Google Spanner

Proseminar Ein-/Ausgabe Stand der Wissenschaft

Hanno Harte

Betreuer: Julian Kunkel

24.6.13

Gliederung:

- Überblick
    - Exkurs SQL
- Funktionsweise
    - Exkurs Paxos
    - True Time
    - Konsistenzsemantik
- Benchmarks
- Zusammenfassung
- Quellen

Überblick:

Google Spanner ist ein seit 2009 existierendes Projekt der Google Inc. zur Vernetzung und Dezentralisierung der Server und Rechenzentren von Google.
Das Projekt umfasst eine globale Datenbank, durch die eine hohe Anzahl an Servern auf der ganzen Welt vernetzt wird. Bisher sind es voraussichtlich ca. eine Million Server in 13 auf dem Planeten verteilten Rechenzentren, was aber auf bis zu 10 Millionen Server ausgebaut werden soll in den kommenden Jahrzehnten.
Diese Daten sind allerdings nicht von Google bestätigt, sodass die wirklichen Werte abweichen können.
Diese Globale Vernetzung soll zu einer hohen Verfügbarkeit von Daten führen.
Erreicht werden soll dies durch einige grundlegende Maßnahmen die durch die Vernetzung möglich werden. Datenredundanzen auf verschiedenen Servern ermöglichen einen schnellen Zugriff und die Möglichkeit Daten zu dem, dem Kunden am nächsten gelegenen, Rechenzentrum zu verschieben.
Nähe zum Klienten ist ein wichtiger Bestandteil des Projektes, denn die daraus resultierenden geringen Latenzen sind eines der Ziele dieser Neuerungen.
Genauer werden die Latenzen beim Lesen einer Datei durch die Nähe zum Klienten, die Latenzen beim schreiben durch die Nähe der Replikationen untereinander und die Verfügbarkeit und Beständigkeit durch die Menge der Replikationen Zugesichert, worauf ich im späteren Verlauf noch weiter eingehen werde. Als eine weitere Sicherheit fungiert automatische Verschiebung von Replikationen auf einen anderen Server oder sogar ein anderes Rechenzentrum sofern Probleme im laufe einer Transaktion auftreten.
Google Spanner ist implementiert als eine multi-versionale Datenbank mit Zeitstempeln, also eine Datenbank, in der die Konsistenz gesichert ist dadurch, dass bei zeitgleichen Zugriffen auf eine Datei beide Zugriffe nicht blockiert werden. Dazu ist es nötig, und von Google erstmals auf globaler Ebene gelungen, die Zeitstempel derart präzise zuteilen zu können, das jede Transaktion von jeder anderen unterschieden werden kann.
Da Google Spanner ein SQL ähnliches relationales Datenbanksystem ist, mit Reihen Spalten und Werten, folgt ein kleiner Exkurs zu SQL. Zu beachten ist jedoch, dass Google Spanner nicht vollständig relational ist, wie SQL, da Reihen ebenfalls einen Namen besitzen müssen.

Exkurs SQL

Structured Query Language, auf deutsch Strukturierte Abfrage Sprache, ist eine Sprache für Datenbanken zur Bearbeitung und Abfrage von Datenbeständen.
Zusätzlich definiert sie die Datenstrukturen, was bedeutet das sie definiert in welcher Struktur Daten verwaltet werden und welche Operationen auf die Daten anwendbar sind. Dies soll natürlich so effizient wie möglich geschehen um einen besseren Zugriff auf die Daten zu erhalten.
SQL wird hauptsächlich für relationale Datenbanksysteme eingesetzt, wobei relational davon kommt, dass die Tabellen in Relationen angeordnet sind und jeweils ein eindeutiger Schlüssel oder eine eindeutige Kombination aus Schlüsseln pro Tabelle existiert.
SQL ist angelehnt an die englische Umgangssprache und unterliegt einigen Normen, um auf möglichst vielen Datenbanksystemen zu laufen und leicht verständlich zu sein.
Folgende Graphik ist ein Beispiel wie die Tabellen in einem mit SQL angelegten Datenbanksystem aussehen könnte.

