The new time series kid on the block

Florian Lautenschlager
@flolaut
That’s what we need a time series database for.
68.000.000.000* time correlated data objects.

How to store such amount of data on your laptop computer and retrieve any point within a few milliseconds?

* collect every 10 seconds 72 metrics x 15 processes x 20 hosts over 1 years
Well we tried that approach…

- Store data objects in a classical RDBMS

But…
- Slow import of data objects
- Huge amount of hard drive space
- Slow retrieval of time series
- Limited scalability due to RDBMS
- Missing query functions for time series data
Hence it felt like ...
But what to do? Chunks + Compression + Document storage!

The key ideas to enable the efficient storage of billion data objects:
- Split time series into chunks of the same size with data objects
- Compress these chunks to reduce the data volume
- Store the compressed chunk and the attributes in one record

Reason for success:
- 32 GB disk usage to store 68 billion data objects
- Fast retrieval of data objects within a few milliseconds
- Fast navigation on attributes (finding the chunk)
- Everything runs on a laptop computer
- … and many more!
That’s all. No secrets, nothing special and nothing more to say.

What comes next?

- Time Series Database - What’s that? Definitions and typical features.
- Why did we choose Apache Solr and are there alternatives?
- Chronix Architecture that is based on Solr and Lucene.
- What’s needed to speed up Chronix to a firehorse.
Definition 1: “A data object $d$ is a tuple of $\{\text{timestamp, value}\}$, where the value could be any kind of object.”

Definition 2: “A time series $T$ is an arbitrary list of **chronological ordered** data objects of one value type”.

Definition 3: “A chunk $C$ is a chronological ordered **part of a time series**.”

Definition 4: “A time series database $\text{TSDB}$ is a specialized database for **storing and retrieving** time series in an **efficient and optimized** way.”
### A few typical features of a time series database

<table>
<thead>
<tr>
<th>Data management</th>
<th>Performance and Operational</th>
</tr>
</thead>
<tbody>
<tr>
<td>■ Data management</td>
<td>■ Rare updates, Inserts are additive</td>
</tr>
<tr>
<td>■ Round Robin Storages</td>
<td>■ Fast inserts and retrievals</td>
</tr>
<tr>
<td>■ Down-sample old time series</td>
<td>■ Distributed and efficient per node</td>
</tr>
<tr>
<td>■ Compression</td>
<td>■ No need of ACID, but consistency</td>
</tr>
<tr>
<td>■ Delta-Encoding</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Describing Attributes</th>
<th>Time series language and API</th>
</tr>
</thead>
<tbody>
<tr>
<td>■ Arbitrary amount of attributes</td>
<td>■ Statistics: Aggregation (min, max, median), …</td>
</tr>
<tr>
<td>■ For time series (Country, Host, Customer, …)</td>
<td>■ Transformations: Time windows, time shifting, resampling, ..</td>
</tr>
<tr>
<td>■ For data object (Scale, Unit, Type)</td>
<td>■ High level analyses: Outlier, Trends</td>
</tr>
</tbody>
</table>

Check out: A good post about the requirements of a time series:  
Some time series databases out there.

- **RRDTool** - [http://oss.oetiker.ch/rrdtool/](http://oss.oetiker.ch/rrdtool/)
  - Mainly used in traditional monitoring systems

- **Graphite** – [https://github.com/graphite-project](https://github.com/graphite-project)
  - Uses the concepts of RRDTool and puts some sugar on it

- **InfluxDB** - [https://influxdata.com/time-series-platform/influxdb/](https://influxdata.com/time-series-platform/influxdb/)
  - A distributed time series database with a very handy query language

- **OpenTSDB** - [http://opentsdb.net/](http://opentsdb.net/)
  - Is a scalable time series database and runs on Hadoop and Hbase

- **Prometheus** - [https://prometheus.io/](https://prometheus.io/)
  - A monitoring system and time series database

- **KairosDB** - [https://kairosdb.github.io/](https://kairosdb.github.io/)
  - Like OpenTSDB but is based on Apache Cassandra

- ... many more! And of course Chronix! - [http://chronix.io/](http://chronix.io/)
“Ey, there are so many time series databases out there? Why did you create a new solution?”

