Praktikum: Betriebsdatenerfassung regenerativer Energieanlagen

Martin.Junghanns (at) studserv.uni-leipzig.de

Gliederung

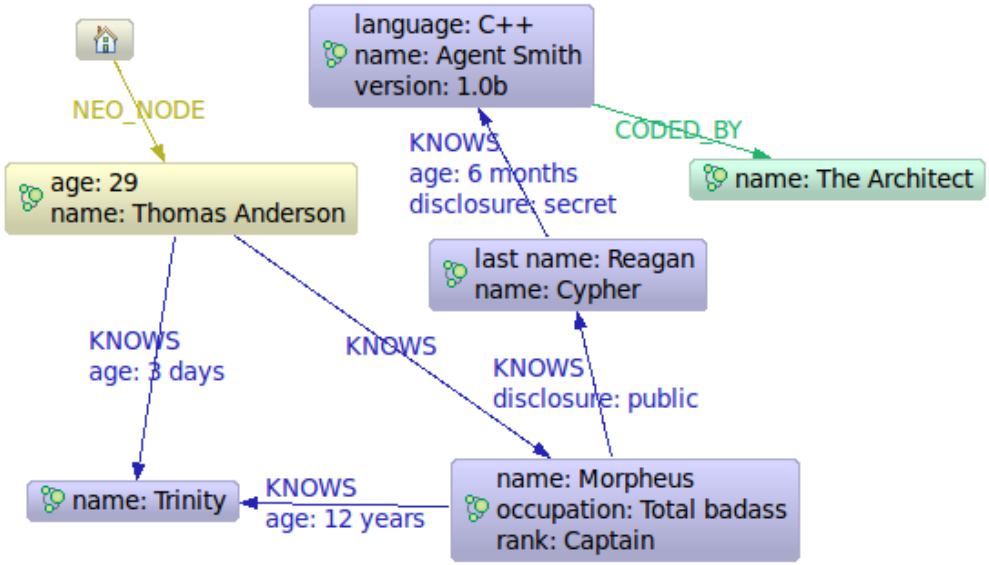
1. Graph- vs. Dokumentendatenbanken
2. MongoDB
3. Datenschema
4. Beispiel Abfragen
5. Ergebnisse
6. Optimierung
7. Fazit

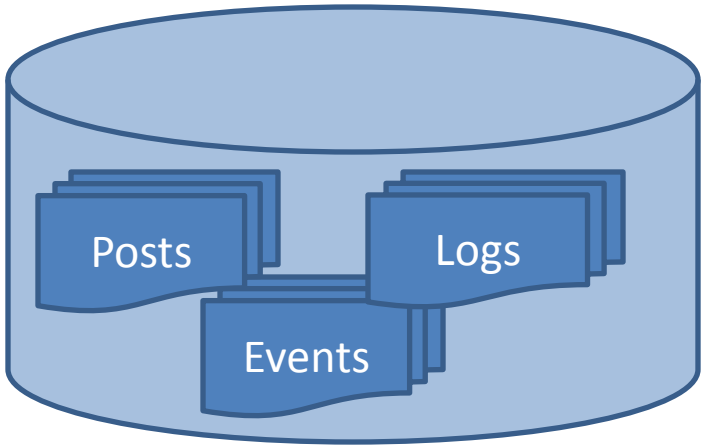
Graph- vs. Dokumentendatenbanken

	Graphdatenbanken Neo4j	Dokumentendatenbanken MongoDB
Datenmodell	Property Graph	Dokumente

Graph- vs. Dokumentendatenbanken

	Graphdatenbanken Neo4j	Dokumentendatenbanken MongoDB
Datenmodell	Property Graph	Dokumente





Graph- vs. Dokumentendatenbanken

	Graphdatenbanken Neo4j	Dokumentendatenbanken MongoDB
Datenmodell	Property Graph	Dokumente
Datenart	Hochgradig vernetzte Daten	Unabhängige, schemalose Daten

Graph- vs. Dokumentendatenbanken

	Graphdatenbanken Neo4j	Dokumentendatenbanken MongoDB
Datenmodell	Property Graph	Dokumente
Datenart	Hochgradig vernetzte Daten	Unabhängige, schemalose Daten
ACID	Ja (READ_COMMITTED)	Nein (atomares In Place Update)

