Modeling Interactions Among Microservices Communicating via FIFO or Bag Buffers

Fei Dai[1], Jinmei Yang[1], Qi Mo[2](✉), Hua Zhou[1], and Lianyong Qi[3]

[1] School of Big Data and Intelligent Engineering, Southwest Forestry University, Kunming, China
daifei@swfu.edu.cn
[2] School of Software, Yunnan University, Kunming, China
[3] School of Information Science and Engineering, Qufu Normal University, Jining, China

Abstract. Interactions among individual microservices communicating asynchronously via FIFO or bag buffers vary significantly even for the same buffer size. Different interactions among microservices will lead to different interaction behaviors, which can make microservices systems malfunction during their execution. However, these two asynchronous communication models with FIFO or bag buffers are seldom distinguished. In this paper, we present new results for the interaction differences between one asynchronous communication model with FIFO buffers and another asynchronous communication model with bag buffers. First, we propose a framework to uniformly define two asynchronous communication models. Second, we model interaction behaviors among microservices as sequences of send and receive message actions under these two asynchronous communication models. Finally, we compare these two asynchronous communication models using refinement checking to show their differences. Experimental results show that the asynchronous communication model with FIFO buffers is included in the asynchronous communication model with bag buffers.

Keywords: Interactions · Microservice · Asynchronous communication · FIFO buffers · Bag buffers

1 Introduction

The cloud computing paradigm drives many IT companies to build microservice systems [1, 2]. A microservices system consists of a set of individual microservices that interact with each other via exchanging messages [3]. Compared with traditional web services, these microservices are much more fine-grained and are independently developed and deployed. These characteristics of microservices drive many IT company's software systems to migrate from monolithic architecture to microservice architecture [4, 5].

In a microservices system, most of the interactions among microservices are asynchronous since synchronous interactions are considered to be harmful due to the multiplicative effect of downtime [4]. These interactions among individual microservices communicating asynchronously via FIFO or bag buffers vary significantly even for the

H. Song and D. Jiang (Eds.): SIMUtools 2020, LNICST 370, pp. 505–518, 2021.
https://doi.org/10.1007/978-3-030-72795-6_41

same buffer size. These different interactions will lead to different interaction behaviors, which can make microservices systems malfunction or deadlock during their execution. However, these two asynchronous communication models with FIFO or bag buffers are seldom distinguished [6].

Most of the existing works are restricted to asynchronous communication via FIFO buffers [7], e.g. [10–16]. However, asynchronous communication via bag buffers is meaningful [7–9] when the ordering of consuming messages does not matter. To ensure the sound execution of microservices systems, it is necessary to distinguish the differences between two asynchronous communication models with FIFO or bag buffers.

In this paper, we present new results for the interaction differences between one asynchronous communication model with FIFO buffers and another asynchronous communication model with bag buffers. Specifically, the contributions of this paper are as follows.

1) Define two asynchronous communication models with FIFO buffers and bag buffers uniformly.
2) Model interactions among microservices under these two different asynchronous communication models uniformly.
3) Compare these two asynchronous communication models using refinement checking to show their differences.
4) Conduct experiments to show the effectiveness of our proposed results.

The rest of this paper is organized as follows. Section 2 discusses related work. Section 3 presents formal definitions of two asynchronous communication models with FIFO or bag buffers. Section 4 formally model interaction behaviors among microservices under two asynchronous communication models. Section 5 compares asynchronous communication models with trace refinement. Section 6 discusses the implementation of our approach and experimental results. Section 7 concludes this paper.

2 Related Work

There have been many works on modeling interactions among peers communicating asynchronously via FIFO buffers. In [10, 11], Bultan et al. first proposed a framework to model interactions of e-service as a sequence of send messages, which is called a conversation. In their model, an e-service is composed of a set of peers where each peer is equipped with one FIFO queue for storing messages sent from other peers. Subsequently, Bultan et.al used to conversation protocol to model interactions of a composite Web service that consists of a set of peers under asynchronous communication with FIFO buffers [12]. In [13, 14], Basu et al. also used conservation protocols to model interactions of four distributed systems that consist of a set of peers communicating via FIFO queues. In [15, 16], Salaün et al. modeled interactions of asynchronously communicating systems with unbounded buffers as a sequence of sending messages via FIFO buffers.

Compared to these works above, we not only consider an asynchronous communication model with FIFO buffers but also consider an asynchronous communication model with bag buffers. Besides, we are particularly interested in modeling interactions among microservice as sequences of send and receive message actions rather than send message actions.

Recently, some research has begun to consider an asynchronous communication model with bag buffers. In [17], Clemente et al. studied the reachability problem for finite-state automata communicating via both FIFO and Bag buffers. In [7], Akroun et al. studied the verification problem for finite-state automata communicating via FIFO and bag buffers. However, our work focuses on the interaction differences between two asynchronous communication models with FIFO or bag buffers.

3 Asynchronous Communication Models

This section presents two asynchronous communication models with FIFO or bag buffers in a unified way.

In a microservices system, microservices interact with each other via messages.

