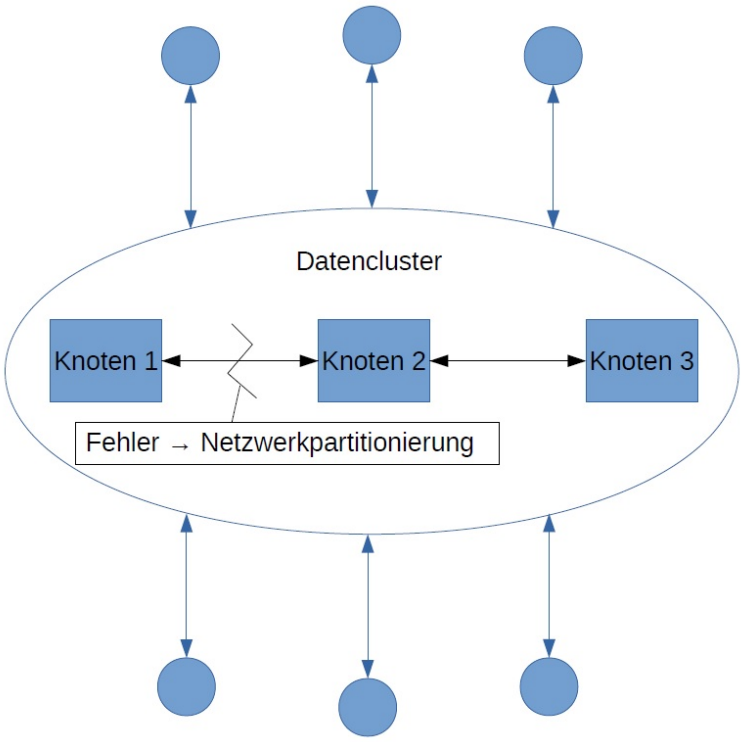


Abbildung 4.2.: Bei einer Netzwerkpartitionierung gehen die Daten von Feldgeräten nicht verloren

Betriebssystem

In heterogen Umgebungen kommen die unterschiedlichsten Betriebssysteme zum Einsatz. Ältere Betriebssysteme werden häufig nicht aktualisiert und es müssen darüber hinaus neue Systeme, und damit neue Betriebssysteme, in das Firmennetzwerk integriert werden. Besonderes Augenmerk liegt hierbei bei den mobilen Geräten. Selbst bei den Android-Geräten gibt es mehrere Versionen von Android.

Open Source

Um die Kosten gering zu halten ist eine möglichst kostenlose Synchronisierungssoftware zu bevorzugen, die bestenfalls Open Source ist. Dadurch ist der Source-Code für jeden einsehbar und Fehler können leichter öffentlich gemacht werden. Trotzdem ist es gut, wenn optional ein Herstellersupport existiert.

4.3. Gegenüberstellung der Technologien/Produkte

Die Gegenüberstellung auf Abbildung 4.3 zeigt, dass alle hier vorgestellten Synchronisationslösungen hinsichtlich der Verfügbarkeit geeignet sind. Einzig HDFS hat einen Schwachpunkt mit dem Namenode als SPoF. Mit Spezialhardware lässt sich dieser Nachteil aber abschwächen. Zudem kann man mit einem Backup von dem Namenode eine schnelle Fehlerbehebung ermöglichen.

Syncthing ist in Bezug auf die Rechenleistung der Knoten erst einmal nur vertikal skalierbar. In P2P-Netzwerken können aber verteilte Algorithmen die Anzahl und Rechenleistung mehrerer Knoten ausnutzen und damit wäre auch bei Syncthing eine horizontale Skalierung möglich. Solche Algorithmen sind sehr mächtig, aber auch aufwendig zu implementieren.

Soll das Produkt in einer heterogenen Umgebung eingesetzt werden, muss es auf unterschiedlichen Plattformen funktionieren. Hierbei fällt HDFS aus dem Rahmen, da Hadoop in Java programmiert ist und damit die Java Runtime (JRE) erfordert. Allerdings unterstützen so gut wie alle aktuellen Betriebssysteme die JRE. Selbst unter Windows XP ist es mit ein paar Tricks möglich Java 8 zu installieren.

In der Kategorie Plattform ist das Feld zweigeteilt, weil Seafile und Syncthing zwei Produkte sind, die in der nächsten Zeit wieder vom Markt verschwinden könnten. Das bedeutet, dass Clients nicht mehr für neue Betriebssysteme entwickelt oder generell nicht weiter gepflegt werden. Im Gegensatz dazu sind GlusterFS, Hadoop und MySQL schon länger auf dem Markt und es ist davon auszugehen, dass sich das nicht so schnell ändert.

Die vielleicht wichtigste Kategorie ist die Skalierbarkeit des Speichers. Hier gilt für Syncthing das gleiche wie bei der Performance. Allerdings ist hier der Speicher ausschließlich vertikal skalierbar. Das würde bedeuten, dass jeder Knoten im P2P-Netz erweitert werden muss wenn der Speicher knapp wird.

	Replikation	CAP	Authentifizierung/ Autorisierung	Fehlertoleranz	Geo-Replikation
Seafile-Cluster (mit Load Balancer und DFS als Backend)	👍	👍	E2E-Verschlüsselung; Anmeldung über Client 👍	Loadbalancer + redundante Server + DFS 👍	durch DFS gegeben 👍
Syncthing	👍	k.I.	Anmeldung über Client 👍	Knoten können jederzeit in das Netz ein- oder austreten 👍	👍
GlusterFS-Cluster	👍	👍	über ACL des Servers 👍	Knoten können im Betrieb ausgetauscht werden 👍	👍
HDFS + HBase	👍	👍	über ACL des Servers 👍	Namenode ist SPoF 👎	Geographically Distributed Hadoop 👍
MySQL Cluster	👍	ACID 👍	MySQL-User 👍	Fehlende Verbindungen + Heartbeat-Nachrichten 👍	👍

	mobile Devices	Einfachheit	Speicherskalierbarkeit	Performance	Kosten
Seafile-Cluster (mit Load Balancer und DFS als Backend)	Clients für mobile OS 👍	Konfiguration von Load-balancer, Servern und DFS	beliebig horizontal 👍	Loadbalancer + saklierbares DFS 👍	Professional Edition kostenpflichtig
Syncthing	Clients für mobile OS 👍	Konfiguration der einzelnen Knoten	vertikal 👎	vertikale Skalierung 👎	kostenfrei
GlusterFS-Cluster	erscheint als Netzlaufwerk 👍	Konfiguration der Server über Kommandozeile	beliebig horizontal 👍	Elastic Hashtable; horizontale Skalierung 👍	kostenfrei
HDFS + HBase	Client muss implementiert werden 👎	MapReduce-Funktionalitäten müssen implementiert werden	beliebig horizontal 👍	MapReduce; horizontale Skalierung 👍	kostenfrei
MySQL Cluster	k.I.	Konfiguration von Server, Storage-Nodes und Management-Node	beliebig horizontal 👍	k.I.	einmalige Zahlung einer Lizenz

	Hardware	Netzwerklast	Netzwerk-partitionierung	Betriebssysteme	Open Source
Seafile-Cluster (mit Load Balancer und DFS als Backend)	commodity hardware 👍	Dedizierte Verbindungen zum DFS und im DFS möglich 👍	Load-Balancer und hochverfügbares DFS 👍	Client für Linux, Windows, MacOSX, Android, IOS 👍	👍
Syncthing	commodity hardware 👍	schnelle Verbindungen zwischen den Peers nötig 👎	k.I.	Client für Linux, Windows, Android 👎	Source Code auf GitHub 👍
GlusterFS-Cluster	commodity hardware 👍	Dedizierte Verbindungen zwischen den Servern empfohlen 👍	Clients können lesen und schreiben 👍	Clients u.a. per NFS und SAMBA; Server erfordern Linux 👍	👍
HDFS + HBase	Spezialhardware beim Namenode empfohlen 👎	k.I.	Clients können lesen und schreiben 👍	API für C, C++, C#, Java; Nodes erfordern Java 👍	👍
MySQL Cluster	commodity hardware 👍	k.I.	k.I.	API für C, C++, Java; DBS für Windows, Unix, MacOSX 👍	👍

Abbildung 4.3.: Vergleich der Produkte bezüglich den Anforderungen

5. Auswahl einer geeigneten Synchronisationstechnologie

Den Vergleich der Produkte über die Anforderungen zu ziehen gestaltet sich als nicht trivial. Häufig müssen diese Eigenschaften in praktischen Versuchen bestätigt bzw. getestet werden. Das bedeutet einen hohen Aufwand, der in dieser Arbeit nicht geleistet werden kann. Aus diesem Grund wurden die Kriterien für Einfachheit, Flexibilität und Performance herangezogen, um ein Produkt auszuwählen. Auf der Abbildung 4.3 wurde bereits ein grober Vergleich versucht.

HDFS/HBase scheidet erst einmal aus, weil die Anwendungen mit Hilfe des Hadoop-Frameworks erstellt werden müssen. Zudem ist HDFS speziell für Big Data entwickelt worden und für Echtzeitanalysen nicht geeignet.

MySQL-Cluster unterstützt die ACID-Eigenschaften, kann aber nur mit strukturierten Daten umgehen. Da aber davon auszugehen ist, dass in Industrie 4.0 eine große Menge an unstrukturierten Daten erzeugt werden ist MySQL-Cluster nicht geeignet.

Seafile benötigt für eine hochverfügbare Datenhaltung einen entsprechenden Backend-Storage. Dieser ist idealerweise wiederum ein DFS mit Hochverfügbarkeitseigenschaften. Seafile wäre also durchaus als übergeordnete Instanz relevant, die auf ein DFS aufbaut.

Syncthing ist eine interessante Idee, um Daten hochverfügbar zu machen. Allerdings müsste Syncthing in Feldversuchen beweisen, dass Administrationsaufwand, Performance und Speicherskalierbarkeit für eine hohe Anzahl an Peers praktikabel ist.

GlusterFS ist ein DFS, welches eine einfache Einrichtung erlaubt. Die Möglichkeit, das Dateisystem für andere Systeme – z.B. Hadoop oder Seafile – zur Verfügung zu stellen, erhöht den Nutzen. Darüber hinaus ermöglicht die horizontale Skalierung sowohl eine Verbesserung der Performance als auch des Speichervolumens. GlusterFS ist Open Source und auf den Einsatz von kostengünstiger Commidity-Hardware ausgelegt. Systeme, die GlusterFS nutzen möchten, können dieses einfach integrieren, da es sich wie ein normales Dateisystem verhält. Aufgrund der aufgeführten Vorteile von GlusterFS wird es in den nächsten Kapiteln mit Hilfe eines praktischen Aufbaus analysiert.