Graph- vs. Dokumentendatenbanken

	Graphdatenbanken Neo4j	Dokumentendatenbanken MongoDB
Datenmodell	Property Graph	Dokumente
Datenart	Hochgradig vernetzte Daten	Unabhängige, schemalose Daten
ACID	Ja (READ_COMMITTED)	Nein (atomares In Place Update)
Abfrage	APIs, Cypher (deklarativ), Gremlin (imperativ)	„Drivers“, MongoQL, MapReduce

Graph- vs. Dokumentendatenbanken

	Graphdatenbanken Neo4j	Dokumentendatenbanken MongoDB
Datenmodell	Property Graph	Dokumente
Datenart	Hochgradig vernetzte Daten	Unabhängige, schemalose Daten
ACID	Ja (READ_COMMITTED)	Nein (atomares In Place Update)
Abfrage	APIs, Cypher (deklarativ), Gremlin (imperativ)	„Drivers“, MongoQL, MapReduce
Skalierbarkeit	M/S Replikation Readperformance	Auto-Sharding (Partitionierung) M/S Replikation Read- + Writeperformance

Graph- vs. Dokumentendatenbanken

	Graphdatenbanken Neo4j	Dokumentendatenbanken MongoDB
Datenmodell	Property Graph	Dokumente
Datenart	Hochgradig vernetzte Daten	Unabhängige, schemalose Daten
ACID	Ja (READ_COMMITTED)	Nein (atomares In Place Update)
Abfrage	APIs, Cypher (deklarativ), Gremlin (imperativ)	„Drivers“, MongoQL, MapReduce
Skalierbarkeit	M/S Replikation Readperformance	Auto-Sharding (Partitionierung) M/S Replikation Read- + Writeperformance
Versionierung	Applikation	Applikation

Graph- vs. Dokumentendatenbanken

	Graphdatenbanken Neo4j	Dokumentendatenbanken MongoDB
Datenmodell	Property Graph	Dokumente
Datenart	Hochgradig vernetzte Daten	Unabhängige, schemalose Daten
ACID	Ja (READ_COMMITTED)	Nein (atomares In Place Update)
Abfrage	APIs, Cypher (deklarativ), Gremlin (imperativ)	„Drivers“, MongoQL, MapReduce
Skalierbarkeit	M/S Replikation Readperformance	Auto-Sharding (Partitionierung) M/S Replikation Read- + Writeperformance
Versionierung	Applikation	Applikation
Pros	Rekursive Abfragen Schemafreiheit Lucene Indexing	Auto-Sharding Schemafreiheit Grid-FS MapReduce

Graph- vs. Dokumentendatenbanken

	Graphdatenbanken Neo4j	Dokumentendatenbanken MongoDB
Datenmodell	Property Graph	Dokumente
Datenart	Hochgradig vernetzte Daten	Unabhängige, schemalose Daten
ACID	Ja (READ_COMMITTED)	Nein (atomares In Place Update)
Abfrage	APIs, Cypher (deklarativ), Gremlin (imperativ)	„Drivers“, MongoQL, MapReduce
Skalierbarkeit	M/S Replikation Readperformance	Auto-Sharding (Partitionierung) M/S Replikation Read- + Writeperformance
Versionierung	Applikation	Applikation
Pros	Rekursive Abfragen Schemafreiheit Lucene Indexing	Auto-Sharding Schemafreiheit Grid-FS MapReduce

Graph- vs. Dokumentendatenbanken

	Graphdatenbanken Neo4j	Dokumentendatenbanken MongoDB
Datenmodell	Property Graph	Dokumente
Datenart	Hochgradig vernetzte Daten	Unabhängige, schemalose Daten
ACID	Ja (READ_COMMITTED)	Nein (atomares In Place Update)
Abfrage	APIs, Cypher (deklarativ), Gremlin (imperativ)	„Drivers“, MongoQL, MapReduce
Skalierbarkeit	M/S Replikation Readperformance	Auto-Sharding (Partitionierung) M/S Replikation Read- + Writeperformance
Versionierung	Applikation	Applikation
Pros	Rekursive Abfragen Schemafreiheit Lucene Indexing	Auto-Sharding Schemafreiheit Grid-FS MapReduce

