HandlerSocket und ähnliche Technologien - NoSQL für MySQL

DOAG SIG Development: DB-Programmierung mal anders
Kassel, 9. Juni 2011

Oli Sennhauser

Senior MySQL Consultant, FromDual GmbH

oli.sennhauser@fromdual.com

FromDual GmbH

- Wit bieten neutral und Hersteller unabhängig:
  - Beratung für MySQL
  - Remote-DBA / MySQL Betrieb
  - Support (ex. MySQL Basic und Silber)
  - Schulung für MySQL
- Wir sind Consulting Partner der Open Database Alliance (ODBA.org)

- Oracle Silver Partner (OPN)

Inhalt

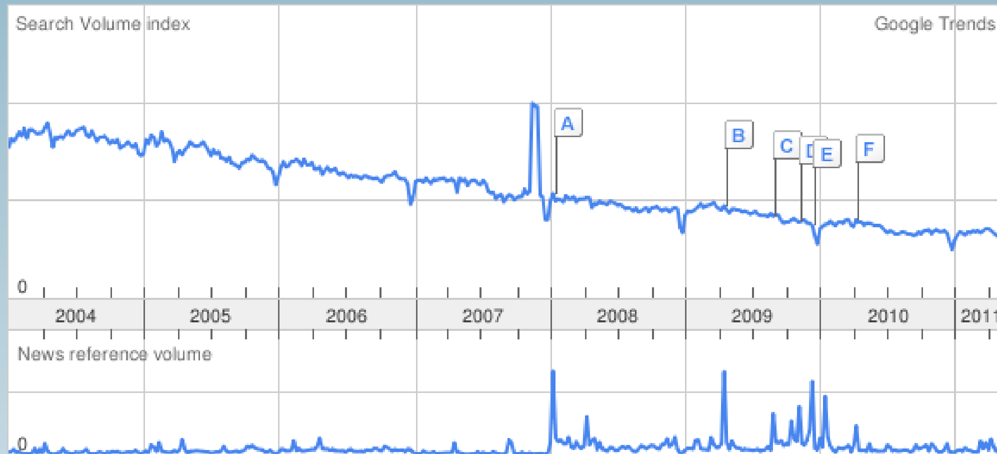
NoSQL für MySQL

- NoSQL Trends
- SQL Overhead
- HandlerSocket
- NDB-API
- BLOB Streaming Engine
- Handler Interface
- Graph Storage Engine

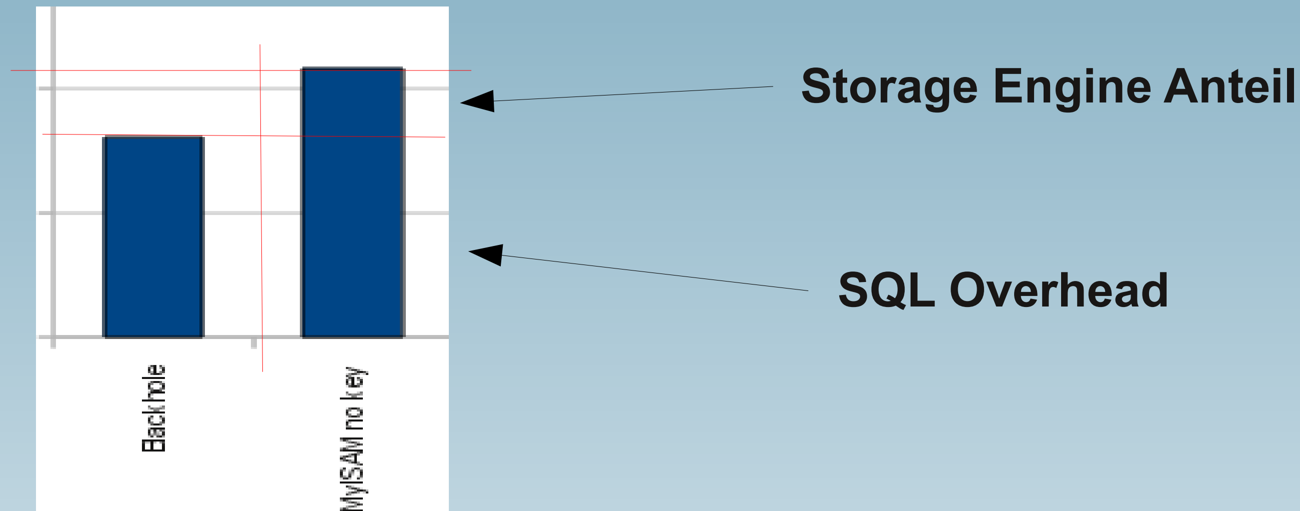
Trends

MySQL

NoSQL



Wo liegt das Problem?



