Introduction to Time Series Database Systems

D. Shah, M. Stolberg, and C. Vaughn

***Abstract*—In recent years, there has been an ever-increasing number of network-attached devices entering industry and our everyday lives. Many of these devices act as sensors, generating a periodic stream of time series data, in a multitude of domains. The compounding effects of data growth as it relates to time series data has created some unique problems for practitioners and application developers. So much so, that specialized solutions to these problems are now required. No more can a simple relational database model effectuate a scalable solution given the amount of data required to store and process. In this paper, we present an overview of emerging, purpose-built, time series databases. Leading time series databases in the market today are specifically designed to handle write-heavy workloads and time-based aggregation. We intend to show what trade-offs are made in order to achieve high scalability time series throughput, while maintaining utility and usefulness.**

***Index Terms*— Time series, Time Series Database (TSDB), Internet of Things (IoT), Distributed Computing, Databases, Cloud Computing, NoSQL**

I. Introduction

With the continuing advancement of technology, we live in an age marked by accelerating generation of data [1]. Digitization of our economy, coupled with an expansion of sensors and telemetry-related data, has created a computing environment that requires specialized tooling to support. By some estimates, each living person will create on average 1.7MB of data each minute by 2020 [2]. Big Data has come to be known for its data-related four V's: Value, Volume, Variety, and Velocity [3]. On a singular basis, a data point arguably has little value. However, once the Volume of that data increases, coupled with high Velocity and high Variety, the Value perceived and achieved can increase markedly. Interestingly, with the recent emergence of IoT, Social Media, log data and other unstructured data sources, we have seen that unstructured data growth amounts to 80-90% of all data growth. This is 10-50 times the rate of structured data alone [3]. Handling data at these rates and volume has necessitated technological advances so that data can be processed, corrected, and analyzed. One of the many advances has been the formalization of Time Series Databases (TSDB) into a category of emerging database technologies, some of which that are leveraged by some of the largest companies in the world [4].

In this paper, we present an overview of the current state of the art for solving data management problems with Time Series Databases. This includes both definitions for TSDB terminology, as well as use cases for TSDBs. We will contrast TSDBs with other NoSQL technologies such as Cassandra, and we will explain how TSDBs contend with the continuously cumulating nature of data streams. Many systems over the recent years have been designed to handle time series data and have attempted to address the necessary performance and data retention issues time series data brings. We intend to survey some of these technologies and describe how TSDBs functionally handle these issues [5], [6], [7], [8], [9].

II. Time Series Definitions & Properties

When talking about time series it's important to baseline on terminology. Much of the industry is still triangulating to a common understanding and language for NoSQL technologies [10].

A. Definitions

Time Series Databases discussed in this paper represent a specific type of data repository for data that involves *time.* This warrants a definition simply to scope our discussion to what a time series is. For the purposes of our discussion we will define time series as the following:

- **Time Series** – An ordered sequence of values or measurements of a variable, at equal or inequal intervals of time.

Time series data has specific properties that make it valuable. To state it bluntly, ordering matters. Not only does ordering uniquely define the value of a time series measurement, but we can definitely state that if we were to change the ordering of a time series, the meaning of the data changes. Underlying patterns of a time series is where most of the latent value lies [11]. Every time series includes specifics properties of durations, values, and ordering that make it unique.

Generally, there are two types of time series; those that track the duration of events (Fig. 1), and those that track the duration between points or measurements (Fig. 2). Most TSDBs are aligned implicitly with the latter, although the former is feasible in most TSDBs where the need exists.

