Streaming Software Analytics

Georgios Gousios
Radboud University Nijmegen
Nijmegen, the Netherlands
g.gousios@cs.ru.nl

Dominik Safaric
Radboud University Nijmegen
Nijmegen, the Netherlands
d.safaric@cs.ru.nl

Joost Visser
Radboud University Nijmegen
Nijmegen, the Netherlands
j.visser@cs.ru.nl

ABSTRACT

In this paper we present a novel software analytics infrastructure supporting for a combination of three requirements to serve software practitioners in utilising data-driven decision making: (1) Real-time insight: streaming software analytics unify static historical and current event-stream data enabling for immediate, nearly real-time insight into software quality, processes and users; (2) Query model: streaming software analytics substantiate for the lack of an integrated event-stream data extraction method by utilising a sophisticated, yet easy to use query Domain Specific Language; (3) Data summarisation: streaming software analytics allow for high level event-stream data summarisation in respect to distinct stakeholder groups. We expect that streaming software analytics will encourage software practitioners to move beyond information toward actionable insight, and enable for the use of analytics as a feedback and decision-support instrument, thus allowing them for an increase in quality of software systems, processes and delivery.

CCS Concepts

•**Applied computing** → **Digital libraries and archives;** •**Information systems** → *Data mining;* •**Human-centered computing** → *Open source software;*

Keywords

Software analytics, streaming, data mining

1. INTRODUCTION AND MOTIVATION

Software engineering is an exceedingly data abundant activity. Nearly all artifacts of a software development project, both static, such as source code, software repositories and issues and dynamic, such as run-time logs and user activity streams, contain information valuable for understanding and optimising both the software products and the processes that created them. Unfortunately, modern organisations often do not utilise this wealth of information as a feedback and decision support instrument. Rather, teams adopt new software development practices and employ the latest and greatest technology, without questioning the results of their decisions. Consequently, this leads to decisions that are often unnecessarily suboptimal or downright wrong [10].

Software analytics aim at utilising data-driven approaches to enable software practitioners to perform data exploration and analysis in order to obtain *insightful* and *actionable* information for completing various tasks around software systems, software users, and software development process [13]. Recently, the field of software analytics has been witnessing a renaissance: driven from practical needs (e.g. competition, fast iteration) large software development organizations and researchers alike, investigate how to extract, process and present analytics to developers and teams.

Despite the recent surge, what we believe is a key aspect of software analytics is neglected in current research: *immediacy.* Software analytics efforts rarely consume real-time data, but rather static historical data stored within pre-populated databases and thus the obtained analysis are usually post-hoc instead of. Model building based on static has proven to be a valuable tool for various tasks, but the generated models feature strong locality properties that forbids them to work across projects. Continuous and localized learning can make models adapt better to local properties. Furthermore, the move towards more flexible release and deployment techniques (usually referred to as devops) makes processing runtime information and combining it with historical data very important for fast iteration.

With this work, we anticipate a surge of demand for both archival and real-time software analytics that cut across production/runtime layers in order to optimise delivery, performance and software quality. For this reason, we propose the unification of archival and current software-related information using data *streams* as the fundamental data access and computation abstraction. Streaming software analytics allow for high-level aggregation and summarisation of near real-time information.

2. STREAMING SOFTWARE ANALYTICS

Despite the existence of tools and methods for extracting data from software development processes, products and ecosystems, three key aspects are missing: integration, composition and real-time operation. To compensate for these deficiencies, we aim at providing the tools and methods to collect, query, aggregate, and summarise software analytics data as stream, enabling software practitioners for the use of analytics as a core feedback loop. In the context of soft-

BIGDSE'16, May 16 2016, Austin, TX, USA
© 2016 Copyright held by the owner/author(s). Publication rights licensed to ACM. ISBN 978-1-4503-4152-3/16/05…$15.00
DOI: http://dx.doi.org/10.1145/2896825.2896832

ware engineering, we expect that streaming software analytics will enable software practitioners to move beyond information toward actionable insight, hence allowing them for an increase of software process and system technical quality.

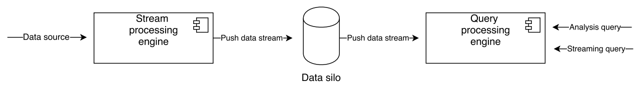

Figure 1: The architectural decomposition of the streaming software analytics platform.

To support for these requirements, we define an architecture of the streaming software analytics through the following three high level architectural components: the data silo, the streaming and querying engine, and the aggregation and summarisation framework. Figure 1 illustrates the architectural decomposition of the system, while subsequent sections of the paper provide a brief overview of each one.

