Performance Models of Event-Driven Architectures

Murray Woodside
Department of Systems and Computer Engineering
Carleton University
Ottawa Canada
cmw@sce.carleton.ca

ABSTRACT

Event-driven architecture (EDAs) improves scalability by combining stateless servers and asynchronous interactions. Models to predict the performance of pure EDA systems are relatively easy to make, systems with a combination of event-driven components and legacy components with blocking service requests (synchronous interactions) require special treatment. Layered queueing was developed for such systems, and this work describes a method for combining event-driven behaviour and synchronous behaviour in a layered queueing model. The performance constraints created by the legacy components can be explored to guide decisions regarding converting them, or reconfiguring them, when the system is scaled.

CCS CONCEPTS

•General and reference~Cross-computing tools and techniques~Performance •Software and its engineering~Software organization and properties~Extra-functional properties~Software performance

KEYWORDS

Software architecture, software performance, event-driven architecture, layered queueing

ACM Reference format:

Murray Woodside. 2021. Performance Models of Event-Driven Architectures. *In the Companion of the 2021 ACM/SPEC International Conference on Performance Engineering, (ICPE'21 Companion), April 19-23, 2021, Virtual Event, France.* ACM, New York, NY, USA. 6 pages. https://doi.org/10.1145/3447545.3451203

1 Introduction

Event-driven architecture (EDA) improves scalability by combining stateless servers and asynchronous interactions. Independent components are loosely coupled through a messaging infrastructure such as message queueing or an event bus. The essntial ideas are described by Richaards in [6], and implementation issues are addressed by Ambre in [1] and by Puri in [5]. The broad outline of a EDA is illustrated by the example in Figure 1 for a simplified e-commerce system. Each component may have multiple load-balanced instances, for scalability.

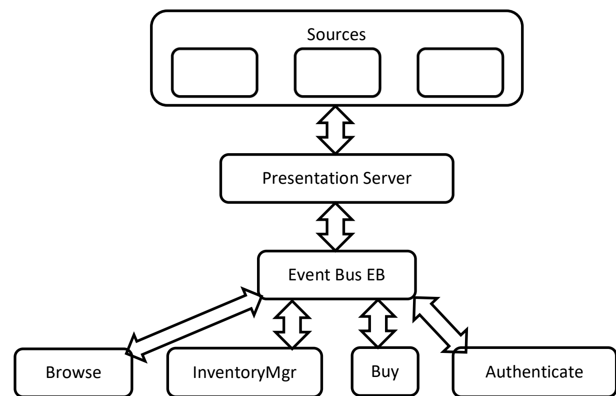

Figure 1 Event-Based Architecture Example

When a component such as "Buy" sends a request to a second component such as InventoryMgr, the request message if handled by the EventBus and the Buy component does not wait for the reply. When the reply is generated it also is handled by the EventBus and may go to a different instance of Buy. To bridge the gap between request and reply the state of the operation at Buy is saved in a Persistence Store (not shown) and retrieved by the instance that takes the reply.

Performance modeling of systems under EDA was addressed by Rathfelder et al in [5] using a simulation-based methodology. They placed particular emphasis in modeling the asyncrhonous communications via a publish/subscribe system. The present paper considers analytic queueing approximations using layered queueing networks (LQNs), which have been described by Franks et al [2] and Woodside [8]. Some previous work by Liu in this direction models publish-subscribe systems in the thesis [3].

The decision to employ EDA, or to convert a legacy system to EDA, is often driven by scalability concerns. The possibilities may include a hybrid solution that integrates a legacy system using RPCs with new event-based components, and our concern here is how to model the performance of such hybrids. This paper

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.
ICPE '21 Companion, April 19–23, 2021, Virtual Event, France
© 2021 Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-8331-8/21/04...$15.00
https://doi.org/10.1145/3447545.3451203

proposes to evaluate the performance of alternatives using LQNs, which were developed to represent the RPC-based architecture directly. To apply LQNs to EDA, the asynchronous interaction must be correctly represented, including the mechanisms used to implement EDA. We consider here the use of persistent storage to store the state of an incomplete service operation, and the use of an Event Bus to transfer messages asynchronously. Additional features of EDA, particularly the handling of transactions (see, e.g. the Saga pattern in [7]) are not fully addressed here.
A method for incorporating these mechanisms will be introduced using the same example, using three alternative architectures:

1. Sync: A direct implementation of the RPC-based architecture with blocking interactions modelled by the LQN in Figure 2, called here a "synchronous" architecture
2. Hybrid: A hybrid architecture in which only one component (here, the Buy server) makes synchronous calls,
3. EDA: as in Figure 1.

