

Monologe – Vortag 2

Nach SQL – Aktuelle Datenbanken

Einreihung

1. Webapplikations-Frameworks:
diesseits und jenseits von Python, Ruby, Java
Scala, Erlang, Lua und C++
2. **Post SQL – Aktuelle Datenbanken**
3. Webserver der nächsten Generation
4. Effiziente und Parallele Programmierung
5. Atomar und Lock-Frei
6. Datenstrukturen, Arbeiten auf komprimierten Daten
7. Projekt „Orkan“ – Framework für Verteilte
Datenbanken und verteilte Systeme

Zyklus

- Projekt „Orkan“
- Framework für Verteilte Datenbanken
 - BigTable, DistributedHash, HBase etc.
- und Verteilte Systeme
 - Funktion wird ‚hier‘ aufgerufen, ‚dort‘ ausgeführt
 - Objekte transparent zwischen Rechnern verschieben
 - geht über RPC hinaus! (Tipp: [Ssrc Web Wispers](#))
- Fokus: Performance und Komfort
- Einsatzfeld: Webapplikationen
 - anderes zwar möglich
 - API und Helfer-Programme aber hierfür ausgelegt

Datenbanken

1. RDBMS
2. Speicherarten
3. Einsatzzweck (beispielhaft)
4. Performance, Metriken
5. Einsatzbeispiele
6. Daten Verteilen
 - Kompression, Hashes, Konsistentes Hashing
 - Universal-Schicht zum Verteilen
 - Performance im Vergleich

1. RDBMS

- MySQL
- Oracle
- MonetDB

- bilden Relationale Algebra nach Codd ab
 - in versch. Speichern
- Abfragesprache, Mächtigkeit
- Programmiersprache (Turing komplett)

2. Speicherarten

- „row-oriented“ - zeilenbasiert
- „column-oriented“ - spaltenbasiert
- Correlation Databases -
- Key/Value Store (aka HashMap, Schlüssel/Wert Speicher)
- „document oriented“ (nicht anerkannt)
- „object oriented“ - objektorientiert
- index only – nur-index Speicher (etwa Array)

Zeilenbasierte Speicher

- klassischer RDBMS Speicher
- MySQL, Oracle, PostgreSQL...
- Daten werden intern per Zeilennummer (vertreten durch Primary ID, PID) adressiert und in sg. „Seiten“ organisiert
- Indices bilden von beliebigen Spalten auf PID ab

PID	Subjekt	Prädikat	Objekt
1	Alex	trinkt	Kaffee
2	Mathias	isst	Kuchen

Zeilenbasierte Speicher

- schneller Zugriff auf Zeile per Primary ID
- redundante Daten, wg. zusätzlicher Indices
- langsam bei vielen Spalten
- Speicherverschwendung durch leere Felder (NULL)
- Zeile schreiben effizient (einfache Serialisierung)
- schlecht bei xIMD Szenarien
 - SIMD moderner Prozessoren nicht ausnutzbar
 - nächster Datenzugriff schwer vorherzusehen (wann fängt Feld X in der nächsten Spalte an?)
 - Caches können nicht effizient befüllt werden (Zeilen können lang werden! BLOB oder TEXT Felder)

Spaltenbasierte Speicher

- alternativer Ansatz; Argument: Codd hat ein Konzept entworfen, keine Implementierung
- MonetDB, Infobright, BigTable (, HBase), SybaseIQ
- Daten werden in Tupeln zu zwei Elementen gespeichert
- eine mehrspaltige Tabelle wird mit internen IDs versehen
- Indices oft erlässlich
 - Inhalt jeder Spalte kann direkt über Position adressiert werden
 - Tupel wird „umgekehrt“, wenn nach bestimmten Wert gesucht wird. Tatsächlich wird grundsätzlich „Zeilenposition(Zeile) + Indexbreite“ adressiert → schnell!

Spaltenbasierte Speicher

OID	Subjekt
1	Alex
2	Mathias

OID	Prädikat
1	trinkt
2	isst

OID	Subjekt
1	Kaffee
2	Kuchen

- Spaltensatz heißt „Binary Association Table“ (BAT) in MonetDB
- Würde Mathias auch einen Kaffee trinken, würde jede BAT einen Eintrag mehr bekommen
- Es folgt eine kleine Übung...

