Analytical Characterization and Efficient Simulation of Batched Arrivals in the Kafka Broker

András Horváth
Department of Computer Science
University of Turin
Turin, Italy
horvath@di.unito.it

Marco Paolieri
Electrical and Computer Engineering
University of Southern California
Los Angeles, USA
paolieri@usc.edu

Benedetta Picano, Enrico Vicario
Department of Information Engineering
University of Florence
Florence, Italy
{benedetta.picano,enrico.vicario}@unifi.it

***Abstract*—Apache Kafka is a key component in event-driven and microservice architectures relying on distributed publish-subscribe messaging for scalable and fault-tolerant streaming of real-time data. To reduce distribution overhead, messages are buffered and dispatched to the broker when either a maximum batch size N is reached or a timeout T expires, enabling control on the trade-off between high throughput and low latency. However, this trade-off has been explored only through empirical studies, referred to specific system deployments and not suited for runtime adaptation to variable workload conditions.**

We provide an analytical characterization of the arrival process induced by Kafka batching policy under Poisson arrivals. The analysis develops on the observation that the time for buffering a full batch and the size of a batch dispatched at expiration of the timeout follow truncated Erlang and Poisson distributions, respectively. Leveraging this insight, we derive closed forms for quantities that characterize the arrival process, and we propose a method for efficient simulation of the process embedded at dispatching times. To support practical implementation, we evaluate solutions for drawing samples from an Erlang distribution provided by NumPy, PyTorch, and R, and we also propose a novel approach based on rejection sampling with proposal function in the family of Kumaraswamy distributions with automated optimization of parameters with respect to the number of phases.

Numerical experimentation shows that: (i) aggregated simulation enabled by the analytical formulation is insensitive to the value of the batch size, and it definitely outperforms fine-grained simulation of individual message arrivals; (ii) the best efficiency is obtained with the NumPy implementation of Erlang, with promising results of the novel approach based on Kumaraswamy, which achieves results comparable to PyTorch and better than R.

***Index Terms*—Kafka Broker, Batched Arrival Process, Rejection Sampling, Kumaraswamy Distribution**

I. Introduction

Apache Kafka [1] is a major platform supporting scalable and fault tolerant streaming of real-time data for a large variety of systems, including data pipelines and event-driven and microservice architectures [2]–[5]. The *Kafka broker* implements the publish-subscribe pattern, in pull-mode, receiving and storing messages from producers, and serving them upon consumers' request. To achieve high throughput and low latency, Kafka producers collect messages into batches before sending them to the Kafka broker. In addition, to ensure low latency, a timeout is used to stop waiting for additional messages and send a batch before it is full; in particular, the configuration parameters `batch.size` and `linger.ms` control the maximum batch size and the timeout from the reception of the first message in a batch, respectively. Kafka exposes a wide array of other tunable parameters, such as buffer memory, message compression, and acknowledgement policies, all of which interact in subtle ways to influence throughput, latency, and reliability. Due to this complexity, most existing studies have relied heavily on simulation, often building customized Kafka clusters or synthetic workloads to explore the design space [6]–[8]. While this body of literature has provided valuable empirical insights, the mechanism resulting from the interaction of batch size and timeout was never made explicit in a closed form supporting predictive evaluation of the effects of the joint setting of the two parameters, by analytic formulation or by efficient simulation able to scale up to volumes involved in practical deployments.

In this paper, we address this gap by proposing an analytical characterization of the interplay between batch accumulation and timeout-triggered dispatch, with specific reference to characteristics and usage scenarios of the Kafka Broker, but also open to adaptation for cloud or other systems where aggregation is driven by the trade-off between throughput and latency [9], [10]. The analysis develops on the observation that the time for buffering a full batch follows a truncated Erlang distribution with shape parameter equal to the batch size, while the size of a batch dispatched at expiration of the timeout follows a truncated Poisson distribution.