2.1 Data siloing

Modern software development projects are more than just the source code that comprises them:

- Software projects utilize a variety of tools for managing versions, issues, builds, releases and deployments.
- More advanced projects also employ static analysis tools to analyze their structure and enforce quality standards.
- After deployment, operation teams track performance and quality metrics, along with operational logs.
- External services exist to inform teams about various topics relating to external system dependencies ranging from security disclosures to community health for language ecosystems.
- Development teams collect process information to help them analyze their performance during development sprints and prioritize their work.
- Marketing and community management teams advertise software products on social media channels and respond to incidents that may harm the products (or the organizations) credibility.

Currently, this wealth of data and metadata about software development is maintained in isolation and dedicated human analysts are required to arbitrarily extract, load, filter and combine the information in order to make sense of it.

What is needed is an infrastructure that aggregates all available information in a centralized location. All possible data, originating in software analysis, software operation and software development processes, needs to be stored in an extensible meta-storage schema that does not abstract over the aggregated data. To enable traceability and data navigation, it is important to cross-link data entries; we believe that one-way parent-child relationships, where applicable, can be sufficient for this purpose. Siloing does not (only) mean warehousing; to enable our vision of streaming software analytics, data must be accessed in the same way, irrespective of whether they are archived or current; consequently, our data representation abstractions should be streams rather than relations. This novel approach of transforming archived data into a stream data representation abstraction enables the simultaneous analysis of the past and the present.

To facilitate data aggregations in a streaming fashion, it is important to compensate for the shortage of mechanisms that allow pushing data to the data silo. To compensate for this, the data silo converts pull-based data sources to push-based ones using purpose specific data sensors that trigger when specific events in the originating data source occur. For instance, a data sensor monitors the project's application store entry and updates the corresponding silo collection with new commits, runtime logs or ratings.

2.2 Stream processing engine

At the core of the streaming software analytics platform is the stream processing engine. The fundamental data representation the stream processing engine works with is a stream, which designates an unbounded length data structure that can push updates to subscribers using the Observer pattern. Operations on streams of data are executed when new data become available, and can result in: (1) new data streams by executing transformation operations such as *join*, (2) single results by evaluating the content of a stream using pre-defined aggregation functions. Importantly, streams can also represent static (e.g. database) data, thus effectively unify processing of both static and dynamic data.

The streaming engine accepts data from incoming data sources, stores them in the data silo and then invokes a series of transformations, depending on the type of the data. On the user facing side, it exposes an API that allows users to invoke the query engine. The transformations are user or system defined functions that extract facts from the incoming data by arbitrarily combining one or more streams with archived data using a custom query language (described below).

While the stream processing engine is the core part of the proposed architecture, the task of creating and testing a stream processing engine from scratch is non trivial. Fortunately, technologies such as Spark Streaming [12], Trill [4] and Reactive Extensions [9] can readily play a central role in our proposed solution. While streaming can be efficient memory-wise, the sheer volume of the data to be processed calls for efficient processing. As computations on streaming systems are usually expressed as idempotent transformations over individual data items, distributing the processing on a cluster of machines can help with performance.

2.3 Query DSL

To extract insightful and actionable information from software analytics, individuals need a broad and growing set of tools. They manage the structure of the data storage, extract it into analysis tools, write ad hoc scripts for data transformation [5]. Because of this, extracting information from software analytics is a tedious and often time consuming task. To compensate for these deficiencies, the industry and research community proposed numerous query engines [9,12] that allow for the extraction of information from data streams. These tools however comprise two significant deficiencies: (i) difficulty of use [5], and (ii) the inability of querying over historical data making them domain agnostic.

To enable ease of use, we aim at extending the capabilities of an existing querying engine by designing a query

Domain Specific Language (DSL). The query DSL will not however directly manipulate streams of data, but rather is an abstraction over the existing querying mechanisms of a querying engine. For the transformation of data streams, the language defines three types of queries. Pull queries, enable for querying over streams of historical data. Push queries, allow for transformations over streams of current data. Lastly, push-pull queries enable for transformations over streams of both historical and current data. Figure 2 illustrates an example of a push query as processed by the querying engine.

```
Observable.from(stacktraces).flatMap{st =>
    commits.find(commit =>
        commit.diffs.find(diff =>
            diff.file == st.file && diff.line == st.line
        ).nonEmpty
    ) match {
        case Some(s) =>
            Observable.just((st.file, s.sha, s.developer
                ))
        case None =>
            Observable.error(...)
    }
}
```

Figure 2: Query combining live stack traces and commit data to identify the developer that created the commit that is causing a stack trace. Returns a combined stream of developer/stack trace information that can be further processed.