Spaltenbasierte Speicher

- Foreign Keys
 - möglich über Hilfstabelle
 - muss aber so nicht modelliert werden:
 - Normalisierung nicht notwendig!
 - Kann eigene Spalte werden
- eine BAT kann wiederum eine BAT enthalten
 - eine BAT kann eine beliebige Datenstruktur enthalten
- JOINS wesentlich effizienter (Designprinzip)
- Leere Spalten nehmen keinen Platz weg – OID in BAT einfach nicht vorhanden
- Parallele Ausführung und Vektorisierung einfach
- SIMD ebenfalls einfach möglich
- effizienteres Read-Ahead – Vorauslesen der Daten

MonetDB

- MonetDB trennt SQL, XQuery, SparQL etc. von der Verwaltung der BATs
- Erweitern ist einfacher als MySQL – einfach entsprechendes Modul oder Plugin schreiben (Hashfunktion als Zehnzeiler auf DVD; zum Vergleich ebenfalls ein Patch für MySQL)
 - MySQL: Plugins nur bei Funktionen möglich, unter Linux auch für Speicher. Ansonsten: Patchen des Quellcodes.

Correlation Databases

- Wie Spaltenorientierte
- Fassen gleiche Feldwerte zusammen
- Serialisieren in einer einzigen BAT (optional!)
- Implementierungen haben einen BAT-Index
- „Illuminate DBMS“

Arbeitsspeicher-Verwaltung

- kann das Betriebssystem schon sehr gut selbst
- DBMS übernehmen diese Aufgabe redundant
 - chaotisches Verhalten möglich, aber selten
 - ineffizient
- Unix: Memory Mapped Files, MMIO, mmap
- MonetDB nutzt diese Techniken

Schlüssel/Wert Speicher

- „Key/Value Stores“ (K/V-S)
- meist nur einzelne BAT
- sehr schnelle adressierung
- neben dem Schlüssel (zur Kollisionsvermeidung) wird sein Hash gespeichert, dieser wird zu Adressierung benutzt
 - oft werden unnötigerweise langsamere kryptographische Hashes benutzt
 - „Schließen von einem Ausgangsbit zu einem Eingangsbit“: muss hier nicht verhindert werden

K/V Vertreter

Verteter	Expiry	bel. Datentyp	MMIO	persistent	Library
Memcache	X	-	-	-	-
Tokyo Cabinet	-	X	X	X	X
Redis	X	X++	-	0	-

- Expiry ist nützlich bei Caches
- Redis unterstützt Listen, Sets. etc
 - entsprechend auch Operationen hierauf
- Persistenz
- Library – oder nur ein eigenes Zugriffsprotokoll

Objekt- und Dokumentorientierte

- etwa MongoDB
 - viel Schall, wenig zu sehen
- Speichern Datentypen direkt, etwa JSON
 - bilden darauf einen Index ab (kann zeilen- oder spaltenbasiert sein)
- möglicherweise bei „unstrukturierten“ Daten gut
- finden derzeit wenig Beachtung, lassen sich leicht nachbilden

3. Einsatzzweck

- Data Warehouse
 - support, confidence – Bayes etc.
 - alle Schüler, >18 Jahre, Gymnasium, Niedersachsen
- Algebra, Analyse
 - Aggregate, Median, Schnitt...
- Cache
 - etwa einer anderen Datenbank
- Datenablage
 - wohin mit den Spielständen? SQLite!

SQL oder nicht SQL?

