



„NoSQL Datenbanken am Beispiel von HBase“

Daniel Georg



N★SQL



No to SQL at all

sondern

Not only SQL

Open-Source Community

Erst im Jahr 2009 gestartet

Community bietet verschiedene Lösungen:
Cassandra, CouchDD, HBase, Hypertable...



Anforderungen

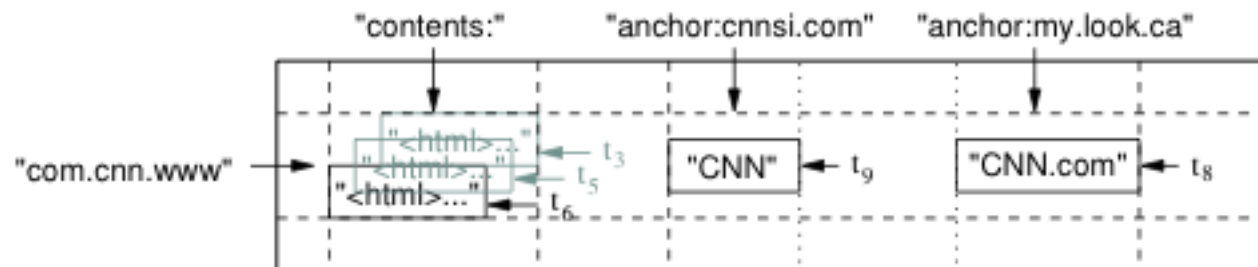
- Performance bei high-load Projekten
- Skalierbarkeit
- Erweiterbarkeit
- Verfügbarkeit
- Antwortzeit



BigTable von Google

- **Datenmodell**

- Eine dünn besetzte, verteilte, sortierte und mehrdimensionale Abbildung.
- Jeder Eintrag in der Abbildung wird als row key, column key, und timestamp : **(row:string, column:string, time:int64) → string** indiziert.

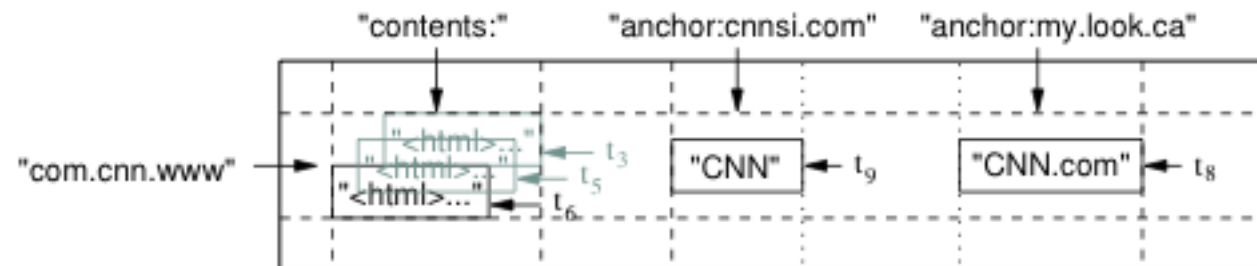




BigTable von Google

- **Datenmodell**

- **Zeilen** in den Tabellen sind nach dem Schlüssel in der lexikographischen Ordnung sortiert.
- Lese- Sreibzugriffe sind atomar.

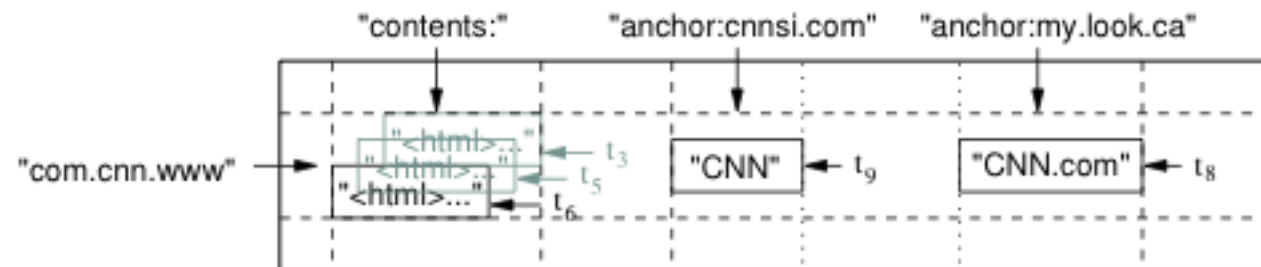




BigTable von Google

- **Datenmodell**

- **Spalten-Schlüssel** sind in Gruppen organisiert und werden Spaltenfamilien (column-family) genannt.
- Der Schlüssel wird aus **family:qualifier** gebildet