5.1. Datencluster mit GlusterFS

Dieser Abschnitt soll zeigen, wie man GlusterFS installiert und sowohl den *Distributed-* als auch den *Disperse*-Modus konfiguriert. Unter anderem wird das Verhalten von GlusterFS bei einem Fehler analysiert und die Performance sowie der Datenverkehr in den beiden Modi beobachtet.

Neben der Möglichkeit den nativen GlusterFS-Client zu nutzen, kann ein Volume per NFS-Freigabe genutzt werden. Welche Unterschiede sich dadurch ergeben ist in Kapitel 5.2.4 erläutert.

GlusterFS ist in den offiziellen Repositorys von Debian in der Version 3.5 verfügbar. Möchte man die aktuellste Version (3.7.x)[1] verwenden ist zwingend Debian in der 64 Bit-Version zu benutzen, da GlusterFS in der 32 Bit-Version nur bis 3.5 zur Verfügung steht.

Bricks vorbereiten

Um GlusterFS zu testen wird in dieser Arbeit ein Cluster mit drei dedizierten Rechnern (*cluster1.example.com*, *cluster2.example.com*, *cluster3.example.com*; Intel Atom 1.66 GHz, 2 GB RAM)) aufgebaut und analysiert. Als Betriebssystem dient Debian 8.2 mit der Kernelversion 3.16. Die Netzwerkanbindung wird über eine PCI-Gigabit-Ethernetschnittstelle realisiert, da die Onboard-Netzwerkschnittstelle nur 100 Mbit/s schnell ist. Die Rechner werden über ein Gigabit-Switch und Cat6-Kabel verbunden. Über einen Performancetest mit Hilfe von *iperf* lassen sich die Übertragungsraten zwischen den Knoten ermitteln. Um die Schritte besser nachverfolgen zu können wurden vorher die IP-Adressen mit den Hostnamen der Rechner in der jeweiligen */etc/hosts*-Datei verknüpft.

```
gluster@cluster1:~$ cat /etc/hosts
127.0.0.1       localhost
127.0.1.1       cluster1.example.com
172.20.2.79     cluster2.example.com
172.20.2.93     cluster3.example.com
```

Listing 5.1: Die Hostnamen müssen in */etc/hosts* eingetragen werden.

Den eigenen Hostnamen schreibt man in */etc/hostname*, der danach gesetzt wird.

```
gluster@cluster1:~$ sudo hostname -F /etc/hostname
```

Listing 5.2: Hostnamen setzen.

[1] Zwischen den Versionen 3.5.x und 3.7.x gibt es größere Unterschiede, wie z.B. den Disperse-Modus.

```
gluster@debiancluster1:~$ iperf -s
```

Listing 5.3: iperf-Server starten

```
gluster@debiancluster2:~$ iperf -c cluster1.example.com
```

Listing 5.4: iperf-Client starten

Auf Abbildung A.1 wurde von einem Windowsrechner die Netzwerkperformance zu den jeweiligen Clusterknoten gemessen. Die theoretische Geschwindigkeit von $1\,\mathrm{Gb/s}$ wird dabei nicht annähernd erreicht, sondern Werte zwischen $470\,\mathrm{Mb/s}$ und $540\,\mathrm{Mb/s}$.

Auf jedem Rechner ist eine $500\,\mathrm{GB}$-HDD verbaut. Die erste Partition (*sda1/ext4*) ist für Debian und ca. $50\,\mathrm{GB}$ groß. Für GlusterFS ist die zweite Partition (*sda3/xfs*)[2] mit $417.2\,\mathrm{GiB}$ reserviert. $2\,\mathrm{GB}$ sind für die Swap-Partition (*sda5*).

Neben der Netzwerkgeschwindigkeit ist die Schreib- und Leseleistung der Festplatten im Cluster ein Kriterium für die Leistungsfähigkeit von GlusterFS. Um Probleme in der Performance von GlusterFS eingrenzen zu können benötigt man die Schreib- und Lesewerte. Hierzu dient das Programm *dd*. Abbildung A.3 zeigt ein Beispiel, um eine $100\,\mathrm{MB}$ große Datei zehn mal auf die Partition zu schreiben. *dd* liefert als Ergebnis die Schreibgeschwindigkeit von durchschnittlich $100\,\mathrm{MB/s}$.[3].

Nach dem Schreibtest wird der Lesetest mit *dd if=testfile of=/dev/null bs=100M count=100* und der vom Schreibtest erzeugten Datei *testfile* ausgeführt. Bei allen drei Festplatten liegt der Lesewert über $100\,\mathrm{MB/s}$.

GlusterFS-Server einrichten

Der erste Schritt, um die Server auf den jeweiligen Bricks einzurichten besteht darin, die Pakete zu installieren. In den offiziellen Debian-Repositorys ist die Version 3.5 des GlusterFS-Servers vorhanden. Um die aktuellste Version zu installieren, muss zuerst der Public-Key von GlusterFS auf dem jeweiligen Rechner mit *gluster@cluster1: $ wget -O - http://download.gluster.org/pub/gluster/glusterfs/3.7/3.7.6/pub.key | apt-key add -* importiert werden.

```
root@cluster1:~$ wget -O - http://download.gluster.org/pub/gluster/glusterfs/\
3.7/3.7.6/pub.key | apt-key add -
```

Listing 5.5: GlusterFS Public-Key importieren

[2] Bevor unter Debian die GlusterFS-Partition mit *mkfs* als *xfs* formatiert werden kann muss das Tool *xfsprogs* installiert werden.

[3] Beim Einsatz von SATA-Festplatten ist darauf zu achten, dass im BIOS der AHCI-Modus eingestellt ist.

Um das Repository optimal nutzen zu können, wird es in */etc/apt/sources.list.d/gluster.list* hinzugefügt.

```
root@cluster1:~$ echo deb http://download.gluster.org/pub/gluster/glusterfs/\
3.7/3.7.6/Debian/jessie/apt jessie main > /etc/apt/sources.list.d/gluster.list
```

Listing 5.6: GlusterFS-Repository bekannt geben

***Hinweis**: Die Befehle aus den Listings 5.5 und 5.6 lassen sich unter Debian 8.2 nur als root-User ausführen!*

Mit *sudo apt-get update* aktualisiert man die Quellen und installiert die neueste Version von GlusterFS mit *sudo apt-get install glusterfs-server*.
Bei der Installation der Server kam es zu folgender Warnung:

W: Possible missing firmware /lib/firmware/rtl_nic/rtl8168g-3.fw for module r8169

Durch die Anpassung der Debian-Quellen in der Datei */etc/apt/sources.list*, lässt sich die Warnung umgehen.

```
gluster@cluster1:~$ cat /etc/apt/sources.list

deb http://ftp.de.debian.org/debian/ jessie main non-free contrib
deb-src http://ftp.de.debian.org/debian/ jessie main non-free contrib

deb http://security.debian.org/ jessie/updates main non-free contrib
deb-src http://security.debian.org/ jessie/updates main non-free contrib

deb http://ftp.de.debian.org/debian/ jessie-updates main non-free contrib
deb-src http://ftp.de.debian.org/debian/ jessie-updates main non-free contrib
```

Listing 5.7: Auszug von /etc/apt/sources.list mit den nichtfreien Quellen

In Listing 5.7 sind neben den freien Quellen zusätzlich die nichtfreien Quellen in *source.list* eingetragen. Anschließend aktualisiert man die Quellen mit "*sudo apt-get update*" und installiert die Firmware mit "*sudo apt-get install firmware-realtek*". Danach verläuft die Installation von GlusterFS ohne Warnung.

Im nächsten Schritt lassen sich die Server zu einem Pool zusammenfassen[4].

```
gluster@cluster1:~$ sudo gluster peer probe cluster2.example.com
gluster@cluster1:~$ sudo gluster peer probe cluster3.example.com
gluster@cluster1:~$ sudo gluster peer status
Number of Peers: 2
```

[4] *cluster1* muss nicht explizit hinzugefügt werden.

```
Hostname: cluster2.example.com
Uuid: c32d821e-ca70-4892-b9b7-2227698339da
State: Peer in Cluster (Connected)

Hostname: cluster3.example.com
Uuid: 4b14eb03-0a6f-413e-91d1-7c62927812ac
State: Peer in Cluster (Connected)
```

Listing 5.8: *cluster2* und *cluster3* zum *Trusted Storage Pool* hinzugefügen.

Vor dem Anlegen eines *share* sind die Partitionen für den Cluster auf allen Knoten in der jeweiligen *fstab* einzutragen, damit diese bei einem Systemstart automatisch gemountet werden. Für *cluster1.example.com* sieht die *fstab* wie folgt aus:

```
gluster@cluster1:~$ cat /etc/fstab
UUID=df73802f-5676-4061-88db-b54a222ab732 / ext4 errors=remount-ro 0 1
UUID=6b71501e-f3d2-4b32-84a8-e75b1e3fe70e /gluster_brick xfs defaults 0 0
UUID=ecfa59ab-8d5d-4c07-b478-97a04f4d3470 none swap sw 0 0
```

Listing 5.9: Um die Partition *sda3* bei Systemstart in den Ordner */gluster_brick* zu mounten, muss diese in die *fstab* eingetragen werden.

Sind die drei Peers zu einem Volume – in diesem Fall drei Replicas[5] – zusammengefasst, kann der Cluster verwendet werden. Der Name des shares ist *glustercluster*. Mit dem Parameter *force* ist es möglich, eine root-Partition zum Pool hinzuzufügen.

```
gluster@cluster1:~$ sudo gluster volume create glustercluster
replica 3 transport tcp
cluster1.example.com:/gluster_brick
cluster2.example.com:/gluster_brick
cluster3.example.com:/gluster_brick force
```

Listing 5.10: *share* mit drei Replicas einrichten.

Nun kann das Volume mit "*sudo gluster volume start glustercluster*" gestartet und der Status abgefragt werden ().

```
gluster@cluster1:~$ sudo gluster volume info

Volume Name: glustercluster
Type: Replicate
Volume ID: d773521b-b605-42e5-9505-65b02dbd8258
```

[5] Werden Replicas verwendet, wird der Modus *Replicate* eingestellt.

```
Status: Started
Number of Bricks: 1 x 3 = 3
Transport-type: tcp
Bricks:
Brick1: cluster1.example.com:/gluster_brick
Brick2: cluster2.example.com:/gluster_brick
Brick3: cluster3.example.com:/gluster_brick
```

Listing 5.11: Nach der Einrichtung kann der Status des Clusters abgefragt werden.