Student

MatrNr	Name
26120	Fichte
25403	Jonas
27103	Fauler

hört

MatrNr	VorlNr
25403	5001
26120	5001
26120	5045

Vorlesung

VorlNr	Titel	PersNr
5001	ET	15
5022	IT	12
5045	DB	12

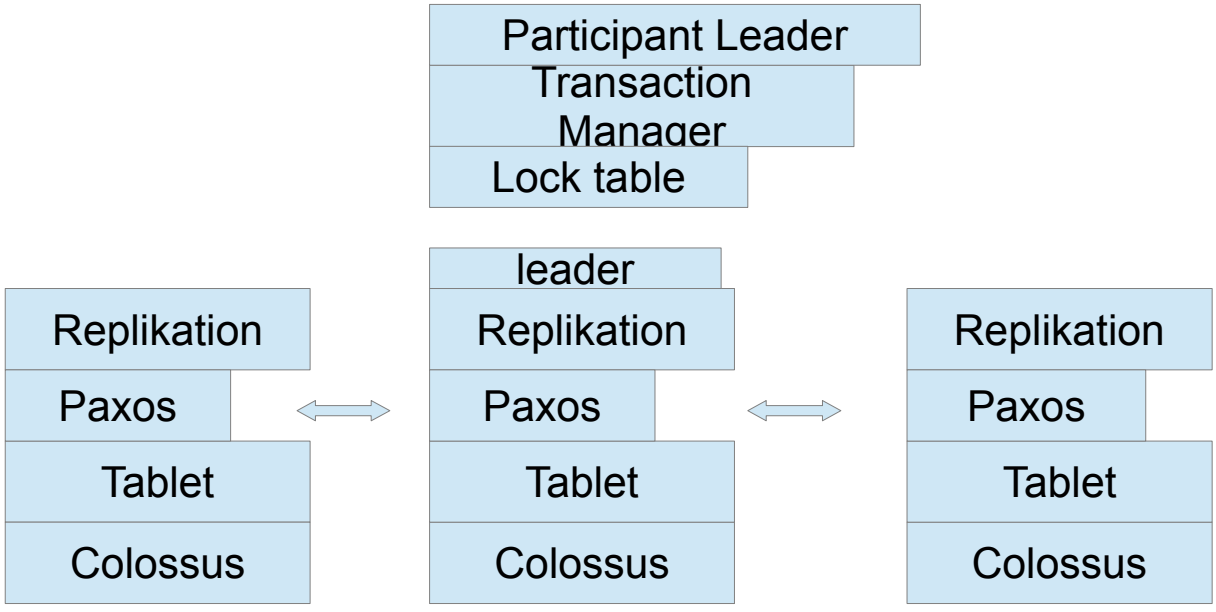
Professor

PersNr	Name
12	Wirth
15	Tesla
20	Urlauber

Es gibt eine Tabelle für Studenten, die die Attribute Matrikelnummer und Name besitzt, wobei die Matrikelnummer den Schlüssel bildet und worüber die Tabelle identifiziert wird. Das gleiche gilt für die Tabellen Professor und Vorlesung. Resultierend aus diesen drei Tabellen folgt eine Tabelle in der verzeichnet ist, welcher Student welche Vorlesung hört mit den Schlüsseln aus beiden dadurch verbundenen Tabellen.