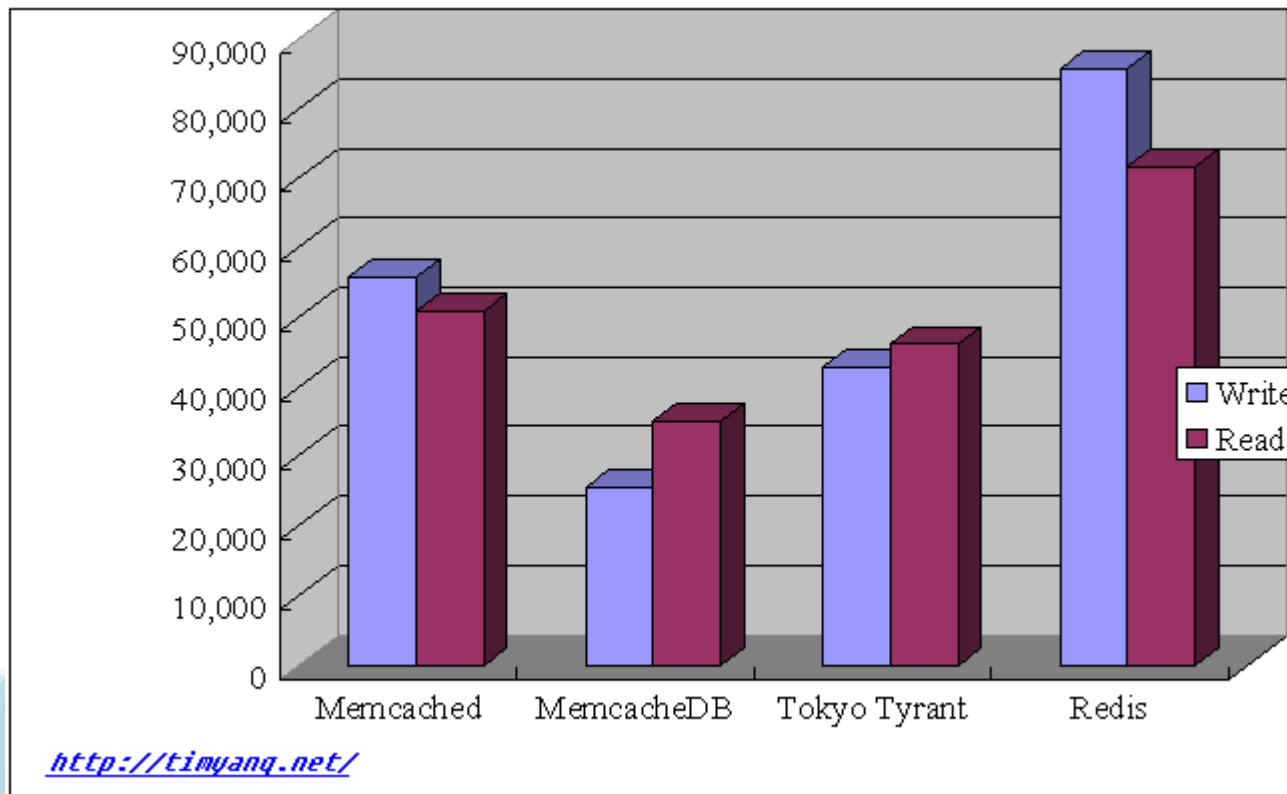
- Hängt vom Einsatzzweck ab.
- Oft entbehrlich, etwa in Blogs, Kontaktdatenbanken, Webseiten...
- SQL Funktionalität wird tatsächlich in oft endlich vielen Abfragen benutzt.
- Häufig unnötigerweise über eine Abstraktionsschicht (JPA, Hibernate...) – bad!
- Mächtiger Syntax etwa in Python vorhanden
 - SQLAlchemy abstrahiert so gut, dass es oft SQL ganz ersetzt

4. Performance

- Metriken entsprechend Einsatzzwecken!
→ Kapitel (3)
- Oracle: „Veröffentlichen von Benchmarks nicht erlaubt.“
 - „DBMS-X slightly slower than MySQL“
 - Bonck PhD Thesis
- Infobright: Spaltenorientierter Speicher für MySQL, kann aber nur SELECT, kein INSERT, DELETE etc.