- SQL Overhead ist 70 – 80% für einfache Queries (~ 1 ms)!
- mit NO SQL → könnten wir bis zu 5 x schneller werden
- SQL ist gemacht für komplexe Abfragen
- NoSQL löst typischerweise einfache Abfragen

Woher rührt der Overhead?

Application / Client

mysqld

Thread Cache

Connection Manager

Parser

Optimizer

User Au-thentication

Access Control

Logging

Command Dispatcher

Table Open Cache (.frm, fh)

Table Manager

Query Cache

Query Cache Module

Table Definition Cache (tbl def.)

Handler Interface

MyISAM	InnoDB	Memory	NDB	PBMS	Aria	XtraDB	Federated-X	...

Was können wir dagegen tun?

- HandlerSocket (2010)
- NDB-API (1997!)
- PrimeBase Streaming Engine (2008)
- Handler Interface (2001/2011)
- Memcached (ab 2006, 2011)
- OQGRAPH SE (2009)

HandlerSocket

- **20. Oktober 2010, Yoshinori Matsunobu:**

  Using MySQL as a NoSQL - A story for exceeding 750,000 qps on a commodity server

Application / Client

mysqld
Connection Manager
User Authentication
Command Dispatcher
Query Cache Module
Parser
Optimizer
Access Control
Table Manager

HandlerSocket Plugin
Connection Manager
Worker Threads

SE-API

Handler Interface

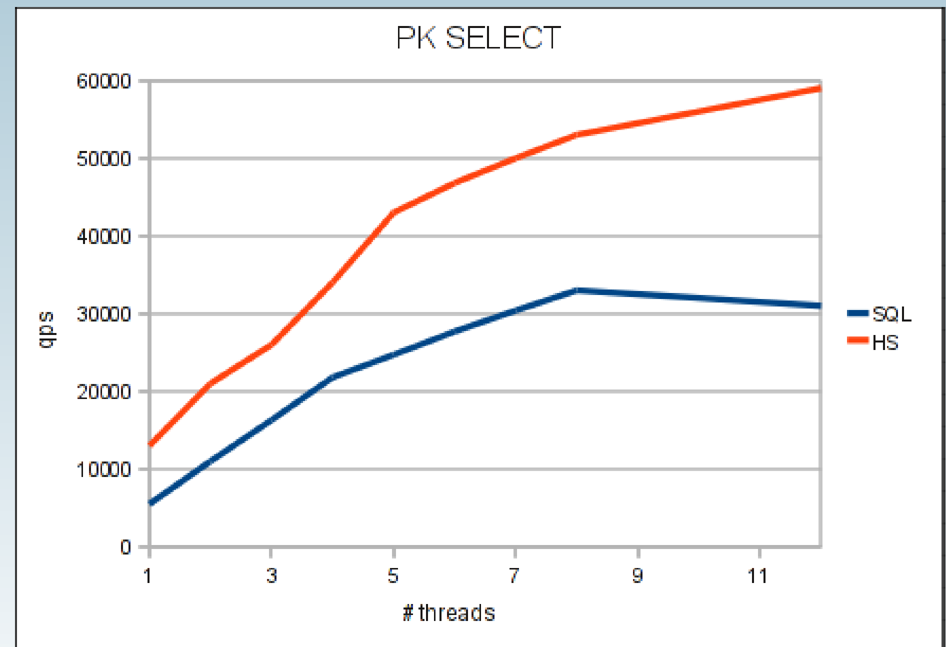
MyISAM | InnoDB | Memory | NDB | PBMS | Aria | XtraDB | Federated-X | ...

SELECT

```
# SELECT * FROM test.test where id = 42;

use Net::HandlerSocket;

my $args = { host => 'master', port => 9998 };
my $hs = new Net::HandlerSocket($args);

my $res = $hs->open_index(0, 'test', 'test', 'PRIMARY', 'id,data,ts');

$res = $hs->execute_single(0, '=', [ '42' ], 1, 0);
shift(@$res);
for (my $row = 0; $row < 1; ++$row) {
  print "$id\t$data\t$ts\n";
}

$hs->close();
```

Infos

- Selber compilieren (einfach!)
- 7.5 mal mehr Durchsatz?!?
- Funktioniert mit MySQL 5.5.8 und MariaDB
- Schneller als mem-cached!?!
- Im Percona-Server 12.3



Features / Funktionalität

- Ermöglicht ganz andere Abfragemuster (siehe auch Handler Interface)
- Viele concurrent Connections
- Hochperformant (200 – 700%)
- Keinen doppelten Cache (vs. MySQL und memcached)
- Keine Dateninkonsistenz (vs. MySQL und memcached)
- Crash-safe
- SQL Zugriff ebenfalls möglich (z. B. für komplexe Reportingabfragen)
- MySQL muss nicht modifiziert/neu gebaut werden?!?
- Funktioniert theoretisch auch für andere Storage Engines (ich hab's nicht ausprobiert).

