Practical Runtime Verification of Performance in event based systems using Metric Temporal Logic

George Fakidis

January 12, 2025

Abstract

Event Based Systems(EBS) are the foundation of modern cloud system infrastructure. Event Driven Architecture(EDA) is a scalable architectural pattern that enables operations at an unprecedented scale. Performance considerations are of utmost importance to ensure cost efficiency and achieving QoS guarantees. In this paper we formalise performance [profiling] of events, in a manner that allows us to reuse existing methodologies and algorithms. We show that performance monitoring of series of events can be reduced to a problem of checking the temporal constraints of the events under study using (event logs). We also create a preliminary tool to integrate this runtime verification into existing technology stacks.

1 Introduction

Will be written last

Software Systems have grown considerably in the last years, and are now used in critical sectors like 1,2,3. Modern large scale systems serve millions of users and execute billions of operations per second. To enable computation at this unprecedented scale, sophisticated scalable architectures are required. One of the most popular and widely adopted is Event-Driven Architecture that centers its attention to messages sent between different systems in order to complete the desired computations. Therefore it is necessary to reason about the performance and correctness software and systems in a manner that can provide guarantees and be relied upon. Event-based systems are complex due to the flexible,unspecified and asynchronous interactions they enable and thus require expressive specification techniques that can be checked at runtime.

[This will be written from Wikipedia and mb a couple foundational papers] Formal Methods are a set of mathematical tools that enable us to reason about software in a complete, sound and verifiable manner using specifications that can be checked rigorously. Formal Methods that pursue completeness can sometimes be unapplicable in certain cases such as 1,2,3. Runtime verification uses data generated by a running system(traces) in order to detect violations of expressed properties and tries to provide guarantees for the system being studied.

A bit more detail of what event-based systems are, how do they look in practice(w. diagrams and a simple example).

Performance in Event-Based Systems can be more precisely expressed as:

- Throughput
- Latency

A brief explanation of Metric Temporal Logic(MTL) and some visual examples and the intuition behind its operators. The paper [Distributed Runtime Verification of metric temporal properties] has a nice paragraph on this, I can write something similar with extra shapes to indicate the intuition.

2 Related Work

Will be written after i do the core of the work below

2.1 Temporal Logic in Event-Based Systems

2.2 Performance in Event-Based Systems

- How is Performance Measured in Event-Based/Event-Driven Systems?
- How are QoS guarantees made in such systems (at a model level)?
- What formal Methods are used in such tasks?

2.3 Quantitative Formal Methods for Event-Based Systems

Formal Methods used in other works.

- UPAAL
- PRISM
- Markov Chains (of Discrete or Continuous time)
- Timed Temporal Logics (TTL $\rightarrow$ TCTL,MTL)
- Probabilistic Temporal Logics (PCTL, PLTL)
- Petri nets

3 Formalising Event-Based Systems

We borrow formalisations for Workflows,"Enactments",Activities and Events from the work of [Early Detection of temporal constraint violations]. Briefly explained, Workflows are an abstract series of abstract events/activities, that have a *start* event and possibly multiple *end* events. (In a software engineering context, workflows are similar to). Activities are the lowest level building block, denoting a unit of work in workflow. Events are generated by the execution of an activity. *Enactments* are the result of a concrete execution of workflow. In order to be able to distinguish between different executions of the same workflow, enactments also include an ID. This ID is included in events generated by activities execution in an enactment. In practice, this ID could be a user's ID that is carried throughout the workflow, thus allowing us to specify and monitor performance properties for the specific sequence of events while the model remains abstract and generic.

Using the definitions explained above, performance in this paper is regarded as the difference between event timestamps in order to keep the model simple. In future work, that difference between event timestamps could be analysed in a more granular level,for example dividing the time required for responding to an event into network latency, database latency and processing latency.

