Cassandra - Eine Einführung

Mikio L. Braun
Leo Jugel

TU Berlin, twimpact

LinuxTag Berlin
13. Mai 2011

Was ist NoSQL?

- Für viele Webanwendungen sind “klassische Datenbanken” nicht die richtige Wahl:
  - Datenbank im Wesentlichen nur Speicher für Objekte
  - Konsistenz nicht unbedingt erforderlich
  - Viele gleichzeitige Zugriffe

NoSQL im Vergleich

Klassische Datenbanken	NoSQL
Mächtige Querysprache	Sehr einfache Querysprache
Skaliert durch größere Server (“Scaling Up”)	Skaliert auf einem Cluster (“Scaling Out”)
Änderungen des Datenbankschemas aufwendig	Kein festes Datenbankschema
ACID: Atomicity, Consistency, Isolation, Durability	In der Regel nur “eventually consistent”.
Transaktionen, Locking, etc.	In der Regel keine Unterstützung für Transaktionen o.ä.

Brewer's CAP Theorem

- CAP: Consistency, Availability, Partition Tolerance
  - **Consistency:** Man erhält keine veralteten Daten.
  - **Availability:** Lese/Schreiboperationen sind immer möglich.
  - **Partition Tolerance:** Bei Netzwerk oder Rechnerausfällen bleiben die restlichen Garantien erhalten.
- Man kann nur jeweils zwei davon garantieren!

Gilbert, Lynch, *Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services*, ACM SIGACT News, Volume 33, Issue 2, June 2002

Homepage	http://cassandra.apache.org
Programmiersprache	Java
Geschichte	• Bei Facebook zur Indexsuche entwickelt, im Juli 2008 als Open Source freigegeben • Seit März 2009 Apache Incubator • Seit Februar 2010 Apache Top-Level
Haupteigenschaften	• strukturierter Key-Value-Store • “eventually consistent” • völlig gleichberechtigte Knoten • Cluster ohne Neustart modifizierbar
Support	DataStax (http://datastax.com)
Lizenz	Apache 2.0

Version 0.6.x und 0.7.x

- Wichtigsten Änderungen in 0.7.x
  - Konfigurationsdateien in YAML statt XML
  - Schemamodifikationen (ColumnFamilies) im laufenden Betrieb
  - Erste Ansätze für Sekundärindexe.
- Allerdings auch anfängliche Stabilitätsprobleme

Inspirationen für Cassandra

- Amazon Dynamo
  - Cluster ohne dezidierte Masterknoten
  - Peer-to-Peer Erkennung, HintedHintoff, etc.
- Google BigTable
  - Datenmodell
  - Benötigt zentrale Masterknoten
  - Bietet wesentlich mehr Möglichkeiten der Kontrolle:
    - welche Daten gemeinsam gespeichert werden sollten
    - On-the-fly Kompression, etc.