Funktionsweise

Angenommen der Klient möchte auf eine von Google abgespeicherte Information zugreifen, so befindet sich diese in einer Tabelle in dem wie zuvor erwähnt SQL ähnlichen relationalen Datenbanksystem.
Wie erwähnt bestehen diese Tabellen aus Reihen, Spalten und Werten, so müssen alle Tabellen je eine geordnete Spalte mit Primärschlüsseln haben, die den Namen der Reihe bilden.
Die Hierarchie der Tabellen wird von jeder Anwendung selbst partitioniert, dennoch muss die oberste Tabelle die sogenannte „directory-table“ sein, deren Schlüssel zu beginn jeder Reihe stehen. Diese hier beschriebenen Tabellen liegen auf den so genannten Spannservern, implementiert anhand von B-Trees auf dem Nachfolger des Google File Systems „Colossus“.
Im Hintergrund läuft auf diesen Spannservern die background Anwendung „Movedir“, die Replikationen von Daten erzeugt und löscht. Die zum Teil vorher in Fragmente zerteilten Daten werden zum größten Teil im, wie der Name backgound Anwendung schon verrät, Hintergrund verschoben, wodurch gesichert ist, dass das verschieben die Daten gegenüber Zugriffen nicht blockiert werden. Erst sobald nahezu die gesamten Daten übertragen sind wird der kleine Rest im Vordergrund verschoben und die Metadaten angepasst.
Innerhalb jedes Spannservers existieren zwischen 100 und 1000 solcher Datenstrukturen, die als Tablet bezeichnet werden und über denen ein so genannter Paxos steht.



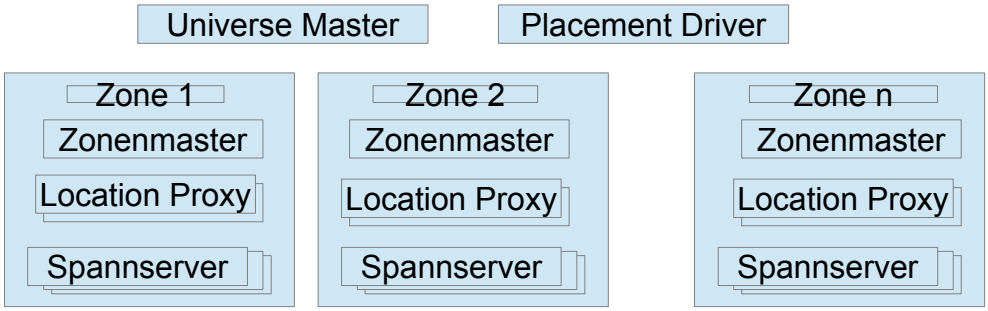
Exkurs Paxos:

Paxos ist eine Familie von Protokollen zur Vereinfachung der Interaktion zwischen verschiedenen Prozessoren.
Hierbei werden Anfragen an das System an verschiedene Prozessoren geschickt von denen jeder eine Replikation erhalten muss und anhand dieser dann eine Aufgabe erfüllt.
Zu diesen Aufgaben gehört als erstes der Client, der die Anfrage für einen schreibenden Zugriff an das System sendet und dann auf eine Antwort wartet.
Dort kommt zunächst der Proposer in Einsatz, welcher die Anfrage des Clienten weiter leitet an die verschiedenen Acceptor.
Einer der Proposer wird zunächst als Leader definiert und übernimmt daraufhin die Koordination.
Acceptor sind der Fehler tolerante Speicher des Protokolls und sind angesammelt in einem so genannten Quorum. Jeder Acceptor in diesem Quorum muss die Anfrage bekommen und akzeptieren. Erst wenn alle akzeptiert haben, wird der Vorgang fortgeführt.
Hier kommt der Learner ins Spiel, der dafür sorgt, dass die Antwort auf die von allen Acceptoren akzeptierte Anfrage an den Client geschickt wird.
Im Bezug auf Spanner wird der long lived leader hervorgehoben. Damit lange klar ist, ob die Transaktion ausgeführt wurde oder nicht, muss der Zeitstempel tt.after(smax) also der Zeitstempel nach der Transaktion von smax also der maximalen Zeit auf true stehen bevor der leader aus seiner Aufgabe entlassen wird.