Client einrichten

Clients im Netzwerk können den Speicher der Bricks nutzen, um Daten abzulegen. Für diese Arbeit dient ein Rechner als Client, der genauso ausgestattet ist wie die Server-Rechner (Intel Atom 1.66 GHz, 2 GB RAM, Gigabit-PCI-Schnittstelle, Debian 8.2). Der Hostname ist *clusterclient.example.com*. Es muss nur die Netzwerkgeschwindigkeit des Client getestet werden, da der Mount von dem Cluster kommt und demnach die Festplatte keinen Einfluss auf die Leistung des Clusters hat. Mehrfaches Durchführen von *iperf* ergibt unterschiedliche Ergebnisse, aber die Werte liegen jederzeit über 470 Mb/s.

Nachdem der Public-Key von GlusterFS importiert wurde, wie in Listing 5.5 durchgeführt, können die Quellen aktualisiert und der GlusterFS-Client installiert werden.

Cluster auf dem Client mounten

Um den Clusterspeicher auf dem Client mit *sudo mount -a* zu mounten, ist ein Eintrag in der */etc/fstab* des Clients nötig und ein entsprechender Ordner anzulegen. Der Eintrag sieht wie folgt aus:

```
gluster@clusterclient:/$ cat /etc/fstab
cluster1.example.com:/glustercluster  /gluster  glusterfs  defaults,_netdev 0 0
```

Listing 5.12: Cluster mounten.

Mit der Option *_netdev* wird festgelegt, dass der Mount einen Netzwerkzugang benötigt. Beim Start wird dadurch erst auf die Netzwerkverbindung gewartet bevor versucht wird, die Quelle zu mounten[6].

Ist der Cluster erfolgreich gemountet, kann dieser auch für Windowssysteme als Samba-Freigabe zur Verfügung gestellt werden.

[6] siehe auch `http://linux.die.net/man/8/mount`

5.2. Cluster testen

Der Cluster soll jetzt mit der Konfiguration *Distributed* getestet werden. Das Hauptaugenmerk gilt, sowohl im Normalbetrieb als auch im Fehlerfall, der Performance.

5.2.1. Erster Test: Übertragungsgeschwindigkeit

GlusterFS kann mit *dd if=/dev/zero of=/gluster/testfile bs=512M count=100 oflag=direct* getestet werden. Um genügend Zeit für die Auswertung zu haben, wird eine $512\,\mathrm{MB}$ große Datei hundertmal in den freigegebenen Ordner geschrieben. Wie auf Abbildung A.4 zu sehen, ist der Netzwerktraffic zu den drei GlusterFS-Servern mit z.B. $112\,\mathrm{Mb/s}$ gleich groß. Das bedeutet aber gleichzeitig einen dreimal so hohen Traffic vom Client aus gesehen (siehe Abbildung 5.1). Der maximale Wert während des ersten Tests liegt bei ca. $113\,\mathrm{Mb/s}$ pro Knoten, was zusammen einen Wert von $339\,\mathrm{Mb/s}$ ergibt. Die mit *iperf* ermittelte maximale Netzwerkgeschwindigkeit von über $560\,\mathrm{Mb/s}$ wird somit nicht erreicht (siehe Abbildung A.1).

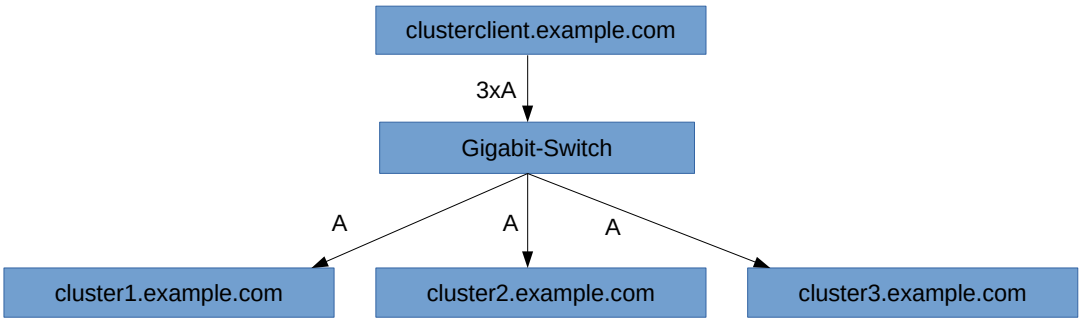


Abbildung 5.1.: Im Distributed-Mode und dem GlusterFS-Client werden die Daten vom Client an die jeweiligen Server geschickt.

Mit dem Programm *top* lässt sich die Auslastung der CPU des Clients überwachen. Laut Abbildung A.5 befindet sich der Prozessor ca. 30 % der Zeit während des Tests im Leerlauf (29,8 % id) und hat somit noch ein paar Reserven.

5.2.2. Zweiter Test: Ein Knoten fällt aus

Im zweiten Versuch wird der Test aus Kapitel 5.2.1 mit den Parametern *bs=100* und *count=10* durchgeführt und die Zeit ermittelt. Nachdem die Werte festgehalten wurden, läuft der Test ein weiteres Mal durch. Diesmal wird aber das Netzwerkkabel von *cluster2.example.com* nach

ca. zwei Sekunden entfernt. Das soll den Ausfall des Knotens simulieren (siehe Abbildung 5.2), wodurch sich bestimmen lässt, wie viel Zeit bei einem Fehler vergeht.

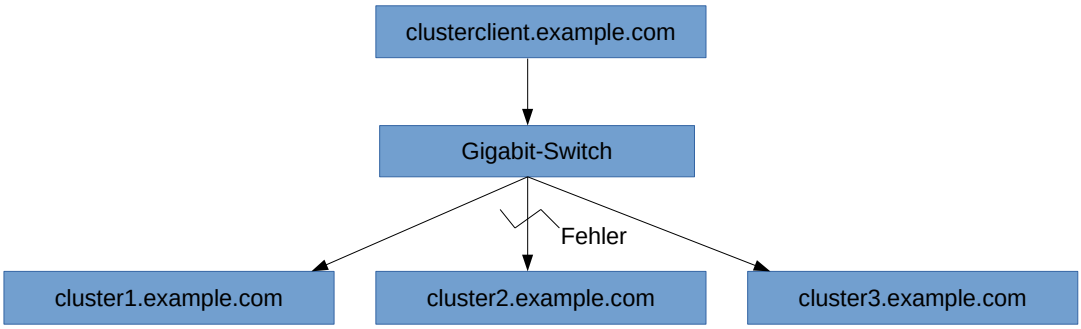


Abbildung 5.2.: Simulation eines Fehlers zum zweiten Server

Der Test läuft ohne Fehler in $52.3\,\mathrm{s}$ mit durchschnittlich $20\,\mathrm{MB/s}$ durch[7]. Mit dem simulierten Netzwerkfehler benötigt der Test $118\,\mathrm{s}$, wobei der durchschnittliche Schreibwert bei $9\,\mathrm{MB/s}$ liegt.

***Hinweis**: Läuft neben dem Test das Programm iftop zur Netzwerkanalyse mit, fallen die Ergebnisse deutlich schlechter aus!*

Warum verschlechtert sich die Schreibperformance so stark?

GlusterFS bietet verschiedene Einstellungsmöglichkeiten, um die Performance des Clusters zu optimieren. Eine davon betrifft die Timeout-Zeit für Knoten. Der Defaultwert beträgt $42\,\mathrm{s}$[Glub]. Das heißt, dass ein Ausfall eines Knotens nach $42\,\mathrm{s}$ als Fehler registriert wird und erst danach die Übertragung weiterläuft. Die Werte verbessern sich signifikant, wenn dieser Wert von $42\,\mathrm{s}$ auf $2\,\mathrm{s}$ gestellt wird. Jetzt liegt der Schreibwert sogar über dem ursprünglichen Wert von $20\,\mathrm{MB/s}$, was daran liegt, dass der Client die Daten nach zwei Sekunden Wartezeit nur noch an zwei Server schicken muss. Somit sinkt das Gesamtdatenvolumen über das Netzwerk von $3\,\mathrm{Gb}$ auf $2\,\mathrm{GB}$. Und darüber hinaus erhöht sich die Netzwerkgeschwindigkeit zu den zwei noch im Cluster befindlichen Knoten.

Ein ähnlicher Test soll zeigen, wie sich die Daten auf den jeweiligen Bricks im Fehlerfall verhalten. Werden die Daten konsistent gehalten wenn der Fehler behoben ist? Auf dem Mount des Clients beinhaltet der Unterordner *temp* eine Knoppix-ISO-Datei mit einer Größe von über $4\,\mathrm{GB}$. Diese Datei wird in den root-Ordner der Freigabe kopiert. Ein ”Netzwerkfehler”

[7] Werte von *dd*

(Netzwerkkabel wird gezogen) am Knoten *cluster2.example.com* tritt während der Übertragung auf. Der Übertragungsvorgang wird für die beiden funktionsfähigen Knoten vollständig abgeschlossen. Danach wird der zuvor fehlerhafte Knoten wieder dem Cluster hinzugefügt.

Sobald der Knoten *cluster2.example.com* wieder funktionsbereit ist, werden die fehlenden Daten von *cluster1.example.com* nachgeladen und die Datei auf dem fehlerhaften Knoten konsistent gehalten. Dies kann man prüfen, indem man die Checksumme der jeweiligen Dateien mit dem Konsolenbefehl *cksum KNOPPIX_V7.6.0DVD-2015-11-21-DE.iso* ermittelt.

5.2.3. Performante und Fehlertolerante Konfiguration

Im Kapitel 3.5 sind die verschiedenen Möglichkeiten, wie man einen Cluster mit GlusterFS konfigurieren kann, aufgezeigt. Der *Disperse*-Modus bietet einen Kompromiss zwischen dem hohen Speicherverbrauch des Replicated-Modus und der nicht vorhandenen Fehlerbehandlung des Striped-Modus. Hierbei kommt der *Erasure Code* zum Einsatz, um die Daten in Blockform auf die Bricks aufzuteilen. Deshalb soll dieses Kapitel dazu dienen, um zu zeigen wie der Cluster statt im *Replicated*-Modus – die Clusterkonfiguration aus den vorherigen Kapiteln (siehe Listing 5.1) – in dem *Dispersed*-Modus erstellt werden kann. Mit drei Knoten bietet sich die EC 1/3-Konfiguration an, bei der die Zusatzinformationen 1/3 des Speicherbedarfs einnehmen und auch bei einem fehlerhaften Knoten der Betrieb des Clusters gewährleistet ist.