The operational architecture, which shows the interactions between the components, is displayed in Figure 1. The Figure uses the notation of Layered Queueing Networks (LQNs), and shows call-return interactions by arrows with *filled arrowheads*. These create blocking interactions, that is the progress of an operation is blocked when a call is made, until the reply is received.

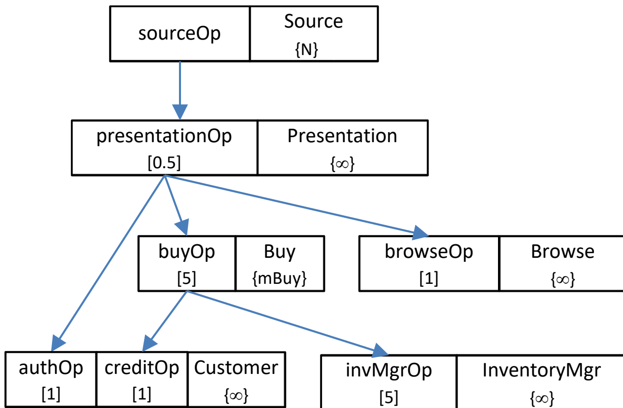

Figure 2 Operational Architecture and LQN Model Structure

The theory of layered queues was developed to address the problem of modeling blocking interactions, and Figure 2 can be interpreted as a LQN. In LQN terms concurrent components are called tasks which offer operations called entries; entries make synchronous (RPC-like) and asynchronous calls to other entries, and a call can be forwarded to additional entries to describe a pipeline of processing.
The Sync alternative is modeled immediately by the LQN defined in Figure 2. To model full or partial EDA, the asynchronous calls must be modeled, however the simple asynchronous calls defined in LQN do not give the most useful model of EDA. We would prefer a model that captures the user response time by retaining the structure and content of the pattern of execution that makes up a user response. There are two ways to capture this: first, from the functional architecture, and second, from the workflow.

We will consider beginning from the functional architecture first, using a transformation to convert a synchronous call into asynchronous interactions using an event bus, which is applied to alternatives 2 and 3.

2. Modeling a Call-Return in an EDA

The modeling of a call-return will be described using Figure 3 which shows a call from the Presentation to the Browse component in LQN notation. One operation of Presentation (invoked by Source) makes y_{PB} calls to browseOp, the operation performed by Browse.

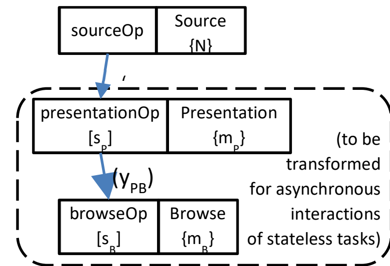

Fig 3 One Synchronous Interaction, to be Transformed

In the EDA style, when PresentationOp makes a call, it first stores its state in a Persistence Server, then sends its call to browseOp via the EventBus. At this point it is completely disengaged from the transaction. When browseOp replies via the EventBus, the state is retrieved from the Persistence Server and the operation continues. The transformation introduces a Persistence server for each component, and calls to write and read the state to/from it.
To disengage the component from the operation of Browse, the *forwarding* interaction in LQN is used together with some pseudo-tasks. Forwarding a call from a server removes it from the return path and means there is no blocking. Pseudo-tasks are LQN model components that do not model software components, but exist purely to describe behaviour. They describe some operation of a system component separately, in order to provide details of the behaviour.
Figure 4 shows the interaction for EBA. To exploit forwarding the execution of presentationOp is divided into two parts, its own execution with the persistence operations, and its calls. Only its own execution is associated with presentationOp itself, then the operation is forwarded to a pseudo-task CallsFromP which represents the delay to make the calls. The pseudo-task makes all calls to whatever further servers are called (here, just synchronous calls to Browse). CallsFromP is modeled as having infinite multiplicity, meaning that it places no limit on the number of simultaneous calls from replicas of Presentation.
The calls are made through a component modeling the EventBus, with two operations, a "client-side" operation to handle the call from Presentation to Browse and forward it, and a "reply" operation to handle the reply. The "reply" operation is the last step in the forwarding path, after which the forwarding semantics imply that a reply is received first at CallsFromP and finally (when all calls are made) at Source.

The transformation clusters all the calls from presentationOp together and moves them to cfpOp. The execution-intervals of presentationOp are modeled separated only by the persistence calls. This structure is imposed by the semantics of forwarding in layered queues. For an analytic mean-value analysis solution (such as is provided by the LQNS solver), the two descriptions are equivalent since they give the same total mean delay for the Caller entry presentationOp.