Southern Methodist University, August 15, 2019, Dallas, Texas

D. Shah is with the Southern Methodist University, Dallas, TX 75205 USA (e-mail: dhyans@smu.edu).

M. Stolberg is with the Southern Methodist University, Dallas, TX 75205 USA (e-mail: mstolberg@smu.edu).

C. Vaughn is with the Southern Methodist University, Dallas, TX 75205 USA (e-mail: cvaughn@smu.edu).

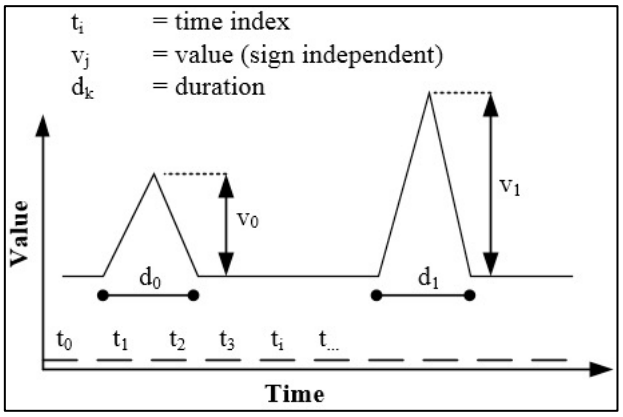

Fig. 1. Time Series - Event Features Tracking Duration

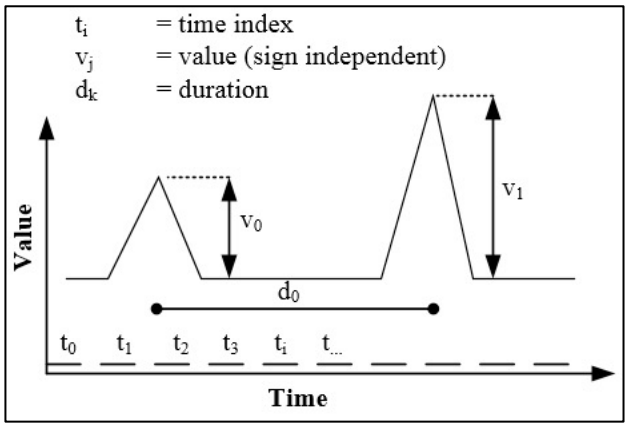

Fig. 2. Time Series - Point Features Tracking Duration

A Time Series Database (TSDB) is also worthy of definition. To that end, we can define a TSDB as follows:

- **Time Series Database** – A database type specifically optimized for time series and ordered, time-segregated data.

TSDBs allow specific capability to create, aggregate, update, destroy, and organize measurements over time. As mentioned, value for time series data is typically measured as a result of the accumulation of measurements over time, and the patterns found as a result. This is in contrast to the primary relational database value model of updating data, where much of the value is reaped from ACID-based updates, not necessarily change tracking over time [11].

B. Time Series Database Motivations

Numerous challenges present themselves with time series data management. Typically, these challenges only become major issues once a database becomes large and in high use [14], [15], [16], [17]. Often the underlying analysis being attempted is relatively uncomplicated, but the workload for aggregation and manipulation of the underlying large, time series dataset can't be reduced to suitable timeframes and resources. These data-intensive applications have a common set of problems:

1. The data volume, complexity, or velocity of acquisition is typically the biggest issue.
2. Users or applications utilizing the data require relatively fast data access and there is a propensity to minimize data movement due to the volume of data.
3. Operationally, there is a desire to scale over time, preferably linearly and predictably, for any future capital outlays to maintain the data and its growth (hardware and software) [18], [19].

Each of these problems are difficult to solve in their own right for Online Transaction Processing (OLTP) or Online Analytical Processing (OLAP) workloads. For time series workloads, specifically, the issues are amplified due to the potential volume and velocity dimensions of the data [3].

C. Time Series Database Uses

There are many scenarios where time series data is utilized. Time series data use varies based on industries and problem domain. The primary domains are related to IoT, monitoring of industrial process controls or IT systems, management of scientific data, and financial use cases [14]. It could be tracking the changes in stock price on an intraday or multi-day basis, the monitoring of temperature or pressure through weather telemetry equipment or smart home devices, tracking the fuel consumption of an engine, or describing the CPU load on a server. Any use case where a regression, or a second derivative and rate of change calculation might be useful, is a potential use for a TSDB (Table I). Time series can either be univariate or multivariate. And, in context, we can thus classify a predominant use into three categories.