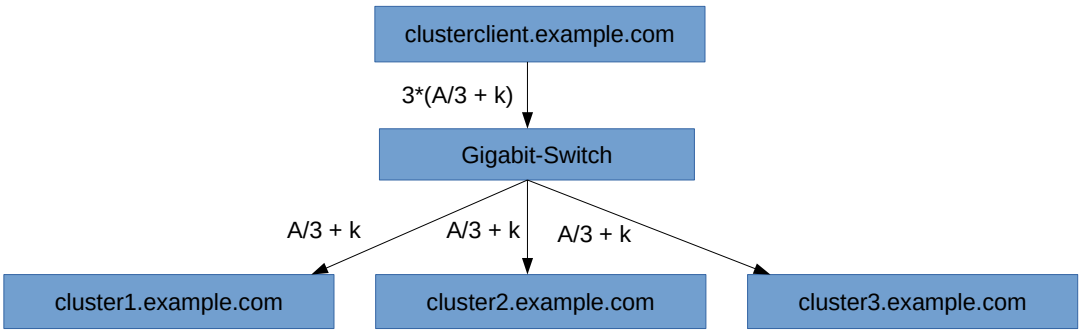


Abbildung 5.3.: Der Cluster im *Dispersed*-Modus. k steht für die Redundanzinformationen.

Die ersten beiden Schritte bestehen darin das existierende Volume zu stoppen und zu löschen.

```
gluster@cluster1:~$ sudo gluster volume stop glustercluster
gluster@cluster1:~$ sudo gluster volume delete glustercluster
```

Listing 5.13: Volume glustercluster stoppen und löschen.

Jetzt wird das neue Volume angelegt. Bei drei Knoten kann die Redundanz maximal eins sein. Der entsprechende Parameter (siehe Abbildung A.6) ist optional und kann beim Anlegen

des Clusters weggelassen werden. GlusterFS versucht dann, den optimalen Wert selber zu bestimmen.

In dieser Konfiguration ergibt sich eine Speicherkapazität von $835\,\mathrm{GB}$. Davon stehen dem Client $826\,\mathrm{GB}$ zur Verfügung, also ca. 2/3 der Summe von den drei Bricks[8]. 1/3 ist für die Redundanzinformationen reserviert.

Der Test mit *sudo dd if=/dev/zero of=/gluster/testfile bs=100M count=10 oflag=direct* ($1\,\mathrm{GB}$) läuft in knapp $41.4\,\mathrm{s}$ durch, wobei die Schreibgeschwindigkeit bei $25.3\,\mathrm{MB/s}$ liegt. Daraus berechnet sich ein Übertragungsvolumen von $41.4\,\mathrm{s} \cdot 25.3\,\mathrm{MB/s} = 1047.42\,\mathrm{MB}$.

Dadurch, dass die Daten in drei gleich große Teile aufgeteilt werden um sie dann zu denn Servern zu übertragen, sollte auch die Schreibgeschwindigkeit im Vergleich zum Replica-Modus deutlich steigen. Verglichen mit den $20\,\mathrm{MB/s}$, die von dem Test aus dem vorherigen Kapitel ermittelt wurden, bedeutet dies einen Geschwindigkeitszuwachs von 26 %[9].

Neben den Kompromissen, die man im Disperse-Modus gegenüber anderen Modi eingehen muss, hat der Modus einen weiteren Nachteil. Wenn Daten häufig vom Cluster gelesen, modifiziert und zurück gespeichert werden, ist der Modus nicht optimal. Dieser Vorgang macht es nötig, die Datenblöcke zusammenzufügen, zum Schluss aufzuteilen und mit den Redundanzinformationen anzureichern. In solchen Fällen sind andere Konfigurationen zwar mit einem höheren Hardwareaufwand verbunden, aber für die Performance besser geeignet. Der *Disperse*-Modus bietet sich somit für Systeme an, die nach dem WORM-Prinzip arbeiten. Hierbei werden die Daten einmal geschrieben und danach nicht wieder verändert.

5.2.4. Volume über NFS freigeben

Neben der Möglichkeit, den nativen GlusterFS-Client zu verwenden um das Volume zu mounten, kann die Freigabe auch über NFS erfolgen. Für einen einfachen Test dient der neu angelegte Ordner */gluster_nfs* auf dem Client. Ein einfacher Konsolenbefehl genügt, um den Cluster über NFS zugänglich zu machen.[10]

```
gluster@clusterclient:~$ sudo mount -t nfs \
cluster1.example.com:/glustercluster /gluster_nfs
```

Listing 5.14: GlusterFS-Volume mit Hilfe von NFS mounten.

Ein Test mit *dd if=/dev/zero of=/gluster_nfs/testfile bs=100M count=10 oflag=direct* und dem Volume im Disperse-Modus, also dem identischen Test aus den vorherigen Kapiteln, ergibt eine Steigerung der Schreibgeschwindigkeit von $25.3\,\mathrm{MB/s}$ auf $27.4\,\mathrm{MB/s}$ ($+8.3\,\%$).

[8] $3 \cdot 418\,\mathrm{GB} \cdot 0.66 \approx 828\,\mathrm{GB}$

[9] $\frac{100\,\% \cdot 25.2\,\mathrm{MB/s}}{20\,\mathrm{MB/s}}$

[10] Ein Eintrag in /etc/fstab sorgt dafür, dass das Volume beim Start des Rechners automatisch mountet.

Wird das Volume über NFS freigegeben verändert sich der Datenverkehr zwischen dem Client und den Servern. Vergleicht man den Netzwerkverkehr von der NFS-Freigabe und der Freigabe mit Hilfe des nativen GlusterFS-Clients ist ersichtlich, dass bei der NFS-Freigabe die Daten ausschließlich an den Server *cluster1.example.com* geschickt werden. Die Weiterleitung geschieht dann über den Server. Das kann je nach Einsatzgebiet Vorteile bringen. Zum einen können die Daten über eine eventuell vorhandene dedizierte Hochgeschwindigkeitsverbindung zwischen den Servern schneller verteilt werden. Zum anderen werden die Daten auf dem jeweiligen Server aufgeteilt und mit redundanten Informationen angereichert. Die Weiterleitung der rechenintensiven Operationen (Festplatte) übernimmt somit, in den meisten Fällen, der leistungsstärkere Server.

Die Abbildungen A.8 und A.7 zeigen den Netzwerkverkehr während Daten auf eine GlusterFS-Freigabe per NFS abgelegt werden. Die vom Client geschriebenen Daten laufen über den Server *cluster1.example.com* an die anderen beiden Server.

5.2.5. Was zeigt der praktische Versuch?

GlusterFS ermöglicht es, mit Hilfe von Standard-Hardware, einen Storage-Cluster aufzubauen, bei dem der Replikationsgrad theoretisch beliebig ist. Allein die Grenzen anderer Hardwarekomponenten (z.B. Netzwerk) limitiert die Anzahl der Replikation. Damit ist allein für den Cluster eine Hochverfügbarkeit gewährleistet.

Sowohl der Aufbau, als auch die Konfiguration des Clusters ist denkbar einfach. Der Administrator muss lediglich grundlegende Linux-Kenntnisse beherrschen. Per Kommandozeile können Knoten hinzugefügt und der Cluster konfiguriert werden.

In dem Versuchsaufbau ist die Netzwerkperformance der limitierende Faktor für die Performance. Bei dedizierten Verbindungen der Server-Knoten und dem Einsatz von Hochgeschwindigkeitsethernet zu den Clients sollte die Performance an die Leistungsgrenze der verbauten Festplatten heranreichen.

Bezüglich der Netzwerkpartitionierung ist GlusterFS gut aufgestellt. Hierbei ist aber zu beachten, dass der Cluster sorgfältig geplant wird. So ist es denkbar, dass an jeder Produktionsanlage in einer Fabrik ein Sub-Cluster mit Replikation steht und diese Sub-Cluster zu einem Gesamtcluster zusammengeschlossen werden. Sollte also eine Produktionsanlage plus Sub-Cluster durch Netzwerkpartitionierung von den anderen Sub-Clustern getrennt werden, stellt dies kein Problem für den Betrieb der Anlage und die Replikation der Daten dar.

Die Kosten beschränken sich ausschließlich auf die Hardware. Sowohl Betriebssystem als auch GlusterFS selber sind Open Source.

5.2.6. GlusterFS und Big Data

GlusterFS zeichnet sich durch eine Vielzahl an Vorteilen gegenüber anderen Produkten aus. Aber ist dieses Dateisystem auch für Big Data geeignet? Zumindest nicht so ohne weiteres! GlusterFS allein ist nicht in der Lage mit dem Big-Data-Problem umzugehen. Allerdings kann GlusterFS in Hadoop integriert werden und somit eine skalierbare und kostengünstige Infrastruktur für Big-Data-Analysen bilden. Gleichzeitig lässt sich der Nachteil des SPoF vom HDFS-Dateisystem beheben.[htta] In dieser Konstellation müssen aber die anderen Nachteile des Hadoop-Framework (vgl. Seite 30) berücksichtigt werden.

6. Zusammenfassung

Für eine hochverfügbare Datenhaltung ist es nicht ausreichend allein den Festplattenspeicher zu berücksichtigen. Weitere Aspekte, wie die Netzwerkstruktur, sinnvolle lokale Verteilung der Speicherknoten oder sogar zeitlich abgestimmte Wartungsarbeiten beeinflussen den Wert für die Verfügbarkeit eines Systems. All diese Punkte müssen zu einem Gesamtkonzept zusammengetragen werden. Darüber hinaus ist es meistens erforderlich, eine geeignete Backupstrategie einzuführen, um versehentlich gelöschte Informationen wiederherzustellen.

Die Arbeit sollte für die Schwerpunkte in Bezug auf Datenhaltung in Industrie 4.0 sensibilisieren. Dabei ist es zum jetzigen Zeitpunkt weniger relevant welche Technologie eingesetzt werden soll. Vielmehr muss sich Gedanken über die Architektur und die IT-Infrastruktur gemacht werden, damit keine Engpässe im Datenverkehr entstehen und so die Performance beeinträchtigt wird.