Leveraging this insight, we derive closed forms for quantities that characterize the arrival process, and we propose an efficient aggregated simulation method that observes the process at dispatching times so as to avoid fine-grained simulation of individual message arrivals. To support practical and efficient implementation of the proposed approach, we evaluate solutions for drawing samples from an Erlang distribution provided by NumPy, PyTorch, and R, and we compare them against the simulation of Erlang distribution by summation of Exponential samples, which is commonly applied in various contexts

This work was partially supported by the European Union under the Italian National Recovery and Resilience Plan (NRRP) of NextGenerationEU, partnership on "Telecommunications of the Future" (PE00000001 - program "RESTART")

[11] [12] [13], but becomes extremely inefficient when the number of phases is as high as the batch size of practical deployments [6] [7]. We also propose a novel approach based on rejection sampling with proposal function in the family of Kumaraswamy distributions with automated optimization of parameters through a multi-parameter *Differential Evolution* (DE) algorithm [14], [15].

Numerical experimentation shows that: (i) aggregated simulation enabled by the analytical formulation is insensitive to the value of the batch size, and it definitely outperforms fine-grained simulation of individual message arrivals (ii) the best efficiency is obtained with the NumPy implementation of Erlang, with promising results of the novel approach based on Kumaraswamy, which achieves results comparable to PyTorch and improving on R.

In the rest of the paper, after a review of related works in Section II, the analytical formulation of the Kafka arrival process is developed in Section III, and the method based on rejection sampling with a Kumaraswamy proposal is presented in Section IV. Numerical experimentation comparing different implementations of the simulation approach is reported in Section V, followed by conclusions in Section VI.

II. Related Work

Several studies have addressed performance engineering of the Kafka platform, focusing on trade-offs between key system parameters such as batch size, timeout configuration, and latency, to achieve objectives of throughput and latency, under different workload conditions.

In [6], accurate predictions of throughput, latency, and disk usage over time are obtained for Kafka cloud services by combining queueing theory and phase-type approximation of empirical observations. The influence of batch size on Kafka's performance under variable network and system conditions is the focus of [7], with reference to a Docker deployment of Kafka. The study remarks the importance of end-to-end latency variability. Given the impact of batch size on message loss rates, the authors propose a reactive batching strategy open to adaptation for different network conditions. The impact of batching strategies on throughput and latency in distributed stream processing subject to varying system conditions is studied in [16], to propose a lightweight control algorithm for dynamic adjustment of batch sizes based on runtime system metrics.

Regarding the generation of pseudo-random samples of gamma (and Erlang) distributions, [17] surveys and classifies algorithms based on acceptance-rejection, series expansion, and transformation techniques, highlighting the trade-offs involved, particularly when dealing with extreme values of the shape parameter. Efficient sampling from classical statistical distributions, including gamma, beta, Poisson, and binomial, is investigated in [18]. The work introduces algorithmic strategies to improve precision and efficiency, with tailored methods for each distribution that leverage transformation and rejection sampling. The problem of generating random samples from phase-type distributions is addressed in [19] [13] [12] [20], with matrix-structure-based algorithms [19] exploiting the embedded Markov chain structure, or with summation of exponential samples (obtained by logarithms on the product of uniform samples) [13] [12].

Rejection sampling for the simulation of gamma distributions for shape parameters $\alpha \geq 1$ is addressed in [21]. The inefficiency of classical rejection sampling methods for complex distributions is tackled in [22], which introduces partial rejection sampling as a more efficient alternative based on partial information about the target distribution. Efficient gamma sampling for $\alpha \geq 1$ is also explored in [21], which introduces an optimized rejection sampling scheme with a carefully constructed proposal distribution.

Usage of Kumaraswamy distribution is addressed in [23] for quantile regression for bounded variables and in [24] with reference to numerical instabilities in the inverse CDF and logarithmic Probability Density Function (PDF). The generalization of the Kumaraswamy distribution is the focus of [25], where the Kumaraswamy-G family is introduced to improve adaptability to skewed and heavy-tailed data.

III. Analytical Characterization

We consider a Kafka-like message generation system, where a producer transmits data to a broker using a dual-trigger batching policy. The producer buffers incoming messages into a batch stored in memory. A batch is dispatched to the broker as soon as either one of the following conditions is met: the number of buffered messages reaches a fixed batch size N; or a timeout T elapses.