NDB-API

- **1997, Mikael Ronström:**

  The NDB Cluster – A parallel data server for telecommunications applications

- **25. November 2008, Jonas Oreland:**

  950'000 reads per second on 1 datanode

MySQL Cluster

Application
NDB-API
Application
NDB-API
Application
Application
Application
Load balancer
SQL Node 1
SQL Node 2
SQL Node 3
...
Mgm Node 1
Mgm Node 2
Data Node 1
Data Node 2
Sw.
Data Node 3
Data Node 4

INSERT

```
// INSERT INTO cars VALUES (reg_no, brand, color);

const NdbDictionary::Dictionary* myDict= myNdb->getDictionary();
const NdbDictionary::Table *myTable= myDict->getTable("cars");

NdbTransaction* myTrans = myNdb->startTransaction();
NdbOperation* myNdbOperation = myTrans->getNdbOperation(myTable);

myNdbOperation->insertTuple();
myNdbOperation->equal("REG_NO", reg_no);
myNdbOperation->setValue("BRAND", brand);
myNdbOperation->setValue("COLOR", color);

int check = myTrans->execute(NdbTransaction::Commit);

myTrans->close();
```

SELECT

```
// SELECT * FROM cars;

const NdbDictionary::Dictionary* myDict= myNdb->getDictionary();
const NdbDictionary::Table *myTable= myDict->getTable("cars");

myTrans = myNdb->startTransaction();
myScanOp = myTrans->getNdbScanOperation(myTable);
myScanOp->readTuples(NdbOperation::LM_CommittedRead)

myRecAttr[0] = myScanOp->getValue("REG_NO");
myRecAttr[1] = myScanOp->getValue("BRAND");
myRecAttr[2] = myScanOp->getValue("COLOR");

myTrans->execute(NdbTransaction::NoCommit);

while ((check = myScanOp->nextResult(true)) == 0) {

  std::cout << myRecAttr[0]->u_32_value() << "\t";
  std::cout << myRecAttr[1]->aRef() << "\t";
  std::cout << myRecAttr[2]->aRef() << std::endl;
}
myNdb->closeTransaction(myTrans);
```

Benchmarks und Zahlen

- `./flexAsynch -ndbrecord -temp -con 4 -t 16 -p 312 -l 3 -a 2 -r 2`
- Aus der MySQL Cluster Test-Suite (`src/storage/ndb/test/ndbapi`)

- 16 number of concurrent threads (-t)
- 312 number of parallel operation per thread
- 3 iterations (-l)
- 2 attributes per table (8 bytes) (-a)
- 4 concurrent connections (-con)
- 2 number of records (-r ?)
- 1 32-bit word per attribute
- 1 ndbmtd (1 NoOfRepl.)

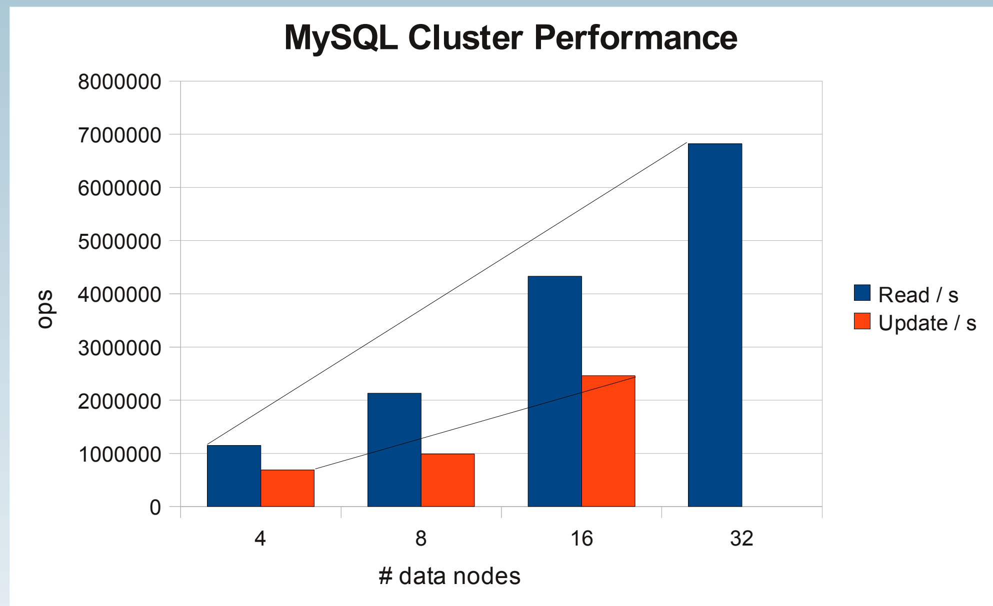
```
insert average: 506506/s min: 497508/s max: 522613/s stddev: 2%
update average: 503664/s min: 495533/s max: 507833/s stddev: 1%
delete average: 494225/s min: 474705/s max: 518272/s stddev: 3%
read   average: 980386/s min: 942242/s max: 1028006/s stddev: 2%
```