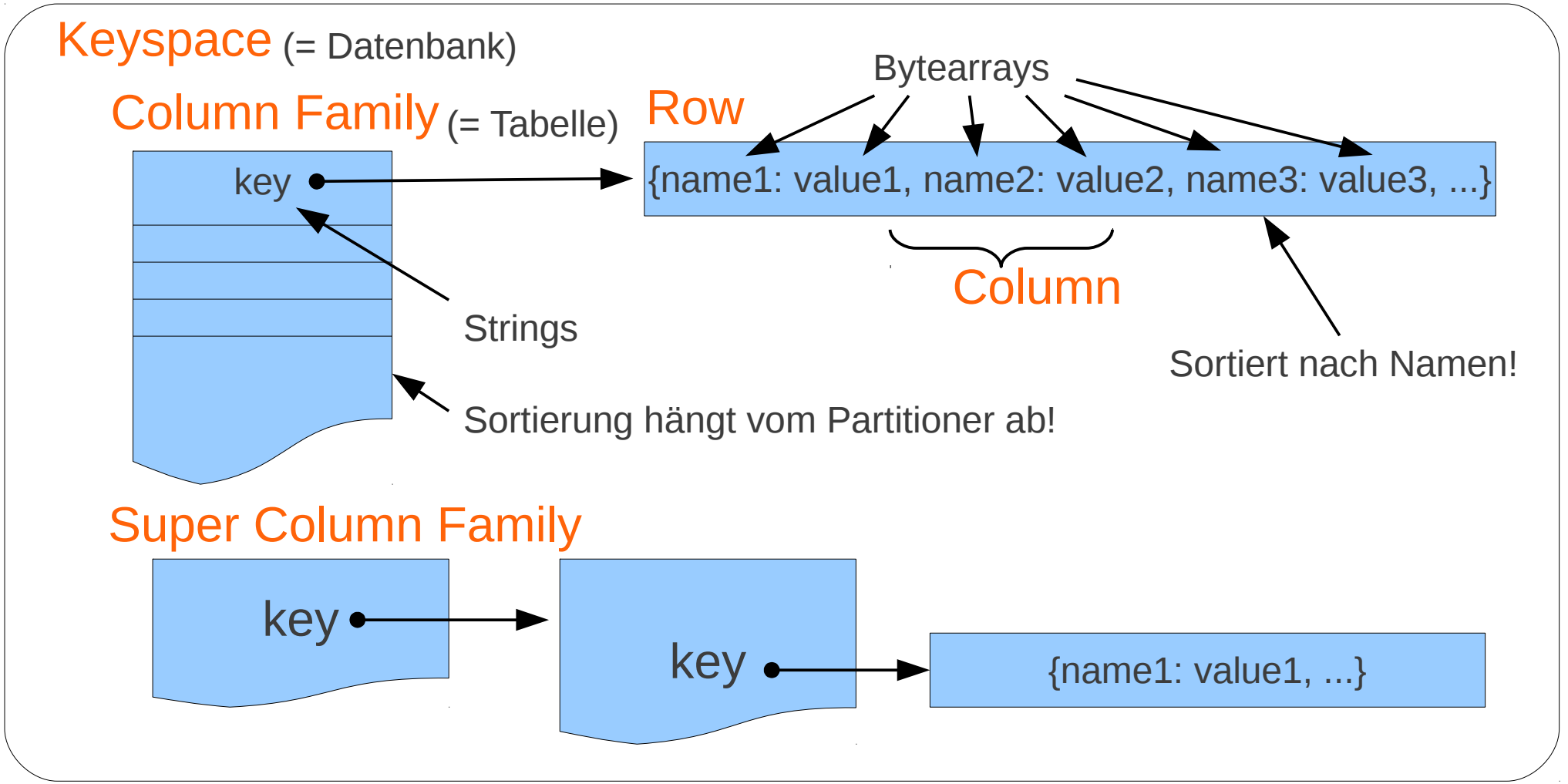
Installation

- tar.gz von http://cassandra.apache.org/download/ herunterladen
- In Verzeichnis auspacken
- ./conf enthält Konfigurationsdateien
- ./bin/cassandra -f startet Cassandra, Ctrl-C zum stoppen

Konfiguration

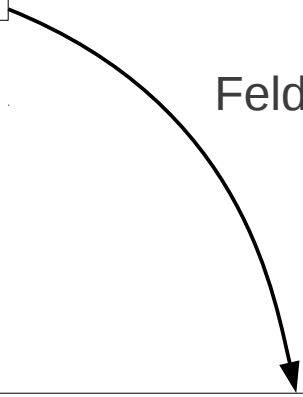
- Datenbank
  - Version 0.6.x: conf/storage-conf.xml
  - Version 0.7.x: conf/cassandra.yaml
- JVM Parameter
  - Version 0.6.x: bin/cassandra.in.sh
  - Version 0.7.x: conf/cassandra-env.sh

Cassandras Datenmodell



Beispiel: Einfacher Objektspeicher

```
class Person {
        long id;
        String name;
        String affiliation;
}
```

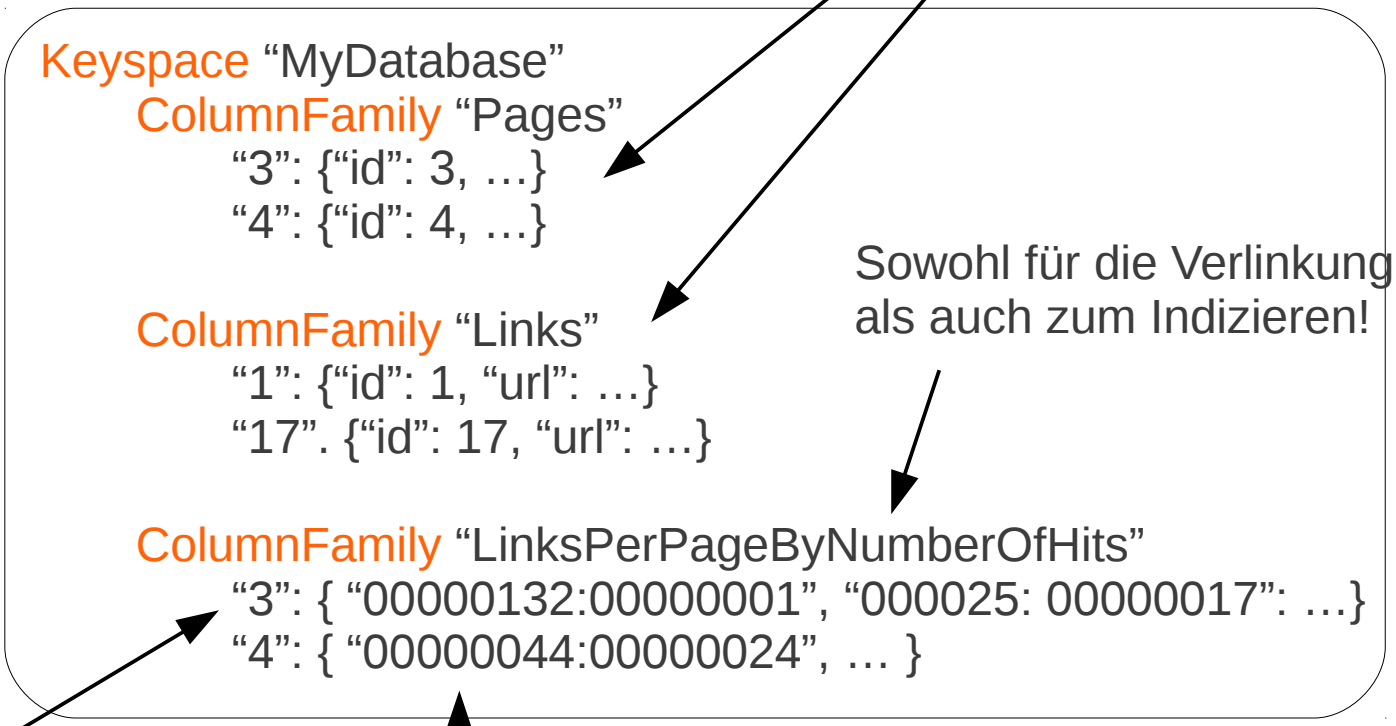


Felder in Bytearrays umwandeln

Keyspace “MyDatabase”:
    ColumnFamily “Person”:
        “1”: {“id”: “1”, “name”: “Mikio Braun, “affiliation”: “TU Berlin”}