For the start of the timeout, we consider two policies: the timeout clock can be started (1) after the dispatch of the previous batch, or (2) after the first arrival following the dispatch of the previous batch (i.e., after the first message of the current batch arrives). In the first case, the time between subsequent batches cannot exceed T but there can be empty batches (notifying that no message has arrived in T time units). In the second case, every batch contains at least one message but the interbatch time can exceed T. In both cases, a message cannot be held in the buffer for a time longer than T.

The analysis of the two cases uses similar technical steps and differences are negligible when the timeout is much higher than the expected time between message arrivals, as in most practical applications. Due to space limitations, we analyze the first case in Section III-A and briefly discuss changes required for the second case in Section III-B.

A. Timeout Clock Starts at Batch Dispatch

We assume that message arrivals follow a homogeneous Poisson process with rate λ, a standard assumption in the modeling of aggregate traffic in distributed systems. The number of messages in a batch and the associated interbatch time will be denoted by the random variables η and ζ, respectively.

The number of message arrivals over an interval of length T follows the Poisson distribution with parameter λT. Considering also the fact that if N messages arrive before the timeout clock exceeds T then a batch of N messages is dispatched, the

number of messages in a batch is the minimum of a Poisson distributed random variable with parameter λT and N, that is,

$$\eta = \min\{\text{Poisson}(\lambda T), N\}$$

and its Probability Mass Function (PMF) is given as

$$p_\eta(k) = P(\eta = k) = \begin{cases} \dfrac{(\lambda T)^k e^{-\lambda T}}{k!} & \text{if } k < N\,, \\ \displaystyle\sum_{i=N}^{\infty} \dfrac{(\lambda T)^i e^{-\lambda T}}{i!} & \text{if } k = N\,. \end{cases} \tag{1}$$

Similarly, the interbatch time is less than T if N messages arrive before the timeout clock exceeds T and in this case its distribution follows the Erlang distribution with parameters λ and N. Otherwise it is equal to T. Accordingly,

$$\zeta = \min\{\text{Erlang}(\lambda, N), T\}$$

with PDF

$$\begin{aligned} f_\zeta(t) &= f_{Erl}(\lambda, N, t) I_{0<t<T} + (1 - F_{Erl}(\lambda, N, T))\delta(t - T) \\ &= \frac{\lambda^N t^{N-1} e^{-\lambda t}}{(N-1)!} I_{0<t<T} + \sum_{i=0}^{N-1} \frac{(\lambda T)^i e^{-\lambda T}}{i!} \delta(t - T) \end{aligned} \tag{2}$$

where f_{Erl} and F_{Erl} are the PDF and the CDF of the Erlang distribution, respectively, I is the indicator function and δ is the Dirac delta function.

The relation between the number of messages and the inter-batch time is $\eta < N \iff \zeta = T$ and, vice versa, $\eta = N \iff \zeta < T$, which implies that

$$P(\eta = N) = \sum_{i=N}^{\infty} \frac{(\lambda T)^i e^{-\lambda T}}{i!} = F_{Erl}(\lambda, N, T) = P(\zeta < T)\,. \tag{3}$$

In turn, Eq. (3) can be calculated efficiently in most computing environments through the *regularized lower incomplete Gamma function* Q defined as:

$$Q(N, \lambda T) := \frac{\gamma(N, \lambda T)}{\Gamma(N)} = \frac{\int_0^{\lambda T} t^{N-1} e^{-t} dt}{\int_0^{\infty} t^{N-1} e^{-t} dt} = \sum_{i=N}^{\infty} \frac{(\lambda T)^i e^{-\lambda T}}{i!}$$

where $\gamma(N, \lambda T)$ is the lower incomplete Gamma function and $\Gamma(N)$ is the Gamma function ($\Gamma(N) = (N-1)!$ for integer N).

From a simulation perspective, pairs (η, ζ) of batch size and inter-batch time can be generated as follows. A uniform random value $u \sim \mathcal{U}(0, 1)$ can be used to determine how the next batch will be triggered:

- If $u \leq P(\eta = N)$, then the batch size is $\eta = N$ and the inter-batch time ζ is generated by sampling the Erlang(λ, N) distribution truncated over $[0, T]$.
- If $u > P(\eta = N)$, then the inter-batch time is $\zeta = T$ and the batch size is generated by sampling the Poisson(λT) distribution truncated over $0, 1, ..., N-1$.