Zusammengefasst soll dies dazu dienen bei großen Systemen gleiche Zustände übermittelt zu bekommen, wodurch sich Daten einfacher von einem Server auf einen anderen Transferieren lassen, sowie eine große Anzahl von Anfragen möglich wird.
Trotz großer Anzahl von Anfragen kann hierdurch auch die korrekte Reihenfolge der Ausführungen garantiert werden.
Dennoch musste die zunächst geplante Anwendung von mehreren Paxos Protokollen pro Einheit zurückgestuft werden auf einen, da die Komplexität Probleme in der Implementierung hervorrief.
Der Paxos schreibt jede Transaktion in zwei logs, den eigenen und den des Tablets.
Diese Aufzeichnungen treten immer wieder repliziert auf und bilden so eine Ansammlung, die Paxos Group genannt, im Tablet gespeichert und vom Paxos verwaltet wird.

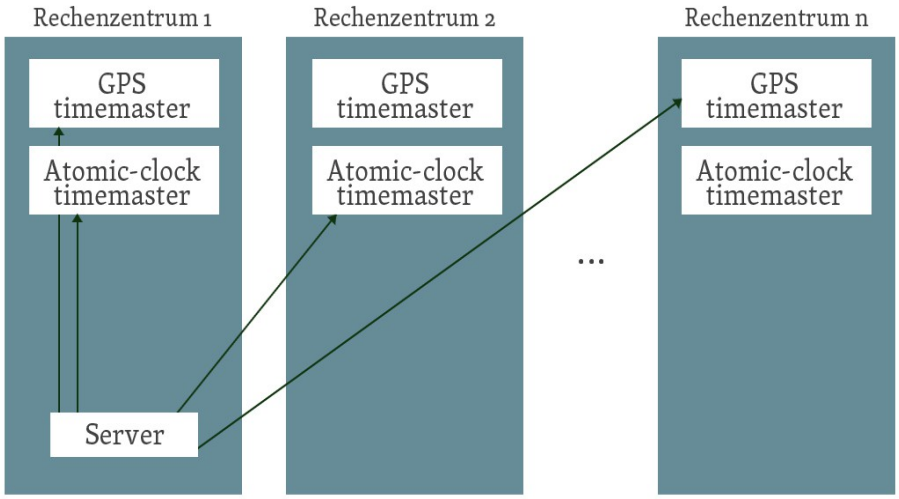
Jeder schreibende Zugriff muss zunächst das Paxos Protokoll initiieren, wogegen lesende Zugriffe direkt vom Tablet verwaltet werden.
Jeder Spannserver implementiert eine auf die Hauptreplikation angewendete lock table, die zur Erhaltung der Konsistenz beiträgt. Sie zeichnet die Abstände der Scdhlüssel von Zuständen auf für das so genannte two phase locking.
Two phase locking besteht zwei Phasen, der Expending Phase, in der nur zu blockierende Daten hinzugenommen werden, und der Shrinking Phase, in der keine mehr hinzugefügt werden sondern nur noch die bereits hinzugefügten wieder aus der Blockierung entlassen werden. Bestimmte Zugriffe setzen diese locks voraus, andere umgehen die lock table.
Sobald hiermit auf mehrere Paxos Groups gleichzeitig zugegriffen werden muss, schaltet sich er Transaction Manager ein und setzt einen der Leader aus den verschiedenen Paxos Groups als Koordinator für die gesamte Transaktion ein.
Um zu wissen welcher Server der richtige für den Client ist, existieren location Proxies die dem Kunden die Position des Spannservers mitteilen.

