Fast and efficient operational time series storage: The missing link in dynamic software analysis

Florian Lautenschlager,[1] Andreas Kumlehn,[2]
Josef Adersberger,[1] and Michael Philippsen[2]

[1]QAware GmbH
Munich, Germany
{florian.lautenschlager|josef.adersberger}@qaware.de

[2]Friedrich-Alexander University Erlangen-Nürnberg (FAU)
Programming Systems Group, Erlangen, Germany
{andreas.kumlehn|michael.philippsen}@fau.de

ABSTRACT

Distributed applications, cloud systems, the Internet of Things, etc. are generating increasing amounts of operational data, such as CPU loads, thread states, memory consumptions, method runtimes, or logs. Many tools continuously collect and analyze such data that is best represented as time series. Typical analyses try to find and localize runtime incidents like outliers, leaks, or trend anomalies. However, these analyses need an efficient use of storage and a fast interactive query execution, that general purpose storage systems do not provide: neither storing operational time series data in general-purpose databases nor in conventional time series databases fulfills these requirements.

We present Chronix, a novel time series storage that is optimized for operational time series and that improves the link between storage and analysis in a dynamic software analysis toolchain. With Chronix a toolchain not only stores data 4–33 times faster and it takes 5–171 times less storage space than with other time series databases, it also executes queries in 15–74% and analyses in 25–74% of the time.

1. INTRODUCTION

In large scale distributed applications, dynamic software analysis is gaining importance. Such applications produce huge amounts of operational data like memory usage statistics, network transfer rates, method runtimes, etc. By means of dynamic software analysis [2] one has to collect, analyze and understand this data to keep these applications up and running.

Fig. 1 shows the three parts of a typical dynamic software analysis toolchain: (A) a collection framework, (B) a time series storage, and (C) an analysis framework. A lot has been done for (A) and (C); specialized tools for a fine-grained, low overhead collection and analysis of operational data have been developed [3, 7, 9, 12, 13, 16]. But existing toolchains often rely on mainstream solutions for (B). Neither general databases nor time series databases specialize and optimize for operational time series data. They use general storage techniques and do not exploit the following specific characteristics of operational time series data: (a) many dimensions are common, (b) there is a large range of expected values along each dimension, (c) the frequencies of time spans tend to vary a lot (point per micro-second, point every few hours), and (d) the individual, concrete value often is of little interest compared to the behaviour of the series of values.

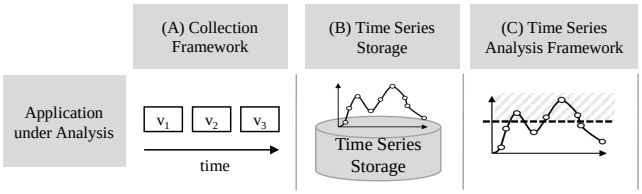

Figure 1: Typical toolchain for dynamic software analysis: (A) collect single data points, (B) store them as time series, (C) analyze the time series.

Chronix is a novel and special purpose time series storage that exploits the specific characteristics and needs of operational time series. Chronix achieves both an effective resource consumption and fast query times, even on large operational time series. It can be plugged into existing software analysis toolchains to speed them up. Its building blocks for efficient storing and querying are semantic compression, attributes (including pre-computed values) and chunking of operational time series data, a carefully selected compression technique, and a multi-dimensional storage.

Sec. 2 covers dynamic analysis toolchains and shows how Chronix fits into them. The architecture of Chronix is presented in Sec. 3. A quantitative comparison to other storage systems and related work follow in Sec. 4 before Sec. 5 concludes.

.

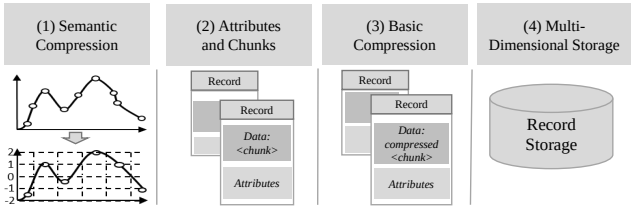

Figure 2: Building blocks of Chronix: (1) semantic compression, (2) chunk the time series data, (3) compress chunks, (4) store them with its attributes.

2. DYNAMIC SOFTWARE ANALYSIS TOOLCHAINS

A **Collection Framework**, (A) in Fig. 1, collects operational data. There is a trade-off between the collection overhead and the granularity of the harvested information. For example, the collection overhead for the global CPU load is very low, whereas method runtimes are more costly to track. Known collection frameworks use various techniques and vary with respect to the different sources of operation data, use cases, and domains [3, 9, 13].

The **Time Series Storage** (B) holds the collected operational data and often a query language support analysis queries. Since relational databases are known to be suboptimal for storing time series data [6] there are time series databases, a few of which are widely used [4, 5, 10]. All of them are built for storing *general* time series and hence they use general storage concepts like round-robin databases, generic key-value stores, or wide-column databases.

A **Time Series Analysis Framework** (C) helps to analyze and understand operational data. It asks the storage for data, analyzes it, and prepares the results for an evaluation. Many domains use time series analysis, each with specialized use-cases and specific analysis frameworks [7, 12, 16].