Definition 1 (Message Set). A message set M is a tuple $(\Sigma, p, send, rec)$ where:

- Σ is a finite set of letters.
- $p \geq 1$ is a non-negative integer number which denotes the numbers of participating microservices.
- *src* and *dst* are functions that associate message $m \in \Sigma$ nonnegative integer numbers $send(m) \neq rec(m) \in \{1, 2, \ldots, p\}$.

We often write $m^{i \to j}$ for a message m such that $send(m) = i$ and $rec(m) = j$.

Definition 2 (Microservice). A microservice $MS_p = (S_p, s_p^0, F_p, A_p, \delta_p)$ is an LTS, where:

- $p \in \{1,2,\ldots,N\}$.
- S_p is the finite set of states.
- $s_p^0 \in S_p$is the initial state.
- $F_p \subseteq S_p$ is the finite set of final states.
- A_p is a set of actions
- $\delta_p \subseteq S_p \times (A_p \cup \{tau\}) \times S_p$ is the transition relation.

For a microservice p, an action over M is either send message action $!m^{i \to j}$ or receive message action $?m^{i \to j}$, with $m \in M_p$. The function $peer(a)$ of an action $a \in A_p$ is defined as $peer(!m^{i \to j}) = i$ and $peer(?m^{j \to i}) = i$.

In a microservice, a transition $t \in \delta_p$ can be one of the following three types:

(1) a send message transition $(s_1, !m^{1 \to 2}, s_2)$ denotes that the microservice MS_1 sends a message m to another microservice MS_2 where $m \in M_1$.
(2) a receive message transition $(s_1, ?m^{1 \to 2}, s_2)$ denotes that the microservice MS_1 consumes a message m from the microservice MS_2 where $m \in M_p$.
(3) an ε-transition (s_1, ε, s_2) denotes that the invisible action of MS_1.

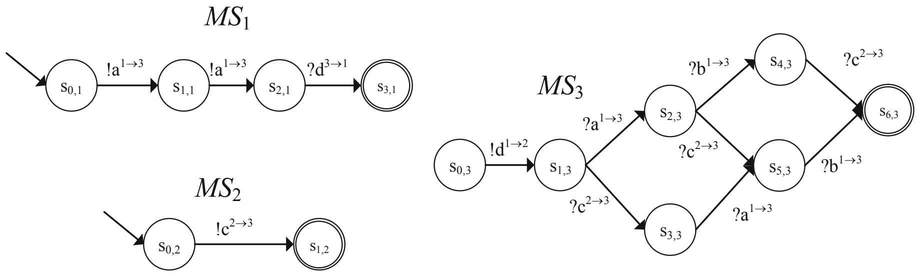


Fig. 1. Motivating example

We often write $s_m \xrightarrow{!m^{1\to2}} s_k$ to denote that (s_m, $!m^{1\to2}$, s_k).

Figure 1 gives a motivating example where microservices are modeled LTS. The initial states of each microservice are subscripted with 0 and marked with an incoming half-arrow. The final states of each microservice are marked with double circles. Each transition of each microservice is labeled with send action (exclamation marks) or receive message action (question marks).

Definition 3 (Actions). Let a microservices system is composed of a set of microservices $MSs = (MS_1, MS_2, \ldots, MS_n)$, the set of actions is:

$$As \triangleq \{!m^{i\to j} \mid\, !m^{i\to j} \in MS.A \wedge MS \in MSs\} \cup \{?m^{j\to i} \mid ?m^{j\to i} \in MS.A \wedge MS \in MSs\}$$

Definition 4 (Execution Sequences). The execution sequences *ES* is set of all finite or infinite sequences of send and receive message actions over *As* such that a received message action is preceded by the send message action.

$$ES \triangleq \left\{ \begin{array}{c} \sigma \in As^* \mid \forall i, j \in dom(\sigma) : \sigma[i] = !m^{x\to y} \wedge \sigma[j] = !m \Rightarrow i = j \\ \wedge \forall i, j \in dom(\sigma) : \sigma[i] = ?m \wedge \sigma[j] = ?m \Rightarrow i = j \\ \wedge\ \forall j \in dom(\sigma) : \sigma[j] = ?m \Rightarrow \exists i \in dom(\sigma) : \sigma[i] = !m \wedge i < j \end{array} \right\}$$

If an action $a \in As$ occurs in an execution sequence σ, we write $a \in \sigma$ and have $\exists j \in dom(\sigma)$ such that $\sigma[j] = a$.

Concerning an execution sequence $\sigma \in ES$, we define the total order.

Definition 5 (Total Order).

$$a_1 \prec_t a_2 \triangleq \exists i, j \in dom(\sigma) : i \leq j \wedge \sigma[i] = a_1 \wedge \sigma[j] = a_2$$

In the following, we define two asynchronous communication models in terms of message ordering and buffer number, that are summed up in Table 1.