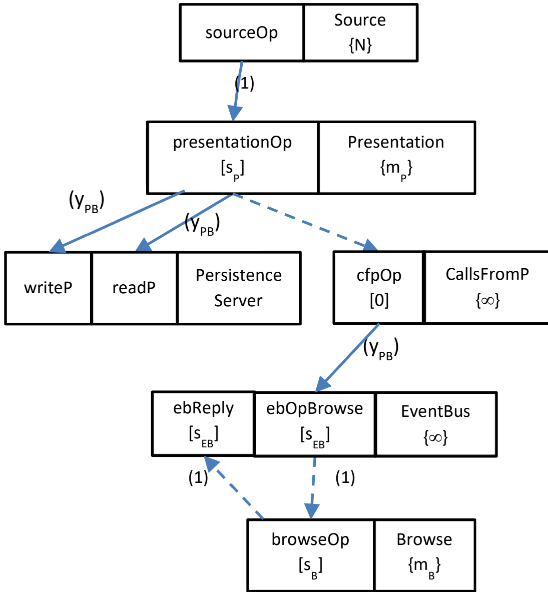

Fig 4. Event-Based Transformation of One Interaction

The use of forwarding means that Source gets a reply when all the operations are complete, so Source sees the correct average response time.

In summary

- the operations with the Persistence server capture the overheads of saving and restoring state, and block the server so the delay of this step is captured,
- the forwarding makes the participation of Presentation asynchronous, and means that EventBus forwards each message it receives, without blocking.
- the CallsFromP pseudo-task imposes the calling pattern without blocking Presentation. Each task sending through EventBus has its own caller pseudo-task.
- Overhead and congestion at EventBus are captured.

The example does not include network delays, which may also important for response time. For a blocking call the round-trip network delay needs to be included in the response delay, and this can be achieved in a variety of ways. One way is to include the total average network delay of all its calls in the "CallsFrom" pseudo-task think time parameter, which adds a pure delay to the operation time.

3. Modeling a Pure Event-Driven Architecture

Figures 3 and 4 define a transformation for each call in the LQN of Figure 1.

1. Each component that makes calls has a persistence server deployed with it, and a CallsFromX pseudo-task created for it.
2. each operation that makes calls adds a write and a read call to the persistence server, for each call it makes.
3. for each operation that makes calls, it forwards to an entry which is added to the CallsFromX pseudo-task, and the calls are moved to the pseudo-task.
4. each call from the pseudo-task is replaced by three calls:

a. a single synchronous call to a "source" EventBus entry

b. forwarding to its destination entry

c. a forwarding call from the destination entry to a "reply" EventBus entry.

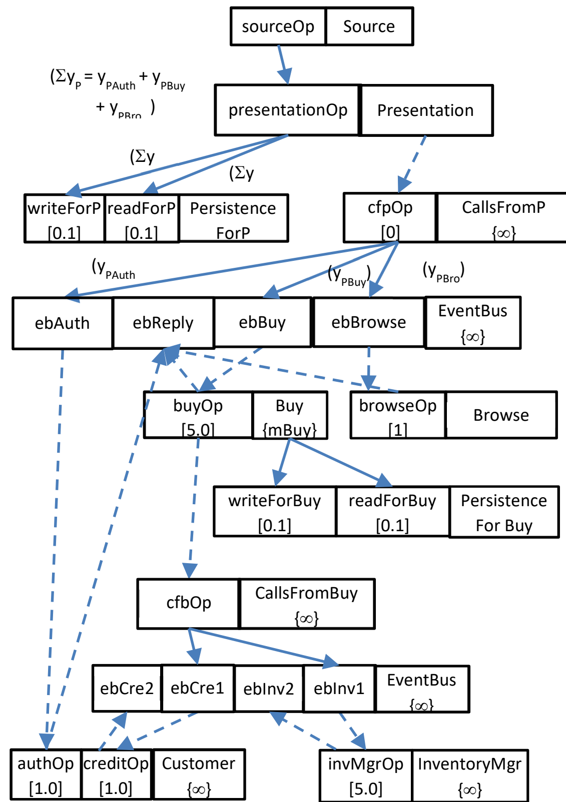

Fig 5 LQN Model for a Fully Event-Based Architecture

Every distinct call must have a separate "client-side" entry in the EventBus, to connect the call to the correct target server entry, but all calls can share a single "reply" entry.

Applying this transformation gives the LQN model in Figure 5.

In Figure 5 a single EventBus messaging server has been assumed, although it is shown twice for convenience, but both EventBus symbols refer to the same component, since they share a name. The deployment can in fact include a network of event buses.