- **Time Series Analysis** – Exploring how a variable change over time
- **Regression Analysis** – Exploring how changes in one or more variables cause shifts in other variables over time
- **Forecasting** – Predicting future values of variables based on historical values

Table I
TSDB Use Examples by Category

Type	Example
Time Series Analysis	Temperature or pressure tracking
Regression Analysis	Economic analysis
Forecasting	Weather or sales forecasting

Cardinality and resolution for a time series varies depending on the area a time series is applied, and the capability of the technology taking the measurements. In the case of tracking temperature or pressure, for example, 15-minute to daily intervals are common [2], [12] (Table II).

In other applications, a time series may have second, millisecond, or even nanosecond scale; nanosecond scale being needed to measure things like energy and quantum experiments [13].

Table II
Climatological Example Data - Asheville, NC January 2017

	Temperature (F)		
Day	Max	Min	Avg
01	54	41	48
02	54	45	50
03	63	47	55
04	54	30	42
05	44	25	34
06	35	25	30
07	25	13	19

D. Time Series Database Data Model Properties

Dreyer et al. [15] presented a time series data model outlining the structural elements of a fully formed time series. It is our view that this model still maintains validity today. The proposed model asserts that a time series contains:

- **Events** – A combination of time-variant events worth recording. This could something like a High Temperature on a sensor, or an opening share price. Event data are scalar typed such as string or integer.
- **Time Series** – Base or derived, chronologically ordered, sequences of event measurements. Normally, time series data of an event are all the same type and predominantly numeric, although this is not required.
- **Groups** – Logical partitions of time series into suites of separate views of data. This could be groups of customers, or groups of sensors, or sales by country. Groups are necessary, and groups can be grouped into other groups by enumeration or computation.
- **Time Series Bases** – The three preceding structural components (Events, Time Series, and Groups) constitute a logical unit to make up a time series base, the size of which depends on the number of time series, the data resolution for each time series, and event periodicity.

In addition to the original model suggested by Dreyer et al., we argue that data retention and down sampling procedures should also be included in the sizing assertion for Time Series Bases as they represent critical data management procedures for a TSDB in our view.

III. Time Series Database Architectural Notes

Time Series data is obviously not a new occurrence in research and industry, and because of this there are numerous examples of classic relational database technology serving time series data well [21], [22], [23], [24]. Simultaneously, there is an emerging and accelerating trend towards solving high volume and velocity time series related data problems by utilizing several types of NoSQL database [3], [4], [5], [6], [7], [8], [9], [11].

For a TSDB to operate well, however, certain base properties of the database must be met. These properties help to optimize the Volume, Velocity, Variety, and Value dimensional concerns of the database management system.

1. **Performant Range Queries** – Typically solved by co-locating series data on disk
2. **Data Management Capabilities**– Supporting compression, expungement, and down-sampling
3. **Scalability** – Data storage and transactional capacity
4. **Usability** – Interfacing for data viewing, manipulation and aggregation

A. Relational versus NoSQL

There are numerous reasons why someone might choose a relational database approach for time series. It could be a question of familiarity, sunk cost in a technology, an ease to execution, or an affinity for strong ACID properties to the data, to name a few. Relational databases can be successfully implemented to handle time series, given enough resources, and assuming the volume of data is well understood.

That being said, the reasons why NoSQL, and distributed computing as an approach, has accelerated is due to the volume of data that many of these problems engender. At scale, a time series database may have to accept 100's of millions of write per day, and store potentially trillions of datapoints [4]. At that scale, there must be a horizontal, distributed computing model to leverage. In that regard, NoSQL variants of databases trade the relational model and ACID properties for scalability.