Table 1. Overview of two asynchronous communication models

	ACM_{FIFO}	ACM_{Bag}
Message ordering	$!m_1^{i\to j} \prec_t \, !m_2^{k\to j}$ $\wedge peer(?m_1^{i\to j}) = peer(?m_2^{k\to j})$ $\Rightarrow \; ?m_1^{i\to j} \prec_t \, ?m_2^{k\to j}$	$!m_1^{i\to j} \prec_t \, !m_2^{k\to j}$ $\wedge peer(?m_1^{i\to j}) = peer(?m_2^{k\to j})$ $\Rightarrow \; (?m_1^{i\to j} \prec_t \, ?m_2^{k\to j}) \vee (?m_2^{i\to j} \prec_t \, ?m_1^{k\to j})$
Buffer types	FIFO buffers	Bag buffers

3.1 Asynchronous Communication Model with FIFO Buffers

An asynchronous communication model with FIFO buffers that is called ACM_{FIFO} requires that the order of receiving message actions is the order of sending message actions and that each microservice has one buffer which is used to store all messages sent from other microservices in a FIFO fashion. The FIFO fashion means that the buffer can be viewed as a queue of messages. In other words, a send message action is to add a message at the tail of the buffer of the destination microservice while the corresponding receive message action is to consume a message at the head of the buffer of the receiver. This asynchronous communication model is used in [10–16].

Figure 2 (a) illustrates ACM_{FIFO}, where each microservice has one FIFO buffer. Figure 2 (b) illustrates an interaction scenario, where the microservice MS_1 sends a message a to the microservice MS_3 before sending a message b to MS_3 and then the microservice MS_3 consumes a from its $buffer_3$, and later b. Note that the messages a and b are stored in the buffer $buffer_3$ in the order they are sent.

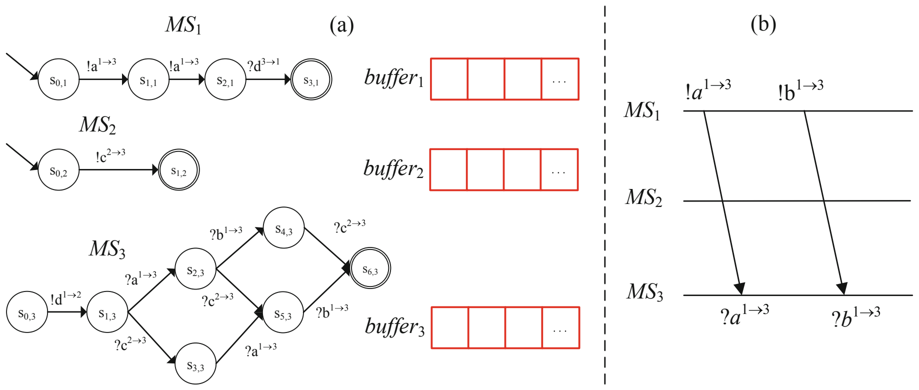


Fig. 2. Asynchronous communication model with FIFO buffers

Definition 6 (Asynchronous communication model with FIFO buffers).

$$ACM_{FIFO} \triangleq \left\{\sigma \in A^* | \forall m_1, m_2 \in M,\ \exists i,j,k \in \mathrm{N}:\ !m_1^{i\to j} \prec_t\ !m_2^{k\to j} \wedge Buffer_j(m_1) \wedge Buffer_j(m_2) \Rightarrow ?m_1^{i\to j} \prec_t\ ?m_2^{k\to j}\right\}$$

3.2 Asynchronous Communication Model with Bag Buffers

An asynchronous communication model with bag buffers that is called ACM_{Bag} requires that the order of receiving message actions is the order of sending message actions and that all messages sent to one microservice from the other microservices are stored in a buffer in a bag fashion. The bag fashion means that the buffer can be viewed as a set. In other words, a receiver microservice can consume messages from its buffer in any order. This asynchronous communication model is used in [7, 17].

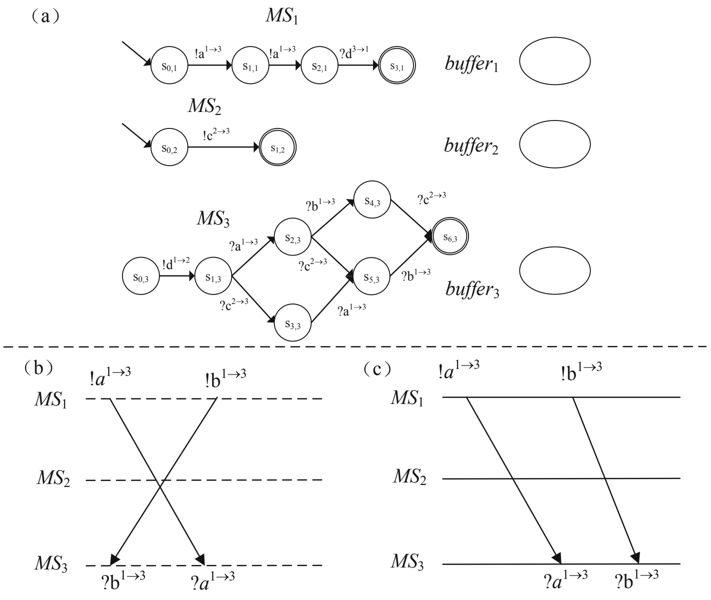


Fig. 3. Asynchronous communication model with bag buffers

Figure 3 (a) illustrates ACM_{Bag}, where each microservice has one bag buffer. Figure 3 (b) and (c) illustrate two interaction scenarios separately, where the microservice MS_1 sends a message a to the microservice MS_3 before sending a message b to MS_3 and then the microservice MS_3 consumes$a$ and b in any order.