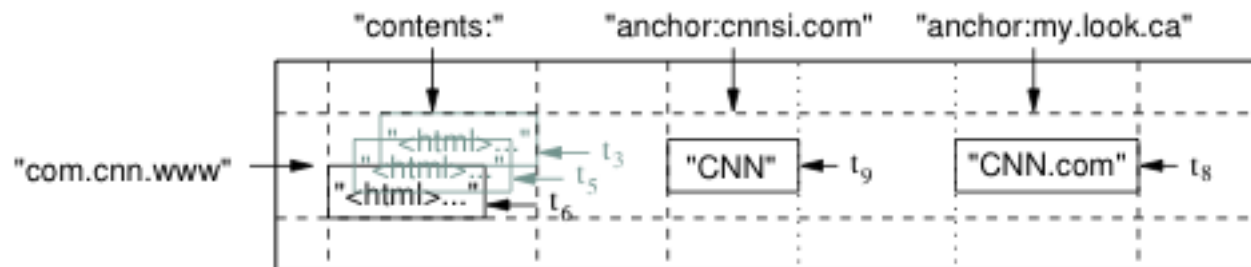




BigTable von Google

- **Datenmodell**

- Mehrere Versionen pro Zeile
- Jede Version hat einen eigenen **Zeitstempel**
- Ermöglicht Automatische Entfernung von veralteten Daten.





Architektur von BigTable

- BigTable-Cluster bestehen aus dem Master-Server und einer Menge von Tablet- Servern.
- Jeder Tablet-Server ermöglicht den Zugriff auf bestimmte Tablets.
- Zur Speicherung des Standortes der Tablets wird eine dreistufige Hierarchie verwendet.



Architektur von BigTable

- **Aufgaben von Master-Server:**
 - Verteilungen von Tablets auf die Tablet- Server.
 - Überwachung von Tablet-Servern, Load-Balancing zwischen Servern.
 - Entfernung von veralteten Daten.
 - Steuert die Erzeugung und Änderungen von Tabellen sowie der Family-Columns.



Architektur von BigTable

- **Aufgaben von Tablet-Server:**
 - Verarbeiten die Lese-Schreibanfragen an die betreuten Tablets.
 - Teilung von Tablets.
 - Dynamisches Hinzufügen oder Entfernen im Cluster.



Architektur von BigTable

- **Dreistufige Hierarchie**

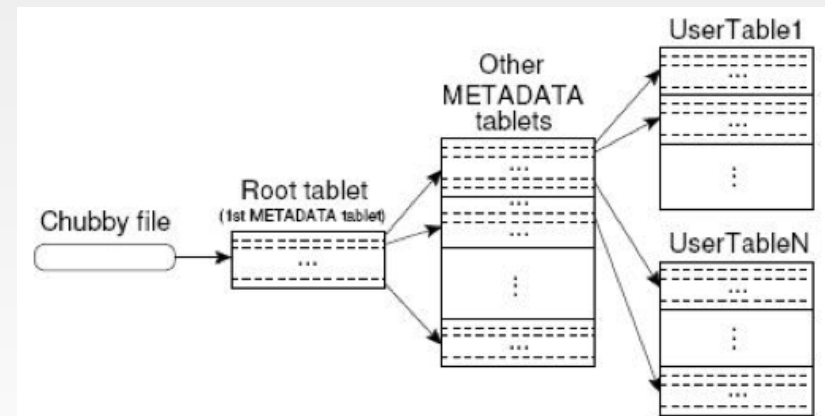
- Root tablet

- Verweise auf Metadaten Tablet
- Im Chubby gespeichert
- Wird Nie aufgeteilt

- Metadaten Tablet

- Verweise auf User Tablets

- User Tablets





Architektur von BigTable

- **Tablets**
 - Im Form einer Datei auf GFS abgelegt.
 - Alle Änderungen werden im Log-File gespeichert.
 - Neue Änderungen im Puffer gespeichert (memtable)
 - Älteren Änderungen in dem SSTables gespeichert.
 - Verdichtung:
 - Aus Memtables wird eine SSTable erzeugt.
 - Aus Mehreren SSTables wird eine SSTable erzeugt.
 - Alle Memtables und SSTables werden in eine SSTable vermischt.



- Eine OpenSource Alternative zum BigTable-System
- Ähnliche Datenmodell und Realisation
- HBase verwendet eine etwas andere Terminologie als BigTable
- Existieren kleine spezifische Unterschiede



- **Unterschiede:**
 - Tablet als Region bezeichnet
 - Server die Regionen betreuen sind RegionServer
 - Die Daten werden auf Familien von Spalten aufgeteilt
 - Jede Familie wird in Store gespeichert
 - Store ist Analog zu SSTable
 - Die Daten werden auf dem HDFS abgelegt und heißen StoreFiles
 - Hbase verwendet kein Chubby Service
 - Ist in Java realisiert



Yahoo PNUTS

- Daten können als geordnete oder gehashte Datenstruktur gespeichert
- Speziell für Webapplikationen entwickelt
- Arbeitet mit Duplikaten
- Hinzufügen von neuen Attributen während der Laufzeit möglich
- Geografisch orientiertes Datenbanksystem