B. Time series with NoSQL

Diack et al. [27] take great pains in refining the original CAP theorem [28] to better explain Consistency, Availability, and Partition Tolerance tradeoffs, and thus we do not cover that in this analysis and refer the reader to that work. However, these seminal tradeoffs cannot be overstated, and the reasons for a shift towards NoSQL for time series are that high scale, resilient TSDBs must now provide:

1. **Write Capacity** - Highly efficient, and scalable, write capacity
2. **Data Distribution** - In order to potentially provide both, read and write scalability, and availability across failure domains, data distribution must be efficient and eventually consistent
3. **Data Replication** - In order to provide data redundancy and persistence, replication capabilities must be reliable and non-blocking
4. **Query Processing** - Efficient query processing must be robust, and preferable optimized for aggregating across *Events* and *Time series*, as well as having the ability to utilize the aforementioned *Groups* as partitions

To contrast and argue the final point on why NoSQL has becoming popular in solving time series data problems, we can review some of the architectural differences between popular databases, both generic and time series specific [29]. From Table III we can see that for a high-volume system that potentially needs wide distribution and coordination across multiple locations, and therefore clusters, with the exception of

Prometheus, only NoSQL variant databases have the necessary capabilities to achieve high availability, partition tolerance, automated data partitioning, and cross cluster replication.

Table III
Technical Features of Select Databases

Data Model	DBMS	Design & CAP Emphasis	Data Partitioning	Replica Scope
RDBMS	MySQL PostGreSQL	Master-Slave CA	Manual	Cluster
NoSQL Document	MongoDB	Multi-Master CP	Hash Range	Cross Cluster
NoSQL Columnar	Cassandra	Multi-Master AP (typically)	Hash Range	Cross Cluster
NoSQL Time series	InfluxDB	Multi-Master AP or CP	Range	Cross Cluster
NoSQL Time series	Prometheus	Master-Slave	Manual	Cluster

Many of the TSDBs exist primarily as an offshoot of NoSQL concepts. In fact, many look very much like their non-time series specific NoSQL counterparts. InfluxDB for example, is architected very similarly to Cassandra in some ways. For example, InfluxDB utilizes a write-ahead log, a memory cache, and index files, just as Cassandra does. However, InfluxDB takes the added steps to swap out the log structured merge tree (LSM) storage engine that Cassandra leverages, with a time series merge tree (TSM) storage engine that pre-sorts all data by series and time and adds compression [11]. Jensen et al. [30] had experimental results which state InfluxDB took only 44% of the time to ingest test data as compared to Cassandra ingesting the same time series data, with a worst-case 64% better storage compression ratio. Having a time optimized storage engine, along with added capabilities for compression, replication, sharding, aggregations and query processing, helps a time series specific database like InfluxDB solve across all four dimensions of write capacity, distribution, replication, and query functionality.

IV. Survey of TSDBs

Several commercially available, and opensource, TSDBs exist as of this writing. And while it is beyond the scope of this paper to enumerate them all, it is beneficial to outline a few in order to give the reader a more holistic view of the technology alternatives.

As of this writing, the most popular and capable time series databases include Graphite, InfluxDB, Kdb+, OpenTSDB, Prometheus, RRDTool, and TimescaleDB [2], [31], [32], [34]. Each of these TSDBs have unique characteristics, but only a select few meet our desired TSDB property criteria of scalability. In fact, out of all the TSDBs analyzed, only InfluxDB and OpenTSDB meet a high, linear scalability, criteria. Said differently, only InfluxDB and OpenTSDB are capable of distributing information and workloads in such a way that there are not immediately apparent impediments to their growth, for a preponderance of cases conceptualized.

Table IV
Popular Time series Database Variants

Database	Data Partitioning	Replication	Access Methods	Linear Scale
Graphite	None	None	API	No
InfluxDB	Sharding	Customizable	API SQL-like	Yes
Kdb+	Sharding	Master-Slave	API SQL-like	No
OpenTSDB	Sharding	Hbase Based	API	Yes
Prometheus	Sharding	Federated	API	No
RRDTool	None	None	Shared Library	No
TimescaleDB	Sharding	Customizable	API Shared Library	No

A. *Graphite*

Graphite is a relatively simple TSDB, with a fixed size datastore. The fixed size of the database makes performance consistent and reliable over time, but also limits its usefulness as a multipurpose TSDB. Graphite has no capabilities for data distribution or replication and is not horizontally scalable.

