DMIF University of Udine

Managing Time Series with MongoDB

Paolo Gallo
paolo.gallo@uniud.it

May 18, 2020

Outline

The United States is beginning to make its transition to self-driving cars.

For this reason United States Department of Transportation is setting up central service to monitor traffic conditions nationwide

Sensors over the interstate system monitor traffic conditions like: car speeds, pavement and weather conditions, etc.

Interstate Highway System

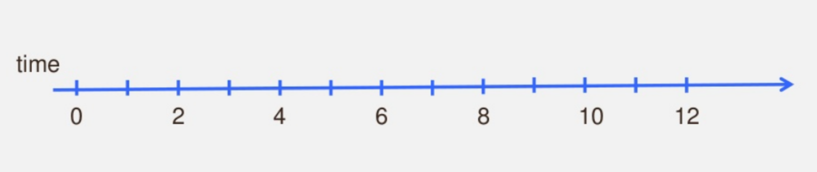
Traffic Sensors

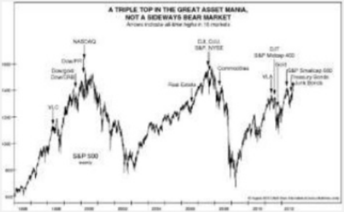
- 16.000 sensors
- Measure
  - Speed
  - Travel Time
  - Weather, pavement, and traffic condition
- Support desktop, mobile, and car navigation systems

Other Requirements

- Need to keep 3 years of history
- Three data centres
  - New York
  - Chicago
  - Los Angeles
- Need to support 5 millions simultaneous users
  - Peak volume (rush hour)
  - Every minute, each request the 10 minute average speed for 50 sensors

A (discrete) *time series* is a sequence of data, measured typically at successive points in time, spaced at uniform time intervals.




A TRIPLE TOP IN THE GREAT ASSET MANIA,
NOT A SIDEWAYS BEAR MARKET
NASDAQ
Real Estate
Commodities
S&P 500

Schema Design Consideration

- Horizontal Scalability
- Store event data
- Support analytical queries
- Find best compromise between:
  - Memory utilisation
  - Write performance
  - Read/Analytical query performance
  - Accomplish with a realistic amount of hardware

Designing for reading, writing, ...

There are different choices in order to store upcoming information with a document DB:

- Document per event
- Document per minute (average)
- Document per minute (second)
- Document per hour
- ...

Document per event

```
{
  segID: "I80_mile34",
  speed: 63,
  ts: ISODate("2016-11-10T22:56:00.00-0500")
 }
```

- Relational centric approach
- Insert driven workload
- Overall performances and resources consumption are same as with a relational DB

Document per minute (average)

```
{
  segID: "I80_mile34",
  speed_num: 18,
  speed_sum: 1256,
  ts: ISODate("2016-11-10T22:56:00.00-0500")
 }
```

- Pre-aggregate to compute average per minute easily
- Update driven workload
- Resolution at the minute level

Document per minute (second)

```
{
  segID: "I80_mile34",
  speed: {0:63, 1:23, 2:45, ..., 59:65},
  ts: ISODate("2016-11-10T22:56:00.00-0500")
 }
```

- Store per-second data at minute level
- Update driven workload
- Pre allocate structure to avoid document move

Document per hour

```
{
  segID: "I80_mile34",
  speed: {0:63, 1:23, 2:45, ..., 3599:65},
  ts: ISODate("2016-11-10T22:00:00.00-0500")
 }
```

- Store per-second data at hourly level
- Update driven workload
- Pre allocate structure to avoid document move
- **Updating last second requires 3599 steps**

Document per hour (by second)

```
{
  segID: "I80_mile34",
  speed:{0: {0:63, ..., 59:45},
         ...,
         59:{0:65, ..., 59:65}
        }
  ts: ISODate("2016-11-10T22:00:00.00-0500")
 }
```

- Store per-second data at hourly level with nesting
- Update driven workload
- Pre allocate structure to avoid document move
- **Updating last second requires 59 + 59 steps**