Beispiel: Index

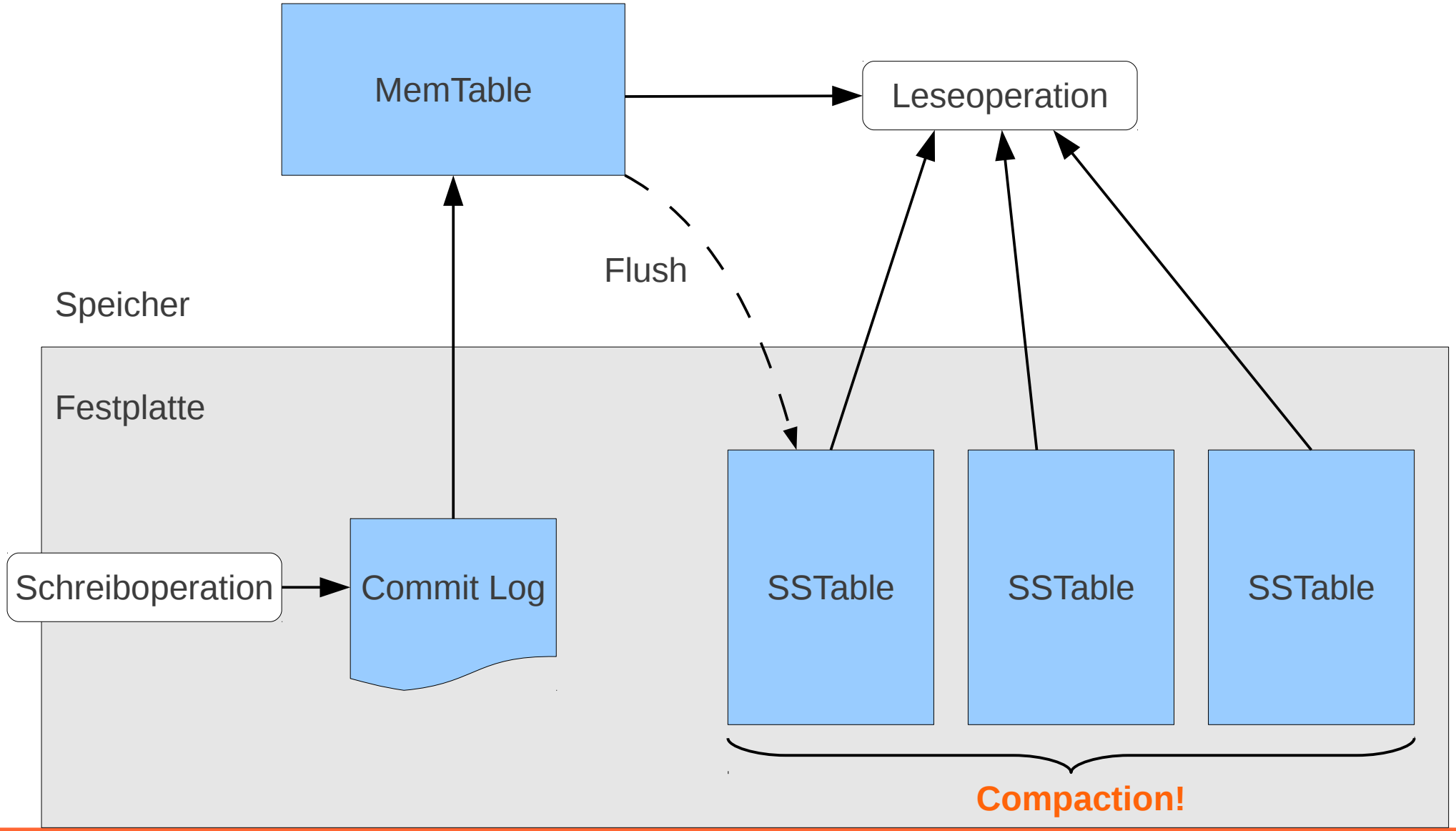
```
class Page {
        long id;
        …
        List<Links> links;
}

class Link {
        long id;
        ...
        int numberOfHits;
}
```



Benötigt man SuperColumnFamilies?

- Meistens kann man SuperColumnFamilies durch mehrere ColumnFamilies ersetzen.
- Da die SuperColumnFamilies die Implementation und das Protokol verkomplizieren gibt es auch Stimmen, die sie ganz abschaffen wollen... .

Cassandras Architektur



Cassandras API

- THRIFT-basierte API

Leseoperationen	
get	Einzelne Column
get_slice	Reihe von Columns
multiget_slice	Reihe von Columns in verschiedenen Rows
get_count	Anzahl von Columns
get_range_slice	Reihe von Columns in einer Reihe von Rows
get_indexed_slices	Reihe von Columns mit Index