Lehren

- Beobachtungen:
  - CPU's sind nicht am Limit. "Igendwo" ist noch Potential !?!
  - Wenn übertrieben wird: CPU ist überlastet und Performance fällt auf 14%!
- Schummeleien:
  - Schau die Parameter genau an!
  - Mit allen anderen Konfigurationen erhielt ich schlechteren Duchrsatz!
  - Sobald eine IP anstatt localhost verwendet wird: 89% Durchsatz
- Lehren:
  - Traue keinen Benchmarks, die Du nicht selber manipuliert hast!

Neuere Resultate

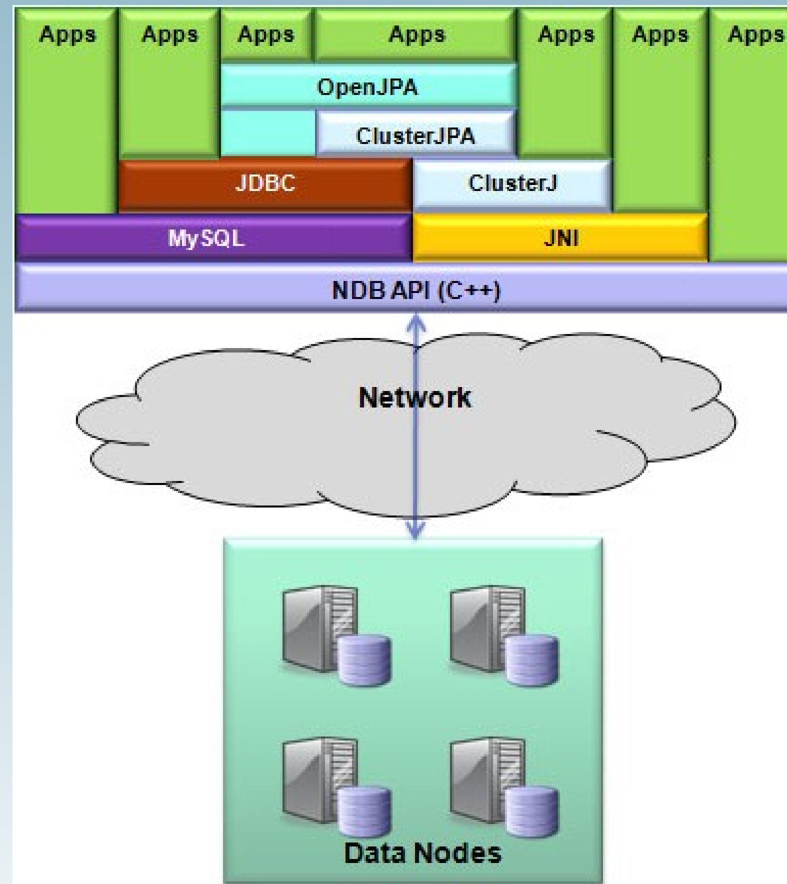
- **Michael Ronström, April/Mai 2011:**
  - **6.82M reads per second**
  - **2.46M updates per second**