Definition 7 (Asynchronous communication model with bag buffers).

$$ACM_{Bag} \triangleq \left\{\begin{array}{r} \sigma \in A^* | \forall m_1, m_2 \in M,\ \exists i,j,k \in \mathrm{N}:\ !m_1^{i\to j} \prec_t\ !m_2^{k\to j} \wedge Buffer_j(m_1) \wedge Buffer_j(m_2) \\ \Rightarrow (?m_1^{i\to j} \prec_t\ ?m_2^{k\to j}) \vee (?m_2^{i\to j} \prec_t\ ?m_1^{k\to j}) \end{array}\right\}$$

4 Modeling Interactions

In this section, we model interaction behaviors among microservices as sequences of send and receive message actions under two asynchronous communication models defined in Sect. 3.

4.1 Modeling Interactions Among Microservices Under the Asynchronous Communication Model with FIFO Buffers

Definition 8 (FIFO buffer). A FIFO buffer is a queue of messages, where a send message action is to add a new message at the tail of the queue and a receive message action is to consume a message from the head of the queue.

Definition 9 (Interaction behavior under the asynchronous communication model with FIFO buffers). An interaction behavior among a set of microservices (MS_1, MS_2, …, MS_N) with $MS = (S_i, s_i^0, F_i, A_i, \delta_i)$ and Qi being its associated buffer under the asynchronous communication model with FIFO buffers is a labeled transition system $IB_{FIFO} = (S, s_0, F, A, \delta)$ where:

- $S \subseteq S_1 \times Q_1 \times S_2 \times Q_2 \times \ldots \times S_N \times Q_N$ where $\forall i \in \{1,2, \ldots, N\}$, $Q_i \in M^*$ with $M = MS_1.M_1 \cup MS_2.M_2 \cup \ldots \cup MS_N.M_N$.
- $s_0 \in S$ such that $s_0 = (s_1^0, \varepsilon, s_2^0, \varepsilon, \ldots, s_N^0, \varepsilon)$.
- $F \subseteq S$.
- $A = MS_1.A_1 \cup MS_2.A_2 \cup \ldots \cup MS_N.A_N$.
- $\delta_p \subseteq S \times (A \cup \{tau\}) \times S$ for $s = (s_1, Q_1, s_2, Q_2, \ldots, s_n, Q_n)$ and $s' = (s'_1, Q'_1, s'_2, Q'_2, \ldots, s'_n, Q'_n)$.

(a) send message action

$$s \xrightarrow{!m^{i \to j}} s' \in \delta \text{ if } \exists i, j \in \{1,2,\ldots,N\} \wedge m \in M:$$

(i) $send\ (m) = i \wedge rec(m) = j \wedge i \neq j$,
(ii) $s_i \xrightarrow{!m^{i \to j}} s_i' \in \delta_i$,
(iii) $Q'_j = Q_j m$,
(iv) $\forall k \in \{1,2, \ldots, N\}: k \neq i \Rightarrow s'_k = s_k$,
(v) $\forall k \in \{1,2, \ldots, N\}: k \neq i \Rightarrow Q'_k = Q_k$.

(b) receive message action

$$s \xrightarrow{?m^{i \to j}} s' \in \delta \text{ if } \exists i, j \in \{1,2,\ldots,N\} \wedge m \in M:$$

(i) $send\ (m) = i \wedge rec\ (m) = j \wedge i \neq j$,
(ii) $s_j \xrightarrow{?m^{i \to j}} s_j' \in \delta_j$,
(iii) $mQ'_j = Q_j$,
(iv) $\forall k \in \{1,2,\ldots,N\}: k \neq j \Rightarrow s'_k = s_k$,

(v) $\forall k \in \{1,2,\ldots,N\}$:$Q'_{\mathrm{k}} = Q_{\mathrm{k}}$.

(c) internal action

$$s \xrightarrow{\varepsilon} s' \in \delta \textit{ if } \exists i, j \in \ \{1,2,\ldots,N\} \wedge m \in M:$$

(i) $s_i \xrightarrow{\varepsilon} s_i' \in \delta_{\mathrm{i}}$,
(ii) $\forall k \in \{1,2,\ldots,N\}$: $k \neq i \Rightarrow s'_{\mathrm{k}} = s_{\mathrm{k}}$,
(iii) $\forall k \in \{1,2,\ldots,N\}$:$Q'_{\mathrm{k}} = Q_{\mathrm{k}}$.

According to Definition 9, there are three following interaction types among microservices communicating asynchronously via FIFO buffers:

(1) a send message action $s \xrightarrow{!m^{i \to j}} s'$ denotes that microservice MS_{i} sends a message m to another microservice MS_{j} where $m \in M_i$ (9a-i). After that, the state of the sender is changed (9a-ii), the message will be inserted to the tail of the *buffer*$_{\mathrm{j}}$ of the receiver (9a-iii), the other microservices' states do not change (9a-iv), and the other buffers do not change (9a-v).