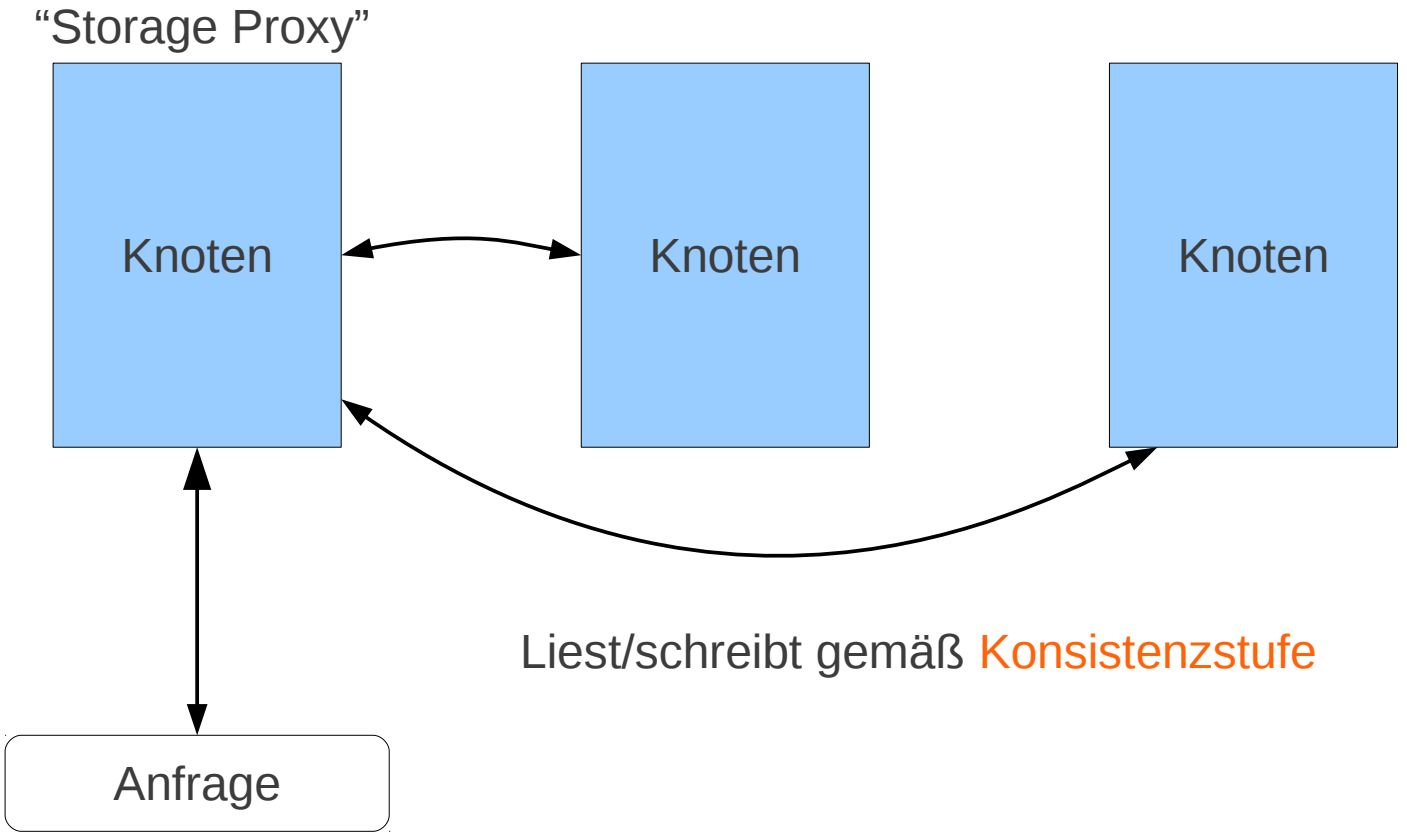
Schreiboperationen	
insert	Einzelne Column
batch_mutate	Mehrere Columns in verschiedenen Rows
remove	Einzelne Column
truncate	Ganze Column Family

Sonstige
login, describe_*, add/drop column family/keyspace

erst in 0.7.x

Cassandra im Cluster

- Völlig gleichberechtigte Knoten
- Zum Bootstrappen ein Seed nötig



Konsistenzstufen und Replikationsfaktor

- Replikationsfaktor: Auf wie vielen Knoten werden die Daten gespeichert?

- Konsistenzstufen:

Konsistenzlevel	
ANY	Ein Knoten hat die Operation bekommen, selbst im Fall von HintedHandoff
ONE	Auf einem Knoten wurde die Operation ausgeführt.
QUORUM	Operation für die Mehrzahl der Knoten ausgeführt / Neuestes Resultat wird zurückgegeben
LOCAL_QUORUM	QUORUM im lokalen Datenzentrum
GLOBAL_QUORUM	QUORUM im globalen Datenzentrum
ALL	Warte, bis alle Knoten die Operation ausgeführt haben.

Verhalten bei Ausfall

- Solange Anforderungen der Konsistenzstufe erfüllt werden können, ist alles in Ordnung.
- **Hinted Handoff:**
  - Eine Schreiboperations für einen ausgefallenen Knoten wird auf einem anderen Knoten gemerkt.
  - Diese Daten sind anschließend nicht lesbar!
- **Read Repair:**
  - Selbst nachdem die Leseoperation abgeschlossen ist, werden die Daten überprüft und ggf. aktualisiert.

Bibliotheken

Python	Pycassa: http://github.com/pycassa/pycass Telephus: http://github.com/driftx/Telephus
Java	Datanucleus JDO:http://github.com/tnine/Datanucleus-Cassandra-Plugin Hector: http://github.com/rantav/hector Kundera http://code.google.com/p/kundera/ Pelops: http://github.com/s7/scale7-pelops
Grails	grails-cassandra: https://github.com/wolpert/grails-cassandra
.NET	Aquiles: http://aquiles.codeplex.com/ FluentCassandra: http://github.com/managedfusion/fluentcassandra
Ruby	Cassandra: http://github.com/fauna/cassandra
PHP	phpcassa: http://github.com/thobbs/phpcassa SimpleCassie: http://code.google.com/p/simpletools-php/wiki/SimpleCassie

Oder was selbstgebautes direkt auf THRIFT http://thrift.apache.org/ :)