B. *InfluxDB*

InfluxDB represents one of the most feature complete NoSQL TSDBs, with strong partition tolerance and availability guarantees. It is architected for high write capacity, distribution, replication, with a feature rich query processor, and no external dependencies. Retention Policy and Continuous Query capabilities also make down sampling data straight forward for managing ongoing data volume.

C. *Kdb+*

Kdb+ is the only closed-source, commercial solution in our list. This is likely due to Kdb+ being largely a financial industry solution, which is where it originated. Kdb+ provides strong ACID properties and is officially classified as a relational database, optimized for time series. Unfortunately, Kdb+ is cumbersome to scale as compared to other NoSQL databases, supporting only a Master-Slave model.

D. *OpenTSDB*

OpenTSDB runs on top of Hbase, and leverages HDFS for much of its distribution and replication capabilities. OpenTSDB is linearly scalable, only accessible through API, and is scalable and performant for write throughput.

E. Prometheus

Prometheus was written in Go in an effort to optimize it for code execution and speed. Prometheus is best known for its high dimension data model, and its system monitoring integration functionality. Scalability is cumbersome as its architected as a Master-Slave model and workloads cannot be automatically distributed without complex configuration.

F. RRDTool

RRDTool is another fixed size datastore. It was originally designed as a high-performance data logging tool to be integrated into applications. RRDTool's size limitations limits its usefulness, and it is not meant for high volume applications.

G. TimescaleDB

TimescaleDB is a PostgreSQL extension intended to optimize PostgreSQL for time series. It is fully ACID, SQL and JOIN compliant. TimescaleDB takes its consistency, availability, and partition tolerance model from PostgreSQL, as well as its replication and partitioning capabilities.

V. Other Common Solutions

Any discussion concerning managing time series data would be incomplete without a quick acknowledgement about how numerous implementations have solved for horizontally scalable time series data management with more general-purpose NoSQL databases. Both MongoDB and Cassandra have both been utilized for this purpose, for example [34].

As we mentioned in Table III, MongoDB is a document-based database. MongoDB provides both read and write scalability, although it is more favored to reads since master-nodes must confirm all writes and are therefore a bottleneck for writes. This potential bottleneck for writes makes MongoDB a less than ideal solution for high velocity time series applications. MongoDB also provides a schema-less architecture, Map Reduce features, and aggregation functions allowing for flexibility in development.

Similarly, Cassandra has also been utilized for time series database implementations. Cassandra is a columnar database, which also provides a schema-less architecture. Cassandra, however, employs a peer-to-peer, read/write anywhere, architecture for scaling that simplifies operations since every node in the cluster is the same, and potentially easily replaced if necessary. Cassandra excels at writes, which is why it is a common choice of technology [35]. However, Cassandra is also limited in its abilities to perform range scans of data, which for certain time series use cases is an important requirement to do well.

It is important to note that both MongoDB and Cassandra represent decent technology choices, for specific circumstances. They can represent good decisions for individuals searching for a highly scalable, multi-purpose, NoSQL solution to act as the core to a time series database. However, as of this writing, true TSDBs are more optimized, and have more time series specific feature sets, than their more generic counterparts. It is our expectation, if it has not already occurred, that TSDBs will become more desirable, specialized tools for managing time series at scale. True TSDBs are worth consideration for anyone with a high volume, high velocity, high variety time series data management problem.

VI. Lessons Learned

Through this paper and this analysis, we learned two key lessons.

- For a truly high scalability solution on time series data management problems, only a few options exist. Only Cassandra, MongoDB, InfluxDB, and OpenTSDB are architected in such a way that would allow near limitless scale on storage, read capacity, and write capacity.
- It's not enough to solve for write capacity and storage growth, you also have to plan for down sampling and data expungement. Most multi-purpose NoSQL databases have some form of data retention functionality for expunging data. However, down sampling your data is largely a task left for you to implement. This is especially poignant since Cassandra does not have broad support for aggregate functions. This is in contrast with a full featured TSDB like InfluxDB, which provides tools to schedule this maintenance and data hygiene.