- **Mehr als lineares Skalieren scheint möglich (Caching-Effekte)!**

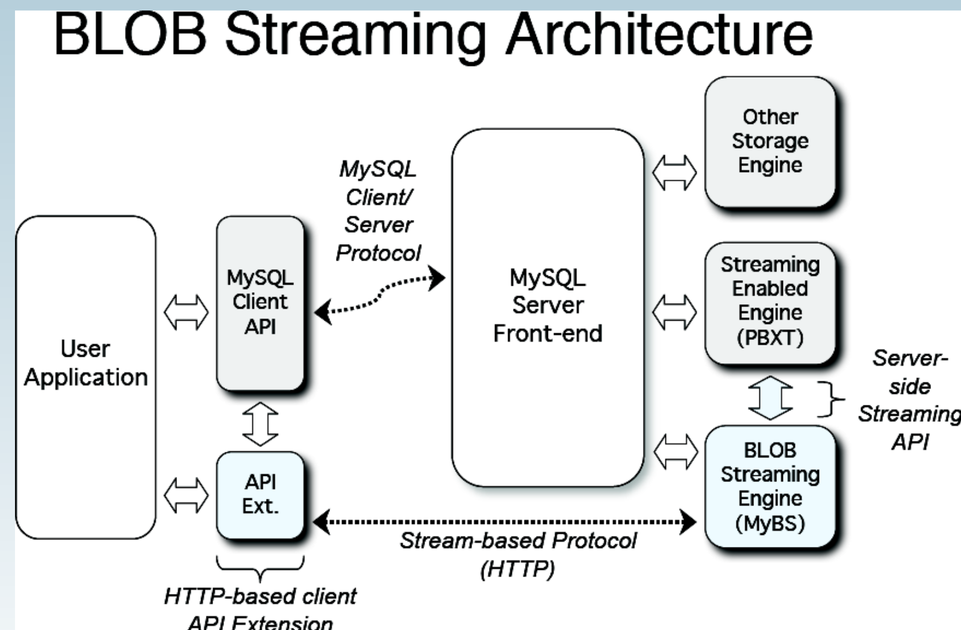
Und wenn wir uns das nicht antun wollen?

- **Ab MySQL Cluster 7.1 (7.0)**



BLOB Streaming Projekt

- April 2008, Paul McCullagh: *Introduction to the BLOB Streaming Project*
- 5. März 2010, Barry Leslie: *Upload 1000+ BLOB's per second!*



Vorteile von BLOB's in der Datenbank

- alt: RDBMS sind nicht schnell mit dem Speichern von BLOB's → BLOB's NICHT in der Datenbank speichern
- neu: Mit NoSQL Technologien wird es viel besser!
- Mit PBSE: atomare Transaktionen → Keine ins Nirgendwo zeigende Referenzen.
- BLOB's im normalen Datenbank-Backup !?!
- BLOB's können repliziert werden
- BLOB's in der DB skallieren besser. Die meisten Filesysteme performen schlecht mit mehr als 2 Millionen Dateien ?

Das Handler Interface

- **Oktober 2001, MySQL Handbuch:** ***A new HANDLER interface to MyISAM tables***
- **27. Dezember 2010, Stephane Varoqui:** ***Using MySQL as a NoSQL: a story for exceeding 450'000 qps with MariaDB***
- **10. Januar 2011, Stephane Varoqui:** ***20% to 50% improvement in MariaDB 5.3 Handler Interface using prepared statement***

Überspringen des Overheads mit dem Handler Interface

Application / Client

mysqld

Thread Cache

Connection Manager

User Au-thentication

Logging

Command Dispatcher

Query Cache

Query Cache Module

Parser

Optimizer

Access Control

Table Manager

Table Open Cache (.frm, fh)

Table Definition Cache (tbl def.)

Handler Interface

MyISAM	InnoDB	Memory	NDB	PBXT	Aria	XtraDB	Federated-X	...

HANDLER Beispiel

```
# MySQL
# SELECT * FROM family;

HANDLER family OPEN;
HANDLER family
   READ `PRIMARY` = (id)
  WHERE id = 1;
HANDLER family CLOSE;


# With MariaDB 5.3

HANDLER family OPEN;
PREPARE stmt
   FROM 'HANDLER family
            READ `PRIMARY` = (id)
           WHERE id = ?';
set @id=1;
EXECUTE stmt USING @id;
DEALLOCATE PREPARE stmt;
HANDLER family CLOSE;

Use persistent connections!!!
```

```
HANDLER tbl OPEN

HANDLER tbl READ idx (..., ..., …)
  WHERE ... LIMIT ...

HANDLER tbl READ idx FIRST
  WHERE ... LIMIT ...
HANDLER tbl READ idx NEXT
  WHERE ... LIMIT ...
HANDLER tbl READ idx PREV
  WHERE ... LIMIT ...
HANDLER tbl READ idx LAST
  WHERE ... LIMIT ...

HANDLER tbl READ FIRST
  WHERE ... LIMIT ...
HANDLER tbl READ NEXT
  WHERE ... LIMIT ...

HANDLER tbl CLOSE
```

Charakteristik des Handler Interfaces

- `HANDLER` ist schneller als `SELECT`:
  - Weniger Parsing
  - Kein Optimizer Overhead
  - Weniger Query-Checking Overhead
  - Die Tabelle muss zwischen zwei Handler-Anfragen nicht gelocket werden
- KEINE konsistente Sicht auf die Daten (dirty reads sind erlaubt)
- Optimierungen sind möglich, welche `SELECT` nicht zulässt
- Traversieren der Daten auf eine Art, welche schwierig bis unmöglich mit `SELECT` zu erlangen ist