To enable for querying over historical data using the query DSL, we intent to construct an extensible querying mechanism that lifts domain objects to the query data pool, and allows both historical data and streaming event data to be queried seamlessly. But, constructing this mechanism constitutes a certain challenge. Data storage systems and querying engines prioritise current data over historical instances. Because of this, the mechanism should assign priority to a certain data source depending on the context of the query, using semantic analysis and state of the art machine learning algorithms.

2.4 Metrics aggregation and summarization

The streaming software analytics platform, when applied on real projects, will generate vast amounts of real-time data; to produce actionable software analytics out of fast moving data streams, the system needs to support the aggregation of streaming data. The aggregation and summarization component encapsulates and executes methods for high-level data aggregation and produces summaries of event-stream data. Figure 3 illustrates an example of a summary output produced by the aggregation and component.

```
Version 1.2.1 (commit e243434229c) of app Foo is
receiving negative feedback (sentiment ratio:2.7%) on
app store. Users complaining about frequent crashes.

Top exceptions in app crash log: NullPointerException
(88%), increased 95%, in version 1.2.1.

Static analysis on commit e243434229c indicates a
function measuring Cyclomatic Complexity above t 15.

Commit e243434229c is 85% bigger than average.
Code review passed with not feedback.
```

Figure 3: Information summarisation example for a mobile application.

For the event stream data to be summarised, we employ the concept of cascading aggregations. First, a set of metrics in conjunction with metric violation thresholds are specified by the user. Next, the user specifies a set of aggregation functions by consolidating pre-defined, and user-defined aggregation functions using the query DSL. In summary, the aggregation functions extract relevant pieces of information from streams of historical and current data. Finally, insight is produced by associating metric violation thresholds to the results of the evaluated aggregation functions. The cascading aggregation concept can be readily applied on static data; its application on streaming data poses additional research challenges.

Initially, the type of required insightful and actionable information depends upon: (i) software development project domain, (ii) development role of an individual [2]. For illustrative purposes consider a server application development project that may require information related to e.g. load monitoring, while a team developing a mobile application may find more interest into e.g. user visible crashes. Hence, a qualitative research for eliciting these diverse requirements should be performed. Secondly, the aggregation system is intended to be designed as a collection of composable stream-based functions. However, what are the building blocks of such as system? How can we effectively aggregate composed streams of data?

3. A RESEARCH AGENDA

With the proposed research, we aim at utilising the wealth of information produced by distinct software development artifacts in order to enable software practitioners to use software analytics as a feedback and decision support instrument. Realising streaming software analytics, therefore requires an understanding of distinct stakeholder information needs, and decisions they influence. But more importantly, we strive to understand how those needs map to software analyses. We believe that community co-ordinated efforts can help realizing the streaming software analytics vision in at least the following ways:

Requirements for analytics Researchers need to identify the stackeholders' information needs by means of qualitative research. Early works by Buse and Zimmerman [3] and Begel and Zimmerman [2] are on this direction, but need to be revisited in the light of real-time analytics and instant feedback.

Infrastructure work Developing analytics pipelines is a complex and error prone task. It is however an area of intense competition (both academic and industrial) and there is ample room for improvement. Software engineering researchers should work together with researchers in other fields (e.g. databases, programming languages) to tailor existing analytics systems to software engineering requirements.

Feedback-driven software engineering Software engineering development methods (e.g. Scrum) are currently based on much folklore and little evidence. As researchers, we should aim to enable feedback loops within the software engineering practice. Developers should be able to easily execute experiments (e.g. A/B tests), collect and correlate the results of applying specific design and development decisions and their outcomes. This way, teams and organizations can organically evolve their tooling and optimize their processes based on data-driven decisions rather than black-box methodologies.

4. RELATED WORK

Data streaming refers to the process of dealing with data points in unbounded length data structures. If we know the size of collection before an algorithm is applied to it, there are several optimizations that can be performed (e.g. reordering). In addition, many mathematical/statistical concepts (e.g. mean) assume bounded-length datasets. As a result, algorithms that are consider standard in every day programming (e.g. `quicksort` or quantiles) do not apply to stream processing [6].

Recent advances in computer processing power coupled with increasing needs for fast data processing, led to a renaissance of the field of stream processing. *Data stores*, both research [8] and commercial ones (e.g. MongoDB and PipelineDB), already offer the ability to expose updates in the stored data as streams and organize data processing in Single Instruction Multiple Data operations. Dedicated *streaming engines*, such as STREAM [1], Apache Spark Streaming [12], Trill [4] and Storm[1] facilitate the application of high level data transformations (usually based on list processing primitives stemming from functional programming) in a distributed network of machines. *Programming models*, such as functional reactive programming [11] and reactive extensions [9] make streams a first class citizen in application code, hiding the complications of asynchronous and concurrent execution behind data storage (e.g. collections) and computation (e.g. higher-order functions) primitives familiar to programmers.

