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

George Fakidis[1]

[1]Athens University of Economics and Business

February 13, 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 demonstrate a formal approach of performance monitoring in an EBS, in a manner that allows us to reuse existing methodologies and algorithms, using a combination of Metric Temporal Logic(MTL) and EBS formalisations. We show that performance monitoring of a series of events can be solved by checking the temporal constraints of the events under study using metric temporal logic. We also design a preliminary tool to integrate this runtime verification into existing technology stacks, which will be demonstrated in future work.

1 Introduction

Software Systems have grown considerably in the last years, and are now used in critical sectors like healthcare,transportation and financial systems. 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(EDA) 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.

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. Runtime verification uses data generated by a running system(event logs,execution traces) in order to detect violations of expressed properties and monitors the system being studied at runtime.

2 Metric Temporal Logic

In this section we give a brief and practical explanation of Metric Temporal Logic, for a more mathematically rigorous presentation of the concepts presented here the reader is referred to [1, 2, 3, 4].

Linear Temporal Logic(LTL) [4] allows us to reason about propositions' truth values across a linear timeline. MTL [5] is an extension of LTL that enhances temporal operators with timing constraints. The

temporal operators are (U)ntil, (R)elease, (N)ext, (F)uture and (G)lobally. This enables us to reason about performance in a concise and mathematically rigorous manner, taking advantage of existing monitoring algorithms for MTL specifications.

To understand MTL, we start with a finite set of propositional variables, denoted by AP. At each point in time there is a state S, that carries the truth values of all AP propositions. MTL formulas are then checked against a timed *trace*, where trace is a (possibly infinte) sequence of states in S. A *timed trace* is a sequence of tuple elements in $S \times T$, where T is a monotonically increasing sequence of timestamps. We can then evaluate MTL specifications on timed traces. MTL provides us a way to concisely specify performance expectations of the system being monitored, in a generic and expressive manner.

The syntax of MTL can be defined as follows:

$$\varphi \ := \ true \mid p \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \neg\varphi \mid \varphi \mathcal{U}_I \varphi \quad p \in \mathbf{AP}$$

U_I denotes the *Until* temporal operator in interval I, from which the rest of the operators are derived from. The remaining operators mentioned previously can be seen in Table 1, which will be further showcased in practical examples.

3 Related Work

In [6], the authors present a formalisation of EBS systems based on Allen's [7] temporal interval relations and provide a checking algorithm for the model. In [8], the authors demonstrate a way to utilise MTL in a partially synchronous distributed system where there is not a single global clock. They also provide a way to evaluate MTL expressions incrementally by rewriting the formulas. In [9], the authors provide a formalisation and the respective algorithms using an approach based on workflows and events to proactively detect timing constraints violations in rules.

In [1], the authors introduce a novel tool to check metric temporal logic formulas. The online runtime verification scenario is not fully handled in the tool. The conceptual framework supports it through the use of "informative prefixes", a construct that defines how partial traces can be used to inform us about the truth value of a formula φ. In [2], the authors demonstrate how MTL can be used as an expressive and readable intermediate specification for timed transition graphs in the context of timed discrete-event systems.

In [10], the authors present a method for checking QoS properties, using a continuous-time Markov Chain(CTMC), in the context of component based systems. They also present an automated tool, and two case studies, one of which is based on a service-based system.

In [11], the authors use the Event-B formal method for modelling and analysis in the deployment of cloud applications.

In [12], the authors generate a model and a monitor of an industrial system (IEC 61499 Standard) using process mining from event logs in order to formally verify event-driven systems.

4 Formalising Event-Based Systems

We borrow formalisations for Workflows,"Enactments",Activities and Events from the work of [9]. Briefly explained, Workflows are an abstract series of abstract events/activities, that have a *start* event and possibly multiple *end* events. *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 distance 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

Operator	Symbol	Meaning
Until	$\varphi \mathcal{U}_I \psi$	φ holds until ψ becomes true within interval I.
Release	$\varphi \mathcal{R}_I \psi$	ψ remains true until the first time φ becomes true within I.
Next	$\bigcirc_I \varphi$	φ holds at the next time step within I.
Finally (Eventually)	$\Diamond_I \varphi$	φ will eventually hold at some time within I.
Globally (Always)	$\Box_I \varphi$	φ must hold at all time points within I.