The mean batch size $\mathbb{E}[\eta]$ and the mean inter-batch time $\mathbb{E}[\zeta]$ can be computed based on Eq. (1) and Eq. (2):

$$\begin{aligned} \mathbb{E}[\eta] &= \lambda T(1 - Q(N-1, \lambda T)) + N\,Q(N, \lambda T)\,, \\ \mathbb{E}[\zeta] &= \frac{N}{\lambda} Q(N+1, \lambda T) + T(1 - Q(N, \lambda T))\,. \end{aligned}$$

Proof. Using Eq. (1),

$$\begin{aligned} \mathbb{E}[\eta] &= \sum_{k=1}^{N-1} k \frac{(\lambda T)^k e^{-\lambda T}}{k!} + N \sum_{k=N}^{\infty} \frac{(\lambda T)^k e^{-\lambda T}}{k!} \\ &= \lambda T \sum_{k=1}^{N-1} \frac{(\lambda T)^{k-1} e^{-\lambda T}}{(k-1)!} + N\,Q(N, \lambda T) \\ &= \lambda T \sum_{k=0}^{N-2} \frac{(\lambda T)^k e^{-\lambda T}}{k!} + N\,Q(N, \lambda T) \\ &= \lambda T(1 - Q(N-1, \lambda T)) + N\,Q(N, \lambda T)\,. \end{aligned}$$

Using Eq. (2),

$$\begin{aligned} \mathbb{E}[\zeta] &= \int_0^T t f_{Erl}(\lambda, N, t)\, dt + T \sum_{i=0}^{N-1} \frac{(\lambda T)^i e^{-\lambda T}}{i!} \\ &= \frac{N}{\lambda} \int_0^T \frac{\lambda^{N+1} t^N e^{-\lambda t}}{N!}\, dt + T(1 - Q(N, \lambda T)) \\ &= \frac{N}{\lambda} F_{Erl}(\lambda, N+1, T) + T(1 - Q(N, \lambda T)) \\ &= \frac{N}{\lambda} Q(N+1, \lambda T) + T(1 - Q(N, \lambda T))\,. \end{aligned}$$

□

The probability that a randomly selected message belongs to a batch with k messages, denoted by L_k, due to a mechanism analogous to that of the *inspection paradox*, is not equal to the probability that a batch contains k messages, i.e., $P(\eta = k)$. L_k is instead given by

$$L_k = \frac{k P(\eta = k)}{\mathbb{E}[\eta]}\,.$$

Table I reports $P(\eta = k)$ and L_k with $N = 8$ where $L_k > P(\eta = k)$ for $N \geq 5$, illustrating that messages are more likely to belong to larger batches.

Next, we study the time elapsed between the arrival of a message and the dispatch of its batch. This quantity is strongly impacted by the position of the message within its batch and the size of the batch. Therefore, we denote by $U_{i,j}$ the elapsed time between message arrival and batch dispatch for a message that is the jth message in a batch of size i.

For $i < N$ (i.e., when the dispatch is triggered by the timeout), assuming Poisson arrivals, the i messages are distributed uniformly over the interval $[0, T]$. Furthermore, the larger the value of j ($1 \leq j \leq i$), the less time the corresponding message has to wait until dispatch. The PDF of $U_{i,j}$, denoted by $f_U(i, j, x)$, can then be obtained from the j-th order statistic of a batch of i samples with uniform distribution, which is distributed as $X \sim \text{Beta}(j, i - j + 1)$, by mapping X to $[0, T]$ and subtracting the result from T (which is equivalent to swapping x/T and $1 - x/T$ in the Beta PDF):

$$f_U(i, j, x) = \frac{1}{T} \cdot \frac{\left(1 - \frac{x}{T}\right)^{j-1} \left(\frac{x}{T}\right)^{i-j}}{(j-1)!\ (i-j)!\ /\ i!}\,.$$

For $i = N$ and $j = N$, $U_{N,N} = 0$ since the dispatch is triggered by the message arrival. For $i = N$ and $j < N$, the

k	0	1	2	3	4	5	6	7	8
$P(\eta = k)$	0.0067	0.0337	0.0842	0.1404	0.1755	0.1755	0.1462	0.1044	0.1333
L_k	0	0.0069	0.0345	0.0863	0.1439	0.1799	0.1799	0.1499	0.2187

