ABSTRACT
In this work we are presenting a time series datasets visualization combining multiple datasets sharing same Y domain into one chart which are then stacked vertically. All interactions are synced between charts which allows easy interaction with all charts at the same time, providing a unifying feel. We see such visualization as a tool for analysis and exploration of a large set of big time series datasets. Visualization runs in the browser and to minimize data transferred to the browser, server side component precomputes downsampled datapoints at various granularity levels.

Author Keywords
Time series; visualization; comparison; big data; storage; downsampling.

ACM Classification Keywords
H.5.3. Group and Organization Interfaces: Web-based interaction

RELATED WORK
Many components used in our approach have been proposed and used separately in the past, but to our knowledge a combined and comprehensive approach have not yet been evaluated.

imMens [8] presents a way for visual querying of big data in the browser using multivariate data tiles and parallel query processing implemented with WebGL for data processing and rendering on the GPU. It uses binned aggregation to enable multiple levels of resolution and then folds them efficiently into data cubes. Although their method can be applied to multiple data types, they focus in geospatial data. Our approach of downsampling datapoints for each granularity level is similar, but we do not use only binned aggregation but a wide set of different downsampling operators which we all precompute on the server side. Additionally, we concentrate only on one data type, time series numerical data.

Nanocubes [7] help real-time exploration of large multidimensional spatiotemporal datasets through new data structure called nanocubes, building upon well-known data cubes approach. It uses hierarchical structure in space and time to allow efficient queries to generate various visualizations. Paper concentrates on efficient storage both in terms of memory and CPU usage. In our approach we are using a simple layered data structure where each granularity level is a collection in our database, leaving to the client to determine which layer it wants to query.

Nanocubes [7] help real-time exploration of large multidimensional spatiotemporal datasets through new data structure called nanocubes, building upon well-known data cubes approach. It uses hierarchical structure in space and time to allow efficient queries to generate various visualizations. Paper concentrates on efficient storage both in terms of memory and CPU usage. In our approach we are using a simple layered data structure where each granularity level is a collection in our database, leaving to the client to determine which layer it wants to query.

Horizon Graphs [2] are an alternative approach to visualizing time series datasets with emphasis on low consumption of vertical space. The idea is to fold graph vertically multiple times onto a smaller vertical space. By that reading of values is mostly impossible, but following of trends and changes by the user is still easy. Because of small vertical space consumption, many such time series datasets can be stacked together on top of each other. The idea of stacking and keeping in sync time series visualizations is something we are using
in our approach as well. We determined that we want more readable visualizations because we are not considering just trends, but analysis and exploration of time series datasets as well.

Stack Zooming [6] provides time series visualization where once you select a region of the dataset you want to look into closer, a new more detailed visualization opens below instead of replacing the main view. In this way user can have a better understanding of both local and global picture of the dataset. We use something similar by providing a constant overview of the whole but simplified dataset below each visualization. User can select and pan to choose timespan shown in the main view.

Cube [1] provides a similar simple implementation of time series storage and it provides downsampling when querying datapoints, but it computes downsampled values on the fly every time. This does allow more complex and arbitrary functions specified by the client, but on the other hand it puts additional load on the server. We believe that precomputing is a better approach especially with current low storage costs and because of the open source nature of the project where users can add custom downsampling operators to the codebase at will.

We implemented a similar visualization\(^1\) in the past. We prerendered on the server static time-series visualization images for various timespans and provided an interface in the browser for users to select time series and timespan. This works surprisingly well in practice, but has few issues. User cannot export time series dataset data when he or she finds a view he or she likes, because all that is available is just a prerendered image. Only prerendered options are available to the user. Most importantly, we have to prerender every time series dataset for every timespan even if it will never be used by any user. We have to prerender this every time time series dataset gets a new datapoint. But this previous visualization contains some similarities to the new one. User can select a timespan for multiple visualizations at the same time. They are all stacked vertically that comparison between them is easier.

\(^1\)An example you can see live at [https://nodes.wlan-si.net/node/fri/](https://nodes.wlan-si.net/node/fri/)

**METHODS**

Our system allows concurrent visualization of multiple time series datasets in the browser and interaction with it.

