# Integration of Disk Based Storage into In-Memory Time-Series Engine

**Marcelo C. Bortolozzo**[1]

[1]Instituto de Informtica – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brazil

mcbortolozzo@inf.ufrgs.br

***Abstract.*** *The growth of the Internet of Things and Big Data applications, has increased the storage and performance requirements on time series storage methods. These databases are split into in-memory and disk based, each with its own shortcomings. This work aims to present an alternative, through the integration between existing time series in-memory engine and disk storage tool. Thus minimizing the faults of each, regarding performance and storage capacity.*

## 1. Introduction

Time Series Data has been obtaining an increasing importance in the last few years, accompanying the rise of Big Data and the Internet of Things (IoT). This is justified by the nature of Time Series Data itself, which always is composed of large, and constantly increasing datasets, providing Big Data Analysis tools with endless information. In the near future, the amount of data generated through IoT devices could reach up to 600 ZB per year by 2020, as predicted by [Index 2016]. All this data is generally represented as time series, and even if not all of it is relevant enough to require storage, the remaining amount of data would still be very large.

The result of this growth, was the need for adequate storage methods, which provide advantages over general purpose databases, thus requiring specialized time series storage methods. They provide more adequate storage through compression, and faster access for this specific kind of data, besides some aggregations through analysis, which speed up the proces. They can be mainly divided into two categories, disk and in memory time series databases.

In spite of being viable solutions, they each have their own advantages and shortcomings. In memory databases are capable of faster processing of requests and insertions, it has a limited single node capacity, requiring larger clusters to handle its data. While disk databases provide a much more flexible storage expansion, it also has much slower processing and insertion.

In this scenario, the aim of this project is to present a viable alternative to these two categories, a solution in which both types of database are interconnected, in order to provide fast processing capabilities, while also having a good single node storage capacity. This might be achieved through the integration of a disk database to the in memory time series engine, both running on the same node of the cluster.

This document will present the preliminary steps for such a project, with the required background of the existing technologies, a more detailed explanation of the proposed product to be implemented and the methods which will be used to make this implementation possible.

## 2. Background

The project presented on this document relies closely on the intrinsic characteristics of time series data, as a they affect the way databases handle the data for better performance. This requires a short presentation on this characteristics, as they will be required for later explanations of the project.

### 2.1. Time Series

Time Series is the name given to a sequence of discrete-time data points, indexed by time, and containing values, thus, each data point, representing a state at a certain point in time. This may, for example, represent the data obtained from several measurement instruments, each possessing a series of values produced, over a period of time. [Brockwell and Davis 2013] This kind of data can be originated from periodic collection, or from events with no fixed frequency, and examples of possible collection of this data can be widely found. For instance, monitoring systems for electrical grids [Carreras et al. 2004], finance [Shasha 1999] or meteorology [Brown et al. 1984].

It is also important to mention, the close relation between Time Series and the quickly growing domain of Internet of Things, as seen in Figure 1. In IoT, millions of Internet enabled devices produce billions of data points every day, which must be collected, stored, and analyzed, and are very often represented as time series data. [Cooper and James 2009]

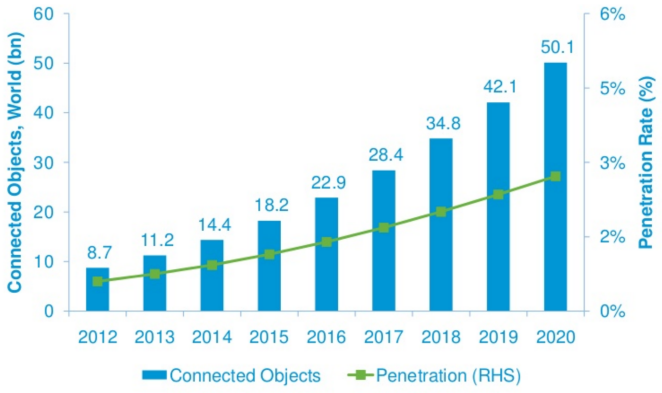


**Figura 1. Number of IoT Devices Expected Worldwide**

The main motivation for the collection of this data is analyzing it, thus allowing the generation of, for example, forecasting models for it. Applications of this can be easily identified from the examples mentioned earlier. For electrical grid monitoring, patterns of undesired conditions of the network can be predicted, while in meteorology, the likely climate conditions on a near future may be inferred. Besides this, many other benefits can be obtained from the analysis, such as anomaly detection, classification or clustering.

It is also important to note some characteristics which are intrinsic to time series and, thus, must be considered in any environment which handles it. Time series has a very high rate of data generation, which could accumulate over years, producing a very large aggregated dataset. This could be a very difficult problem to handle, were the relevancy of the data equal for all the collected points over the years, which it's not. The newly

collected measurements must be treated with higher relevancy, as they're usually the ones providing the input for short term forecasts, while the older data points, the historical data, are less frequently accessed, thus creating requirements for data storage and access, which differ from standard DBMS.