Graph- vs. Dokumentendatenbanken

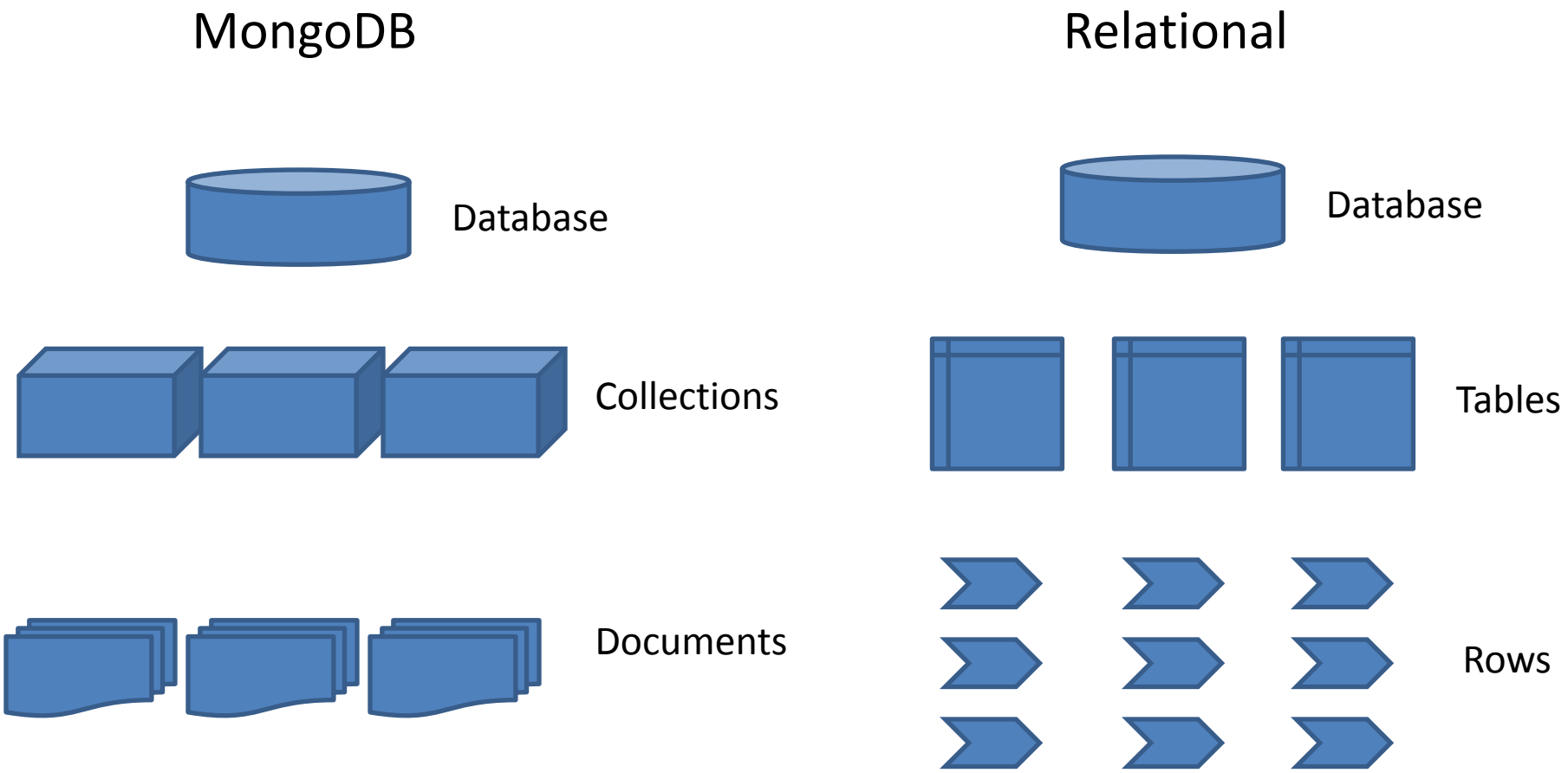
	Graphdatenbanken Neo4j	Dokumentendatenbanken MongoDB
Datenmodell	Property Graph	Dokumente
Datenart	Hochgradig vernetzte Daten	Unabhängige, schemalose Daten
ACID	Ja (READ_COMMITTED)	Nein (atomares In Place Update)
Abfrage	APIs, Cypher (deklarativ), Gremlin (imperativ)	„Drivers“, MongoQL, MapReduce
Skalierbarkeit	M/S Replikation Readperformance	Auto-Sharding (Partitionierung) M/S Replikation Read- + Writeperformance
Versionierung	Applikation	Applikation
Pros	Rekursive Abfragen Schemafreiheit Lucene Indexing	Auto-Sharding Schemafreiheit Grid-FS MapReduce