Es ist nicht so einfach abzuschätzen wie groß das Datenaufkommen in der Smart Factory sein wird. Beschränken sich die Informationen auf Sensordaten? Oder finden Audio- und Videoverarbeitung verstärkt Anwendung? Spätestens dann muss die Storage-Strategie Big-Data kompatibel sein. Mit dem Hadoop-Framework gibt es heute bereits ein Mittel, um diese Datenmengen zu verarbeiten. Doch für Echtzeitanalysen ist dieses Konzept nicht geeignet. Auf der anderen Seite hält der Markt viele Produkte bereit, hochverfügbare Datenspeicher kostengünstig zu erstellen. Eine der einfachsten Möglichkeiten bietet wohl GlusterFS, mit dem unterschiedliche, auf die speziellen Bedürfnisse angepasste, Konfigurationen des Clusters eingestellt werden können. Doch die Konkurrenz zu GlusterFS ist groß. Es gibt bereits unzählige alternative DFS, die in dieser Arbeit nicht berücksichtigt werden konnten. Produkte, die sich durch Einfachheit und starker Performance auszeichnen, werden sich dann wohl durchsetzen.

Kommen unstrukturierte Daten eher seltener vor, bieten sich weiterhin relationale Datenbanken an. MySQL-Cluster ermöglicht es, diese Daten hochverfügbar zu speichern. Allerdings müssen hierzu die Daten immer in einer strukturierten Form vorliegen. Zudem müssen die Leistungsgrenzen festgestellt werden, damit die Reaktionsfähigkeit der Datenbank langfristig gesichert ist.

Mobile Geräte – die vermutlich verstärkt in der Smart Factory zum Einsatz kommen – können über verschiedene Wege mit einem Datencluster kommunizieren. Entweder über die

vom Betriebssystem zur Verfügung gestellten Freigaben oder (meist besser) über Drittsoftware wie zum Beispiel Seafile. Dadurch lassen sich neue Geräte auf einfache Art und Weise per Client auf dem Gerät in das System hinzufügen. Weitere Vorteile wie Load-Balancing ergänzen diese Lösung.

P2P-Netzwerke, wie sie z.B. Syncthing verwendet, können in Zukunft durchaus konkurrenzfähig sein, wenn es darum geht hochverfügbare Datencluster zu erstellen. Dies bedarf aber zusätzlicher Recherchen oder sogar Forschungsarbeit. Das Potenzial dieser Netze bezüglich verteiltes Rechnen oder verteilter Ressourcen kann durchaus interessant für zukünftige Aufgaben im Umfeld Industrie 4.0 sein.

Ein Kernproblem, das sich in dieser Arbeit herausgebildet hat, ist die Bewältigung von Big-Data bei gleichzeitiger Echtzeitanalyse von Daten. Keines der hier betrachteten Produkte beherrscht beide Kriterien. Damit soll das letzte Kapitel dienen um die Lambda-Architektur einzuführen.

7. Ausblick – Die Lambda-Architektur

Wie schon mehrfach in dieser Arbeit angedeutet, ist bei Abfragen auf umfangreichen Daten eine Parallelverarbeitung unerlässlich, möchte man zeitnah Ergebnisse aus einem großen Pool an Daten gewinnen. Der populärste derzeitige Vertreter, der diese Aufgabe bewältigt, ist das Hadoop-Framework mit dem Map-Reduce-Verfahren. Doch scheitert diese Technologie wenn es um die Echtzeitauswertung von Informationen geht.

> ***Analogie**: In dem Buch "Seven Concurrency Models in Seven Weeks" von Paul Butcher wird das Hadoop Framework mit einem Truck verglichen der viele Transportgüter von A nach B transportieren muss. Für diese Aufgabe ist der Truck prädestiniert. Allerdings würde niemand auf die Idee kommen ausschließlich ein kleines Päckchen auf diese Art zu verschicken. Dafür sind andere Transportmöglichkeiten geeigneter.[But14] Dieser Vergleich soll verdeutlichen, dass man zwischen kleinen und großen Datenmengen unterscheiden muss. Übersteigt die Datenmenge einen bestimmten Punkt, ist es nötig die Strategie für die Auswertung zu ändern.*

Um beiden Problemen gerecht zu werden, wurde die Lambda-Architektur entworfen. Diese Architektur besteht aus drei Teilen: dem Batch-Layer, dem Serving-Layer und dem Speed-Layer. In jedem der drei Layer werden unterschiedliche Tools verwendet, um entsprechende Aufgaben bestmöglich zu bearbeiten. Dabei gibt es keine Vorgabe, welche Tools das sein müssen. Darüber gibt die Lambda-Architektur keine Auskunft.

Batch-Layer

Der Batch-Layer ist dafür zuständig, bestimmte Abfragen auf alle Daten durchzuführen und die Ergebnisse zu speichern. Zwei Dinge müssen vom Batch-Layer erfüllt werden: unveränderbare Speicherung immer neu hinzukommender Eingangsdaten und das Ausführen beliebiger Abfragefunktionen auf diese Daten.[MW15, S. 16] Diese Eigenschaften erfüllt beispielsweise Hadoop. Der PseudoCode für den Batch-Layer könnte so aussehen:

```
function runBatchLayer() :
```

```
while(true) :
  recomputeBatchViews()
```

Listing 7.1: Pseudocode für den Batch-Layer (Quelle: [MW15])

Da der Batch-Layer alle Daten auswertet, ist mit einer hohen Latenz der Ausführung zu rechnen. Ein Aktualisierungsvorgang kann durchaus mehrere Stunden dauern. Daten, die neu hinzu, kommen werden beim nächsten Durchlauf berücksichtigt.

Serving-Layer

Der Batch-Layer erstellt Ansichten als Resultat von Funktionen, die auf den Quelldaten angewendet werden. Der Serving-Layer speichert jeweils das aktuellste Ergebnis und stellt es anderen Programmen zur Verfügung. Der Vorteil dieser Konstellation ist, dass es trivial wird die Daten aktuell zu halten. Ein Beispiel für eine Serving-Layer-Datenbank ist ElephantDB, die nur ein paar tausend Codezeilen umfasst.[MW15, S. 17]

Speed-Layer

Die Daten im Serving-Layer werden aktualisiert, sobald der Batch-Layer die neue Berechnung durchgeführt hat. Der Vorgang kann aber ein paar Stunden dauern. Für Echtzeitanalysen ist dieser Weg deshalb unbrauchbar. Diese Aufgabe wird vom Speed-Layer erfüllt. Der Speed-Layer verwendet ausschließlich aktuelle Daten und führt keine Berechnungen auf den Daten aus.

Beispiel

Hadoop bietet sich für den Batch-Layer an. Dabei können die Daten in HDFS fehlertolerant gespeichert werden, um dann eine skalierbare, parallelisierte Verarbeitung mit Hilfe von Map-Reduce durchzuführen. Aktuelle Daten werden in eine Message Queue wie *Kafka* gespeichert und zyklisch mit einem *ETL*-Tool (z.B. Campus) in das Dateisystem abgelegt.[httb]

Die Ergebnisse können in einer Read-Only-Datenbank wie SploutSQL abgelegt werden, die den Serving Layer bildet.[httb]

Das Stream-Framework Storm könnte im Speed-Layer zum Einsatz kommen. Neue Daten werden in eine so genannte Topologie eingespeist, weiter verarbeitet und dann in Datenbanken wie Cassandra oder HBase solange gespeichert, bis der Batch Layer aufgeholt hat.[httb]

Jedes Tool kann durch jedes andere Tool oder Framework, das die gleiche Aufgabe erledigt, ersetzt werden. Eine Alternative zu Hadoop ist Spark, Splout kann man auch durch ElephantDB ersetzen, Storm durch Samza, S4 oder Akka, und so weiter.[httb]

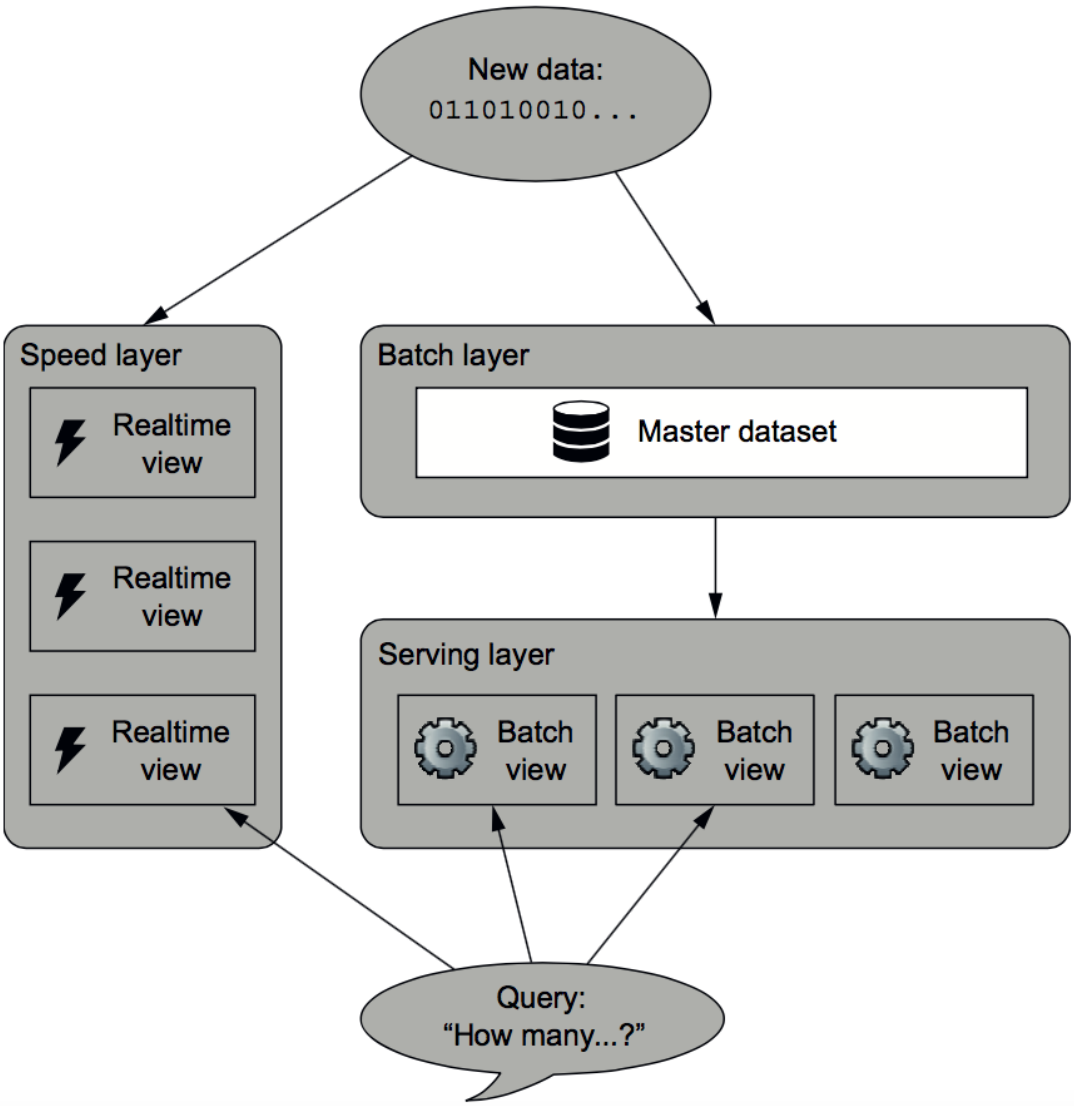


Abbildung 7.1.: Aufbau der Lambda-Architektur
Quelle: [MW15, S. 19]