Characterizing write differences

- Example: data generated every second
- For one minute:
  - *Document Per Event* $\longrightarrow$ 60 Writes
  - *Document Per Minute* $\longrightarrow$1 Write, 59 updates
- Transition from write driven to update driven
  - individual writes are smaller
  - performance and concurrency benefits

Characterizing read differences

- Example: data generated every second
- Reading data for a single hour requires:
  - *Document Per Event* $\longrightarrow$ 3600 reads
  - *Document Per Minute* $\longrightarrow$ 60 reads
- Read performance is greatly improved:
  - optimal with tuned block and read ahead
  - fewer disks seeks

Characterizing memory differences

- _id index for 1 billion events:
  - *Document Per Event* $\longrightarrow$ 32 Gb
  - *Document Per Minute* $\longrightarrow$ 0.5 Gb
- _id index plus segId and ts index:
  - *Document Per Event* $\longrightarrow$ 100Gb
  - *Document Per Minute* $\longrightarrow$ 2 Gb
- memory requirements significantly reduced:
  - fewer shards
  - lower capacity servers

Quick analysis

- Writes:
  - 16.000 sensors, 1 update per minute
  - 16.000 / 60 = 267 updates per second
- Reads:
  - 5 millions simultaneous users
  - Each request data for 50 sensors per minute

Reads: impacts of alternative schemas

Query: Find the average speed over the last ten minutes

10 minute average query		
Schema	1 sensor	50 sensors
1 doc per event	10	500
1 doc per 10 min	1.9	95
1 doc per hour	1.3	65

10 minute average query with 5M users	
Schema	ops/sec
1 doc per event	42M
1 doc per 10 min	8M
1 doc per hour	5.4M

Writes: impacts of alternative schemas

1 Sensor - 1 Hour		
Schema	**Inserts**	**Updates**
doc/event	60	0
doc/10 min	6	54
doc/hour	1	59

16000 Sensors – 1 Day		
Schema	**Inserts**	**Updates**
doc/event	23M	0
doc/10 min	2.3M	21M
doc/hour	.38M	22.7M

Memory: impacts of alternative schemas

1 Sensor - 1 Hour		
Schema	# of Documents	Index Size (bytes)
doc/event	60	4200
doc/10 min	6	420
doc/hour	1	70

16000 Sensors – 1 Day		
Schema	# of Documents	Index Size
doc/event	23M	1.3 GB
doc/10 min	2.3M	131 MB
doc/hour	.38M	1.4 MB

Document structure

```
{  _id: ObjectId("5382ccdd58db8b81730344e2"),
linkId: 900006,
date: ISODate("2014-03-12T17:00:00Z"),
data: [
     { speed: NaN, time: NaN },
     { speed: NaN, time: NaN },
     { speed: NaN, time: NaN },
     ...
  ],
  conditions: {
     status: "Snow / Ice Conditions",
     pavement: "Icy Spots",
     weather: "Light Snow"
  }
}
```

Query: two indexes required

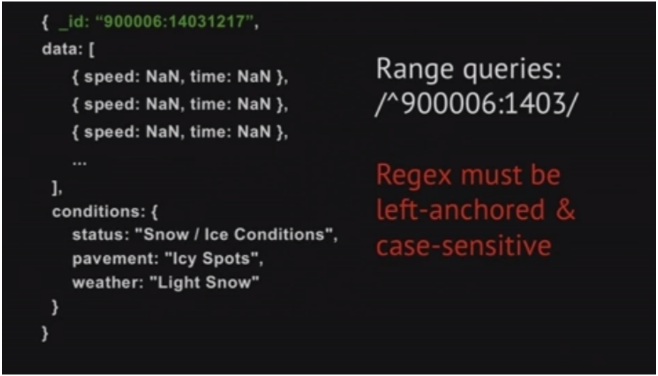
```
{
"segId" : "20484097",
"ts" : ISODate("2013-10-10T23:06:37.000Z"),
"time" : "237",
"speed" : "52",
"pavement": "Wet Spots",
"status" : "Wet Conditions",
"weather" : "Light Rain"
}
```