Graph- vs. Dokumentendatenbanken

	Graphdatenbanken Neo4j	Dokumentendatenbanken MongoDB
Datenmodell	Property Graph	Dokumente
Datenart	Hochgradig vernetzte Daten	Unabhängige, schemalose Daten
ACID	Ja (READ_COMMITTED)	Nein (atomares In Place Update)
Abfrage	APIs, Cypher (deklarativ), Gremlin (imperativ)	„Drivers“, MongoQL, MapReduce
Skalierbarkeit	M/S Replikation Readperformance	Auto-Sharding (Partitionierung) M/S Replikation Read- + Writeperformance
Versionierung	Applikation	Applikation
Pros	Rekursive Abfragen Schemafreiheit Lucene Indexing	Auto-Sharding Schemafreiheit Grid-FS MapReduce


mongoDB

Datenmodell – Überblick



Datenmodell – Dokument

- BSON Dokumente
- Schlüssel-Wert-Paare

```
{
        „boolValue“     : true,
        „intValue“      : 23,
        „stringValue“   : „Ich bin dein Vater“,
        „array“         : [ 42, „in“, [23,42,1337], true],
        „embeddedDoc“   : {
                                „stringValue“ : „…wtf oO“
                          }
}
```

Datenschema

Datenschema – Measurings

- Collection **measurings**
- Beispiel:

```
{
   „timestamp“ : 1314277800000, // Unix-Timestamp (long)
   „value“     : 1337, // numerischer Wert (int)
   „stationID  : „singwitz“, // Anlagenname (string)
   „partID“    : „wr“, // Bauteilart (string)
   „serialNo   : 42, // laufende Nummer (int)
   „datatype“  : „gain“, // Datenart (string)
   „opt1“      : „string“ // Zeichenkette bei upc, pdc
   „opt2“      : 0 // laufende Nummer bei upc, pdc
}
```

Datenschema – Stations

- Collection **`stations`** (update on copy)
- Beispiel:

```
{
   „stationID“    : „wendlinghausen2“, // Identifier (string)
   „creationDate“ : 1234L, // Versionierung
   … // Metadaten
}
```

Abfragen

Abfragen – Beispiel Query 1

- Wieviele Einträge hat Zeitreihe XY insgesamt/im Zeitintervall [von,bis]?

```
db.measurings.find(
{
   "datatype"       : "gain",
   "stationID"      : "wendlinghausen2",
   "serialNo"       : 1,
   "timestamp"      :
   {
       $gt : 1269953100000,
       $lt : 1269970200000
   }
}).count();
```

Abfragen – Beispiel Query 4

- Wie ist der Zeitpunkt des ältesten/neuesten Eintrags in Zeitreihe XY?

```
db.measurings.find(
{
   // Selektion
   "datatype"   : „gain“,
   "stationID"  : "wendlinghausen2",
   "serialNo"   : 1
},
{
   // Projektion
   „timestamp“ : 1
}).sort(
{
   „timestamp“ : [1|-1] // Index!
}).limit(1);
```

Abfragen – Beispiel Query 6

Wie ist der Verlauf des Wirkungsgrades für den Wechselrichter XY im Zeitintervall [von,bis]?

Abfragen – Beispiel Query 6 / 1

```
// function map(void) -> void

map = function() {
        var r = {pac:0, total_pdc:0};

        if(this.datatype == "pac") {
                r.pac = this.value;
        } else {
                r.total_pdc = this.value;
        }
        emit(this.timestamp, r);
}
```

Abfragen – Beispiel Query 6 / 2

```
// function reduce(key, array_of_value) -> value

reduce = function(key, values) {
	var r = {pac : 0, total_pdc:0};

	values.forEach(function(v) {
		r.pac += v.pac;
		r.total_pdc += v.total_pdc;
	});

	return r;
}
```