Neu aus den MySQL Labs

- **Memcached Plugin für InnoDB (5.6)**

Application

SQL

Memcached protocol

mysqld

MySQL Server

memcached plugin

innodb_memcache

local cache (optional)

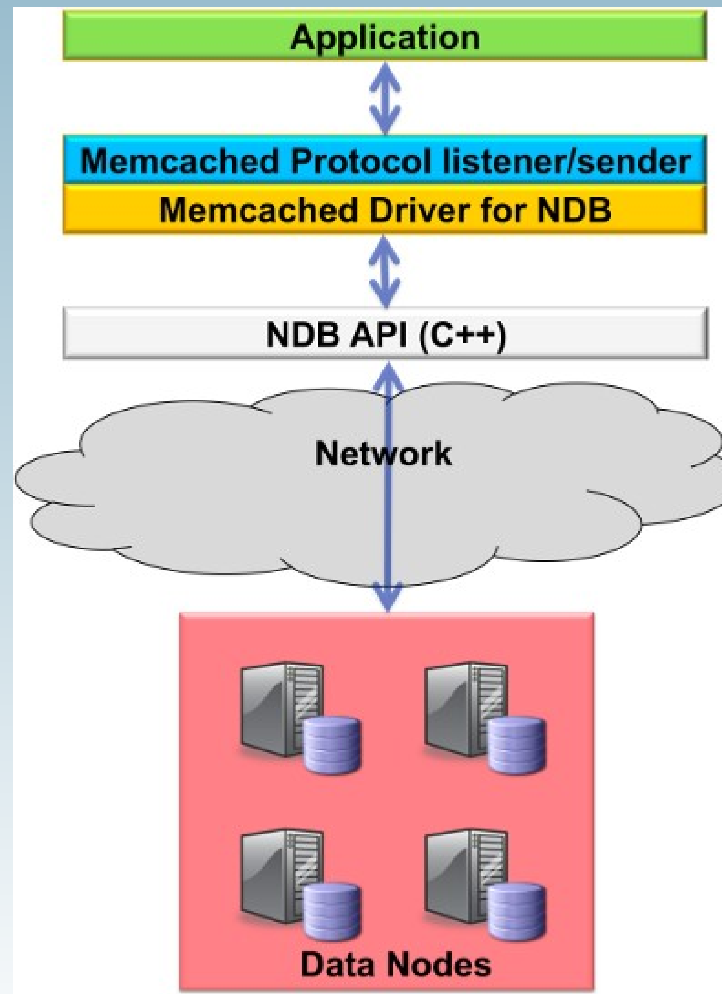
Handler API

InnoDB API

InnoDB Storage Engine

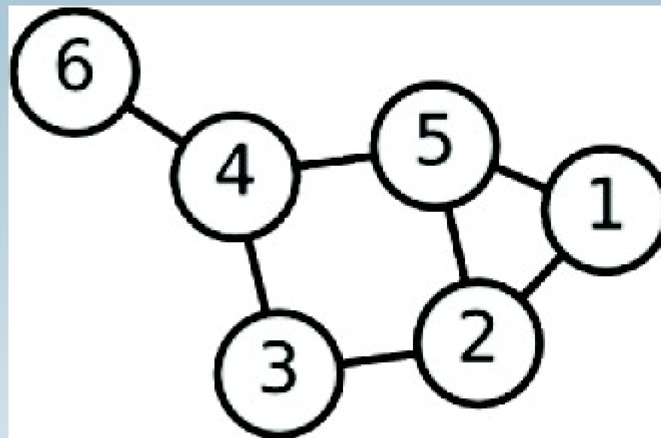
Neu aus den MySQL Labs

- **Memcached API für MySQL Cluster (7.2)**



Eine Graphen Storage Engine

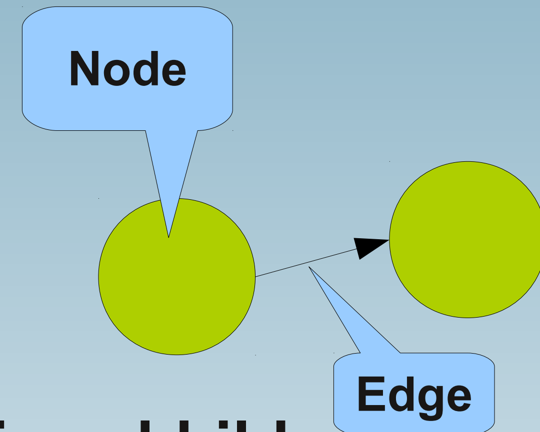
- **5. Mai 2009, Arjen Lentz: *OQGRAPH Computation Engine for MySQL, MariaDB & Drizzle***



- **In MariaDB 5.1 ff.**
- **Verfügbar für MySQL 5.0 ff.**

Wie fühlt sich das an?