A. Anhang

```
C:\Users\bredemeier\Downloads\iperf>iperf.exe -s
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 64.0 KByte (default)
------------------------------------------------------------
[  4] local 172.20.1.106 port 5001 connected with 172.20.2.82 port 58161
[ ID] Interval       Transfer     Bandwidth
[  4]  0.0-10.0 sec   561 MBytes   471 Mbits/sec
[  4] local 172.20.1.106 port 5001 connected with 172.20.2.79 port 37078
[  4]  0.0-10.0 sec   649 MBytes   544 Mbits/sec
[  4] local 172.20.1.106 port 5001 connected with 172.20.2.85 port 39097
[  4]  0.0-10.0 sec   564 MBytes   472 Mbits/sec
```

Abbildung A.1.: Netzwerkgeschwindigkeit von cluster1.example.com (172.20.2.82), cluster2.example.com (172.20.2.79) und cluster3.example.com (172.20.2.85) zu einem Windows-Rechner (172.20.1.106)

```
gluster@cluster1:~$ sudo hdparm -W0 /dev/sda
[sudo] password for gluster:

/dev/sda:
 setting drive write-caching to 0 (off)
 write-caching =  0 (off)
```

Abbildung A.2.: Mit hdparm den Festplatten-Cache deaktivieren

```
gluster@cluster1:~$ sudo hdparm -W0 /dev/sda
[sudo] password for gluster:

/dev/sda:
 setting drive write-caching to 0 (off)
 write-caching =  0 (off)
gluster@cluster1:~$ dd if=/dev/zero of=testfile bs=100M count=100 oflag=direct
100+0 Datensätze ein
100+0 Datensätze aus
10485760000 Bytes (10 GB) kopiert, 101,962 s, 103 MB/s
```

```
gluster@cluster2:~$ sudo hdparm -W0 /dev/sda
[sudo] password for gluster:

/dev/sda:
 setting drive write-caching to 0 (off)
 write-caching =  0 (off)
gluster@cluster2:~$ dd if=/dev/zero of=testfile bs=100M count=100 oflag=direct
100+0 Datensätze ein
100+0 Datensätze aus
10485760000 Bytes (10 GB) kopiert, 100,646 s, 104 MB/s
```

```
gluster@cluster3:~$ sudo hdparm -W0 /dev/sda
[sudo] password for gluster:

/dev/sda:
 setting drive write-caching to 0 (off)
 write-caching =  0 (off)
gluster@cluster3:~$ dd if=/dev/zero of=testfile bs=100M count=100 oflag=direct
100+0 Datensätze ein
100+0 Datensätze aus
10485760000 Bytes (10 GB) kopiert, 110,816 s, 94,6 MB/s
```

Abbildung A.3.: Mit dd die Schreibgeschwindigkeit der Festplatten ermitteln

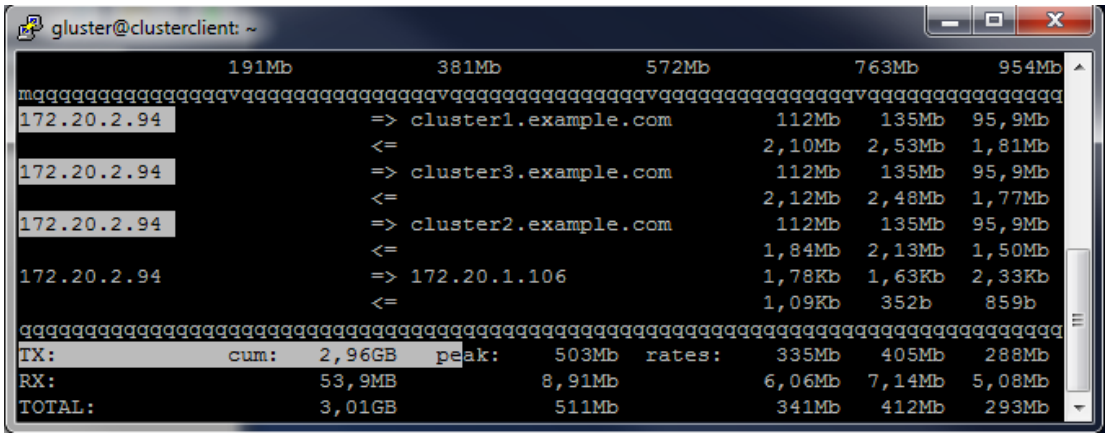


Abbildung A.4.: Mit *iftop* lässt sich die Geschwindigkeit zu den Servern ermitteln.

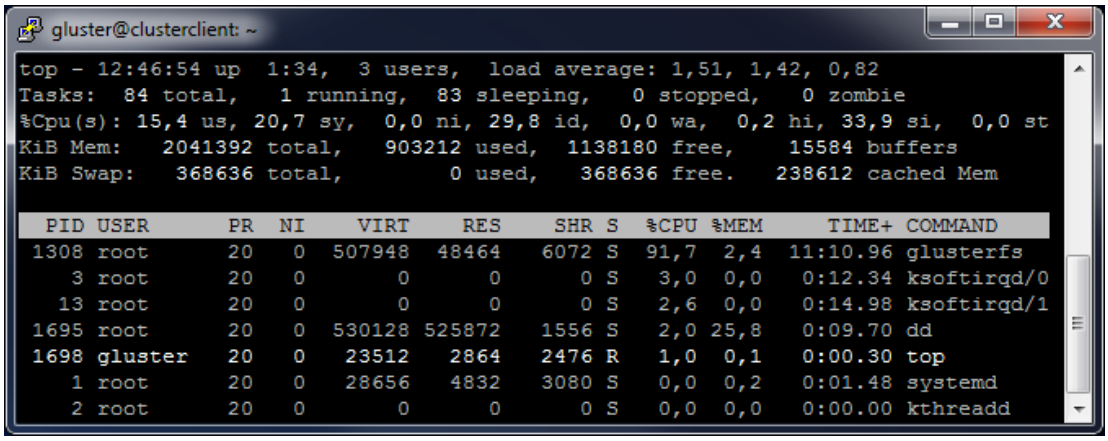


Abbildung A.5.: Mit *top* lässt sich die CPU-Auslastung des Clients im ersten Test überwachen.

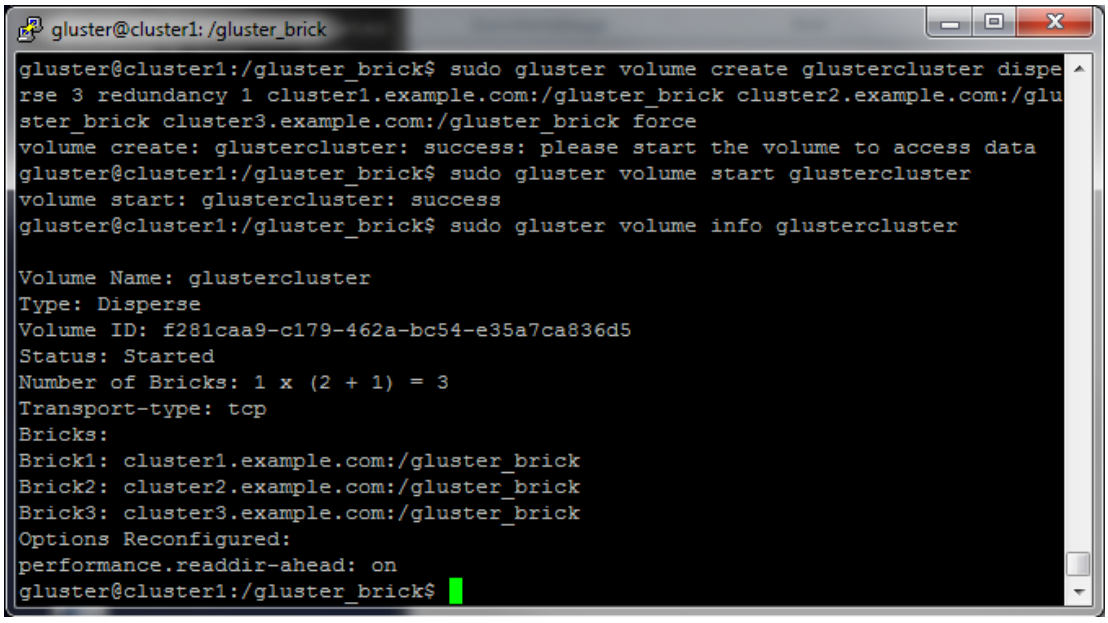


Abbildung A.6.: Werden drei Knoten im Disperse_Modus angelegt, ist der maximale Wert für Redundancy 1 (Number of Bricks: 1 x (2 + 1) = 3).