The known collection and analysis frameworks are well proven and tested in production systems and they are optimized and specialized on operational data. But up to now available toolchains make use of time series databases that are built for storing general time series data and that use general storage techniques. Since they do not exploit the specific characteristics and needs of operational time series this causes weak storage efficiency and slow query performance. The Chronix time series storage is the missing specific link between collection and analysis of operational data.

3. ARCHITECTURE OF CHRONIX

The key data type of Chronix is called a *record*. It stores a chunk of time series data in a compressed binary large object. The record also stores technical fields, time stamps for start and end, that describe the time range of the chunk of data, and a set of arbitrary user-defined attributes. Storing records instead of individual pairs of time stamp and value has two major advantages: (1) a reduced storage demand due to compression and (2) almost constant query times for accessing a chunk due to indexable attributes and a constant overhead for decompression.

The architecture of Chronix has the four building blocks shown in Fig. 2 that we discuss below. It is well-suited to the parallelism of multi-core systems. All blocks can work in parallel to each other to increase the throughput.

Semantic Compression is optional and reduces the amount of time series with the objective of storing fewer records. It uses techniques that exploit knowledge on the shape and the significance of a time series to remove irrelevant details even if some accuracy is lost, e.g. dimensionality reduction through aggregation, or more sophisticated approaches like Wavelets. Chronix uses the time stamp compaction of Shafer et al. [11], but can also be extended with arbitrary techniques.

Attributes and Chunks breaks down time series into chunks of `n` data points that are serialized into `c` Bytes. It also calculates the attributes and the pre-calculated values of the records. Part of this serialization is a novel *Date-Delta Compaction* that compares the deltas between time stamps. It serializes only the value if the aberration of two deltas is within a defined range, otherwise it writes both the time stamp and the value to the record's data field.

Then **Basic Compression** uses gzip, a lossless compression technique that operates on `c` consecutive bytes. Only the record's data field is compressed to reduce the storage demand while the attributes remain uncompressed for access. Compression of operational time series data yields a high compression rate due its value characteristics. In spite of the decompression costs when accessing data, compression actually improves query times as data is processed faster.

The **Multi-Dimensional Storage** holds the records in a compressed binary format. Only the fields that are necessary to locate the records are visible as so-called dimensions to the data storage system. Queries can then use any combination of those dimensions to locate records. Chronix uses Apache Solr [1] as it ideally matches the requirements. Furthermore Chronix has built-in analysis functions, e.g, a trend and outlier detector, to optimize operational time series analyses. These functions are used in several approaches that detect anomalies in time series data [12, 14, 15].

Chronix can be *tuned* on training data to find the best chunk size `c` and to pick the best compression technique. More detail on Chronix can be found in [8].

4. EVALUATION

This evaluation compares the write throughput, storage efficiency, and query performance of an example toolchain that employs Chronix or another time series storage (Graphite [4], InfluxDB [5], and OpenTSDB [10]). Graphite internally uses a round-robin database and encodes the attributes in a dot-separated name of the time series, InfluxDB uses a key-value store and stores attributes in key-value pairs per series, and OpenTSDB is based on a wide-column database and stores attributes in key-value pairs.

For the benchmark we store operational time series data of three real-world projects in each database (write throughput), measure the storage demand on disk, and execute 72 queries collected in real-world analyses. For the evaluation, we randomly retrieve $20 \cdot 72$ time ranges of size `r` (in days) from the databases to stabilize the results.

All measurements were conducted on a server (Intel Xeon E5-2650L v3 CPU, 16 GB RAM, 160 GB SSD, Ubuntu 14.04.2x64) and each database is configured as single node.

Write Throughput. For the tree projects, Table 1 shows the write throughput for storing the raw time series data into the databases, using their HTTP interfaces.

Table 1: Write throughput in MByte/seconds.

Project	CSV (MByte)	Graphite	InfluxDB	OpenTSDB	Chronix	Chronix (SC)
1	129	4.96	2.74	0.61	11.73	11.73
2	5,400	5.24	2.69	0.63	21.26	22.88
3	9,600	4.19	2.73	0.55	22.86	24.12
avg	5,043	4.80	2.72	0.59	18.61	19.58

Table 2: Storage demands in MBytes.

Project	CSV	Graphite	InfluxDB	OpenTSDB	Chronix	Chronix (SC)
1	129	758	144	254	4	2
2	5,400	14,000	11,000	263	226	74
3	9,600	26,000	21,000	758	632	162
total	15,129	40,758	32,144	1,275	862	238

With Chronix, the toolchain's write throughput is 4-33 times higher than with the other times series databases. The main reasons are different underlying storages and that the others lack compression or ship single data points (Graphite).

Storage Efficiency. Table 2 lists the storage demands of the data in the various time series databases, including index files.

Even without Semantic Compression (SC), Chronix achieves a compression rate of 94% and takes only 68% of the storage of the best alternative. With Semantic Compression enabled, Chronix can further lower the storage demands by almost a factor of 4. In total, by employing Chronix instead of a general purpose time series database a toolchain takes 5–171 times less space.