Abfragen – Beispiel Query 6 / 3

```
// function finalize(key, value) -> final_value

finalize = function(k, r) {
	if(r.total_pdc > 0) {
		return {timestamp : k,
			wirkungsgrad : r.pac / r.total_pdc };
	} else {
		return {timestamp : k,
			wirkungsgrad : 0 };
	}
}
```

Abfragen – Beispiel Query 6 / 4

```
db.runCommand(
{
    mapreduce : „measurings“,
    map   : map,
    reduce : reduce,
    out : { inline : 1 },
    query : {     „datatype“ :
                            { "$in" : [ „pdc“ , „pac“]},
                  „stationID"       : "1555“,
                  „serialNo“        : 1,
                  „timestamp“       :
                            {
                                    „$gt“:   1295743851828,
                                    "$lt“:   1295916651828
                            }
    },
    finalize : finalize
});
```

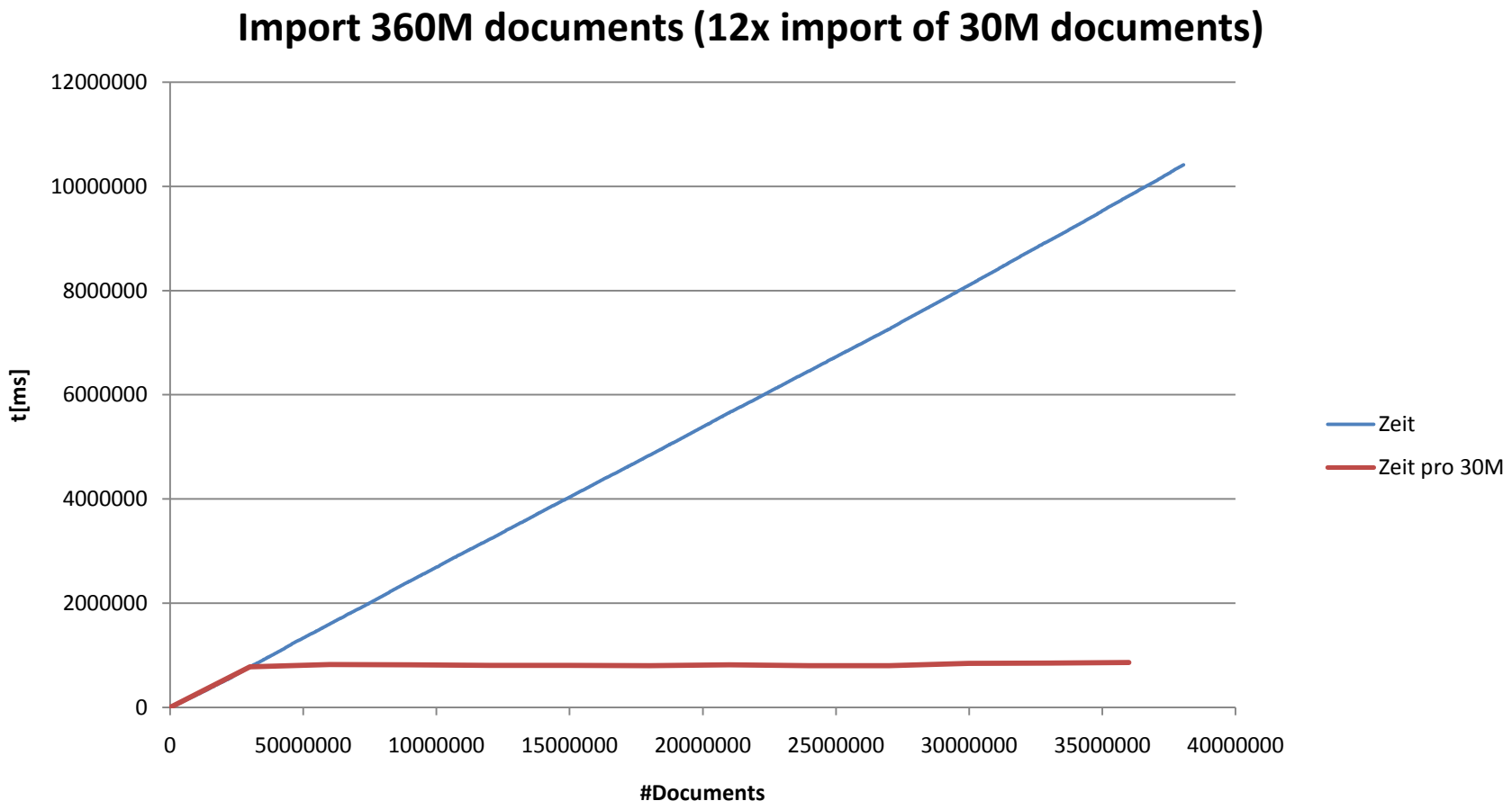
Ergebnisse

Testbed 4 – Single Server

- 4x Intel ® Xeon® CPU E5649 @2.54Ghz
- 6GB RAM
- HDD: 140GB

- Single mongod instance (journaling off)
- Indexe
  - Primary Index `<_id>`
  - MultiKey-Index `<stationID, inverterID, datatype, timestamp>`