(2) a receive message action $s \xrightarrow{?m^{i \to j}} s'$ denotes that microservice MS_{j} consumes a message m sent from microservice MS_{i} where $m \in M_i$ (9b-i). After that, the state of the receiver is changed (9b-ii), the message at the head of the *buffer*$_{\mathrm{j}}$ of the receiver will be consumed (9b-iii), the other microservices' states do not change (9b-iv), and the other buffers do not change (9b-v).

(3) an internal action $s \xrightarrow{\varepsilon} s'$ denotes that microservice MS_{i} executes an internal action (9c-i). After that, the other microservices' states do not change (9c-ii) and the other buffers also do not change (9c-iii).

We use $IB^k_{FIFO} = (S^k, s^k_0, F^k, A^k, \delta^k)$ to define the interaction behavior of a microservices system under the asynchronous communication model via FIFO buffers, where each buffer's bound is set to k. This IB^k_{FIFO} can be obtained from Definition 9 by allowing executing send message actions if the buffer of the receiver is not full (i.e., the buffer has less than k messages).

4.2 Modeling Interactions Among Microservices Under the Asynchronous Communication Model with Bag Buffers

Definition 10 (Bag buffer). A bag buffer is a multiset of messages.

Given a set messages{m1, m2, m3}, We often write B(m1) = 2, B(m2) = 2, and B(m3) = 1 to denote that a bag B = {m1, m1, m2, m2, m3}.

Definition 11 (Interaction behavior under the asynchronous communication model with bag buffers). An interaction behavior among a set of microservices (MS_1, MS_2, …, MS_{N}) with $MS_i = (S_i, s_i^0, F_i, A_i, \delta_i)$ and B_{i} being its associated buffer under the asynchronous communication model with bag buffers is a labeled transition system $IB^{Bag}_{Mailbox} = (S, s_0, F, A, \delta)$ where:

- $S \subseteq S_1 \times B_1 \times S_2 \times B_2 \times \ldots \times S_N \times B_N$ where $\forall i \in \{1,2, \ldots,N\}$, $B_i \subseteq M^*$ with $M = MS_1.M_1 \cup MS_2.M_2 \cup \ldots \cup MS_N.M_N$.
- $s_0 \in S$ such that $s_0 = (s_1^0, \varepsilon, s_2^0, \varepsilon, \ldots, s_N^0, \varepsilon)$.
- $F \subseteq S$.
- $A = MS_1.A_1 \cup MS_2.A_2 \cup \ldots \cup MS_N.A_N$.
- $\delta_p \subseteq S \times (A \cup \{tau\}) \times S$ for $s = (s_1, B_1, s_2, B_2, \ldots,s_n, B_n)$ and $s' = (s'_1, B'_1, s'_2, B'_2, \ldots, s'_n, B'_n)$.

(a) send message action

$$s \xrightarrow{!m^{i\to j}} s' \in \delta \ if\ \exists i, j \in \{1, 2,\ldots,N\} \wedge m \in M:$$

(i)$send\ (m) = i \wedge rec\ (m) = j \wedge i \neq j$,
(ii) $s_i \xrightarrow{!m^{i\to j}} s_i' \in \delta_i$,
(iii)$B'_j = B_j \cup \{m\}$,
(iv) $\forall k \in \{1,2,\ldots,N\}$: $k \neq i \Rightarrow s'_k = s_k$,
(v) $\forall k \in \{1,2,\ldots,N\}$:$k \neq i \Rightarrow B'_k = B_k$.

(b) receive message action

$$s \xrightarrow{?m^{i\to j}} s' \in \delta \ if\ \exists i, j \in \{1, 2,\ldots,N\} \wedge m \in M:$$

(i)$send\ (m) = i \wedge rec\ (m) = j \wedge i \neq j$,
(ii) $s_j \xrightarrow{?m^{i\to j}} s_j' \in \delta_j$,
(iii)$B'_j = B_j$-$\{m\}$,
(iv) $\forall k \in \{1,2,\ldots,N\}$: $k \neq j \Rightarrow s'_k = s_k$,
(v) $\forall k \in \{1,2,\ldots,N\}$:$B'_k = B_k$.

(c) internal action

$$s \xrightarrow{\varepsilon} s' \in \delta \ if\ \exists i, j \in \{1, 2,\ldots,N\} \wedge m \in M:$$

(i) $s_i \xrightarrow{\varepsilon} s_i' \in \delta_i$,,
(ii) $\forall k \in \{1,2,\ldots,N\}$:$k \neq i \Rightarrow s'_k = s_k$,
(iii) $\forall k \in \{1,2,\ldots,N\}$:$B'_k = B_k$.

Note that the differences between Definition 9 and Definition 11 are the condition (11a-iii) and the condition (11b-iii).

Similarly, we use $IB^k_{Bag} = (S^k, s^k_0, F^k, A^k, \delta^k)$ to define the interaction behavior of a microservices system under the asynchronous communication model via bag buffers, where each buffer's bound is set to k.

5 Comparison of Two Asynchronous Communication Models

This section compares these two asynchronous communication models defined in Sect. 3 using refinement checking [18].