- Es ist ähnlich wie die MEMORY SE (Persistenz, Locking, trx)
- Wir sprechen in:
  - Node/Item/Vertex und
  - Edge/Connection/Link
- Edges haben eine Richtung
- Wir können Netzwerke und Hierarchien abbilden
  - (Familienbeziehungen, Freunde von Freunden, kürzeste Strecke von A nach B)
- Um mit der OQGRAPH SE zu sprechen brauchen wir “latches” (welcher Algorithmus zu verwenden ist)
- Es ist eine “computational engine” nicht eine Storage Engine!



Einfaches Beispiel: Meine Familie

```
INSERT INTO family VALUES
  (1, 'Grand-grand-ma')
, (2, 'Grand-ma')
, (3, 'Grand-uncle')
, (4, 'Grand-aunt')
, (5, 'Grand-pa')
, (6, 'Mother')
, (7, 'Uncle 1')
, (8, 'Uncle 2')
, (9, 'Father')
, (10, 'Me')
, (11, 'Sister');

INSERT INTO relation (origid, destid)
VALUES
  (1, 2), (1, 3), (1, 4)
, (2, 6), (2, 7), (2, 8)
, (5, 6), (5, 7), (5, 8)
, (6, 10), (6, 11)
, (9, 10), (9, 11);
```

```
SELECT f1.name AS parent, f2.name AS child
  FROM relation AS r
  JOIN family f1 ON f1.id = r.origid
  JOIN family f2 ON f2.id = r.destid;

+----------------+-------------+
| parent         | child       |
+----------------+-------------+
| Grand-grand-ma | Grand-ma    |
| Grand-grand-ma | Grand-uncle |
| Grand-grand-ma | Grand-aunt  |
| Grand-ma       | Mother      |
| Grand-ma       | Uncle 1     |
| Grand-ma       | Uncle 2     |
| Grand-pa       | Mother      |
| Grand-pa       | Uncle 1     |
| Grand-pa       | Uncle 2     |
| Mother         | Me          |
| Mother         | Sister      |
| Father         | Me          |
| Father         | Sister      |
+----------------+-------------+
```

Netzwerk-Abfragen

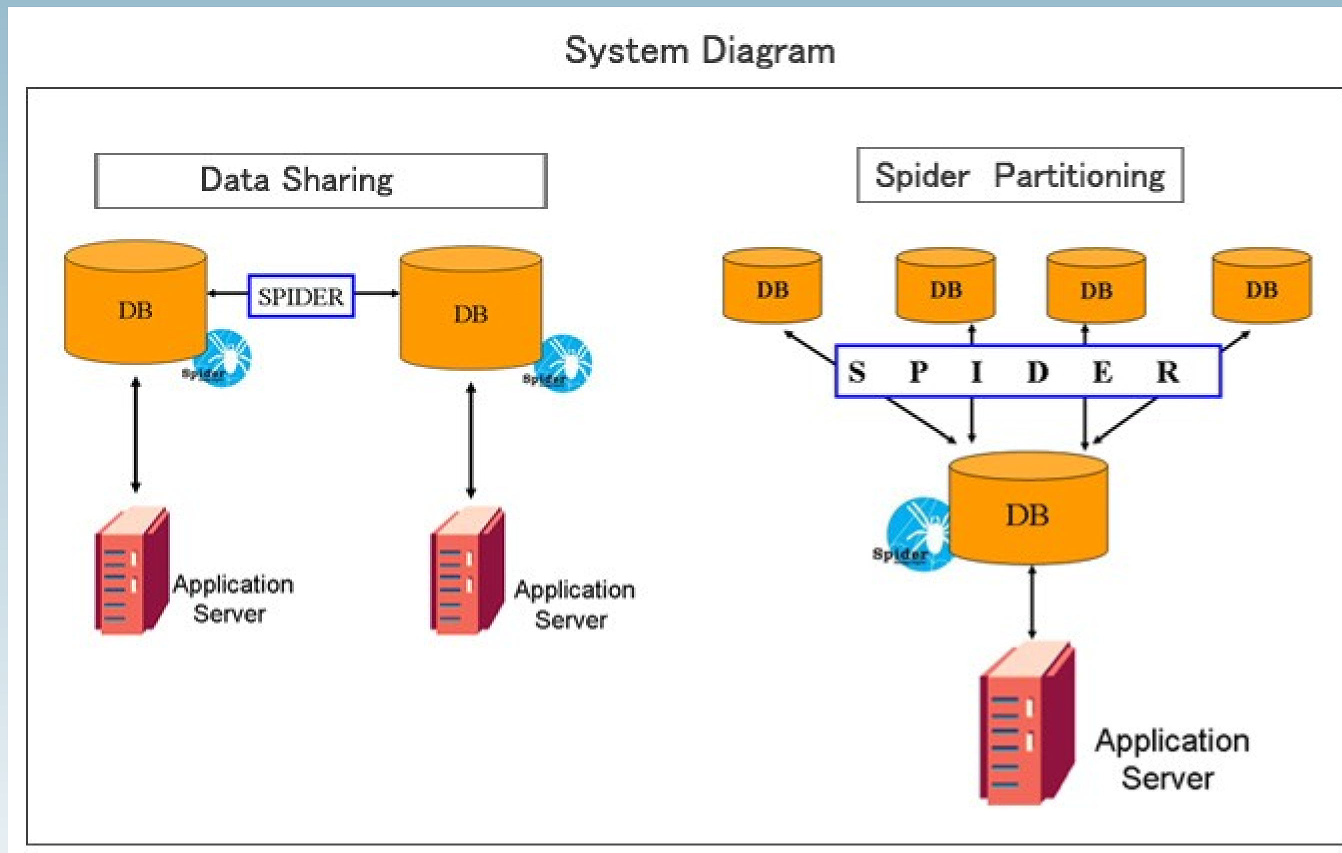
```
SELECT GROUP_CONCAT(f.name SEPARATOR ' -> ')
       AS path
  FROM relation AS r
  JOIN family AS f ON (r.linkid = f.id)
 WHERE latch = 1
   AND origid = 1
   AND destid = 10
ORDER BY seq;

+--------------------------------------------+
| path                                       |
+--------------------------------------------+
| Grand-grand-ma -> Grand-ma -> Mother -> Me |
+--------------------------------------------+
```