4. Comparison of Alternative Architectures

To complete the set of alternative models, Fig 6 shows a model for the hybrid architecture with a legacy Buy server making synchronous calls to creditOp and invMgrOp. The operations for persistence and for the EventBus were assumed to require 0.1 unit of CPU time, the other operations are labelled with their CPU demands. Tasks are labeled with their multiplicity, representing thread pool size; stateless servers are modeled as infinite for convenience.

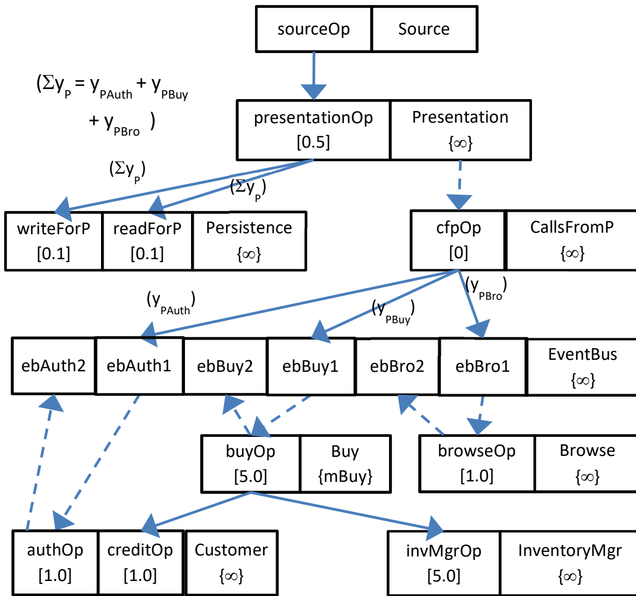

Fig 6 LQN Model of EBA with Legacy "Buy" Server

The question of system scalability is addressed by considering increasing values of N, the number of Sources (typically, users). Figure 7 shows the system throughput obtainable for the three architectures. The top line for EBA scales smoothly and continues to scale beyond 100 users, assuming that additional processing resources are added as needed. The other curves represent both the Sync and Hybrid versions, which give almost identical throughputs for different values of mBuy, the thread pool size for the Buy server. Thus the hybrid architecture buys no improvement in scalability, because the Buy server is the limiting factor in the performance of the system. Adding event-based operations to the other components did not yield any scalability benefit.

5. Modeling Event-Based Architectures via Workflows

An alternative structure can be used to model pure event-based systems with LQNs, which is better for some purposes. It is outlined here for completeness, but the example is not worked out in full. If the workflows of the system are known, the workflows themselves are first modeled by LQN *activity graphs* (see [2] or [8] for details of activity graphs) embedded in workflow pseudo-tasks. Since the pseudo-tasks invoke the execution of the workflows they have the role of EBA Orchestrators.

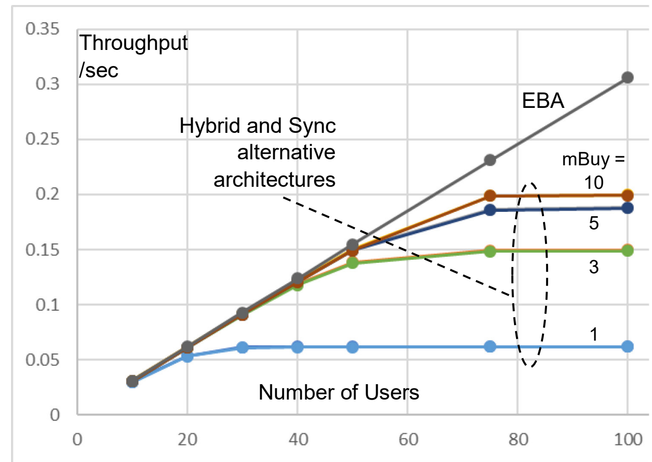

Fig 7 Throughput vs Number of Request Sources

LQN activity graph semantics include the basic flow semantics of workflows, with sequence, alternative and parallel branching of the flow, and loops. In the workflow graph each activity executes a workflow step by making calls to model elements, which can be tasks to execute operations, or other orchestrators. Figure 8 shows an example describing a workflow over some of the system elements as we considered above (the Presentation server is left out). Note that the arrows between activities (which are suffixed "Act" here) represent precedence, while the arrows from activities to entries represent synchronous calls.