Definition 12 (Trace Refinement). Let ACM_{i} where $i \in \{1,2\}$ be two asynchronous communication models, for any a set of microservices (MS_1, MS_2, …, MS_{n}), ACM_1 is a trace refinement of ACM_2 if and only if $traces(IB_1) \subseteq \mathrm{traces}(IB_2)$, where IB_1 is the interaction behavior among microservices (MS_1, MS_2, …, MS_{N}) with under the ACM_1 and IB_2 is the interaction behavior among microservices (MS_1, MS_2, …, MS_{N}) with under the ACM_2:

$$ACM_1 \prec_r ACM_2 \triangleq for\, any\, a\, set\, of\, microservcies\, (MS_1,\ MS_2, ...MS_n)\ traces(IB_1) \subseteq traces(IB_2)$$

Definition 13 (Model Inclusion). Let ACM_{i} where $i \in \{1,2\}$ be two asynchronous communication models. ACM_1 is included in ACM_2 iff ACM_1 is a trace refinement of ACM_2 and ACM_2 is not a trace refinement of ACM_1:

$$ACM_1 \subseteq ACM_2 \triangleq (ACM_1 \prec_r ACM_2) \wedge (ACM_2! \prec_r ACM_1)$$

Theorem 1 (Comparison of ACM_{FIFO} and ACM_{Bag} with regard to Refinement).

Proof. The proof follows directly from Definition 9 and 11. Since the FIFO buffers are the special case of bag buffers, for any a set of microservices (MS_1, MS_2, …, MS_{n}), $traces(IB_1) \subseteq \mathrm{traces}(IB_2)$ is always true, but not vice versa, where IB_1 is the interaction behavior among microservices (MS_1, MS_2, …, MS_{N}) with under the ACM_{FIFO} and IB_2 is the interaction behavior among microservices (MS_1, MS_2, …, MS_{N}) with under the ACM_{Bag}.

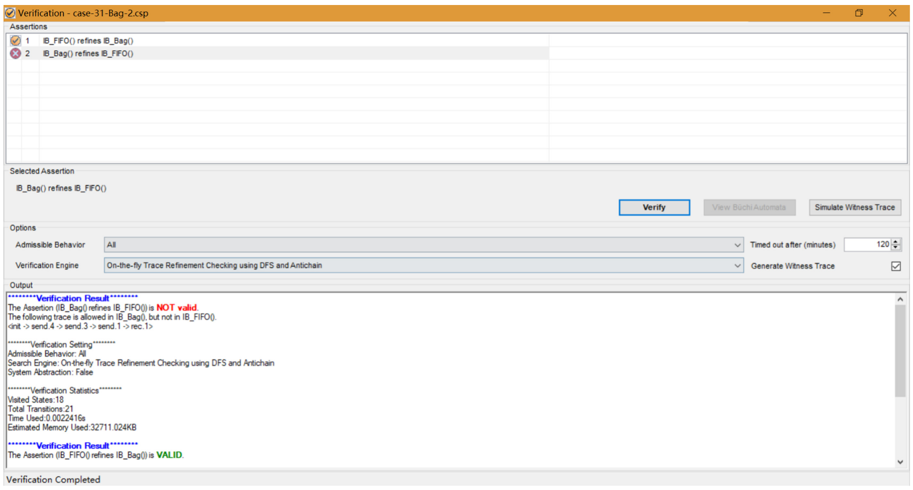
6 Implementation and Experiments

6.1 Implementation

We have implemented our approach under the support of the Process Analysis Toolkit (PAT) tool [19]. The snapshot of the comparison result of the running example using refine checking can be found in Fig. 4. This figure shows that the running example with FIFO buffers is included in that with bag buffers.

For the asynchronous communication model via FIFO buffers, we can use channel communication to implement it. More specifically, for a send message action ($s \xrightarrow{!m^{i \to j}} s'$), the operation of channel output can be written as $buffer_{\mathrm{ij}}$!m, where $buffer_{\mathrm{ij}}$ is a channel. For a receive message action ($s \xrightarrow{?m^{i \to j}} s'$), the operation of channel input can be written as $buffer_{\mathrm{ij}}$?m.

For the asynchronous communication model via bag buffers, we define a new data type *Bag* using the C# library editor and compiler in the PAT to implement it. The *Bag* has two important operations to support message emission and reception, namely *add* and *remove*. For the *add* operation $buffer_{\mathrm{ij}}.add(m)$ which is used to denote a send message action, the message m is stored in the *Bag* $buffer_{\mathrm{ij}}$ if the $buffer_{\mathrm{ij}}$ is not full yet. For the *remove* operation $buffer_{\mathrm{ij}}.reomve(m)$ which is used to denote a receive message action, the message m in the *Bag* $buffer_{\mathrm{ij}}$ is retrieved and if the buffer is not empty.



```
1  IB_FIFO() refines IB_Bag()
2  IB_Bag() refines IB_FIFO()
```

IB_Bag() refines IB_FIFO()