VII. Conclusion

The age of accelerating data generation is requiring us to think about managing data in new ways, and this is especially true for time series data. We have reviewed not only a proposed data model for time series data that should guide TSDB development and technology choices, but also some of the architectural tenants utilized for TSDBs, as well as a few popular TSDBs that exist today.

Today's environment of data velocity and volume is forcing technology developers and researchers to begin to specialize technology along the dimensions of time. As a result, we are just now seeing an emerging technology landscape that is bringing us special purpose tools which can ingest, organized, replicate, store, and analyze time series data, like never before. Some of these technologies can do this providing availability and partition tolerance, in a highly distributed setting. And, other technologies can provide us consistency and availability in more centralized settings. These TSDBs are especially suited and built for the time series data model and can help us manage the enormity of data that we may acquire.

Appendix

An online video educational series for Time Series Databases is provided through the link below. This covers overviews of the market, the motivations for TSDB design and adoption, as well as a hands-on introduction to InfluxDB. We strongly recommend the reader review this video to learn more about the current state of Time Series Database technologies. https://bit.ly/2zaNAND

Acknowledgment

D. Shah, M. Stolberg, and C. Vaughn thank Dr. Daniel Engels, Professor of Practice, Office of the Provost, with Sothern Methodist University, for his stewardship and guidance

for this work.

REFERENCES

[1] Kambatla, K., Kollias, G., Kumar, V., & Grama, A. (2014). Trends in big data analytics. Journal of Parallel and Distributed Computing, 74(7), 2561-2573.

[2] Petre, I., Boncea, R., Radulescu, C. Z., Zamfiroiu, A., & Sandu, I. (2019). A Time series Database Analysis Based on a Multi-attribute Maturity Model. Studies in Informatics and *Control*, *28*(2), 177-188.

[3] Jin, W. (2019, June). Big Data Technology Progress And Development Trend. In Journal of Physics: Conference Series (Vol. 1237, No. 2, p. 022102). IOP Publishing.

[4] Pelkonen, T., Franklin, S., Teller, J., Cavallaro, P., Huang, Q., Meza, J., & Veeraraghavan, K. (2015). Gorilla: A fast, scalable, in-memory time series database. Proceedings of the VLDB Endowment, 8(12), 1816-1827.

[5] T. Oetiker, "RRDtool," [Online]. Available:http://oss.oetiker.ch/rrdtool/.

[6] C. Davis, "Graphite," Orbitz, [Online]. Available: https://graphiteapp.org/.

[7] B. Sigoure, "OpenTSDB - A distributed, scalable monitoring system," in OSCON, 2011.

[8] "Riak TS - NoSQL time series database," Basho Technologies, [Online]. Available: http://basho.com/products/riak-ts/.

[9] "InfluxData (InfluxDB) - Time series database monitoring analytics," [Online]. Available: https://www.influxdata.com/.

[10] Ramzan, S., Bajwa, I. S., & Kazmi, R. (2019). Challenges in NoSQL-Based Distributed Data Storage: A Systematic Literature Review. Electronics, 8(5), 488.

[11] S. Y. Syeda Noor Zehra Naqvi, "Time series databases and influxdb," Studienarbeit, Universite Libre de Bruxelles, 2017, URL: ´http://cs.ulb.ac.be/public/ media/teaching/influxdb 2017.pdf [retrieved:2018-08-14].

[12] Quick Links. (n.d.). Retrieved August 14, 2019, from https://www.ncdc.noaa.gov/data-access/quick-links#loc-clim

[13] Pujol Priego, L., & Wareham, J. D. (2019). Obsessed with time? White rabbit at CERN. White Rabbit At CERN (March 2019). ESADE Business School Research Paper, (272).

[14] Jensen, S. K., Pedersen, T. B., & Thomsen, C. (2017). Time series management systems: A survey. IEEE Transactions on Knowledge and Data Engineering, 29(11), 2581-2600.