4 Formal properties in MTL

From the non my work stuff this will come first as it is important

Metric Temporal Logic can be used as a specification language to specify desired properties of an event-based system. When attempting to specify formal properties in a non-trivial software system it is important to understand *Safety* and *Liveness* properties[Lamport, On proving the correctness of multiprocess (concurrent) programs], a distinction practically useful. Safety is about checking whether *bad things* happen, it is about prohibiting invalid behaviour, thus only allowing for safe states throughout execution.

Liveness is about checking *good things* that must happen during an execution. It is important to note that the thing we check can involve an infinite number of steps.

Generally it is more difficult to specify and check liveness properties than safety properties[citation needed].

In the context of an event-based system we provide some indicative properties we would want to check. This will be made into a table that specifies what we want to check and the related MTL formula.

- Event A should always be followed by Event B within a certain time.
- Event A should be responded with Event B or Event C within a certain time(useful to specify where failures can happen and how).
- Event A should happen periodically.

- Event chains, e.g. Event A should be followed by Event B which should then be followed by Event C or Event D. This is especially useful when checking that execution of workflows, as defined above, conform to performance standards.
- For integrity reasons an Event B should not happen if an Event A has not come before that.
- Event A and Event B should not happen until another event happens.

In a practical scenario, the above properties will represent policy that must be adhered to by the implemented software.

I will write some simple things we usually test in such systems.

5 Integration of verification process into running EBS

This section will probably be done after everything as it is a bit optional now that i think about it

6 Software Solution Discussion

This will be written first as it is the core of my work

6.1 Functional Requirements

6.2 Design Goals

6.3 Proposed Architecture

7 Proof of Concept Demonstration

This should be around 2-3 days of intense focused work, I will go TDD in order to have the infra ready to produce the results

First will come a couple lines of a few implementation details. I will make it in python, where each service is a Kafka Consumer/Producer and I will have an API that I will run where the client will submit his request. The way i will demonstrate that my system works is the following, each request being processed will have a small percentage of ”failure”, in our case a thread.sleep() command that makes our system slower than usual and thus fail the runtime verification test. What we want to observe is that our system can catch these scenarios.

7.1 Case Study

This mostly revolves around writing the case study and doing the graphic

This is where i will present the case study that i will then run as an example. I will attempt to make my case study more complex in order to encapsulate more types of temporal properties demonstrated. Using my software to do the verification. I will probably do an integrated e-commerce system: User clicks pay, payment event is emitted, payment event does the processing and emits payment completed or payment failed due to lack of funds.

The payment completed event is consumed by a notification system that sends an email to the user, and by a supply chain software that updates the inventory and emits an inventory update event.

The payment completed event is also consumed by the shipment delivery system that adds a new shipment to be done, emitting a NewShipment event that within some time also emits a ShipmentScheduled event that notifies the user when the package ordered will arrive. Finally when the user receives the package, the person reponsible for its delivery submits a Shipment Completed Event.

Write those in metric temporal logic.

The diamond is a bit small for some reason $\diamond$ $\Box$ $\triangle$ Things I want to check:

1. After the user paying, a payment completed or a payment failed event arrives in X time
2. Succession of events in a timely manner(in the happy path and the simple failure path)
3. If a user clicks twice for the same payment that only one payment completed event is fired and user was not doubly charged(this is more of a logic thing than a performance thing).
4. No items should arrive without a Shipment Scheduled event before them (ensuring process efficiency and integrity).
5. If the user specified same day delivery, the Shipment Completed Event should be within 24 hours of the payment completed event.

I will make a simple diagram (as a pdf), that i will include here, to showcase the different services, and the events through which they communicate.

7.2 Results

This is where i will put the results. Results will be of the form: Table of Scenarios, Detected by our program or not.

8 Future Work

- Responding Online(Autoscaling?) to violations of metric temporal properties

- Synchronising Automatic Test Generation and Runtime Verification Monitor Generation from metric temporal properties in order to integrate performance checking in both development and production environments.
- Generating Metric Temporal Properties from event logs using process mining.

Something should go in there

A

Appendix A