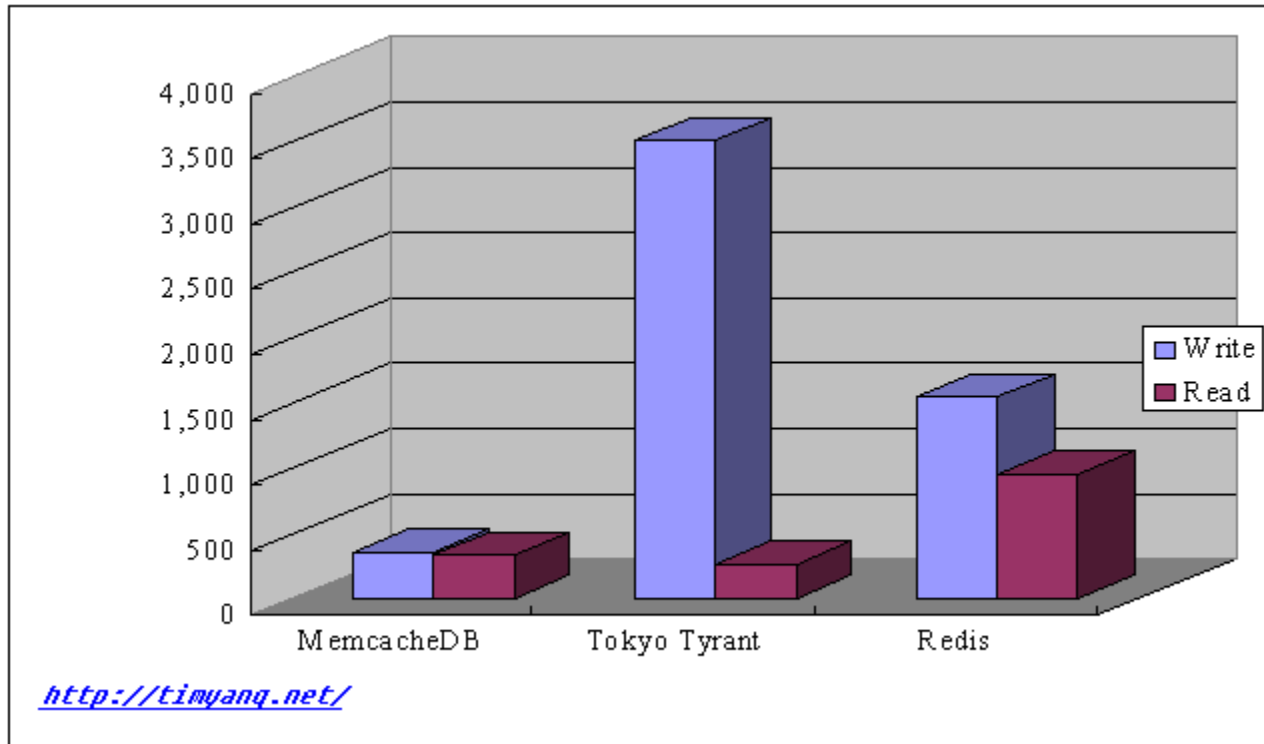
K/V Storages

RPS; 100b Data; Key := Int [1-5.000.000]



K/V Storages

RPS; 20k Data; Key := Int [1-500.000]