TWIMPACT: Eine Anwendung

- Echtzeitanalyse von Twitter
- Trendanalyse basierend auf Retweets
- Sehr hohes Datenaufkommen (mehrere Millionen Tweets am Tag, ca. 50/s)

TWIMPACT: twimpact.jp

TWIMPACT: twimpact.com

Anwendungsprofil

- Informationen über Tweets, User, Retweets
- Textmatching für nicht-API-Retweets
- Retweetshäufigkeiten und Userimpact
- Operationsprofil:

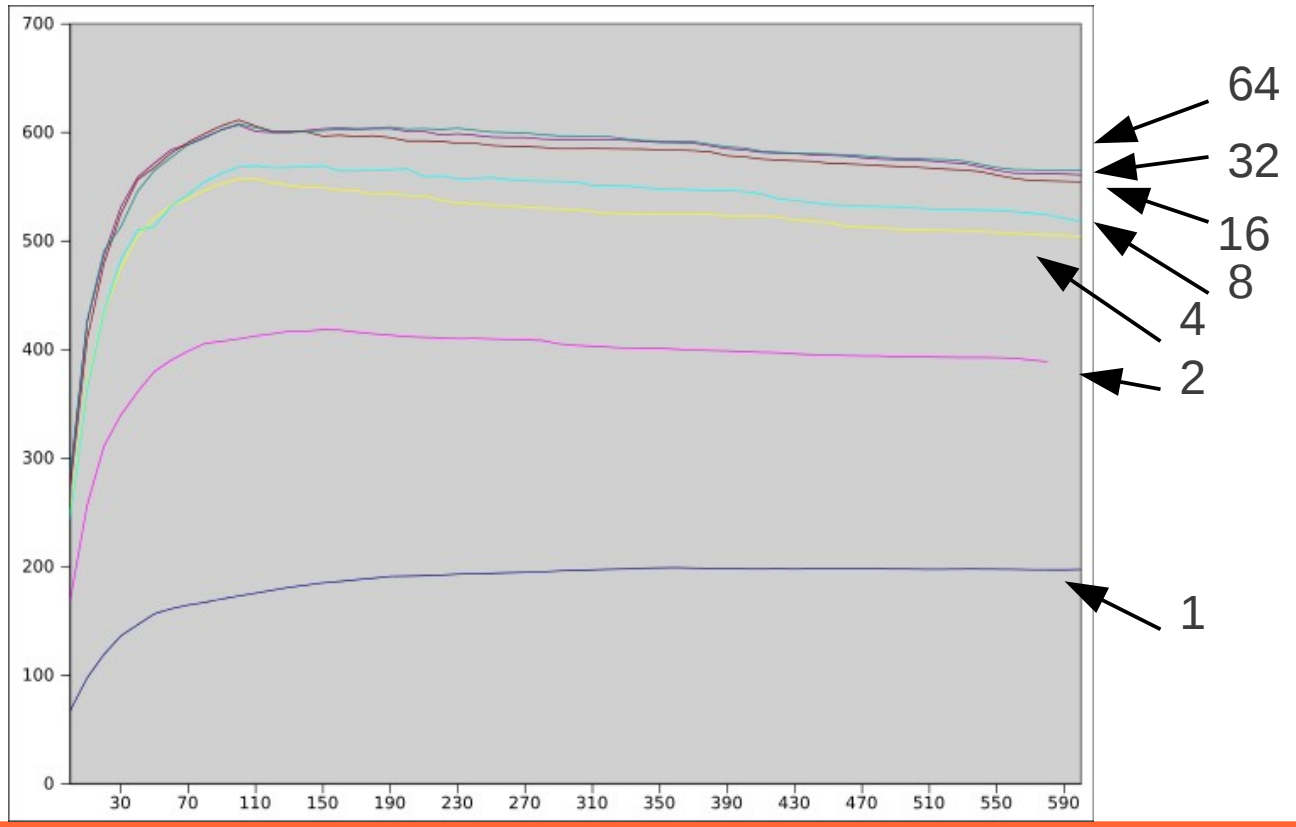
	get_slice (alle)	get	get_slice (Bereich)	batch_mutate (eine Row)	insert	batch_mutate	remove
Anteil	50.1%	6.0%	0.1%	14.9%	21.5%	6.8%	0.8%
Dauer	1.1ms	1.7ms	0.8ms	0.9ms	1.1ms	0.8ms	1.2ms

Praktische Erfahrungen mit Cassandra

- Sehr stabiler Betrieb
- Leseoperationen verhältnismäßig teuer
- Multithreading bringt große Performanceverbesserung
- Relativ aufwendiges Tuning erforderlich
- Clusterbetrieb bringt nicht unbedingt Verbesserung
- Compaction führt zu bis zu 50% Performanceeinbruch