**Design Principles**

Design principles guiding us were:

- visualization should be in the browser
- interactive, zoomable, mouse interface
- scalable, RESTful API
- support for huge datasets
- minimization of “operation mode” switching
- no need to understand the technical background of the underlying system
- only minimal amount of data should be transferred to the browser

**Structure**

The system consists of three components:

- server side time series storage (Python + MongoDB [5])
- HTTP RESTful interface (Python + Django [3])
- client visualization (JavaScript + Highstock [4])

Server side component store datasets into the database and take care of downsampling datapoints into multiple granularity levels. As datapoints are inserted, it automatically downsamples datapoints to lower granularity levels. Highest supported resolution for datapoints is a second, and then the server side component will downsample them.

It is important to notice that downsampling can happen for the datapoint value and the datapoint timestamp. Our server component supports that. It takes a list of datapoints for a timespan at a higher granularity level and creates a downsampled value and downsampled timestamp for a datapoint at a lower granularity level. User can configure what exactly this downsampled datapoint contains. For example, user can configure that it contains a mean, minimum and maximum of all values from a timespan. Same for the timestamp, for example, you can configure that timestamp for the datapoint contains first, last and mean timestamps of all datapoints from a timespan.

All downsampling timespans for all stored time series are equal and rounded at reasonable boundaries (for example, hour granularity starts and ends at full hour). This allows easier alignment between datapoints when visualizing them.

**Supported value downsamplers are:**

- average of all datapoints
- median of all datapoints
- sum of all datapoints
- minimum value of all datapoints
- maximum value of all datapoints
- sum of squares of all datapoints
- standard deviation of all datapoints
- count, number of all datapoints
- the most often occurring value of all datapoints
- the least often occurring value of all datapoints
- for each value number of occurrences in all datapoints

**Supported time downsamplers are:**

- average of all timestamps
- median of all timestamps
• the first timestamp of all datapoints
• the last timestamp of all datapoints

Highest supported resolution is second and datapoints are downsampled to 10 seconds, 1 minute, 10 minutes, 1 hour, 10 hours, 1 day level.

Additionally, server side component supports derived time series. Derived time series are automatically generated from other stored time series datapoints as new datapoints are inserted to those them. For example, you can create a time series which compute derivative of another time series. Or sums multiple time series together.

Server side component is implemented in Python.

The server side component is exposed through RESTful HTTP interface with web applications. By using a standard HTTP interface we allow easy building of alternative visualizations and easy exporting of datapoints once an interesting timespan has been found. Our client visualization uses this same interface to retrieve its own data to visualize. Because we precompute all downsampled datapoints in advance on the server, client can request values from multiple downsampling operators at once, to allow for our visualization using both mean, min and max downsampled values. HTTP interface is implemented in Django.

On the client side, we are building upon Highstock JavaScript visualization library. We fetch information about available time series datasets from the server and then combine those which have matching Y domain together into one chart. These charts we then vertically stack together. When user interacts with any of visualized time series datasets, we manipulate all time series datasets the same to keep them in sync and easier to compare.

Implementation Details
Metadata for time series is stored in one MongoDB collection, datapoints are stored separately in combined datapoints “granularity” collections, where “granularity” is one of the possible granularity levels.

When performing downsampling, we have to differentiate between two timestamps:
• Datapoint timestamp is the timestamp of the datapoint that has been inserted for a given granularity level. On the highest granularity level it is always second precision. On lower granularity levels it is a dictionary of multiple values, depending on time downsamplers settings for a given time series.
• Internal datapoint timestamp (stored in datapoint’s internal id) is based on a timespan for the given granularity level. For example, if a datapoint was inserted at 31-07-2012 12:23:52, then the downsampled internal timestamp for the timespan this datapoint is in for hour granularity would be 31-07-2012 12:00:00 and for month granularity would be 01-07-2012 00:00:00.

Based on highest granularity setting, inserted datapoints are stored in the collection configured by highest granularity and only lower granularity values are downsampled. Requests for granularity higher than highest granularity simply return values from highest granularity collection. Highest granularity is just an optimization to not store unnecessary datapoints for granularity levels which would have at most one datapoint for their granularity timespans.

RESULTS
For each time series dataset visualized, we displayed initially whole dataset by using downsampled datapoints. We draw a line through points representing mean value at each downsampled timespan. In the background layer we draw area around the line where with minimum and maximum values as limits. See Figure 1 for an example on a simulated data generated randomly twice on three intervals. Visualization supports among others selecting, panning, zooming, and highlighting user interactions. All time series are visualized as parallel inter-locked series on one chart, which are then in turn in parallel inter-locked themselves. Any interaction with any of them invoke interaction with others as well.