RDF Storage in „Orkan“

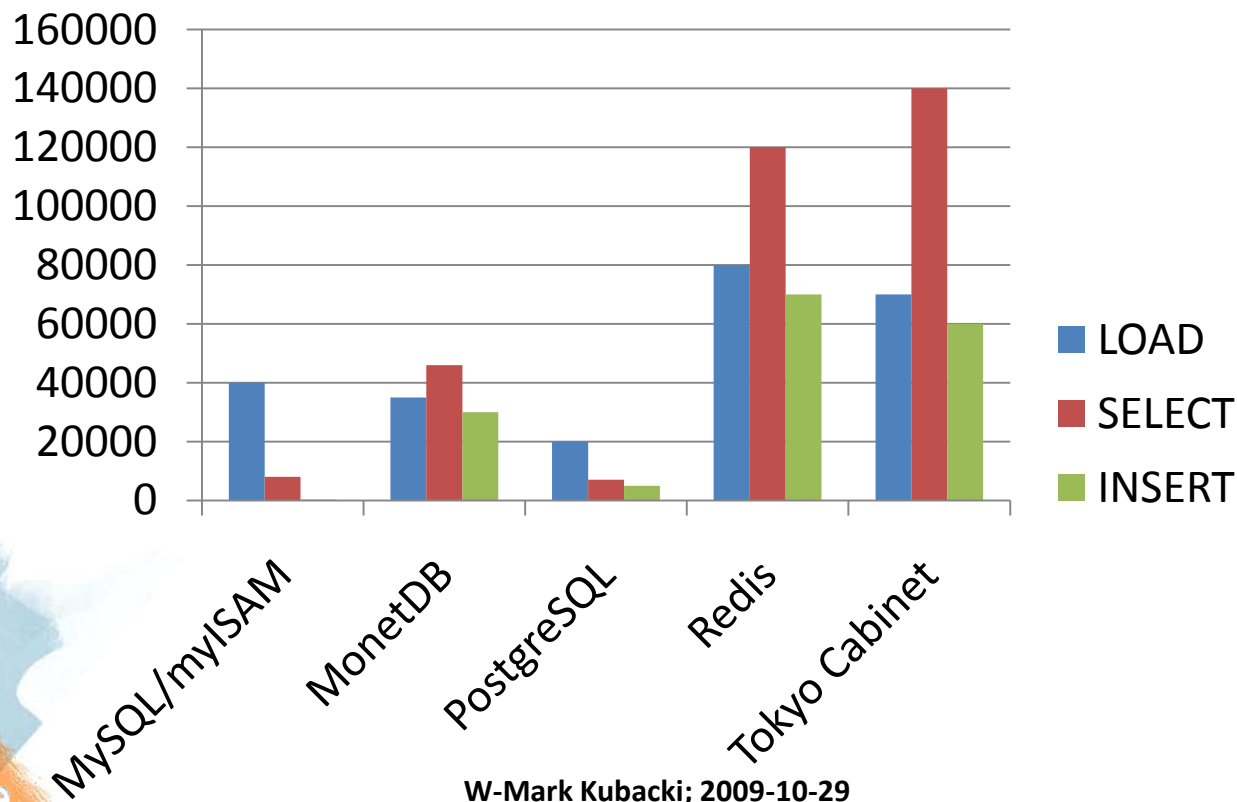
- Eigener Test, RDF Daten der DBPedia (3.2)
- Drei-Elementige Tupel,
 - HASH(Subjekt:Prädikat) → Objekt
 - HASH(Objekt:Prädikat) → Subjekt
 - ...
 - Hash ist FNV1A_64 (64bit)
 - Wert ein SET (besondere Form einer Liste)

RDF Storage in „Orkan“

- Drei Arten von Datenabfragen:
- unmittelbare: (Subjekt:Prädikat)
 - kann direkt über K/V Storage beantwortet werden
- mittelbare: Subjekt(Object) mit Prädikat
 - Rohdaten werden z.B. an Applikation gesendet, die Filtert weiter
 - mehrere unmittelbare Anfragen (ggf. EXIST)
- komplexe: Subjekt:Prädikat mit Eigenschaft(Object)
 - etwa Umkreissuche
 - ohne RDBMS: Wie mittelbare Abfrage
 - mit RDBMS: Daten werden dynamisch dorthin gesandt und ausgewertet
- Zum letzteren gibt es eine elegantere Lösung!

RDF Storage für „Orkan“

- unmittelbare Anfrage – Key jetzt Hashes!
- Ops/s, RDF Daten



5. Einsatzbeispiele

- Cache:
 - ganze Seiten können in K/V Storage als Wert gespeichert werden, Schlüssel ist dann die URI
 - Nginx kann direkt aus K/V Storage lesen
 - nur noch bei Veränderungen muss Python, Java o.Ä. aufgerufen werden
 - Zeilen oder Seiten (Tabellen~) können zwischengespeichert werden, entlastet MySQL und Co.
 - braucht Expiry oder Invalidation-Policy
- Dekoratoren für TurboGears trivial,
- Wrapper für Ruby vorhanden