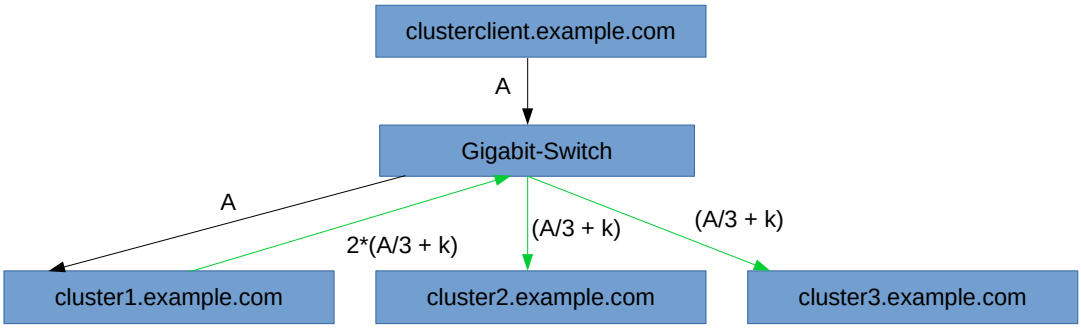


Abbildung A.7.: Freigabe des Volumes über NFS.

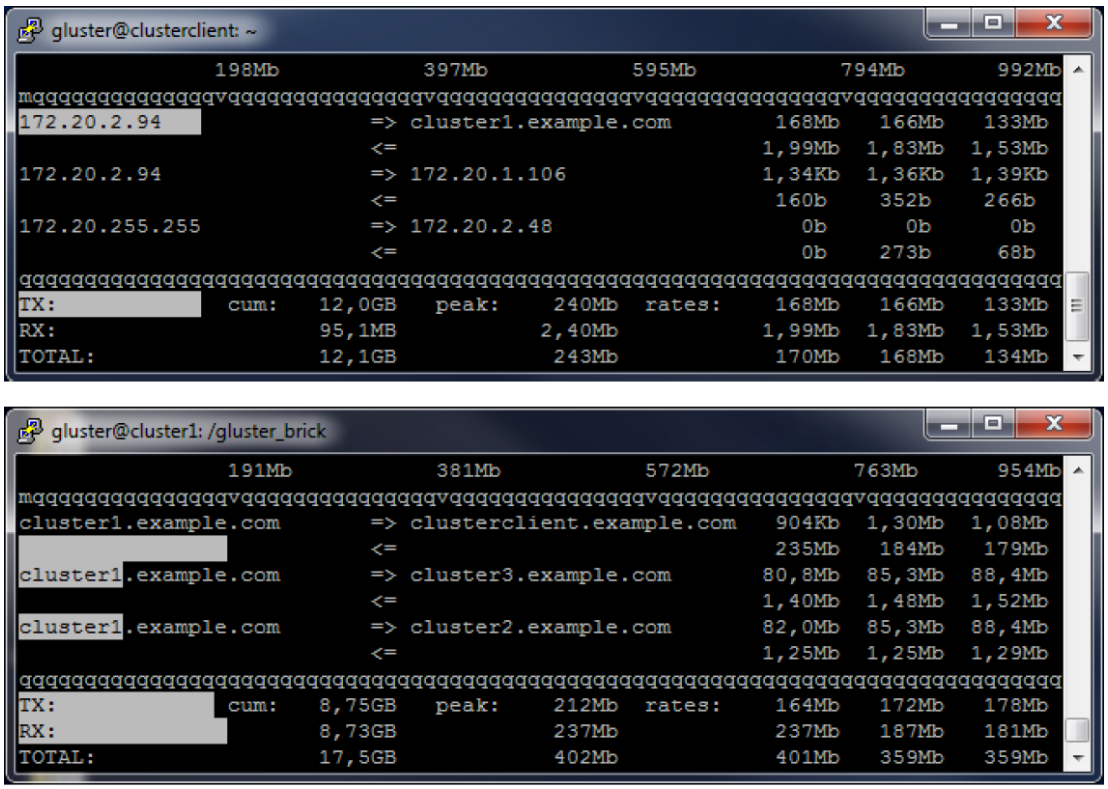


Abbildung A.8.: Netzwerktraffic bei NFS-Freigabe (oben *clusterclient.example.com*, unten *cluster1.example.com*)

Literatur

[Ash09] Kevin Ashton. »That ”Internet of Things” Thing«. In: *RFID Journal* (Juni 2009). http://www.itrco.jp/libraries/RFIDjournal-ThatInternetofThings pdf Zugriff am 19.10.2015.

[Azu] Microsoft Azur. *Verwenden von MapReduce mit Hadoop in HDInsight*. https://azure.microsoft.com/de-de/documentation/articles/hdinsight-use-mapreduce/ Zugriff am 19.11.2015.

[Ben14] Günther Bengel. *Grundkurs Verteilte Systeme*. 7. Aufl. Springer Vieweg, 2014.

[BHV14] Thomas Bauernhansl, Michael ten Hompel und Birgit Vogel-Heuser. *Industrie 4.0 in Produktion, Automatisierung und Logistik: Anwendung · Technologien · Migration*. Springer Vieweg, Mai 2014.

[Bor] Dhruba Borthakur. *HDFS Architecture Guide*. https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html. Zugriff am 15.10.2015.

[Bro09] Julian Browne. »Brewer's CAP Theorem«. In: (Jan. 2009). http://www.julianbrowne.com/article/viewer/brewers-cap-theorem Zugriff am 19.10.2015.

[But14] Paul Butcher. *Seven Concurrency Models in Seven Weeks: When Threads Unravel*. 1. Aufl. O'Reilly, Juli 2014.

[Cht+] Ekaterina Chtcherbina u. a. »Peer to Peer in Theorie und Praxis«. In: (). http://alt.euk.cs.ovgu.de/EuK/lehre/lehrveranstaltungen/Wintersemester0304/P2P_1/Chtcherbina_JS_04_02.pdf Zugriff am 05.11.2015.

[Cze11] Christop Czernohous. *Pervasive Linux*. Springer, 2011.

[DMS13] Benjamin Depardon, Gael Le Mahec und Cyril Seguin. *Analyses of Six Distributed File Systems*. Techn. Ber. https://hal.archives-ouvertes.fr/ Zugriff am 05.10.2015. HAL-Inra, 2013.

[Dok] Tachler's DokuWiki. »Hochverfügbarkeit GlusterFS«. In: (). http://www.dokuwiki.tachtler.net/doku.php?id=tachtler:hochverfuegbarkeit_glusterfs Zugriff 23.11.2015.

[Fan] Laurent Fanichet. »Big Data-Medienarchive mittels Object Storage bewältigen«. In: (). https://confluence.oceanobservatories.org/download/attachments/16418744/mysql-cluster-technical-whitepaper.pdf.

[FK] Thomas Findling und Thomas König. *MapReduce Konzept*. http://dbs.uni-leipzig.de/file/ Zugriff am 20.11.2015.

[Fre14] Jonas Freiknecht. *Big Data in der Praxis*. 1. Aufl. Hanser, Okt. 2014.

[GB10] Eva Geisberger und Manfred Broy. *Integrierte Forschungsagenda Cyber-Physical Systems*. http://www.acatech.de/?id=1405. Zugriff am 16.09.2015. 2010.

[Glua] Gluster.org. *GlusterFS Documentation*. https://gluster.readthedocs.org/en/latest/ Zugriff 14.01.2016.

[Glub] Gluster.org. *GlusterFS Documentation Community*. http://www.gluster.org/community/documentation/index.php/Gluster_3.2:_Setting_Volume_Options Zugriff 21.01.2016.

[Gro] Havard Research Group. »AVAILABILITY ENVIRONMENT CLASSIFICATIONS«. In: (). http://www.hrgresearch.com/ Zugriff 24.11.2015.

[Hal10] Stephan Haller. *The Things in the Internet of Things*. http://www.iot-a.eu/public/news/resources/TheThingsintheInternetofThings_SH.pdf. Zugriff am 19.10.2015. Nov. 2010.

[Han+13] Uwe Hansmann u. a. *SyncML: Synchronizing and Managing Your Mobile Data*. Feb. 2013. URL: http://www.informit.com/articles/article.aspx?p=31064.

[HPS] Thomas Hornung, Martin Przyjaciel-Zablocki und Alexander Schätzle. *Große Datenmengen mit Map-Reduce und Hadoop verarbeiten*. http://www.linux-magazin.de/Ausgaben/2012/04/Hadoop Zugriff am 19.11.2015.

[htta] http://www.redhat.com/. *Gluster Community Delivers Big Data Solution to Advance Enterprise Analytics*. http://www.redhat.com/de/about/press-releases/gluster-community-delivers-big-data-solution-to-advance-enterprise-analytics Zugriff 08.02.2016.

[httb] http://www.soutier.de/blog/2014/02/23/lambda-architektur/. *Skalierbare Datenhaltung mit der Lambda-Architektur*. http://www.soutier.de/blog/2014/02/23/lambda-architektur/ Zugriff 10.02.2016.

[Kal14] Thomas Kaltschmidt. »95 Prozent aller Geldautomaten laufen mit Windows XP«. In: (Jan. 2014). http://www.heise.de/newsticker/meldung/95-Prozent-aller-Geldautomaten-laufen-mit-Windows-XP-2088583.html Zugriff am 20.10.2015.

[Kli] Bernd Kling. »Edward Snowden bezeichnet Dropbox als ”datenschutzfeindlich”«. In: (). http://www.zdnet.de/88198985/edward-snowden-bezeichnet-dropbox-als-datenschutzfeindlich/?PageSpeed=noscript Zugriff am 09.11.2015.

[Klü14] Oliver Klünter. »Mobile Betriebssysteme im Vergleich: Firmentauglichkeit, Sicherheit, MDM«. In: (Feb. 2014). http://www.searchnetworking.de/meinung/Mobile-Betriebssysteme-im-Vergleich-Firmentauglichkeit-Sicherheit-MDM Zugriff am 20.10.2015.

[Kry] Kryptowissen.de. *Asymmetrische Verschlüsselung*. http://www.kryptowissen.de/asymmetrische-verschluesselung.html Zugriff am 30.10.2015.

[KS] Bejoy KS. *Word Count - Hadoop Map Reduce Example*. http://kickstarthadoop.blogspot.de/2011/04/word-count-hadoop-map-reduce-example.html Zugriff 15.03.2016.

[Len] Dr. Mario Lenz. »Big Data trifft Industrie 4.0«. In: (). http://remote-services2014.we-conect.com/cms/media/uploads/events/wc1415/dokumente/empolis-big-data-trifft-4.0.pdf.

[Lex] Datenbanken Online - Lexikon. *Datenbanken*. http://wikis.gm.fh-koeln.de/wiki_db/Datenbanken/DokumentenorientierteDatenbank Zugriff am 16.11.2015.

[Man03] Klaus Manhart. *Daten immer up to Date*. http://www.tecchannel.de/kommunikation/handy_pda/401715/daten_immer_up_to_date/. März 2003.