Another important characteristic that is taken into account is that time series data is indexed by its timestamp, which could be regularly spaced or not. This results in another range of possible optimizations which should be taken into account when storing and accessing this type of data.

Based on this information, the databases which specialize in handling time series data, have to differ from normal Database Management Systems (DBMS), in order to provide better performance, and be competitive in the market. As this project also relies on specialized time series databases, a short background on some of the ways these improvements take place will be presented next.

## 2.2. Time Series Databases

The large amount of data being collected constantly, and the restrictions created by an analysis which focuses its operations on the latest data available, creates more specific requirements, differing from what could be offered by general-purpose DBMS. These restrictions will be subdivided in different categories, to provide a better explanation.They are: the need for specialized storage optimization, access speed, and integrated analysis functions.

### 2.2.1. Storage

The first point which we will consider will be the efficient storage, as it immediately comes to mind when handling very large datasets. The standard DBMS solutions dont provide any special kind of compression for time series data, and as can be seen in [Pelkonen et al. 2015], the potential for it is very compelling. With better storage methods, the disk space storage is reduced, and, as in the case of the in-memory engines, the memory usage too.

Time Series also has as one of its main characteristics, as mentioned, the gradual reduction of the relevancy of data, as it grows older. This allows, in some cases, for periodic aggregation of historical data, greatly reducing the storage usage, and making the analysis and access to it easier.

And finally, there is also the difference between disk and in memory storage, the former being capable of handling much more data, when considering a single node case. This last situation is closely connected with the core of this project, which hopes to provide a hybrid solution minimizing shortcomings and maximizing the benefits of each solutions.

### 2.2.2. Access Speed

In this section, the relevancy of access speed in Time Series Databases (TSDB), which can be expected to be high, as the general use cases for time series require real-time analysis

and monitoring of large datasets, will be discussed. The first point mentioned, still in connection to storage, is that the most relevant data for time series is usually the most recent. By organizing the data according to this known characteristic, faster access can be provided for recent data, while still having the possibility of having reasonable access times for older data.

Another important requirement of time series is generated by the way data is queried from the tables, which even in many cases relies upon granularization queries (e.g. the data is stored using milliseconds resolution, but on certain queries the data is aggregated into larger time unites, with different possibilities of aggregating the data values in each of the new time units). This is similar to the aggregation that may be done for storage, discussed in the last paragraph, but in this case, the data is only aggregated for the query result.

Once again, there is a large difference between the two models of database presented, however, this time the in memory storage is the one providing better results. It provides much faster access speed than disk storage, due to the intrinsic differences between memory and disk access.

### 2.2.3. Analysis Functions

The last aspect, which will be presented are time series analysis functions, which consist of providing some integrated operations, useful to time series data, directly on the TSDB. This functions include correlation, histogram generation, exponential smoothing, and other statistical functions, for example. While these could all be obtained by querying the data and running the functions, this method provides a faster, and more economical execution of the most common functions used by these systems, as they are placed closer to the data, and may, for instance, take advantage of distributed techniques.

## 3. Related Work

A very wide list of proprietary and open-source TSDBs exists, many disk based and some in memory. Some benchmarks of the open-source options available can be found at [Acreman 2016]. These database provide the basic functionality required for time series data, as well as being expected to have a better performance than general purpose database, when handling this kind of data. They are, however, usually restricted to either being completely in memory, or completely disk based (dismissing possibly existing caching mechanisms). Therefore they are restricted by the shortcomings mentioned in previous sections, such as storage space, for memory, and performance for the disk.

There is one example which is more closely related to this project, and thus, worth mentioning in more detail. The recently developed, and released as open-source code, Gorilla database, by Facebook [Pelkonen et al. 2015]. The proposed architecture relies on a in-memory database acting as a front end for incoming time series data, and a underlying cluster of disk based storage. This provides fast access to the data which is stored on the first instance, which is in memory, while still providing long term storage on the disk. However, the provided architecture relies on two separate instances communicating, which results in a more complex process for the managing the data.

As can be seen, most of the solutions are either purely in-memory or disk based, which results in the already explored undesired issues. The Gorilla example, which manages to handle most of this issues, by integrating a clustered in-memory database, to an underlying cluster of disk storage, introduces a new issue, by having two completely independent structures communicating, and handling the same datasets.

## 4. Work Proposal

This work proposes the integration of a disk based storage system to an already existing in-memory database, but doing it in such a way that the disk storage is completely transparent to the querying and insertion process. This can be achieved through the integration of the disk storage on the node level of an existing in-memory time series database cluster, as will be detailed in this section.

This solution aims to solve the presented issues of using disk or memory only databases, while providing the simplest querying and insertion possible. The first part can be achieved through the integration of the different types of storage, while taking into account the intrinsic characteristics of time series data presented in the background section.