TABLE I: $P(\eta = k)$ and L_k with $\lambda = 1, T = 5, N = 8$

previous $N-1$ arrival times follow a uniform distribution over an interval whose distribution is Erlang(λ, N) truncated over the interval $[0, T]$. It follows that for $j < N$ we have

$$f_U(N, j, x) = \int_x^T \frac{f_{Erl}(N, \lambda, t)}{F_{Erl}(N, \lambda, T)} \cdot \frac{1}{t} \cdot \frac{\left(1 - \frac{x}{t}\right)^{j-1} \left(\frac{x}{t}\right)^{N-1-j}}{\frac{(j-1)!\ (N-j-1)!}{(N-1)!}} dt$$
$$= \frac{e^{-\lambda x} \lambda (\lambda x)^{N-1-j}\ Q(j, \lambda(T-x))}{(N-1-j)!\ Q(N, \lambda T)}.$$

Finally, individual message arrival times can also be easily generated from pairs (η, ζ) of batch size and interbatch time:

- If $\eta < N$ and $\zeta = T$, then generate η uniform random variables on $[0, T]$.
- If $\eta = N$ and $\zeta < T$, then the last arrival is at ζ, while the others can be obtained by generating $N - 1$ uniform random variables on $[0, \zeta]$.

B. Timeout Clock Starts at First Message Arrival

In this case, every batch contains at least one message, which arrives after an exponentially distributed delay, which starts the timeout. Dispatch is then triggered either by the arrival of an additional $N - 1$ messages or by the timeout. Accordingly, the same quantities derived in Section III-A can now be obtained by modifications following these redefined equations:

$$\eta = 1 + \min\{\text{Poisson}(\lambda T), N - 1\} \quad (4)$$
$$\zeta = \text{Exp}(\lambda) + \min\{\text{Erlang}(\lambda, N - 1), T\} \quad (5)$$

IV. Efficient Simulation of the Arrival Process

The analytical formulation presented in Section III opens the way to an *aggregated simulation* scheme, where samples are drawn from closed-form distributions of the batch-level dynamics of the Kafka broker: for each batch, a Bernoulli sample with weights identified by Eq. (3) pre-selects whether the batch should be full or the timeout should expire, which results, respectively, in either (*i*) a truncated-Poisson increment of the count of dispatched messages with deterministic advancement of time (by the timeout), or in (*ii*) a deterministic increment of messages (by the batch size) with truncated Erlang advancement of time. This approach avoids the fine-grained stochastic simulation of individual message arrivals, with major reduction of computational complexity and substantial immunity with respect to the batch size. To effectively exploit this approach, an efficient solution is required to generate samples from the truncated Erlang that characterizes the interarrival time of full batches.

To benchmark the proposed approach, we implemented the aggregated simulation using a rejection approach where the truncated Erlang distribution is simulated by repeatedly drawing a sample from the distribution Erlang(λ, N) until obtaining a value lower than T. In turn, the generation of Erlang samples was performed using alternative baseline implementations:

- The *Exponential Sum* sampler draws N independent exponential samples with rate λ (using the inverse transform method, i.e., $-\log(1 - U)/\lambda$) and returns their sum.
- The *NumPy Gamma* method uses `np.random.gamma` to directly sample from the target distribution (using Marsaglia and Tsang's gamma method [26], with normal samples generated with the ziggurat method [27]).
- The *R Gamma* method uses the `rgamma` function from the R environment via `rpy2` bindings (using the method of [28] to generate gamma samples with shape greater than 1 from exponential samples obtained with [29]).
- The *PyTorch Gamma* method leverages `torch.distributions.Gamma` (implementing Marsaglia and Tsang's gamma method [26], with normal samples generated using the Box-Muller transformation [30]).

In addition, we also experimented with a novel approach:

- The *Kumaraswamy-based rejection sampling* implements a rejection approach, using the Kumaraswamy distribution as proposal function: the *target distribution* is Erlang(λ, N) truncated over $[0, T]$; candidate samples are generated by inverse transform from a scaled Kumaraswamy distribution with support $[0, T]$, where the parameters a, b and the overscaling factor p are pre-optimized offline using a Differential Evolution optimizer (implemented in `scipy.optimize`) for each configuration of parameters N and λT. The acceptance test is implemented using the `scipy.stats.gamma` package for evaluation of the target truncated Erlang PDF.