[Mat10] Friedemann Mattern. *Vom Internet der Computer zum Internet der Dinge*. http://link.springer.com/article/10.1007/s00287-010-0417-7. Zugriff am 16.10.2015. 2010.

[McL14] Ryan McLaughlin. »Mobile Security: Vor- und Nachteile bei der Verwendung von Container-Lösungen«. In: (Mai 2014). http://www.searchnetworking.de/sonderbeitrag/Mobile-Security-Vor-und-Nachteile-bei-der-Verwendung-von-Container-Loesungen Zugriff am 20.10.2015.

[Mey] Mathias Meyer. »The Simple Magic of Consistent Hashing«. In: (). http://www.paperplanes.de/2011/12/9/the-magic-of-consistent-hashing.html.

[MW15] Nathan Marz und James Warren. *Big Data: Principles and Best Practices of Scalable Realtime Data Systems*. MANNING, Mai 2015.

[Nei12] Yves Neißen. *Synchronisation Markup Language*. Techn. Ber. Fachhochschule Aachen, 2012.

[Pol] Malte Pollmann. *Industrie 4.0: IT-Sicherheit als elementare Voraussetzung*. http://www.datensicherheit.de/aktuelles/industrie-4-0-sicherheit-elementare-voraussetzung-24523. Zugriff am 16.10.2015.

[Rah94] Erhard Rahm. *Mehrrechner-Datenbanksysteme*. http://dbs.uni-leipzig.de/buecher/mrdbs/index.html Zugriff am 19.10.2015. Addison-Wesley, 1994.

[RSS15] Erhard Rahm, Gunter Saake und Kai-Uwe Sattler. *Verteiltes und Paralleles Datenmanagement*. Springer Vieweg, 2015.

[RT] Mikael Ronström und Lars Thalmann. »MySQL Cluster Architecture Overview«. In: (). https://confluence.oceanobservatories.org/download/attachments/16418744/mysql-cluster-technical-whitepaper.pdf.

[Sch02] Chana R. Schoenberger. »The Internet of Things«. In: *Forbes* (März 2002). http://www.forbes.com/forbes/2002/0318/155.html Zugriff am 19.10.2015.

[Sea] Seafile.org. »Installation im Cluster«. In: (). http://handbuch.seafile-server.org/deploy_pro/deploy_in_a_cluster.html Zugriff 23.11.2015.

[Sei] Udo Seidel. »Newcomer: Ceph und Gluster-FS«. In: (). http://www.linux-magazin.de/Ausgaben/2013/02/Ceph-und-Gluster.

[Sic] Bundesamt für Sicherheit in der Informationstechnik. *Regelung des Passwortgebrauchs.* https://www.bsi.bund.de/DE/Themen/ITGrundschutz/ITGrundschutzKataloge/Inhalt/_content/m/m02/m02011.html Zugriff am 13.11.2015.

[Spe] Speicherguide.de. »Verfügbarkeit komplexer Systeme«. In: (). http://www.speicherguide.de/management/hochverfuegbarkeit/verfuegbarkeiten komplexer-it-systeme-14053.aspx.

[SPS11] Stephan Spitz, Michael Pramateftakis und Hoachim Swoboda. *Kryptographie und IT-Sicherheit: Grundlagen und Anwendungen.* 2. Aufl. Vieweg + Teubner, März 2011.

[Tri] Andrew Tridgell. *The rsync algorithm.* Techn. Ber. https://rsync.samba.org/tech_report/ Zugriff am 03.11.2015.

[Ubu] Wiki Ubuntuuser. *Unison.* https://wiki.ubuntuusers.de/unison Zugriff am 02.11.2015.

[WCa] Thomas Wieland und Ekaterina Chtcherbina. »Peer review«. In: (). http://www.cpp-entwicklung.de/downld/chtch_wieland_p2p.pdf Zugriff am 05.11.2015.

[WCb] Thomas Wieland und Ekaterina Chtcherbina. »Peer-to-Peer – Anwendungsbereiche und Herausforderungen«. In: (). http://elk.informatik.fh-augsburg.de/oss/etc/P2P.pdf Zugriff am 05.11.2015.

[Web] Unison Webpräsenz. *Unison File Synchronizer.* http://www.cis.upenn.edu/~bcpierce/unison/ Zugriff am 02.11.2015.

[Whi10] Tom White. *Hadoop – The Definitive Guide.* 2. Aufl. O'REILLY, Nov. 2010.

[Wik] Wikipedia. *Festplattenlaufwerk.* https://de.wikipedia.org/wiki/Festplattenlaufwerk Zugriff am 20.11.2015.

[Xu] Fangming Xu. »BEEP - The Blocks Extensible Exchange Protocol Core«. In: (). https://www.wiselib.org/courses/ws0203/skm-svs/articles/xu.pdf Zugriff am 05.11.2015.

Glossar

Backdoor

Hintertüren oder Backdoors sind Teile eines Programmes, die eine für den Endbenutzer nicht gewollte Funktionalität bieten. Das könnte z.B. das Ausspionieren des Rechners ermöglichen.

Cluster

Logische Zusammenfassung von Festplatten unterschiedlicher Rechner.

Code-to-Data-Prinzip

Statt sich die Daten zu holen und diese dann auf einem Rechnerknoten auszuführen, wird beim Code-to-Data-Prinzip der Analysecode auf den verteilten Knoten ausgeführt. Dadurch wird eine verteilte Datenanalyse ermöglicht und die Eingabedaten müssen nicht über das Netzwerk geschickt werden.

Distributed Hash Table

Peers und Daten werden einem Hashwert zugeordnet. Hierdurch lassen sich Ressourcen in einem reinen P2P-System lokalisieren. Der Hashwert kann z.B. aus der IP oder der MAC-Adresse des Peers erstellt werden.

Erasure Code

Erasure Code ist eine Art der Forward Error Correction (FEC), bei dem ein Datensatz n in m Fragmente aufgeteilt und mit zusätzlichen Informationen k angereichert, auf verschiedene Knoten im System verteilt werden ($n = m + k$). Bei der EC 10/16 Konfiguration werden zu zehn Grundzeichen sechs weitere Zechen hinzugefügt. Wenn die sechzehn Datenfragmente verteilt gespeichert werden, könnten sechs Knoten ausfallen und die Zeichen wären dennoch reproduzierbar.

ETL

ETL steht für Extract, Transform, Load und beschreibt einen Prozess, bei dem Daten aus mehreren gegebenenfalls unterschiedlich strukturierten Datenquellen in einer Zieldatenbank vereinigt werden.

Heartbeat

Ein Netzwerkteilnehmer teilt einem anderen Netzwerkteilnehmer mit, dass er noch im Netzwerk erreichbar ist (am Leben).

heterogenen System

Heterogene Umgebung bedeutet in Bezug auf Betriebssysteme, dass Rechner in einem Netzwerk mit unterschiedlichen Betriebssystemen laufen z.B. Linux und Windows.

Infiniband

Serielle Hochgeschwindigkeitsübertragungstechnik, die meistens als Cluster-Verbindungstechnik zum Einsatz kommt.

Kafka

Kafka ist ein in Scala implementiertes, verteiltes Messaging-System. Es kann über ein Java-API auch in Java-Applikationen verwendet werden. Im Wesentlichen kann man mit Kafka eine Reihe persistenter Queues mit hohem Durchsatz bereitstellen. Ursprünglich wurde es konzipiert, um bei LinkedIn der Flut von Logs Herr zu werden.

Losgröße

Anzahl der produzierten Menge eines Produktes ohne Unterbrechung. Die Losgröße 1 wird als optimal angesehen, ist heutzutage aufgrund der *Rüstkosten* i.d.R. nicht sinnvoll.

Mean Time Between Failures

Beschreibt die Betriebsdauer zwischen zwei aufeinanderfolgenden Ausfällen.

Mean Time To Repair

Beschreibt die Zeit, die für eine Reparatur bei einem Ausfall benötigt wird.

Mirroring

Gegen Datenverlust werden häufig die Daten als exakte Kopie auf einer separaten Partition gespeichert.

Perfect Forward Secrecy

Schlüsselaustauschprotokoll, bei dem für jede Sitzung ein neuer Sitzungsschlüssel vereinbart wird.

Rüstkosten

Im Zeitablauf werden auf einer Maschine verschieden Produkte gefertigt. Für das jeweilige Produkt bedarf es einer mehr oder weniger aufwändigen Anpassung/Vorbereitung der

Maschine. Dieser sogenannte Rüstvorgang erzeugt Kosten. Ziel ist es, diese Rüstkosten so gering wie möglich zu halten.

Shared-Nothing-Architektur

In einer Distributed-Computing-Architektur erfüllt jeder Knoten unabhängig und eigenständig seine Aufgaben mit seinem eigenen Prozessor und den zugeordneten Speicherkomponenten (Festplatte, Hauptspeicher).

Single Point of Failure

Wenn der Ausfall einer Komponente zu einem Versagen des gesamten Systems führt.

soziotechnische Systeme

Menschen (soziales System), die in einem System mit Technologien (technisches System) verknüpft sind, mit dem Ziel ein Ergebnis zu erzielen z.B. ein Fließband.

Spurhalteassistent

Sorgt dafür, dass der Autofahrer gewarnt wird, falls das Auto einen bestimmten Abstand zur Fahrbahnmarkierung unterschreitet.

Striping

Um die Performance eines Speichersystems zu beschleunigen, können die Daten aufgeteilt und auf mehreren Festplatten gespeichert werden.

Transport Layer Security

TLS oder besser bekannt als Secure Sockets Layer (SSL) in Version 3.0 ist ein hybrides Verschlüsselungsprotokoll, das häufig bei HTTPS eingesetzt wird.

Ubiquitous Computing

Kurz ubicomp (engl.) bzw. Rechnerallgegenwart.

Universal Plug and Play

Mit UPnP ist es Anwendungen möglich Portfreigaben für Router ohne Eingriff eines Benutzers zu konfigurieren. Damit können diese Anwendungen durch eine Firewall mit der Gegenstelle kommunizieren.

Zero-Knowledge-Prinzip

Bei Cloud-Diensten gewährt dieses Prinzip eine erhöhte Privatsphäre, da die Daten einzig und allein vom Kunden entschlüsselt werden können.

Abkürzungsverzeichnis

ACC Adaptive Cruise Control, Deutsch: Abstandsregelung

SyncML Synchronisation Markup Language

WORM write once read many

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 17.03.2016 Steffen Bredemeier