Table 1: Temporal Operators in Metric Temporal Logic (MTL)

example dividing the time required for responding to an event into network latency, database latency and processing latency.

5 Formal properties in MTL

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 [13] , a distinction practically and conceptually 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.

In the context of an event-based system we provide some indicative properties we would want to check. In Table 2 we specify the following properties to be checked. In future work, those could be expanded upon by investigating alerting and monitoring techniques and practices along with testing methodologies for EDA systems.

- Event A should always be followed by Event B within a certain time(response 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 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.
- Event A and Event B should not happen until another event C happens.

In a practical scenario, the above properties will represent policy that must be adhered to by the implemented software, and responses to violations shall be specified in more detail.

6 Software Solution Design

In this section we will talk more in detail about how to implement runtime verification of performance using metric temporal logic.

6.1 Design Goals

The target software architecture we intend to verify is Event-Driven Architecture(EDA), therefore any stated desired goal should consider the constraints of a distributed event-driven architecture.

We outline the following Design Goals and explain them more in detail.

- Online execution for real-time violation reports.
- Integration with the existing event-driven ecosystem.(Ease of Integration with a new system)

Description	MTL Formula
Event A should always be followed by Event B within a certain time t:	$\square(A \implies \Diamond_{\leq t} B)$
Event A should be responded with Event B or Event C within a certain time t:	$\square(A \implies \Diamond_{\leq t}(B \vee C))$
Event chains, e.g., Event A should be followed by Event B which should then be followed by Event C or Event D:	$\square(A \implies \Diamond_{\leq t_1}(B \wedge \Diamond_{\leq t_2}(C \vee D)))$
Event A and Event B should not happen until another event happens:	$\square(\neg(A \vee B)\mathcal{U}_{\mathcal{I}} C)$

Table 2: Metric Temporal Logic (MTL) Formulas for Event Constraints

- Horizontal Scalability and Distributed Computation
- System Independent
- Performance

Online execution is important as the order of magnitude of the data an offline algorithm would need to store is impractical. It also provides valuable timely feedback, useful for addressing performance issues early on and avoiding costly downtimes. Finally the runtime verification problem by its very nature is more accurately expressed by online algorithms, an offline mode would not differ significantly from a simple constraint checking problem using the system logs. In this paper, we focus on the online case.

Being *system independent* practically means that the solution does not have a model of the underlying system, only a generic view of it. Decoupling from a concrete implementation is necessary, due to the fact that in EDA, the only common denominator between different systems is the Event Bus, which is also further segmented into different independent topics. Assuming a specific technology stack is an approach that is unrealistic for complex large-scale systems.

Performance is important to the degree that existing systems are not affected by the addition of the monitoring component into the system. As with all performance measurement tools its resource consumption should be a negligible fraction of the total resource consumption of the system. Data structures and algorithms should be carefully considered to achieve low time and space complexity. Such large-scale systems operate on the order of magnitude of billion operations processed per second thus complexity is crucial in practice.

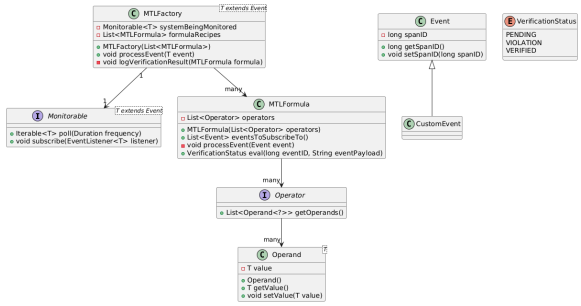
For verification to be practical it should be close to the software development process and not implemented in a separate isolated tool. Instead, a solution should be part of the suite of tools that is already used in EDA systems. This way not only the functionality is easier to adopt in the software development process, but also benefits from a rich ecosystem to support it.