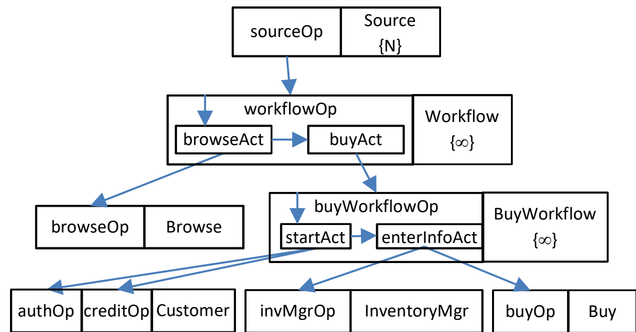

Figure 8 LQN based on Workflow

In this alternative structure the messages between tasks are not represented as such in the model. The workflow tasks are infinite, assigned to notional infinite hosts, and do no execution. They only act as orchestrators. The calls to task entries are all synchronous because they just invoke the operation and return; no task makes calls in this form of model.

To complete an EBA model using an EventBus for messaging, two elements are added to Figure 8 to give Figure 9:

- Persistence servers for each task which makes calls, with a number of calls equal to the total request messages sent. In this case only the Buy server makes calls,

- messaging delay calls to insert the EventBus delays into the response. Each activity which invokes an operation that makes calls, adds a call to a "client-side" entry and one to a "reply" entry of EventBus for each call. In the Figure these are added to buyAct.

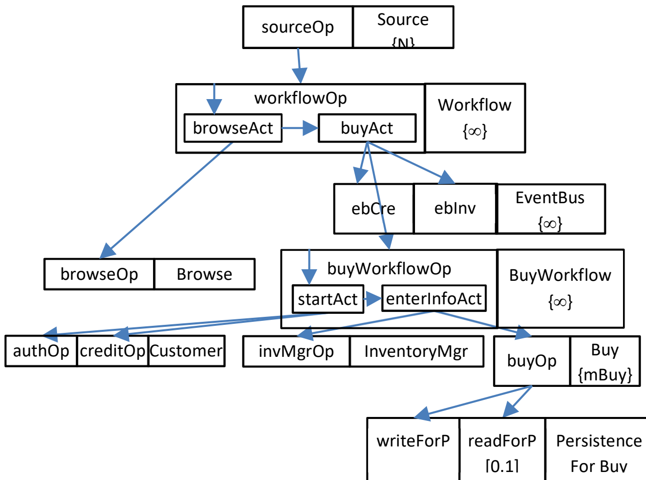

Figure 9 LQN Based on Workflow with Persistence and EventBus Operations Added.

With this approach to modeling a workflow it is straightforward to model additional features that may affect performance. The example of admission control will be described here. The behaviour of admitted jobs is modeled by a separate workflow pseudo-task with multiplicity equal to the size limit for admission. For example to control the number of users who can enter the buying workflow to a maximum of 7, the multiplicity of the BuyWorkflow pseudo-task would be set to 7 instead of infinity. Similarly the behaviour of a transaction feature implemented in an event-based style such as the Saga pattern [7] can be modeled by a workflow pseudo-task.

ACKNOWLEDGMENTS

This research was supported by NSERC, the Natural Sciences and Engineering Research Council of Canada, through its Discovery Grants program.

REFERENCES

[1] Tanmay Ambre, "Architectural considerations for event-driven microservices-based systems", IBM Developer, July 10, 2020. Online at https://developer.ibm.com/depmodels/microservices/articles/eda-and-microservices-architecture-best-practices/.

[2] Greg Franks, Tariq Al-Omari, Murray Woodside, Olivia Das, Salem Derisavi, "Enhanced Modeling and Solution of Layered Queueing Networks", IEEE Trans. on Software Eng. Aug. 2008

[3] Helen (Huai) Liu, *Multilevel Performance Analysis of Scenario Specification for a Presence System*, MSc thesis, Carleton University, October 2002.

[4] Sumeet Puri, "The Architect's Guide to Implementing Event-Driven Architecture", Solace, 2020. Online at https://solace.com/resources/white-papers/wp-download-architects-guide-to-implementing-event-driven-architecture

[5] Christoph Rathfelder, Benjamin Klatt, Kai Sachs & Samuel Kounev "Modeling event-based communication in component-based software architectures for performance predictions", Software & Systems Modeling v 13, (2014), 1291–1317

[6] Mark Richards, *Event-Driven Architecture*, chapter 2 in "Software Architecture Patterns", O'Reilly, 2015

[7] Chris Richardson, *Microservices Patterns*, Manning Publications, Nov. 19 2018

[8] Murray Woodside, "Tutorial Introduction to Layered Modeling of Software Performance", online at http://www.sce.carleton.ca/rads/lqns/lqn-documentation/tutorialh.pdf