Performance durch Multithreading

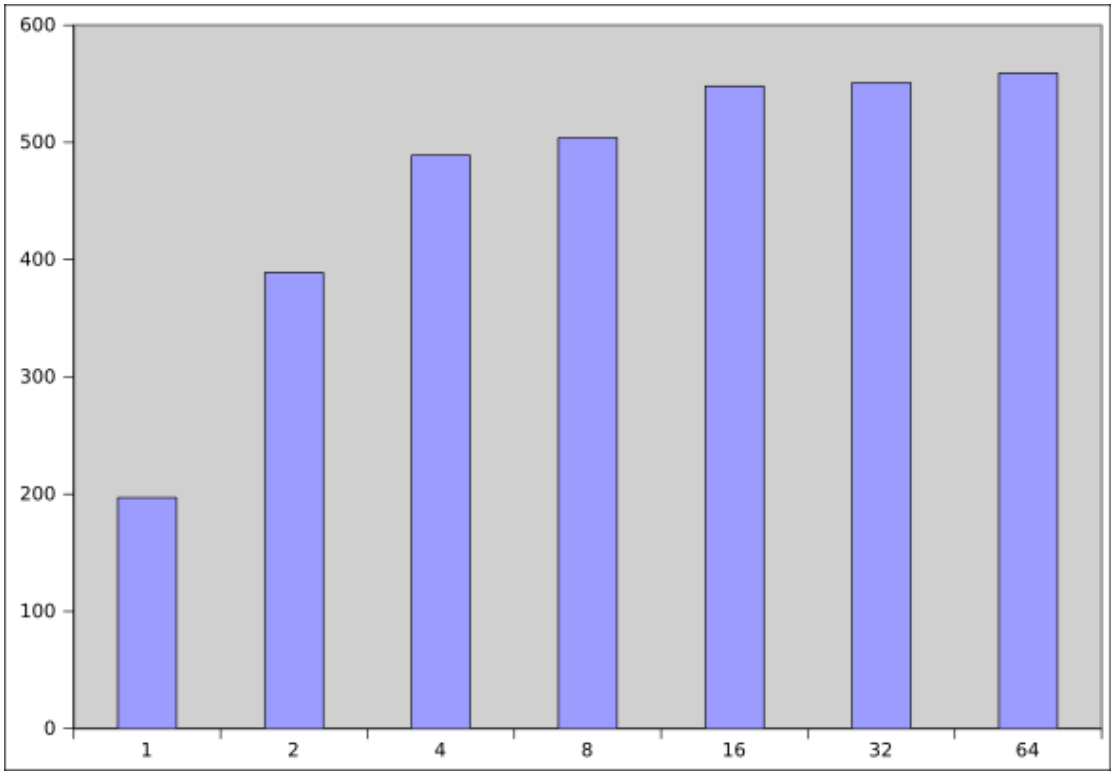
- Multithreading bringt erheblich mehr Performance.
- Multithreading ohne Transaktionen/Locking?



Core i7,
4 cores
(2 + 2 HT)

Performance durch Multithreading

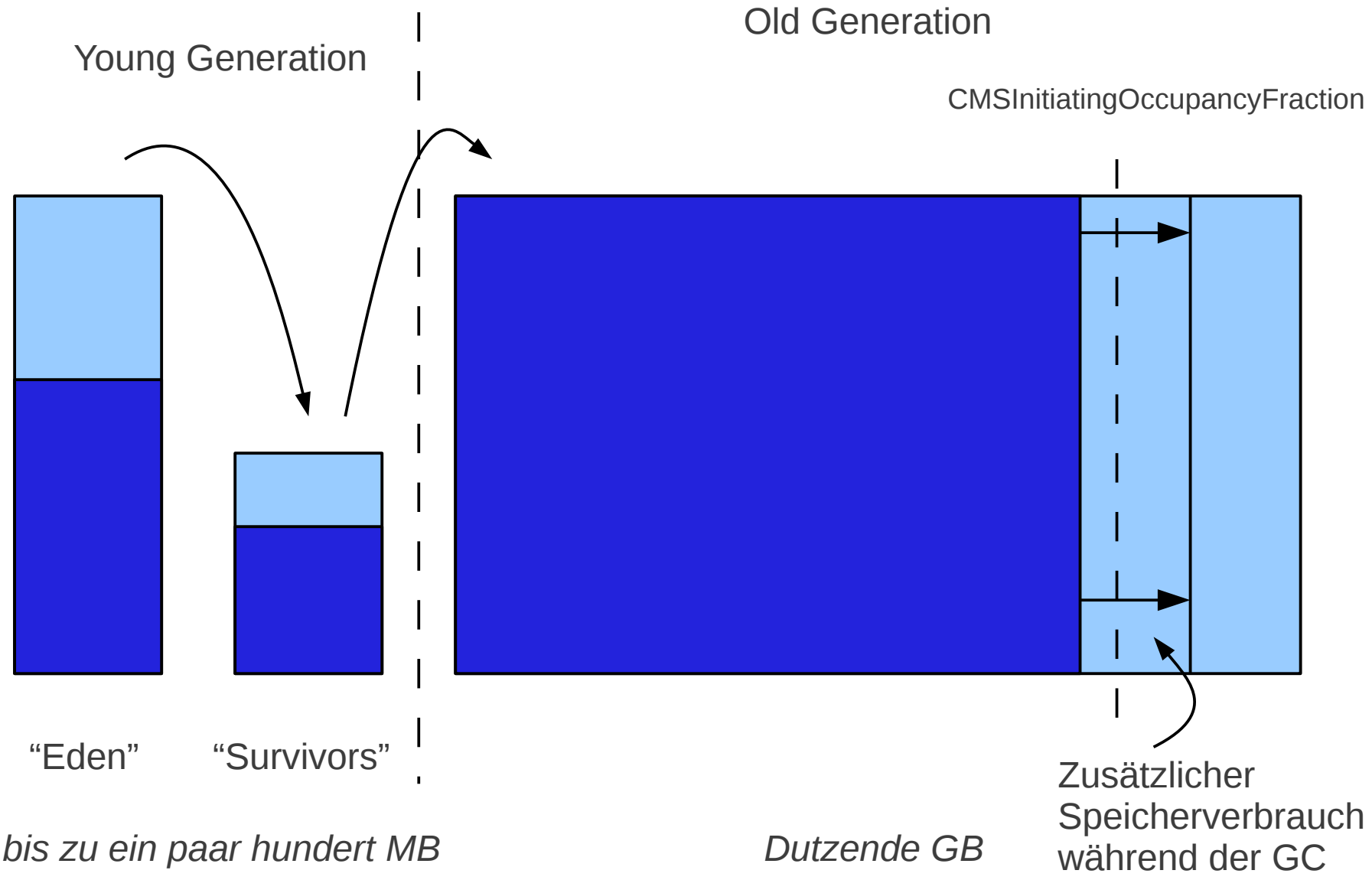
- Multithreading bringt erheblich mehr Performance.
- Multithreading ohne Transaktionen/Locking?