“Our tool has been around for a good few years, and in the beginning there was no time series database that complies our requirements. And there isn’t one today!”

Our Requirements

- A fast write and query performance
- Run the database on a laptop computer
- Minimal data volume for stored data objects
- Storing arbitrary attributes
- A query API for searching on all attributes
- Large community and an active development

That delivers Apache Solr

- Based on Lucene which is really fast
- Runs embedded, standalone, distributed
- Lucene has a built-in compression
- Schema or schemaless
- Solr Query Language
- Lucidworks and an Apache project
Let’s dig deeper into Chronix’ internals.

Image Credit: http://www.taringa.net/posts/ciencia-educacion/12656540/La-Filosofia-del-Dr-House-2.html
Chronix’ architecture enables both efficient storage of time series and millisecond range queries.

1. Semantic Compression
2. Attributes and Chunks
3. Basic Compression
4. Multi-Dimensional Storage

- Optional

Record data:
- <chunk>
- attributes

Record:
- data:compressed <chunk>
- attributes

Optional Storage

68 Billion Points
1 Mio. Chunks * 68.000 Points
~ 96% Compression
The key data type of Chronix is called a record. It stores a compressed time series chunk and its attributes.

```plaintext
record{
  data:compressed{<chunk>}
  //technical fields
  id: 3dce1de0-...-93fb2e806d19
  version: 1501692859622883300
  start: 1427457011238
  end: 1427471159292
  //optional attributes
  host: prodI5
  process: scheduler
  group: jmx
  metric: heapMemory.Usage.Used
  max: 896.571
}
```

- **Data:compressed{<chunk of time series data>**}
  - Time Series: timestamp, numeric value
  - Traces: calls, exceptions, …
  - Logs: access, method runtimes
  - Complex data: models, test coverage, anything else…

- **Optional attributes**
  - Arbitrary attributes for the time series
  - Attributes are indexed
  - Make the chunk searchable
  - Can contain pre-calculated values
Chronix provides specialized aggregations, transformations, and analyses for time series that are commonly used.

### Aggregations (ag)
- Min / Max / Average / Sum / Count
- Percentile
- Standard Deviation
- First / Last
- Range

### Analyses (analysis)
- Trend Analysis
  Using a linear regression model
- Outlier Analysis
  Using the IQR
- Frequency Analysis
  Check occurrence within a time range
- Fast Dynamic Time Warping
  Time series similarity search
- Symbolic Aggregate Approximation
  Similarity and pattern search

### Transformations (tr)
- Bottom/Top n-values
- Moving average
- Divide / Scale
- Vectorisation
Only scalar values? One size fits all? No! What about logs, traces, and others? No problem – Just do it yourself!

```java
public interface TimeSeriesConverter<T> {
    /**
     * Shall create an object of type T from the given binary time series.
     */
    T from(BinaryTimeSeries binaryTimeSeriesChunk, long queryStart, long queryEnd);

    /**
     * Shall do the conversation of the custom time series T into the binary time series that is stored.
     */
    BinaryTimeSeries to(T timeSeriesChunk);
}
```

Chronix Kassiopeia (Format)

- Time Series framework that is used by Chronix.

- Time Series Types:
  - **Numeric**: Doubles (the time series known to be the default)
  - **Thread Dumps**: Stack traces (e.g. java stack traces)
  - **Strace**: Strace dumps (system call, duration, arguments)
That’s the easiest way to play with Chronix. A single instance of Chronix on a single node with a Apache Solr instance.

Java 8 (JRE)

Your Computer

Chronix - 0.2

Solr - 6.0.0

Solr plugins

Chronix-Query-Handler
Chronix-Response-Writer
Chronix-Retention

Lucene

8983

Java 8 (JRE)
Chronix-Client

HTTP
Plain
Json + Binary
Binary + Binary
Json + Json
Code-Slide: How to set up Chronix, ask for time series data, and call some server-side aggregations.