Die Spannserver werden verwaltet in so genanten Zonen, in der es einen Zonenmaster und bis zu einigen tausend Spannservern gibt. Der Zonenmaster bildet hierbei die Verwaltungseinheit.
Zonen müssen nicht zwangsläufig lokal voneinander getrennt sein, es können auch mehrere Zonen in einem Rechenzentrum existieren, um beispielsweise verschiedene Anwendungen von einander abzugrenzen. Somit ist auch die Möglichkeit gegeben Zonen beliebig hinzuzufügen oder zu entfernen.
Den Zonen Übergeordnet sind ein Universe Master und ein Placement Driver.
Der Universe Master enthält eine Übersicht über alle Zonen und Zustände für interaktives debugging. Der Placement Driver verschiebt automatisiert Daten zwischen Zonen und kommuniziert regelmäßig mit den Spannservern, um Überfüllung zu beheben, Daten zu finden die verschoben werden müssen und Replikationen abzugleichen.
Der gesamte Einsatzbereich mit allem nun erwähnten nennt sich Universe.



True Time:

Das NTP (Network Time Protocoll) wurde normalerweise dazu benutzt Server in einem Rechenzentrum mit den Atomuhren der Welt zu verknüpfen. Es war jedoch sehr Fehler intolerant, wie sich im July 2012 anhand einer Schaltsekunde zeigt, durch die viele Systeme mit NTP abstürzten.
Um eine derartige Abhängigkeit zu vermeiden hat Google True Time entwickelt, welches eine eigene API hierfür bildet. Google schaffte hierfür eigens einige Atomuhren und GPS Reciever an. Zwei verschiedene Arten von Zeitmessungen, um Fehlerquellen auszuschalten die beispielsweise durch die Beeinflussung von GPS durch Radiowellen entstehen können. Verbunden sind diese Atomuhren und GPS Reciever mit den Masterservern in den jeweiligen Rechenzentren von Google.



Ein Hintergrundprogramm, der so genannte time slave daemon, läuft auf jedem Server im Hintergrund und gleicht ständig die Zeit mit den Masterservern im eigenen und in anderen Rechnezentren ab. So soll gewährleistet werden, dass in allen Rechenzentren von Google immer die gleiche Zeit auf den Servern vorliegt.
Zwischen den einzelnen Synchronisierungen fügt der daemon ein epsilon ein, welches die worst case Abweichung der lokalen Uhr von der durch die Master vorgegebene Zeit darstellt. Diese liegt in der Regel zwischen 1 und 7 ms, wovon 0-6 die so genannte „Sägezahn“ Abweichung sind, bei einem Abstimmungsintervall von 30 Sekunden und einer Abweichungsrate von 200 Microsekunden pro Sekunde.
Die fehlende Millisekunde ergibt sich aus der Kommunikationsverzögerung mit dem Master. Daraus folgt, dass die Durchschnittsabweichung bei ca. 4 ms liegt.

Abweichungen hiervon können durch Fehler entstehen wie zum Beispiel die Unerreichbarkeit eines Masters, die sich dann auf das gesamte Rechenzentrum auswirkt.

Method	Returns
TT.now()	Ttinterval:[earliest, latest]
TT.after(t)	True if t definitely passed
TT.before(t)	True if t has definitely not arrived

Mit den in dieser Tabelle gezeigten Methoden wird der Zeitstempel in True Time erzeugt. Ttintervall ist die absolute Zeit, in der TT.now() benutzt wurde, also die Zeit des Zugriffes, und TT.before() sowie TT.after() bilden die Wrapper Klassen um TT.now().

Konsistenzsemantik:

Die Konsistenz ist bei den verschiedenen Arten von Zugriffen unterschiedlich gesichert. Beginnen möchte ich hier mit den Read Only Transaktionen, bei der sichergestellt sein muss, dass alle Zugriffe in einer Transaktion die gleiche Version der Datenbank sehen. Dazu muss ein Zugriff als read only vordeklariert sein und sichert dann zu, dass dieser nicht blockiert.
Bei diesem Zugriff wird auf eine beliebige aktuelle Replikation des Originals zugegriffen. Snapshot reads, reads bei denen es möglich ist den Zeistempel zu wählen oder eine Obergrenze sodass Spanner selbst den Zeistempel festlegt, bilden die Implementation dieser Reads. Bei diesen reads wird anhand von s-read = tt.now().latest der zeistempel des reads festgelegt, der auf den zeitlich nächstgelegenen Zeitstempel zugreift der externe Konsistenz inne hatte. Die bei dem read benötigten Schlüssel werden vorher im scope Ausdruck zusammengefasst.
Sobald die Schlüssel zusammengefasst und der Zeitstempel gewählt wurde wird der read unter allen Umständen ausgeführt und es ist nicht nötig die reads zu buffern, da der read nun bei einem Server Ausfall auf einem anderen Server beziehungsweise auf einer anderen Replikation weitergeführt werden kann. Hierzu wird der Zeitstempel und die Leseposition erneut gesendet und es tritt kein Abbruch auf.

Die Zeitstempel werden in einer Paxos Group in monoton steigender Reihenfolge übergeben. Bestehend aus e-start und e-commit muss das e-commit immer kleiner sein als das einer zweiten Transaktion, die später begonnen hat.
Die read write Transaktionen funktionieren etwas anders, da hier die Datei verändert wird und somit für eine bestimmte Zeit blockiert werden muss.
Writes werden auf dem Client gebuffered und erst nach Abschluss der Transaktion wird der Zeitstempel für dieselbe erzeugt.
Zur Erhaltung der Konsistenz und speziell um deadlocks zu vermeiden verwendet Google Spanner das so genannte wound wait Verfahren. In diesem Verfahren geht es um die Sperren die von read write Transaktionen gehalten werden. Im Falle von mehreren Sperren auf die selbe Datei so wird die zeitlich gesehen jüngere neu gestartet und die ältere Transaktion bekommt die Zuweisung. Hält eine Ältere Sperre bereits die Zuweisung so muss die Jüngere warten, bis die ältere die Zuweisung freigibt.
Aufrecht gehalten werden diese Sperren anhand von keep alive messages die dafür sorgen, dass eine Sperre so lange erhalten bleibt, bis alle reads gelesen und die writes gebuffered sind. Erst danach kann das two phase commit beginnen.
Es wird ein Koordinator gewählt, dessen Identität und alle gebufferten writes an alle beteiligten Paxos Groups gesendet wird. Das nun folgende two phase commit besteht, wie der Name schon sagt aus zwei Phasen und ist ein Protokoll zur Sicherung von Transaktionen bei Fehlern.
Die erste Phase besteht aus einer Nachricht des Koordinators an alle ausführenden Stellen, dem Ausführen der Transaktion und dem Schreiben in zwei logs, einem undo und einem redo log. Daraufhin wird an den Koordinator eine Nachricht gesandt, ob die Transaktion erfolgreich war oder nicht.
In der zweiten Phase reagiert der Koordinator auf die Nachricht.
Ist sie in allen Stellen mit ja beantwortet, so wird erneut eine Nachricht an die ausführenden Stellen geschickt, die Transaktion fertiggestellt und alle locks werden freigegeben.
Ist die Nachricht in einer Stelle jedoch negativ so wird vom Koordinator eine Rollback Nachricht ausgegeben und anhand des undo logs wird in allen Stellen die Transaktion rückgängig gemacht.

Im Bezug auf Spanner schicken alle nicht Koordinator leader einen zunächst gesetzten prepare Zeitstempel an den Koordinator, welcher dann einen Zeitstempel für die gesamte Transaktion festlegt. Hier muss der Zeitstempel wieder größer sein als jeder der zuvor schon einmal festgelegt wurde und ebenfalls größer oder gleich tt.now().latest.
Tritt während der Transaktion kein Time-out ein, so wird vom Koordinator in den Paxos eine Ausführungsaufzeichnung geschrieben.
Um zu garantieren, dass der verwendete Zeitstempel in der Vergangenheit liegt, wartet der Koordinator nun auf tt.after(s), wobei s der zuvor gewählte prepare Zeitstempel ist.
TT.after(s) muss hierbei mindestens 2* epsilon also 2 mal die Standartabweichung sein.
Der neue Zeitstempel wird nun an alle leader und den Client gesendet und die locks gelöst.
Write locks werden vom leader vorausgesetzt, um zuzusichern das auch wirklich keine Überschneidungen möglich sind.