Query Performance. Table 3 shows in the upper part the query time of the $20 \cdot 72$ queries. For a query `q` of size `r` we do not pick a time series that is shorter than `r`.

With Chronix (especially with Semantic Compression (SC) engaged) the toolchain yields the fastest query performance and can save 26–85% of the total access times on the query mix, compared to a toolchain that employs one of the other time series databases.

The lower part of Table 3 shows the times for an outlier and a trend analysis using the inter quartile range and a linear regression. For these analyses, the toolchain can exploit Chronix' built-in query functions (e.g. `q=metric:myMetric &fq={!ANALYZE detect=trend}`) whereas a few basic queries need to be used instead when general purpose time series databases are used. This speeds up things by 26–75%.

Table 3: Query times for $20 \cdot 72$ queries in sec.

r	q	Graphite	InfluxDB	OpenTSDB	Chronix	Chronix (SC)
1	30	12.3	20.4	12.8	12.3	10.2
7	30	15.1	100.3	30.3	13.6	11.0
14	10	5.5	33.6	10.7	4.5	3.8
91	2	2.3	17.7	4.2	1.1	0.9
total		35.2	172.0	58.0	31.5	25.9
Outlier		48.6	95.1	66.5	41.2	36.3
Trend		41.3	175.3	62.7	35.7	30.1
total		89.9	270.4	129.2	76.9	66.4

5. CONCLUSION

Typical toolchains for dynamic software analysis suffer from the lack of a specialized time series storage. Chronix is such a novel storage that is targeted towards operational time series and clearly outperforms related time series databases in terms of storage demands and query runtimes. When plugged into toolchains Chronix can not only import data 4–33 times faster and reduce storage demands by a factor of 5–171. It can also save 26–85% of the query times and executes specific analyses 26–75% faster.

Chronix is open source. See `http://www.chronix.io/`. We also plan to publish the benchmark and the data set as open source. Currently, Chronix is a proof of concept but we are committed to create a production-ready release.

Acknowledgments

In part founded by the Bavarian Ministry of Economic Affairs and Media, Energy and Tech. (IUK-1303-0002, IUK430/001).

6. REFERENCES

[1] Apache Solr. Search engine. http://lucene.apache.org/solr/. [15. Aug. 2015].

[2] T. Ball. The concept of dynamic analysis. In *Softw. Eng. ESEC/FSE'99*, pages 216–234, Toulouse, France, 1999.

[3] collectd. System statistics collection daemon. https://collectd.org/. [10. Aug. 2015].

[4] C. Davis. Graphite. https://github.com/graphite-project/. [10. Aug. 2015].

[5] P. Dix, J. Shahid, and T. Persen. InfluxDB. http://influxdb.com/. [10. Aug. 2015].

[6] T. Dunning and E. Friedman. *Time Series Databases: New Ways to Store and Access Data*. O'Reilly, 2014.

[7] Etsy. Skyline. https://github.com/etsy/skyline/. [10. Aug. 2015].

[8] F. Lautenschlager, A. Kumlehn, J. Adersberger, and M. Philippsen. Chronix: Efficient Storage and Query of Operational Time Series. In *Proc. Intl. Conf. Softw. Anal., Evo., and Re-Eng.*, Osaka, Japan, 2016. [Submitted].

[9] Metrics. Metrics. http://metrics.dropwizard.io/3.1.0/. [10. Aug. 2015].

[10] OpenTSDB. The Time Series Database. http://opentsdb.net/. [10. August 2015].

[11] I. Shafer, R. Sambasivan, A. Rowe, and G. Ganger. Specialized Storage for Big Numeric Time Series. In *Proc. Conf. USENIX Hot Topics Storage & File Systems*, pages 1–5, San Jose, CA, 2013.

[12] O. Vallis, J. Hochenbaum, and A. Kejariwal. A Novel Technique for Long-Term Anomaly Detection in the Cloud. In *Proc. Conf. USENIX Hot Topics Cloud Comput.*, pages 15–15, San Diego, CA, 2014.

[13] A. van Hoorn, J. Waller, and W. Hasselbring. Kieker: A framework for application performance monitoring and dynamic software analysis. In *Proc. Intl. Conf. Perf. Eng. (ICPE'12)*, pages 247–248, Boston, MA, 2012.

[14] C. Wang, K. Viswanathan, L. Choudur, V. Talwar, W. Satterfield, and K. Schwan. Statistical Techniques for Online Anomaly Detection in Data Centers. In *Intl. Symp. Integrated Network Management (IM)*, pages 385–392, Dublin, Ireland, 2011.

[15] A. Wert, J. Happe, and L. Happe. Supporting Swift Reaction: Automatically Uncovering Performance Problems by Systematic Experiments. In *Proc. Intl. Conf. Software Engineering (ICSE)*, pages 552–561, San Francisco, CA, 2013.

[16] Yahoo. EGADS. https://github.com/yahoo/egads/. [10. Aug. 2015].