So far the research community has investigated a number of software analytics infrastructures. Han et al. proposed StackMine [7], an infrastructure aimed at streaming events containing call stack signatures for the identification of impactful performance bugs. Fisher et al. [5] investigated onto an infrastructure for providing an integrated and collaborative environment for both real-time and archived data exploration analysis.

Streaming analytics are already being used in a variety of application scenaria, such as advertising, online fraud detection, user behaviour monitoring, sensor network data processing etc. To the best of our knowledge, there is no application of streaming analytics in the field of software engineering.

5. CONCLUSION

In this paper we introduced the concept of streaming software analytics. Streaming software analytics aim at unifying the representation of historical and current data as streams, and enable high-level aggregation and summarisation of near real-time information using a common query DSL. We expect that streaming software analytics will enable the use of analytics as a feedback and decision support instrument, thus increasing the quality of software systems and accelerating their delivery.

6. ACKNOWLEDGMENTS

This paper is a part of a research project funded by The Netherlands Organisation for Scientific Research (NWO)[2], project acronym CodeFeed.

[1]http://storm.apache.org
[2]http://www.nwo.nl/en/

7. REFERENCES

[1] BABCOCK, B., BABU, S., DATAR, M., MOTWANI, R., AND WIDOM, J. Models and issues in data stream systems. In *Proceedings of the Twenty-first ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems* (New York, NY, USA, 2002), PODS '02, ACM, pp. 1–16.

[2] BEGEL, A., AND ZIMMERMANN, T. Analyze this! 145 questions for data scientists in software engineering. Tech. Rep. MSR-TR-2013-111, October 2013.

[3] BUSE, R. P. L., AND ZIMMERMANN, T. Information needs for software development analytics. In *Proceedings of the 34th International Conference on Software Engineering* (Piscataway, NJ, USA, 2012), ICSE '12, IEEE Press, pp. 987–996.

[4] CHANDRAMOULI, B., GOLDSTEIN, J., BARNETT, M., DELINE, R., FISHER, D., PLATT, J. C., TERWILLIGER, J. F., AND WERNSING, J. Trill: A high-performance incremental query processor for diverse analytics. VLDB Very Large Data Bases.

[5] FISHER, D., CHANDRAMOULI, B., DELINE, R., GOLDSTEIN, J., ARON, A., BARNETT, M., PLATT, J. C., TERWILLIGER, J. F., AND WERNSING, J. Tempe: An interactive data science environment for exploration of temporal and streaming data. Tech. Rep. MSR-TR-2014-148, November 2014.

[6] GABER, M. M., ZASLAVSKY, A., AND KRISHNASWAMY, S. Mining data streams: A review. *SIGMOD Rec. 34*, 2 (June 2005), 18–26.

[7] HAN, S., DANG, Y., GE, S., ZHANG, D., AND XIE, T. Performance debugging in the large via mining millions of stack traces. In *Proceedings of the 34th International Conference on Software Engineering* (Piscataway, NJ, USA, 2012), ICSE '12, IEEE Press, pp. 145–155.

[8] IDREOS, S., GROFFEN, F., NES, N., MANEGOLD, S., MULLENDER, K. S., AND KERSTEN, M. L. Monetdb: Two decades of research in column-oriented database architectures. *IEEE Data Engineering Bulletin 35*, 1 (2012), 40–45.

[9] MEIJER, E. Reactive extensions (rx): Curing your asynchronous programming blues. In *ACM SIGPLAN Commercial Users of Functional Programming* (New York, NY, USA, 2010), CUFP '10, ACM, pp. 1–1.

[10] STRIGINI, L. Limiting the dangers of intuitive decision making. *IEEE Softw. 13*, 1 (1996), 101–103.

[11] WAN, Z., AND HUDAK, P. Functional reactive programming from first principles. *SIGPLAN Not. 35*, 5 (May 2000), 242–252.

[12] ZAHARIA, M., DAS, T., LI, H., HUNTER, T., SHENKER, S., AND STOICA, I. Discretized streams: Fault-tolerant streaming computation at scale. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles* (New York, NY, USA, 2013), SOSP '13, ACM, pp. 423–438.

[13] ZHANG, D., HAN, S., DANG, Y., LOU, J.-G., ZHANG, H., AND XIE, T. Software analytics in practice. *IEEE Software, Special Issue on the Many Faces of Software Analytics 30*, 5 (2013), 30–37.