Admissible Behavior: All
Verification Engine: On-the-fly Trace Refinement Checking using DFS and Antichain
Timed out after (minutes): 120
Generate Witness Trace ✓

```
********Verification Result********
The Assertion (IB_Bag() refines IB_FIFO()) is NOT valid.
The following trace is allowed in IB_Bag(), but not in IB_FIFO().
<init -> send.4 -> send.3 -> send.1 -> rec.1>

********Verification Setting********
Admissible Behavior: All
Search Engine: On-the-fly Trace Refinement Checking using DFS and Antichain
System Abstraction: False

********Verification Statistics********
Visited States:18
Total Transitions:21
Time Used:0.0022416s
Estimated Memory Used:32711.024KB

********Verification Result********
The Assertion (IB_FIFO() refines IB_Bag()) is VALID.
```

Verification Completed

Fig. 4. The snapshot of comparison results of the running example

6.2 Experiments

For validating our approach, we use two datasets. The first dataset is obtained from [7], which contains more than 6 hundred examples. The second dataset contains more than 1 hundred examples, which are collecting from the literature on the close subject.

We conducted several modeling and comparing experiments on these two datasets. All the experiments were performed on the same Windows laptop running on a 2.5 GHz Intel Core i7 processor with 8 GB of memory.

Table 2 shows some of these examples, where we use FIFO buffers and bag buffers as asynchronous communication models and compare these two models using refinement checking in the PAT tool. The second column describes each case, where cases (1), (2), (3), (4) and (5) are from the first dataset and the other cases are from the second dataset. The third column shows the number of microservices in each case. The fourth column shows the number of messages exchanged among microservices. The fifth column shows the interaction behavior with buffers of size 2 under the ACM_{FIFO}(each tuple (X, Y) denotes the number of states X and transitions Y in the IB^2_{FIFO}), the interaction behavior with buffers of size 2 under the ACM_{Bag} (IB^2_{Bag}), and the comparisons of these two interaction behaviors, where " $\rightarrow$" denotes whether IB^2_{FIFO} is a trace refinement of IB^2_{Bag} and " $\leftarrow$" denotes whether IB^2_{Bag} is a trace refinement of IB^2_{FIFO}. The fifth column shows IB^3_{FIFO}, IB^3_{Bag}, and their comparisons. It should be noted that the word "large" in the fifth column means that we cannot build the state space of interaction behavior using PAT.

The experimental results show that IB^k_{FIFO} is a trace refinement IB^k_{Bag} where $k \in \{2,3\}$, but not vice versa (see, e.g., case (1), (2), (3), (5) and (6)). Furthermore, the experimental results also show that the ACM_{FIFO} with FIFO buffers is included in the ACM_{Bag} with bag buffers

Table 2. Experimental results

Id	Description	$\lvert MSs\rvert$	$\lvert MS\rvert$	$k=2$				$k=3$			
				IB^2_{FIFO}	IB^2_{Bag}	$\rightarrow$	$\leftarrow$	IB^3_{FIFO}	IB^3_{Bag}	$\rightarrow$	$\leftarrow$
(1)	EX31	3	4	35/56	35/62	$\prec_r$	$!\prec_r$	44/75	44/87	$\prec_r$	$!\prec_r$
(2)	EX38	3	3	13/15	13/16	$\prec_r$	$!\prec_r$	13/15	13/16	$\prec_r$	$!\prec_r$
(3)	EX43	4	5	large	large	$\prec_r$	$!\prec_r$	large	large	$\prec_r$	$!\prec_r$
(4)	EX152	4	6	23/31	23/31	$\prec_r$	$\prec_r$	23/31	23/31	$\prec_r$	$\prec_r$
(5)	EX155	3	6	34/47	large	$\prec_r$	$!\prec_r$	45/65	large	$\prec_r$	$!\prec_r$
(6)	Train station [20]	4	8	67/117	67/129	$\prec_r$	$!\prec_r$	94/171	94/192	$\prec_r$	$!\prec_r$
(7)	Figure 1 [8]	3	4	16/21	16/21	$\prec_r$	$\prec_r$	16/21	16/21	$\prec_r$	$\prec_r$
(8)	Booking system [21]	4	7	24/31	24/31	$\prec_r$	$\prec_r$	24/31	24/31	$\prec_r$	$\prec_r$
(9)	Online shopping [22]	3	6	13/13	13/13	$\prec_r$	$\prec_r$	13/13	13/13	$\prec_r$	$\prec_r$
(10)	Figure 8 [23]	3	3	16/20	16/20	$\prec_r$	$\prec_r$	16/20	16/20	$\prec_r$	$\prec_r$

7 Conclusions

In this paper, the interaction differrences between one asynchronous communication mod-el with FIFO buffers and another asynchronous communication model with bag buffers are discussed. First, we formally define two asynchronous communication models with FIFO or bag buffers in terms of message ordering and buffer number. Second, we model interactions among microservices as sequences of send and receive message actions under these two asynchronous communication models. Third, we compare these two asynchronous communication models using refinement checking. Our experiments show that the asynchronous communication model with FIFO buffers is included in the asynchronous communication model with bag buffers.

The future work is to study whether our proposed results are suitable for peer-to-peer communication rather than mailbox communication and compare more asynchronous communication models using refinement checking.