- Create a connection to Solr and set up Chronix
  ```java
  solr = new HttpSolrClient("http://localhost:8913/solr/chronix/")
  chronix = new ChronixClient(new KassiopeiaSimpleConverter<>()
      , new ChronixSolrStorage(200, groupBy, reduce))
  ```

- Define and range query and stream its results
  ```java
  query = new SolrQuery("metric:*Load*")
  chronix.stream(solr, query)
  ```

- Call some aggregations
  ```java
  query.addFilterQuery("ag=max,min,count,sdiff")
  stream = chronix.stream(solr, query)
  ```

  Group chunks on a combination of attributes and reduce them to a time series.

  Get all time series whose metric contains **Load**

  Signed Difference: First=20, Last=-100 ➔ -80
That's the four week data that is shipped with the release!
Tune Chronix to a firehorse. Even with defaults it’s blazing fast!
We have tuned Chronix in terms of chunk size, and compression technique to get the ideal default values for you.

■ Tuning Dataset
  ■ Three real-world projects
  ■ 15 GB of time series data (typical monitoring data)
  ■ About 500 million points in 15k time series
  ■ 92 typical queries with different time range and occurrence

■ We have measured:
  ■ Compression rate for serval compression techniques (T) and chunk sizes (C).
  ■ Query time for all 92 queries in the mix (range + aggregations)

■ What we want to know: Ideal values for T and C
We have evaluated several compression techniques and chunk sizes of the time series data to get the best parameter values.

T = GZIP + C = 128 kBytes

For more details about the tuning check our paper.

Florian Lautenschlager, Michael Philippsen, Andreas Kumlehn, Josef Adersberger
Chronix: Efficient Storage and Query of Operational Time Series
International Conference on Software Maintenance and Evolution 2016 (submitted)
Compared to other time series databases Chronix’s results for our use case are outstanding. The approach works!

- We have evaluated Chronix with:
  - InfluxDB, Graphite, OpenTSDB, and KairosDB
  - All databases are configured as single node

- Storage demand for 15 GB of raw csv time series data
  - Chronix (237 MB) takes 4 – 84 times less space

- Query times on imported data
  - 49 – 91 % faster than the evaluated time series databases

- Memory footprint: after start, max during import, max during query mix
  - Graphite is best (926 MB), Chronix (1.5 GB) is second. Others 16 to 39 GB
The hard facts. For more details I suggest you to read our research paper about Chronix.

Florian Lautenschlager, Michael Philippsen, Andreas Kumlehn, Josef Adersberger

Chronix: Efficient Storage and Query of Operational Time Series

International Conference on Software Maintenance and Evolution 2016 (submitted)
Now it’s your turn.
The whole Chronix Stack. Not yet *completely* implemented.
Outlook: A powerful way to work with time series. A Chronix Cloud, a Spark Cluster, and an analysis workbench like Zeppelin.

Various Applications as Workbench

Zeppelin
Java
Scala

Spark Cluster

Spark Node
Spark Node
Spark Node
Spark Node

Chronix Cloud

Chronix Node
Chronix Node
Chronix Node
Chronix Node

Chronix and Spark.
Time Series Processing with Apache Spark – Josef Adersberger, Wed, 3:00 pm
Other interesting related talks:

Real-world Analytics with Solr Cloud and Spark – Johannes Weigend, Wed, 3:00 pm

Time Series Processing with Apache Spark – Josef Adersberger, Wed, 3:00 pm
Code-Slide: Use Spark to process time series data that comes out right now from Chronix.

- Create a ChronixSparkContext
  
  ```java
  conf = new SparkConf().setMaster(SPARK_MASTER).setAppName(CHRONIX)
  jsc = new JavaSparkContext(conf)
  csc = new ChronixSparkContext(jsc)
  sqlc = new SQLContext(jsc)
  ```

- Define and range query and stream its results
  
  ```java
  query = new SolrQuery("metric:*Load*")
  rdd = csc.queryChronixChunks(query, ZK_HOST, CHRONIX_COLLECTION,
                              new ChronixSolrCloudStorage());
  ```

- Play with the data
  
  ```java
  DataSet<MetricObservation> ds = rdd.toObservationsDataset(sqlContext)
  rdd.mean()
  rdd.max()
  rdd.iterator()
  ```

Set up Spark, a JavaSparkContext, a ChronixSparkContext, and a SQLContext

Get all time series whose metric contains Load

Dataset to use Spark SQL features