Import

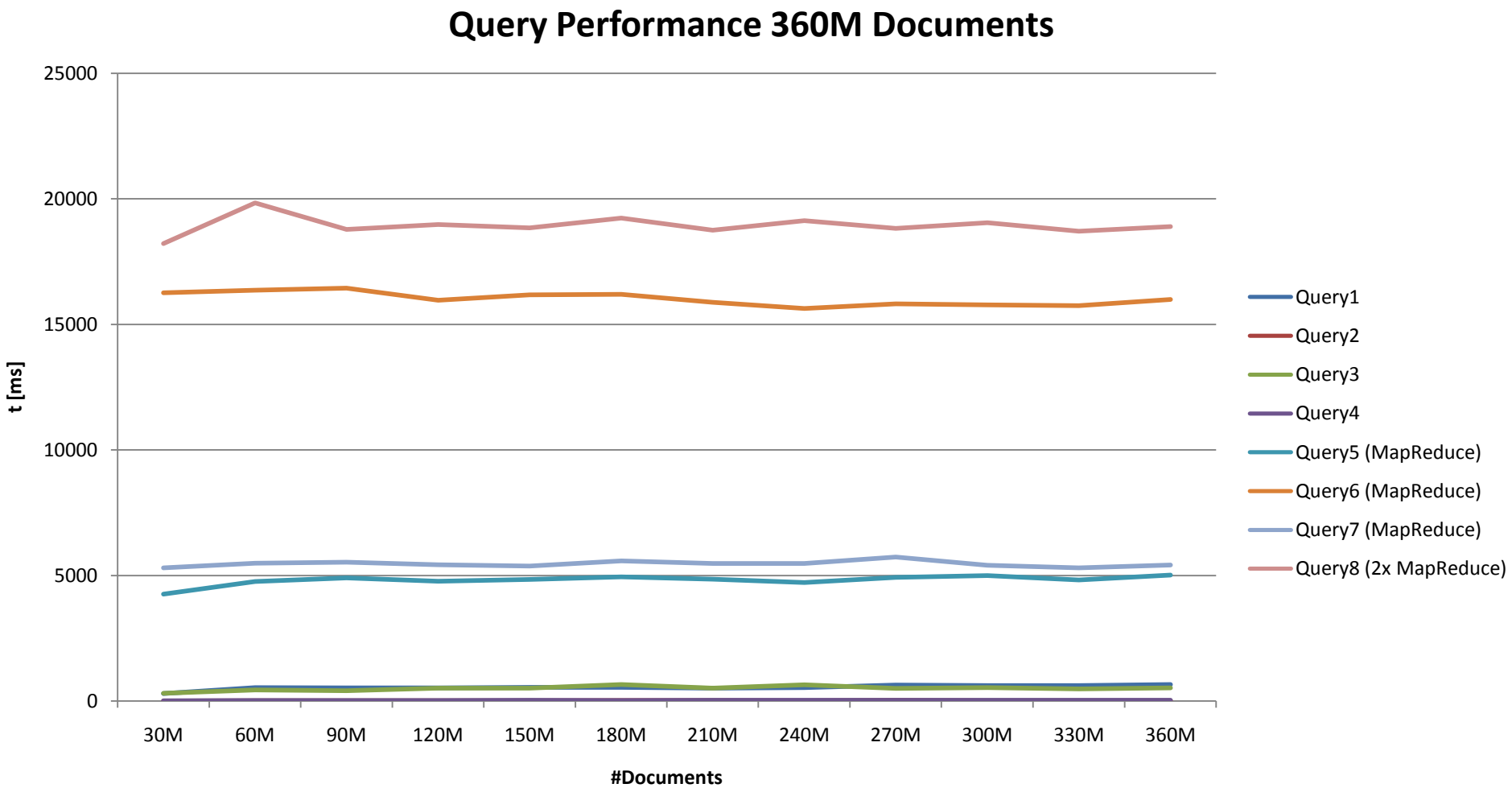

Import 360M documents (12x import of 30M documents)
12000000
10000000
8000000
6000000
4000000
2000000
0
t[ms]
0
50000000
10000000
15000000
20000000
25000000
30000000
35000000
40000000
#Documents
Zeit
Zeit pro 30M

Import – Stats

```
> db.measurings.stats()
{
        "ns" : "BIS_mongo_eval.measurings",
        "count" : 405653636,
        "size" : 35479802848,
        "avgObjSize" : 87.46329306413514,
        "storageSize" : 44114304784,
        "numExtents" : 51,
        "nindexes" : 2,
        "lastExtentSize" : 2146426864,
        "paddingFactor" : 1,
        "flags" : 1,
        "totalIndexSize" : 44796361232,
        "indexSizes" : {
                "_id_" : 13161340528,
                "c_1_e_1_f_1_a_1" : 31635020704
        },
        "ok" : 1
}

[junghanns@eum-prak-01 databases]$ du testbed4/
4       testbed4/_tmp
94277992
```

Queries


Query Performance 360M Documents
25000
20000
15000
10000
5000
0
t [ms]
30M
60M
90M
120M
150M
180M
210M
240M
270M
300M
330M
360M
#Documents
Query1
Query2
Query3
Query4
Query5 (MapReduce)
Query6 (MapReduce)
Query7 (MapReduce)
Query8 (2x MapReduce)

Optimierung

Optimierung – Document size

- Collection **measurings**
- Beispiel:

```
{
„timestamp“ : 1314277800000,
„value“ : 1337,
„stationID“ : „23“,
„partID“ : „wr“,
„serialNo“ : 42,
„datatype“ : „gain“,
[„opt1“ : „string“,]
[„opt2“ : 0]
}
```

```
{
„a“ : 1314277800000,
„b“ : 1337,
„c“ : „23“,