Acknowledgment. This work has been supported by the Project of National Natural Science Foundation of China under Grant No. 61702442 and 61862065, the Application Basic Research Project in Yunnan Province Grant No. 2018FB105, the Major Project of Science and Technology of Yunnan Province under Grant No. 202002AD080002 and No. 2019ZE005.

References

1. Qi, L., Chen, Y., Yuan, Y., Fu, S., Zhang, X., Xu, X.: A QoS-aware virtual machine scheduling method for energy conservation in cloud-based cyber-physical systems. World Wide Web **23**(2), 1275–1297 (2019). https://doi.org/10.1007/s11280-019-00684-y
2. Xiaolong, X., et al.: A computation offloading method over big data for IoT-enabled cloud-edge computing. Future Gener. Comput. Syst. **95**, 522–533 (2019). https://doi.org/10.1016/j.future.2018.12.055
3. Zhou, X., et al.: Poster: benchmarking microservice systems for software engineering research. In: Proceedings of the International Conference on Software Engineering: Companion Proceedings (ICSE 2018), pp. 323–324 (2018)
4. Zhou, X., et al.: Delta debugging microservice systems with parallel optimization. IEEE Trans. Serv. Comput. **99**, 1 (2019)
5. Zhou, X., et al.: Delta debugging microservice systems. In: Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, pp. 802–807 (2018)
6. Chevrou, F., Hurau, A., Quéinnec, P.: On the diversity of asynchronous communication. Formal Aspects Comput. **28**(5), 847–879 (2016)
7. Akroun, L., Salaün, G.: Automated verification of automata communicating via FIFO and bag buffers. Formal Methods Syst. Des. **52**(3), 260–276 (2017). https://doi.org/10.1007/s10703-017-0285-8
8. Finkel, A., Lozes, É.: Synchronizability of communicating finite state machines is not decidable. In: Proceedings of the 44th International Colloquium on Automata, Languages, and Programming ICALP 2017, vol. 122, pp. 1–14 (2017)
9. Barbanera, F., van Bakel, S., de Liguoro, U.: Orchestrated session compliance. J. Logical Algebraic Method Program. **86**(1), 30–76 (2017)
10. Bultan, T., Fu, X., Hull, R., Su, J.: Conversation specification: a new approach to design and analysis of e-service composition. ACM 1–58113–680–3/03/0005, WWW, May 20–24, Budapest, Hungary (2003)
11. Xiang, F., Bultan, T., Jianwen, S.: Conversation protocols: a formalism for specification and verification of reactive electronic services. In: Ibarra, O.H., Dang, Z. (eds.) CIAA 2003. LNCS, vol. 2759, pp. 188–200. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-45089-0_18
12. Bultan, T., Su, J., Fu, X.: Analyzing conversations of web services. IEEE Internet Comput. **10**(1), 18–25 (2006)
13. Basu, S., Bultan, T., Quederni, M.: Deciding choreography realizability. ACM SIGPLAN Notices **47**(1), 191–202 (2012)
14. Basu, S., Bultan, T.: Automated choreography repair. In: Stevens, P., Wąsowski, A. (eds.) FASE 2016. LNCS, vol. 9633, pp. 13–30. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49665-7_2
15. Ouederni, M., Salaün, G., Bultan, T.: Compatibility checking for asynchronously communicating software. In: Fiadeiro, J.L., Liu, Z., Xue, J. (eds.) FACS 2013. LNCS, vol. 8348, pp. 310–328. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-07602-7_19
16. Akroun, L., Salaün, G., Ye, L.: Automated analysis of asynchronously communicating systems. In: Bošnački, D., Wijs, A. (eds.) SPIN 2016. LNCS, vol. 9641, pp. 1–18. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-32582-8_1
17. Clemente, L., Herbreteau, F., Sutre, G.: Decidable topologies for communicating automata with FIFO and bag channels. In: Baldan, P., Gorla, D. (eds.) CONCUR 2014. LNCS, vol. 8704, pp. 281–296. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44584-6_20

18. Yang, L., Chen, W., Liu, Y.A., Sun, J., Zhang, S., Less, J.: Verifying linearizability via optimized refinement checking. IEEE Trans. Softw. Eng. **39**(7), 1018–1039 (2013)
19. Sun, J., Liu, Y., Dong, J.S.: Model checking CSP revisited: introducing a process analysis toolkit. In: Margaria, T., Steffen, B. (eds.) ISoLA 2008. CCIS, vol. 17, pp. 307–322. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-88479-8_22
20. Salaün, G., Bultan, T., Roohi, N.: Realizability of choreographies using process algebra encodings. IEEE Trans. Serv. Comput. **5**(3), 290–302 (2012)
21. Poizat, P.: Checking the realizability of BPMN 2.0 choreographies. In: Proceedings of the 27th Annual ACM Symposium on Applied Computing, pp. 1927–1934 (2012)
22. Bultan, T.: Modeling interactions of web software. In: International Workshop on Automated Specification and Verification of Web Systems, pp. 45–52 IEEE (2006)
23. Bultan, T., Chris, F., Xiang, F.: A tool for choreography analysis using collaboration diagrams. In: Proceedings of the 7th IEEE International Conference on Web Services (ICWS 2009), Los Angeles, CA, July 6–10, pp. 856–863 (2009)