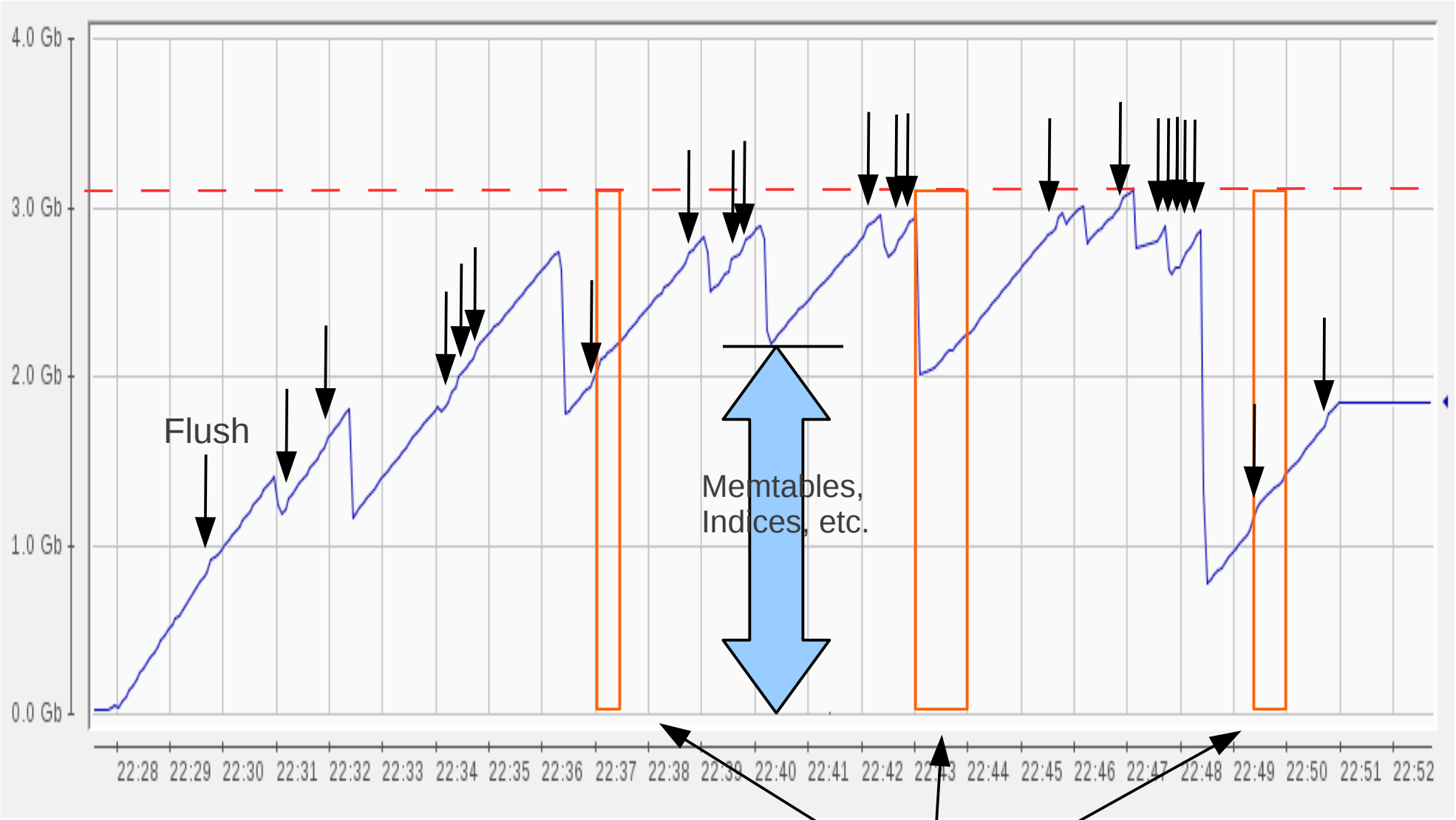
Cassandra Tuning

- Tuningmöglichkeiten:
  - Größe der Memtables, Thresholds für Flushes
  - Größe des JVM Heaps
  - Häufigkeit, Tiefe der Compaction
- Wo?
  - MemTableThresholds etc. in conf/cassandra.yaml
  - JVM Parameter in conf/cassandra-env.sh