„d“ : „wr“,
„e“ : 42,
„f“ : „gain“,
[„g“ : „string“,]
[„h“ : 0]
}
```

"avgObjSize" : 158.59368266666667 (Byte)

"avgObjSize" : 107.956212 (Byte)

Optimierung – Index

- Wenn möglich, Unique Value als Primärindex
  - _id
- Compound Index Reihenfolge beachten
  - Frühe Einschränkungen
- Index nach Möglichkeit im Hauptspeicher

Optimierung – Queries

- MapReduce deutlich langsamer als Mongo Query Language
  - Nutzt lokal nur eine CPU
  - Nutzt keine Indexstrukturen
- MapReduce: Dokumentenmenge einschränken!
- <query>.explain() zur Analyse verwenden
- <query>.hint(„my_idx“) zur Zuweisung von Index

Optimierung – Sharding

- Horizontale Verteilung der Dokumente
  - Basierend auf Sharding Key
- Verteilung von Schreib- und Leseanfragen
- Sharding Key sinnvoll wählen
  - Gleichmäßige Verteilung
  - Exact Match

Fazit

Fazit – Pros

- Aktives Projekt
  - Viele „Driver“ (Java, Python, Ruby, C, …)
  - Große, aktive Community (IRC, Mailing List)
  - Sehr gute Dokumentation + Videos
- Ausgelegt auf Skalierbarkeit (Partitionierung)
  - Skalierung von Schreib- und Lesezugriffen
  - Automatische Lastbalancierung
- Einfache Installation
- Konstanter Antwortzeit-Scaleup im Experiment

Fazit – Cons

- Hoher Speicherbedarf
  - Dokumente (Overhead)
  - Index
- MapReduce < Mongo Query Language
  - Schlecht für komplexe Queries
- Partitionierung nicht für verteiltes Lesen optimiert
  - Mongo Query Language > Map Reduce

Danke für die Aufmerksamkeit.

Fragen?