latch = 1: Find shortest path (Dijkstra)

```
SELECT r.weight, r.seq, f.name
  FROM relation AS r
  JOIN family AS f ON (r.linkid = f.id)
 WHERE r.latch = 2
   AND r.destid = 10;

+--------+------+----------------+
| weight | seq  | name           |
+--------+------+----------------+
|      3 |    6 | Grand-grand-ma |
|      2 |    5 | Grand-pa       |
|      2 |    4 | Grand-ma       |
|      1 |    3 | Father         |
|      1 |    2 | Mother         |
|      0 |    1 | Me             |
+--------+------+----------------+
```

latch = 2: Find originating nodes
(Breadth-first search)

Sharding mit der Spider-SE



Übersicht

Technologie	r / w	Trx	Sprachen
MySQL	ja / ja	ja	“alle”, SQL
HandlerSocket	ja / ja	nein	C++, Perl, Ruby, PHP, Java, Pyhton, ...
NDB-API (MySQL Cluster)	ja / ja	ja	C++, Java, (SQL)
PBSE	ja / ja	ja	C++, SQL
Handler Interface	ja / nein	nein	“alle”, SQL
OQGRAPH SE	ja / ja	nein	“alle”, SQL
Memcached-API	ja / ja	nein	C++, Perl, Ruby, PHP, Java, Pyhton, ...

Zusammenfassung

- SQL ist gut für komplexe Abfragen
- NoSQL üblicherweise für einfache Abfragen
- Vorsicht mit Performance-Zahlen!
- Architektur / Programmierung wird komplexer
- Bessere Performance ist möglich

- Aber es ist verdammt interessant!

Literatur

- Using MySQL as a NoSQL - A story for exceeding 750,000 qps on a commodity server: http://yoshinorimatsunobu.blogspot.com/2010/10/using-mysql-as-nosql-story-for.html
- 950k reads per second on 1 datanode: http://jonasoreland.blogspot.com/2008/11/950k-reads-per-second-on-1-datanode.html
- Scalable BLOB Streaming Infrastructure for MySQL and Drizzle: http://www.blobstreaming.org/
- HandlerSocket: Why did our version not take off? http://pbxt.blogspot.com/2010/12/handlersocket-why-did-out-version-did.html
- Using MySQL as a NoSQL: a story for exceeding 450000 qps with MariaDB: http://varokism.blogspot.com/2010/12/using-mysql-as-nosql-story-for_27.html
- HANDLER Syntax: http://dev.mysql.com/doc/refman/5.5/en/handler.html
- GRAPH Computation Engine – Documentation: http://openquery.com/graph/doc

Q & A

Fragen ?

Diskussion?

Wir haben noch Zeit für persönliche und indviduelle Beratungen...