Yahoo PNUTS

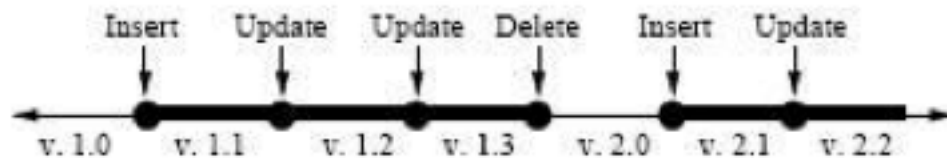
- **Anfragemodell**
 - Selektionen und Projektionen auf die einfachen Tabellen
 - Beim Update oder Delete muss erstmal der Primärschlüssel deklariert werden
 - Unterstützt keine Join- oder Gruppierungs Anfragen



Yahoo PNUTS

- **per-record- time-consistency**

- Die Repliken bewegen sich immer vorwärts auf dem Zeitstrahl.
- Jeder Datensatz hat eine Nummer
- Nummer besteht aus der Generation und der Versionsnummer
- Jedes Insert ist eine neue Generation
- Bei jeden Update wird eine neue Version erzeugt.



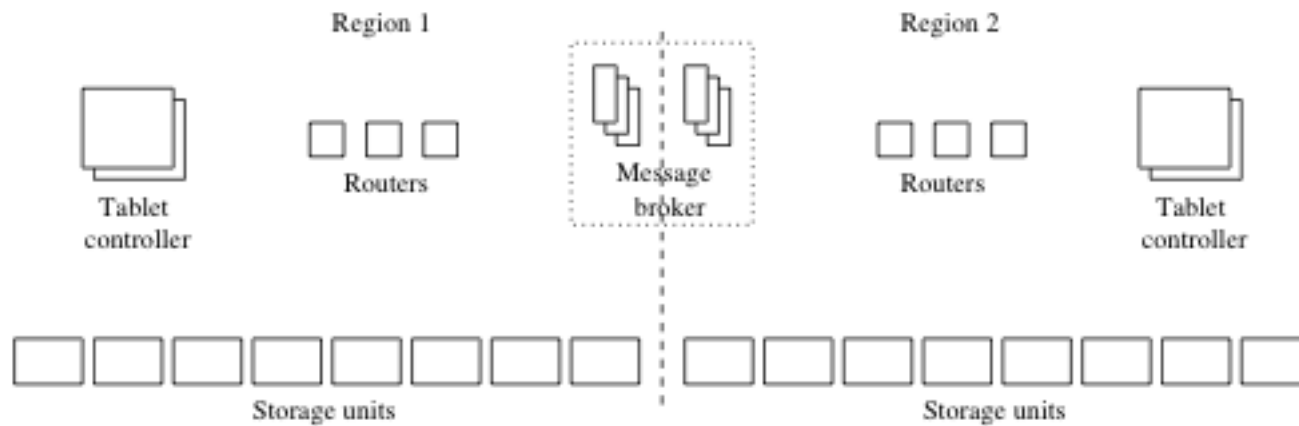


Yahoo PNUTS

- **per-record- time-consistency** bietet eine Reihe von API Funktionen:
 - Read-any
 - Read-critical (requered_version)
 - Read-lastes
 - Test- and-set-write(required)



Architektur von Yahoo PNUTS





Architektur von Yahoo PNUTS

- Das System wird in Regionen geteilt.
- Jedes Region hat alle benötigte Systemkomponenten und eine Kopie von allen Tabellen.
- Verwendung von Archiven oder Backups unnötig.
- Tabellen sind in Tablets unterteilt.
- Jedes Tablet ist innerhalb einer Region nur auf einen Server gespeichert.



Architektur von Yahoo PNUTS

- **Storage-Units**
 - Hier werden die Tablets gespeichert
 - Beantworten die `get()`, `set()` und `scan()` anfragen
 - Bei Hash-Tabellen wird ein Unix Dateisystem verwenden
 - Für die geordnete Tabelle wird `mySQL` mit `InnoDB` verwendet



Architektur von Yahoo PNUTS

- **Router**
 - Bestimmt welche Storage-Unit für einen zu lesenden oder zu schreibenden Datensatz zuständig ist.
 - Der Router speichert eine Liste mit den Intervallen und den dazugehörigen Storage-Units.
- **Tablet-Controller**
 - Führt mappen
 - Teilt die Tablets



Architektur von Yahoo PNUTS

- **Yahoo Message-Broker**
 - Ersatz zum Redo-log
 - Für das Replizieren und Protokollieren zuständig
 - Änderungen werden erstmal dem YMB übergeben, danach verteilt YMB asynchron die Änderungen auf verschiedene Regionen und sie werden danach auf ihre eigenen Repliken angewandt.
 - Die Nachrichten gehen nicht verloren bevor sie auf die Datenbank angewendet wurden.