Einsatzbeispiele

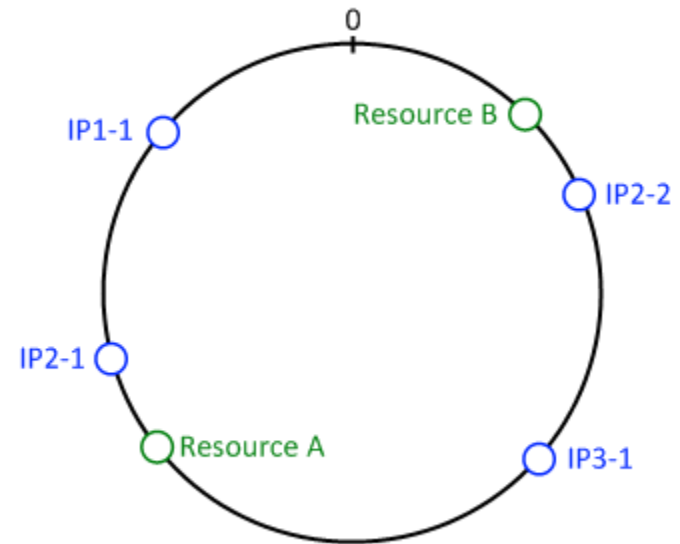
- Twitter: Konversationen
- Emails: Nachrichten-Queues
- Suchmaschinen: Verteilte BAT (kommt gleich)

6. Daten Verteilen

- Wo ist das Datum?
- Zordnung Datum → Knoten
 - explizit (Sharding)
 - transparent (Gruppenkommunikation: Broadcast)
 - implizit (transparent)
- Ein Knoten kann (permanent) ausfallen – was dann?

Daten Verteilen

- Sicherheit braucht Redundanz
- Entweder zwei von:
 - Sicherheit
 - Performance
 - Sparsamkeit
- Konsistentes Hashing
- Alle Knoten a priori bekannt?
- Failover and Restore



Failover and Restore

- Braucht Redundanz:
 - GET x: X und X+1 haben Datum
 - schnellster Antwortet
 - beide Antworten
 - X fällt aus: X+1 antwortet
- Neues Datum während Ausfall:
 - X+1 führt Logdatei
 - X-1 ersetzt derweil X (etwa permanent)
 - X wieder da:
 - erhält Daten aus Logdatei von X+1, dann kann X-1 löschen
 - oder implizit: X-1, X und X+1 haben neue Hashes,
 - X lauscht auf Antworten und erhält Daten nach und nach von X+1 (Löschen im Falle von NONE – Logdatei!)
- Geschickter: mehrere Knoten haben den gleichen Hash, bilden Untergruppe
- Es gibt einen Overhead, den man aber in Parallelität verstecken kann.
- UNIQUE Constraint nicht nötig: PID enthält immer das gleiche Datum.