Übersicht über die GC der JVM



Cassandras Speicherverhalten



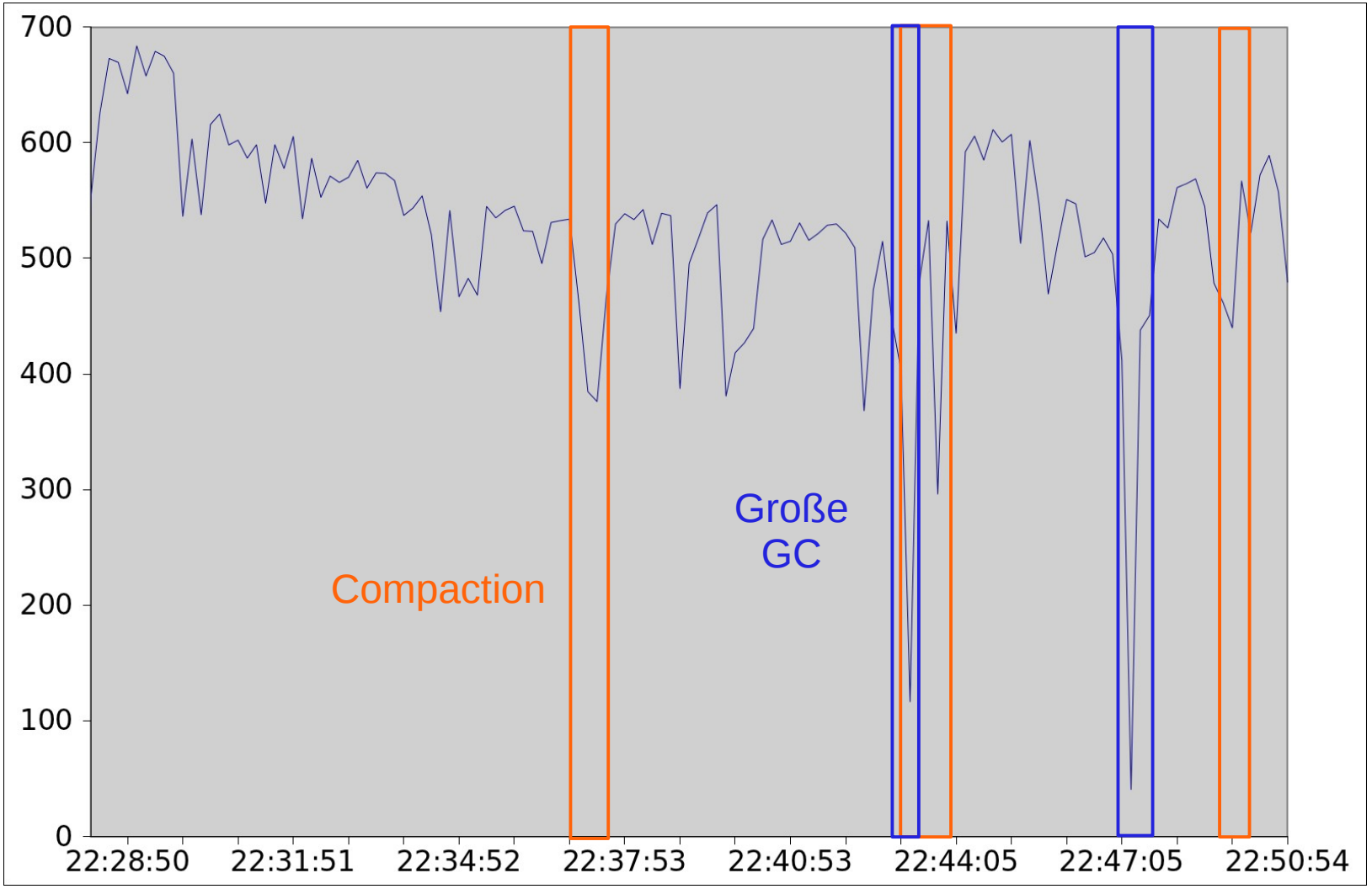
Memtablegröße: 128M, JVM Heap: 3G, #CF: 12

Compaction

Cassandras Speicherverhalten

- Memtables überleben extrem lange (mehrere Stunden)
  - Landen in der Old Generation
  - Bei GC müssen mehrere Dutzend GB aufgeräumt werden.
  - Heap zu klein, oder GC zu spät ausgelöst
    ⇒ “GC storm”
- Tradeoff:
  - I/O Last vs. Speicherbrauch
- Compaction nicht vernachlässigen!

Die Auswirkungen von GC und Compaction



Cluster und Einzelknoten

- Im Vergleich:
  - 1 Cluster mit six-core CPU und RAID 5 aus 6 Festplatten
  - 4 Cluster mit six-core CPU mit RAID 0 aus 2 Festplatten
- Einzelcluster lieferte konsistent 1,5-3 Mal mehr Performance.
- Gründe:
  - Overhead durch Netzwerkkommunikation/Konsistenzstufen, etc.
  - Festplattenperformance ausschlaggebend.
  - Cluster noch zu klein
- Effektiv verfügbarer Plattenplatz:
  - 1 Cluster: 6 * 500 GB = 3TB bei RAID 5 = 2.5 TB (83%)
  - 4 Cluster: 4 * 1TB = 4TB bei Replikationsfaktor 2 = 2TB (50%)

Alternativen

- MongoDB, CouchDB, redis, sogar memcached... .
- Persistenz: Festplatte oder RAM?
- Replikation: Master/Slave oder Peer-to-Peer?
- Sharding?
- Trend zu komplexeren Querysprachen (Javascript), Map-Reduce Operationen, etc.

Fazit: Cassandra

- Gut skalierende Plattform
- Aktive Benutzer- und Entwicklercommunity
- Leseoperationen relativ teuer
- Für optimale Leistung detailliertes Tuning nötig
- Je nach Anwendung “eventually consistent” und fehlende Transaktionen/Locking problematisch.

Links

- Apache Cassandra http://cassandra.apache.org
- Apache Cassandra Wiki http://wiki.apache.org/cassandra/FrontPage
- DataStax Dokumentation für Cassandra http://www.datastax.com/docs/0.7/index
- Mein Blog: http://blog.mikiobraun.de
- Twimpact: http://beta.twimpact.com