~70 bytes per document

A Smart Sample Index

```
{ _id: "900006:14031217",
data: [
    { speed: NaN, time: NaN },
    { speed: NaN, time: NaN },
    { speed: NaN, time: NaN },
    ...
 ],
 conditions: {
    status: "Snow / Ice Conditions",
    pavement: "Icy Spots",
    weather: "Light Snow"
 }
}
```

Saves an extra index

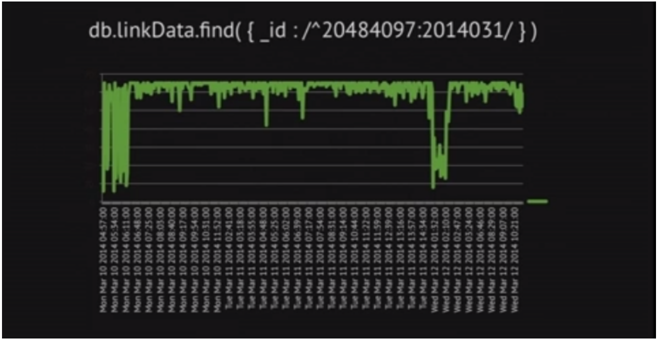

{ _id: "900006:14031217",
data: [
{ speed: NaN, time: NaN },
{ speed: NaN, time: NaN },
{ speed: NaN, time: NaN },
...
],
conditions: {
status: "Snow / Ice Conditions",
pavement: "Icy Spots",
weather: "Light Snow"
}
}
Range queries:
/^900006:1403/
Regex must be
left-anchored &
case-sensitive

Preallocate data

```
{ _id: "900006:140312",
data: [
    { speed: NaN, time: NaN },
    { speed: NaN, time: NaN },
    { speed: NaN, time: NaN },
    ...
 ],
 conditions: {
    status: "Snow / Ice Conditions",
    pavement: "Icy Spots",
    weather: "Light Snow"
 }
}
```

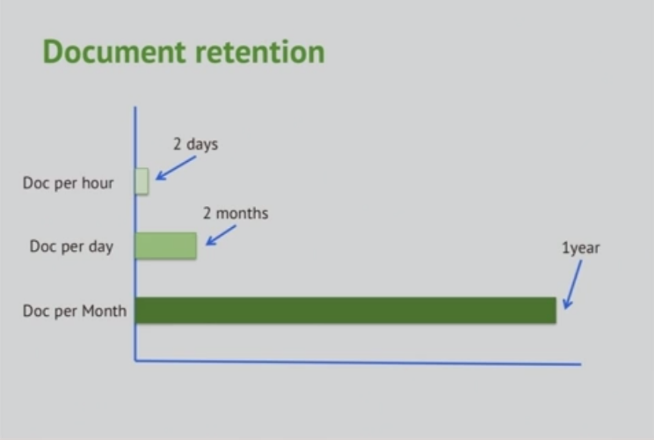
Pre-allocated,
60 element array of
per-minute data

Chart



Rollup

```
{ _id: "20484097:20140204",
  hours: [
    { speed: { sum: 1889, count: 60 }
      time: { sum: 20562, count: 60 },
      conditions: {
        status: "Snow / Ice Conditions",
        pavement: "Icy Spots",
        weather: "Light Snow"
      }
    },
    { speed: {m: 1892, count: 60 },
      time: {sum: 20442, count: 60 },
      conditions: {
        status: "Snow / Ice Conditions",
        pavement: "Slush",
        weather: "Light Snow"
      }
    }
  ]}
```


Document retention
2 days
Doc per hour
2 months
Doc per day
1year
Doc per Month

What is the average speed for a segment ?

```
>db.linkData.aggregate{
      {$match:{"_id:"/^20484097/}},
      {$project:{"data.speed":1,linkId:1}},
      {$unwind:"$data"},
      {$group:{_id:"$linkId",ave:$avg{"$data.speed"}}

};

{_id:"20484097",ave:"47.3665646546"}
```