[15] Dreyer, W., Dittrich, A. K., & Schmidt, D. (1994). Research perspectives for time series management systems. ACM SIGMOD Record, 23(1), 10-15.

[16] Przymus, P., & Kaczmarski, K. (2014). Time series queries processing with GPU support. In New Trends in Databases and Information Systems (pp. 53-60). Springer, Cham.

[17] Deri, L., S. Mainardi, and F. Fusco, tsdb: a compressed database for time series, in Proceedings of the 4th international conference on Traffic Monitoring and Analysis. 2012, Springer-Verlag: Vienna, Austria. p. 143-156.

[18] Wang, W., Xu, L., & Gupta, I. (2015, March). Scale Up vs. scale out in cloud storage and graph processing systems. In 2015 IEEE International Conference on Cloud Engineering (pp. 428-433). IEEE.

[19] Przymus, P., & Kaczmarski, K. (2014). Time series queries processing with GPU support. In New Trends in Databases and Information Systems (pp. 53-60). Springer, Cham.

[20] Han, J., M. Song, and J. Song, A Novel Solution of Distributed Memory NoSQL Database for Cloud Computing, in Proceedings of the 2011 10th IEEE/ACIS International Conference on Computer and Information Science. 2011, IEEE Computer Society. p. 351-355.

[21] Bonnet, P., Gehrke, J., & Seshadri, P. (2001, January). Towards sensor database systems. In International conference on mobile data management (pp. 3-14). Springer, Berlin, Heidelberg.

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

[23] Chen, Y., Dong, G., Han, J., Wah, B. W., & Wang, J. (2002, January). Multi-dimensional regression analysis of time series data streams. In VLDB'02: Proceedings of the 28th International Conference on Very Large Databases (pp. 323-334). Morgan Kaufmann.

[24] Goldring, R. D. (1997). U.S. Patent No. 5,603,024. Washington, DC: U.S. Patent and Trademark Office.

[25] Tahmassebpour, M. (2017). A new method for time series big data effective storage. Ieee Access, 5, 10694-10699.

[26] Dunning, T., & Friedman, E. (2015). Time series databases. New Ways to Store and Access data.

[27] Diack, B. W., Ndiaye, S., & Slimani, Y. (2013). CAP Theorem between Claims and Misunderstandings: What is to be Sacrificed. International Journal of Advanced Science and Technology, 56, 1-12.

[28] Simon, S. (2000). Brewer's cap theorem. CS341 Distributed Information Systems, University of Basel (HS2012).

[29] Mazumdar, S., Seybold, D., Kritikos, K., & Verginadis, Y. (2019, February 11). A survey on data storage and placement methodologies for Cloud-Big Data ecosystem. Retrieved August 14, 2019, from https://link.springer.com/article/10.1186/s40537-019-0178-3

[30] Jensen, S. K., Pedersen, T. B., & Thomsen, C. (2018). ModelarDB: modular model-based time series management with spark and cassandra. Proceedings of the VLDB Endowment, 11(11), 1688-1701.

[31] FAQ - OpenTSDB - A Distributed, Scalable Monitoring System. (n.d.). Retrieved August 15, 2019, from http://opentsdb.net/faq.html

[32] Introduction to Multi-Datacenter Replication. (2018, January 18). Retrieved August 15, 2019, from https://www.influxdata.com/blog/multiple-data-center-replication-influxdb/

[33] Engines Ranking. (n.d.). Retrieved August 15, 2019, from https://db-engines.com/en/ranking/time series dbms

[34] Wlodarczyk, T. W. (2012, December). Overview of time series storage and processing in a cloud environment. In 4th IEEE International Conference on Cloud Computing Technology and Science Proceedings (pp. 625-628). IEEE.

[35] Barata, M., & Bernardino, J. (2016). Cassandra's Performance and Scalability Evaluation. In DATA (pp. 127-134).

[36] Retrieval using standard aggregate functions. (n.d.). Retrieved August 15, 2019, from https://docs.datastax.com/