The proposed solution will first be presented, and later on the methods used to solve the existing issues will be presented. The basic architecture proposed can be seen on Figure 2

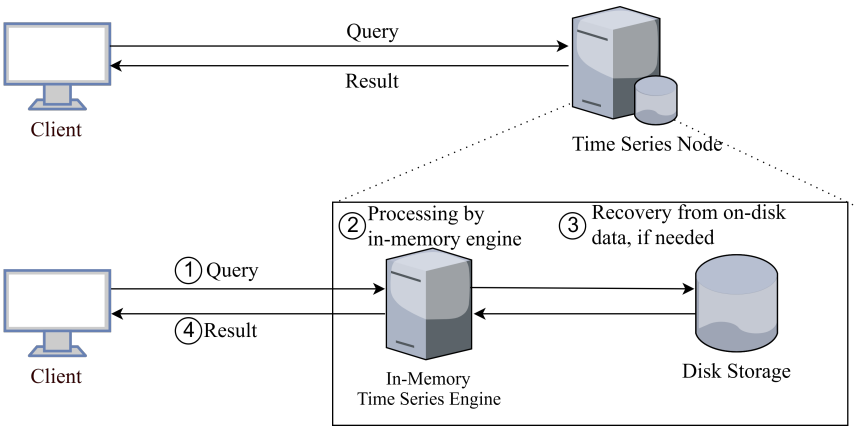


**Figura 2. Proposed Solution Architecture**

The image presents two different views of the same situation, which consists of a query being sent by a client to a single node an in-memory time series engine, and the result being returned. This engine is a generic pre existing in memory engine, which would then be modified, to add disk storage capabilities to it. The first view (on the top half of the figure), presents a completely external view of the process, therefore showing that the integration of a disk database would not modify the interfaces, and overall functionalities of the engine (i.e. it would be a transparent integration between the engine and the disk storage).

The second part of the image, shows the actual modifications which would take place, internally, on the engine node. It consists of two separate structures, the pre-existing

time series in-memory engine and the newly added disk storage system. Naturally, modifications to the in-memory engine would also need to be made, but would not interfere in any way with its previous functioning. The main idea of this architecture is to provide a transparent integration, in which the disk storage of data is a viable alternative, but without interfering or hindering in any with the existing memory functionalities.

The results obtained from this integration would, thus, try to provide the best features from the disk storage, minimizing its disadvantages and being completely transparent for querying. With the disk storage, the stored data in it would be persistent, in opposition to the memory stored measurements, and the limit for the node capacity would be set by the disk capacity, which is cheaper to increase. However the data stored on the disk would have a much slower access speed. A solution found for this was to keep only the more relevant data points in memory while the rest could then be automatically flushed to disk.

Given that time series would be the type of data stored, it is safe to assume that a good estimation for data relevancy is it's age, thus keeping the more recent data in-memory. Therefore, the performance for the more relevant datapoints (e.g. more recent data) would not be affected, and the performance for disk storage could be improved by making it possible for it to work closely together with the memory, with mechanisms such as pre-fetching and chunking. And finally, because more relevant data is more frequently accessed, impact on the average access speed would be minimal, though the worst case time would still correspond to disk access.

This architecture also provided freedom enough to have any kind of disk storage integrated to it, as all of the operations on the data could be, if needed, handled in-memory. However, in the case of time series, as there are specific disk storage databases for it already, some of the operations could be pushed down to the disk component of the architecture, reducing the load on the memory part, which would then receive pre-processed partial results from the disk. In the case of a general purpose DBMS, no support for the time series specific aggregations, analytical functions, and storage optimizations for time series would exist, thus, possibly causing an impact in performance. Therefore, preference for the more specialized databases should be given, when choosing a candidate, if possible.

## 5. Conclusion

As we have seen on the presentation made on this document, good performance and storage capacity are vital requirements for time series databases. Many options are available for such purpose, however, none of them take advantage properly of both the memory and disk environment for its storage mechanisms. Therefore, the work proposed on this document, tries to present a viable alternative for handling the interface between disk and memory storage for time series data, providing disk-like storage capacity, while still keeping the performance as close to in memory databases as possible.

## Referências

Acreman, S. (2016). Time-series database benchmarks.

Brockwell, P. J. and Davis, R. A. (2013). *Time series: theory and methods*. Springer Science & Business Media.

Brown, B. G., Katz, R. W., and Murphy, A. H. (1984). Time series models to simulate and forecast wind speed and wind power. *Journal of climate and applied meteorology*, 23(8):1184–1195.

Carreras, B. A., Newman, D. E., Dobson, I., and Poole, A. B. (2004). Evidence for self-organized criticality in a time series of electric power system blackouts. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 51(9):1733–1740.

Cooper, J. and James, A. (2009). Challenges for database management in the internet of things. *IETE Technical Review*, 26(5):320–329.

Index, C. G. C. (2016). Forecast and methodology, 2015-2020 white paper.

Pelkonen, T., Franklin, S., Teller, J., Cavallaro, P., Huang, Q., Meza, J., and Veeraraghavan, K. (2015). Gorilla: A fast, scalable, in-memory time series database. *Proc. VLDB Endow.*, 8(12):1816–1827.

Shasha, D. (1999). Tuning time series queries in finance: Case studies and recommendations. *IEEE Data Eng. Bull.*, 22(2):40–46.