Horizontal scaling of a service, or program more generally, is about increasing the load it can handle, by increasing the number of nodes (machines) it is being executed on. Vertical scaling involves increasing the resources of machines already being used. *Horizontal scalability* is one of the main features of EDA and thus the monitoring component should be able to operate in a distributed manner. Such systems rely on cloud infrastructure that already operates with horizontal scalability in mind. Vertical scalability is impractical for systems operating at this scale.

6.2 Proposed Architecture

The basic concept for this architecture is to isolate the core components of verification from implementation details. A system is being monitored(Monitorable), where events are polled from,

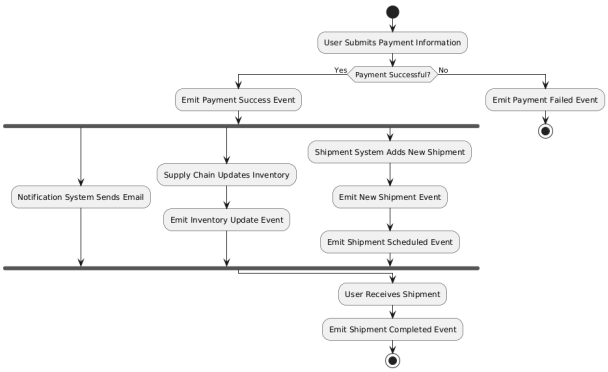
to be processed by the MTLFactory. The MTLFactory is responsible for receiving events and notifying the MTLFormulas to update their verification status. MTLFormulas are just logical expressions that can be evaluated when receiving a new event. The verification status can then be used to notify third-party systems and provide a Dashboard for the system's performance.



7 Case Study: An Integrated E-Commerce System

We will showcase how this approach would look like in the scenario of an integrated e-commerce system. The process we are attempting to verify is the fundamental user journey in e-commerce. It begins with a user that has a full cart and submits the payment information in the website. The response can either be a successfull payment event or a failed payment event. Once the payment goes through, the event emitted is consumed by a notification system that sends an email to the user. In addition, the successfull payment event is consumed by a supply chain software that updates the inventory and emits a respective inventory update event. Lastly, the successfull payment event is consumed by the shipment delivery system that adds a new shipment to be done, emitting the respective new shipment event, which is followed by a shipment scheduled event that notifies the user when the package ordered will arrive.

Finally, when the user receives the package, the person responsible for its delivery submits a shipment completed event, and the user journey finishes.



A few desirable properties for which we would want to check that they hold and know when they don't due to system failures are the following:

1. After the user paying(proposition CL), a payment completed(proposition PC) or a payment failed(proposition PF) event arrives in X time
2. A shipment should be scheduled(proposition SCH) within 24 hours of the user having paid.
3. If the user specified same day delivery, the Shipment Completed(proposition SC) Event should be within 24 hours of the payment completed event.

The properties of the system can be expressed in MTL in the following manner:

$$\square(CL \implies \Diamond_{\leq 5sec}(PC \vee PF)) \quad (1)$$

$$\square(PC \implies \Diamond_{\leq 24h} SCH) \quad (2)$$

$$\square(SDD \implies \Diamond_{\leq 24h} SC) \quad (3)$$

8 Future Work

The research presented in this work attempts to apply runtime verification in performance monitoring using MTL for event-based systems. However to fully adopt MTL in enterprise EBS there needs to be additional work. Below, we outline key directions for extending this work:

- *Rigorous Integration with Application Protocols*: A promising direction is the integration of MTL-based runtime verification with widely used application protocols such as *GraphQL*, *gRPC*, and

REST. These protocols are fundamental to modern distributed systems, and ensuring their compliance with temporal performance properties could significantly enhance system reliability and efficiency. Future work could focus on developing protocol-specific adapters that translate protocol-level events into MTL-compatible formats, enabling seamless verification across diverse communication paradigms.

- *Autoscaling Infrastructure Resources in Response to Violations*:
  Another critical area of exploration is the dynamic adjustment of infrastructure resources based on violations of MTL properties. By coupling runtime verification with *autoscaling mechanisms*, systems could respond to performance degradation captured by violations of temporal constraints.