When user zooms in, as in Figure 2, only data for the granularity levels most suitable for the timespan requested is send to the user. Not just mean, but minimum and maximum is visualized as well so that user can get better feeling how data is distributed. Depending on data such view can be unclear, so user can choose to highlight particular time series. In Figure 3 blue time series are highlighted.

Zooming in fully makes system visualize not downsampled, original, datapoints (see Figure 4). There is no range to visualize at this granularity level.

Figure 5 shows example visualization of four time series datasets with real data. Because datapoints are not random, minimum and maximum area around the mean is much more consistent. You can highlight only time series to easier read from the visualization, or simply use a tooltip as shown in...
Figure 1. Initial view for 6 time-series datasets where each two have matching Y domain so are visualized together.

Figure 2. Zoomed in so that minimum and maximum range area around mean is clearly visible.
Figure 3. Blue time series datasets are highlighted so it is easier for them to be read.

Figure 4. Zoomed in fully to direct datapoints that were inserted into time series.
Figure 5. Visualization using real data of packet loss on links between network devices. You can see that minimum value of “ilq” is consistently lower than that of “lq” for a given link, but average “ilq” and “lq” over multiple links is similar.

Figure 6. Highlighting blue time series datasets from real data.
In Figure 7 you can see a detail of the interface. You can see both smoothed navigator time series visualization below and main view above. As you move mouse over time series a tooltip with exact values is displayed. You can see both line representing mean values and area representing minimum and maximum values at given timespans.

**DISCUSSION**

We presented our visualization to users for unstructured evaluation. Some users have been familiar with other time series visualizations and some users were novice users for this type of visualizations. We got generally positive and enthusiastic feedback. The only main drawback users complained about is initial delay while visualization is initializing and reading information about existing streams, preparing placeholders into which to draw time series datasets. When many time series datasets exists in the system this can take more than 10 seconds and even trigger browser’s “page is unresponsive” warning. More precise evaluation is needed and comparison with other related visualizations.

Looking back to our own implementation and comparing to other existing implementation of similar visualizations we do not know of any other comparable visualization with such feature set. Time series visualization generally deal with visualizing only individual datasets and even with recent move to web browser-based visualizations they still provide visualization of only one dataset. The exception is Horizon Graphs [2], which is rather specialized visualization.

Current implementation assumes that this visualization is integrated into existing web service which has read-only time series datasets to visualize. It does not provide ways for edit datasets or add new datasets. This is seen as out of scope for the visualization.

In the course of designing this visualization, initially we put all time series datasets to one chart ignoring possible differences between Y domains. We found this very confusing so we decide to implement vertical stacking of same Y domain charts. Despite taking more vertical space this has shown as clearer to read and analyze. Highlighting additionally improved usability of the visualization.

**FUTURE WORK**

Our work is preliminary and many future venues of improvements are possible. Current implementation lacks any optimizations and improving client performance for rendering time series datapoints in the browser is definitely one future venue. But more interesting would be to support other types of values, not just numerical values. For example, we could support time series datasets with discrete events stored which could be overlaid as vertical lines on top of numerical time series. In this way user could see when some event occurred and easier understand possible changes of behavior.

Another interesting value type would be support for graphs. So that we could visualize a time series of networks changing through time. As you would move timespan around, graph would dynamically move into new configuration with minimal movements. It would be interesting to think about how to create granularity levels for graphs. What is average graph from a set of multiple graphs in a timespan?

Because our visualization works online, easy sharing of each visualization configuration could be provided as well. So once a user finds an interesting visualization configuration, he or she can share the unique URL for it.

Currently we provide only line graphs. We could provide other types of visualizing time series datasets: stacked graphs, areas, background color regions. The issue here is that we should assure for visualizations to not become too crowded, but highlighting can help here as well.

Because we are already preprocessing time series datasets on the server, we could also compute some time series properties, like auto-correlation, and then visualize this on top of original datapoints.

**CONCLUSION**

The visualization we presented in this paper combines some of existing best practices in the visualization field and uses them for comparative analyze of multiple time-series datasets simultaneously. It provides selecting, panning, zooming, and highlighting. By vertically stacking charts and syncing interactions between them, we retain correspondence between time series datasets and allow easier comparison without sacrificing clarity by overlaying too many layers on top of each other. The visualization has proven doable and usable and satisfied our design principles and goals.

**ACKNOWLEDGMENTS**

We would like to thank Jernej Kos, who helped wrote the backend data storage code. The project is open source and had few other contributors in the past. We would like to thank them as well. Their work motivated us to implement the system and visualization presented in this paper.

**REFERENCES**