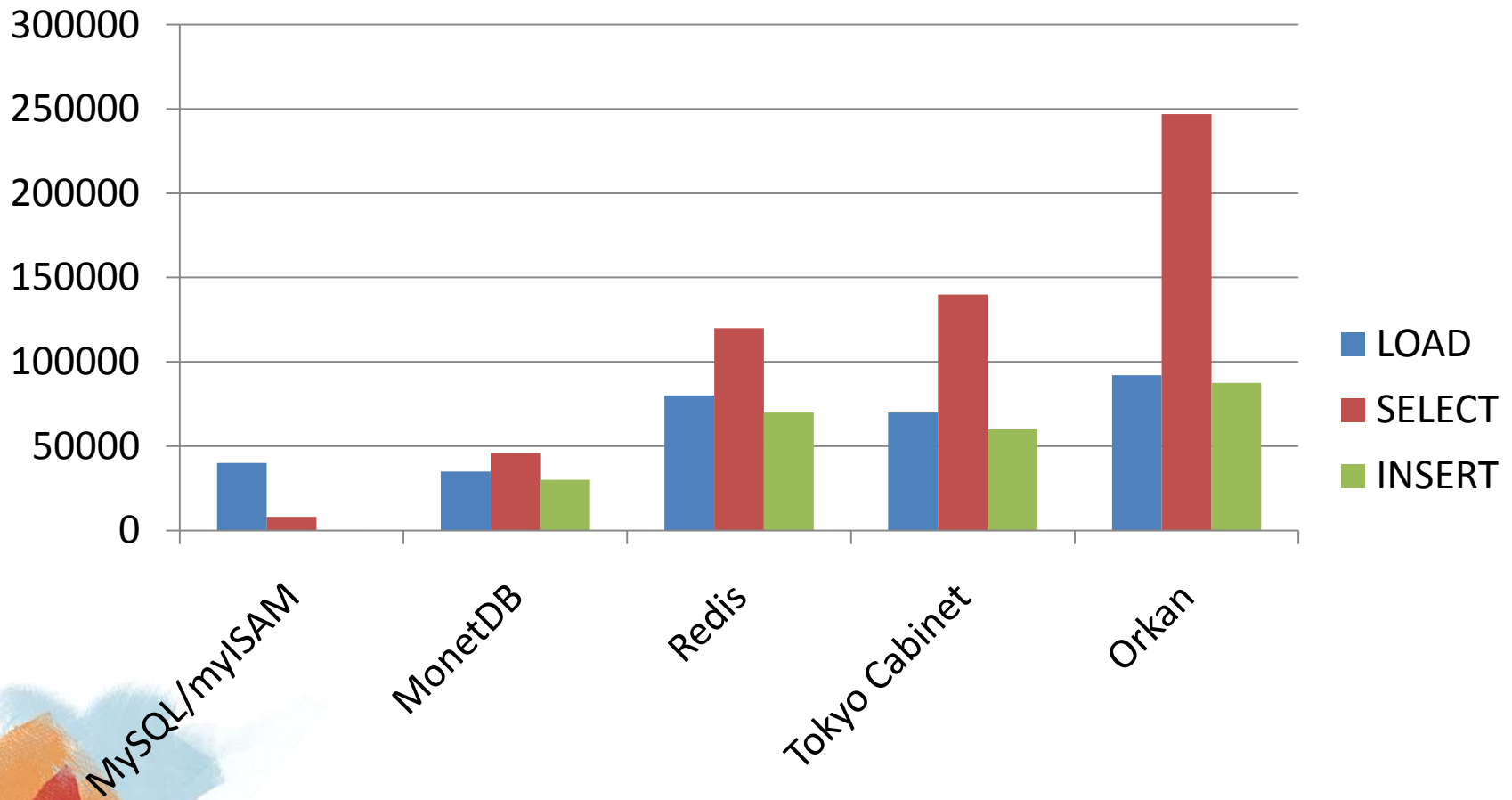
Komplexe Anfragen

- Drei Arten der Anfragen:
- mittelbar: GET X – K/V antwortet direkt
- unmittelbar: mehrere mittelbare werden in Applikation kombiniert
- komplex: MonetDB als Zwischenspeicher, dann Algebra und veränderliches Ergebnisset.
 - ähnlich „Promises“ von LISP

Performance - Orkan

- 4 Nodes, hier aber auf eine normalisiert
- IO/second
- Datenbestand wird beherrschbar
(DBPedia ganz in den jeweiligen Hauptseichern)
- Schlüssel sind Hashes, deshalb langsames Schreiben wg. „Random IO“
- mein Rechner (Phenom 9550, 8 GB DDR2 RAM, Adaptec 3805 @4x WD6400AAKS)

Performance - Orkan



Fazit

- Durch geschickte Verteilung lässt sich ein großer Datenbestand noch effizienter händeln
 - Skalierung (seriellen Anteil verstecken)
- Keine Krypto-Hashes bitte!
- Gruppenkommunikation!
 - mehrere Ringe! Torus
- Spaltenorientierte DBMS

Was ist auf der DVD?

- Virtuelle Maschine mit Gentoo 10.1
 - braucht einen Prozessor mit min SSE4a
 - min. 1 GiB zugewiesenen Arbeitsspeicher
- Prototypen und Quellcode
- Intel ICC C/C++ Compiler (mit Patch f. AMD)
- Auszug aus dem DBPedia Datenbestand zum Nachvollziehen der Benchmarks

Womit ist „Orkan“ programmiert?

- C/C++, [Boost](#); Python
- Pre-Parser für C++ Annotationen
(Quellcodes nutzen Boost Signals/Slots direkt)
- [Redland](#) Bibliothek für RDF
- [Spread Toolkit](#) für Gruppenkommunikation
- [libatomic](#) 1.2
- Threading, [Events](#), Parallele Programmierung