Die so genannten shema change transactions sind nicht blockierende Transaktionen die die Tabellenstruktur im Bezug auf Reihen und Spalten verändern.
Durchgeführt werden diese wie eine normale Transaktion bei der zunächst ein Zeitstempel in der Zukunft festgelegt wird (prepare phase) und nach der Transaktion mit einem Zeitstempel t synchronisiert wird. Der Zeitstempel der Transaktion muss vor t liegen und somit ist auch hier die Konsistenz zugesichert.
Diese genaue Spezifikation und somit die Behandlung einer shema change transaction als normale Transaktion wird erst durch True Time möglich.

Benchmarks:

Um ihr Projekt zu testen hat google auf einer Plattform mit ca 50 Paxos Groups und Maschinen mit 4 GB Ram sowie AMD Barcelona 2200MHz getestet. Die Client Maschinen waren hierbei auf ein andere Maschine ausgelagert wodurch eine Netzwerkdistanz von etwas weniger als 1 ms entstand. Die Latenz des Paxos lag bei ca 9 ms plus eine Wartezeit von 5 ms.
Getestet wurde anhand von Zugriffen mit 4 kb Größe sowohl reads als auch writes. Dabei wurde festgestellt, dass bei steigender Zahl von Replikationen die Latenz relativ konstant blieb und der Datendurchsatz bei reads linear mit der Anzahl der Replikationen wuchs.

Two phase commit wurde separat getestet an 3 Zonen á 25 Spannservern in der laut google die Skalierbarkeit sehr gut funktioniert haben soll.

	Latenz (ms)	
participants	mean	99th percentile
1	17.0 + - 1.4	75.0 + - 34.9
2	24.5 + - 2.5	87.6 + - 35.9
5	31.5 + - 6.2	104.5 + - 52.2
10	30.0 + - 3.7	95.6 + - 25.4
25	35.5 + - 5.6	100.4 + - 42.7
50	42.7 + - 4.1	93.7 + - 22.9
100	71.4+ - 7.6	131.2 + - 17.6
200	150.5 + - 11	320.3 + - 35.1

Zusammenfassung:

Durch Google Spanner wird eine Globale Vernetzung aller Rechenzentren von Google erreicht und dazu die Latenzen minimiert.
Es existiert eine hohe Fehlertoleranz, besonders im Bezug auf Zeit durch die worst case Variable epsilon.
Die wohl größte Errungenschaft des Projektes ist die Möglichkeit einen eindeutigen Zeitstempel zu erzeugen trotz großer Distanzen und Schwierigkeiten mit der Zeitgleichschaltung. Denn es sind ja denkbare Szenarien, dass sehr viele Menschen gleichzeitig von der ganzen Welt auf eine Datei Zugreifen und trotzdem kann der Zeitstempel eindeutig zugewiesen werden.
Ein weiterer wichtiger Punkt hierfür sind die nicht blockierenden Reads, wodurch es weiterhin möglich ist zeitgleich eine Datei zu lesen.

Quellen:

1] http://static.googleusercontent.com/external_content/
untrusted_dlcp/research.google.com/en//archive/spanner-osdi2012.pdf

[2] http://business.chip.de/news/Google-Spanner-Redundanz-fuer-10-Millionen-Server_42369620.html

[3] http://www.wired.com/wiredenterprise/2012/11/google-spanner-time/

[4] Vorlage: http://www.neogrid.de/Bilder-Lexikon.php?Bild_Nr=8&Feld=Google-Spanner&Serie=no

[5] http://de.wikipedia.org/wiki/SQL