- *Generating Metric Temporal Propositions from Event Logs Using Process Mining*:
  Automating the generation of MTL propositions from event logs using *process mining* techniques represents a significant step toward making runtime verification more accessible. Process mining can uncover patterns and temporal relationships in event logs, which can then be formalized as MTL properties. Future work could focus on developing algorithms and tools to extract meaningful MTL propositions from historical event data, reducing the manual effort required for specification and enabling more comprehensive verification.

These future directions aim to expand the applicability and impact of MTL-based runtime verification, making it a more integral part of modern EBS. By addressing these challenges, we can move closer to integrating performance considerations in EBS in a non-invasive manner.

9 Conclusion

In this paper, we presented a formal approach to runtime performance verification in event-based systems (EBS) using Metric Temporal Logic (MTL). We demonstrated how MTL enables precise specification and monitoring of performance constraints, ensuring QoS guarantees in large-scale, event-driven architectures.

We also proposed a scalable software solution for integrating runtime verification into existing systems with minimal overhead. Through a case study of an e-commerce system, we illustrated its practical application, showcasing how MTL can be used to enforce performance properties and detect violations in real-time.

References

[1] M. Hendriks, M. Geilen, A. R. B. Behrouzian, T. Basten, H. Alizadeh, and D. Goswami, "Checking Metric Temporal Logic with TRACE," in *2016 16th International Conference on Application of Concurrency to System Design (ACSD)*, (Torun, Poland), pp. 19–24, IEEE, June 2016.

[2] A. Dhananjayan and K. T. Seow, "A metric temporal logic specification interface for real-time discrete-event control," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 44, no. 9, pp. 1204–1215, 2014. Publisher: IEEE.

[3] P. Thati and G. Roşu, "Monitoring Algorithms for Metric Temporal Logic Specifications," *Electronic Notes in Theoretical Computer Science*, vol. 113, pp. 145–162, Jan. 2005.

[4] A. Pnueli, "The temporal logic of programs," in *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, pp. 46–57, Oct. 1977. ISSN: 0272-5428.

[5] R. Koymans, "Specifying real-time properties with metric temporal logic," *Real-Time Systems*, vol. 2, pp. 255–299, Nov. 1990.

[6] T.-B. Trinh, H.-P. Nguyen, D.-H. Nguyen, V.-K. To, and N.-T. Truong, "Checking Temporal

Constraints of Events in EBS at Runtime," *Cybernetics and Information Technologies*, vol. 24, pp. 82–97, Mar. 2024.

[7] J. F. Allen, "Maintaining knowledge about temporal intervals," *Communications of the ACM*, vol. 26, pp. 832–843, Nov. 1983.

[8] R. Ganguly, Y. Xue, A. Jonckheere, P. Ljung, B. Schornstein, B. Bonakdarpour, and M. Herlihy, "Distributed runtime verification of metric temporal properties," *Journal of Parallel and Distributed Computing*, vol. 185, p. 104801, Mar. 2024.

[9] I. Mackey, R. Chimni, and J. Su, "Early detection of temporal constraint violations," *Information and Computation*, vol. 296, p. 105114, Jan. 2024.

[10] C. Paterson and R. Calinescu, "Observation-enhanced QoS analysis of component-based systems," *IEEE Transactions on Software Engineering*, vol. 46, no. 5, pp. 526–548, 2018. Publisher: IEEE.

[11] A. Mammar, M. Belguidoum, and S. H. Hiba, "A formal approach for the correct deployment of cloud applications," *Science of Computer Programming*, vol. 232, p. 103048, 2024. Publisher: Elsevier.

[12] M. Xavier, V. Dubinin, S. Patil, and V. Vyatkin, "A Framework for the Generation of Monitor and Plant Model From Event Logs Using Process Mining for Formal Verification of Event-Driven Systems," *IEEE Open Journal of the Industrial Electronics Society*, vol. 5, pp. 517–534, 2024. Conference Name: IEEE Open Journal of the Industrial Electronics Society.

[13] L. Lamport, "What good is temporal logic?," in *IFIP congress*, vol. 83, pp. 657–668, 1983.