More details on the latter approach are provided in the next two subsections.

A. Kumaraswamy-based Rejection Sampling

The Kumaraswamy distribution is defined by the PDF

$$f_{KS}(x; a, b) = a\, b\, x^{a-1}(1 - x^a)^{b-1}, \quad x \in [0, 1], \quad (6)$$

where a and b are positive-valued shape parameters that permit ductile fit of a large variety of shapes. As a crucial trait, the Kumaraswamy distribution has a closed-form inverse CDF

$$F_{KS}^{-1}(u; a, b) = \left(1 - (1 - u)^{1/b}\right)^{1/a}, \quad u \in [0, 1], \quad (7)$$

which enables fast and efficient sample generation via inverse transform sampling. Closed-form invertibility and shape ductility make it a natural candidate as a proposal distribution for rejection sampling over bounded domains.

In our case, the target is the Erlang distribution truncated over $[0, T]$, which is proportional to the following function:

$$f_{TrErl}(x; N, \lambda) = \frac{\lambda^N x^{N-1} e^{-\lambda x}}{\Gamma(N) \cdot F_{Erl}(T; N, \lambda)}, \quad x \in [0, T], \quad (8)$$

and rejection sampling can be implemented through the following standard algorithm:

1) Draw m uniform samples $u_i \sim \mathcal{U}(0, 1)$.
2) Transform them using the inverse CDF of the Kumaraswamy: $x_i = \left(1 - (1 - u_i)^{1/b}\right)^{1/a}$.
3) Evaluate the acceptance probability:

$$\alpha_i = \frac{f_{TrErl}(x_i; N, \lambda)}{p \cdot f_{KS}(x_i; a, b)}.$$

4) Accept the sample x_i if $v_i < \alpha_i$, with $v_i \sim \mathcal{U}(0, 1)$.

In this implementation, the support of Kumaraswamy distribution is remapped through a change of variable so that the inverse CDF ranges within an interval $[x_{\text{left}}, x_{\text{right}}]$ identified so that the target truncated Erlang density is lower than a threshold ϵ for any $x < x_{\text{left}}$ and for any $x > x_{\text{right}}$:

$$x_{\text{left}} = Q_{Erl}(\varepsilon),$$
$$x_{\text{right}} = \min\left(Q_{Erl}(1 - \varepsilon),\ T\right),$$

where $Q_{Erl}(p)$ denotes the quantile function of the complete Erlang distribution on $[0, \infty)$ and $\varepsilon \ll 1$ is a small threshold. Due to the restricted support of the proposal function, the simulated distribution will be exactly equal to 0 for any $x \notin [x_{\text{left}}, x_{\text{right}}]$. Unit measure within the significant interval $[x_{\text{left}}, x_{\text{right}}]$ is still guaranteed by the general properties of rejection sampling and the threshold ϵ permits the error be controlled to be as small as required. As a major advantage, the restriction permits the shape parameters a, b and the overscaling factor p be optimized to obtain a tighter fit of the target distribution, and thus a higher acceptance rate, in the interval where this concentrates its mass of probability. Note that an exact implementation of this focusing approach might be obtained by the adoption of a piecewise proposal function, with Kumaraswamy covering the core body of the target distribution, and other (invertible) distributions covering the intervals $[0, x_{\text{left}}]$ and $[x_{\text{right}}, T]$.

As a detail crucial for efficiency, note that in this implementation all the steps are fully vectorized using NumPy, avoiding explicit Python loops. Besides, for numerical precision, the truncated Erlang density is evaluated in logarithmic form, to avoid underflows for high values of the Erlang shape parameter N (as occurring in practical application where N is equal to the batch size):

$$\begin{aligned}\log\left(f_{TrErl}(x;\ N, \lambda)\right) = &\ \left((N-1) \cdot \log(x)\right) + \left(N \cdot \log(\lambda)\right) \\ &- (\lambda \cdot x) - \log\left(\Gamma(N)\right) - \log\left(F_{Erl}(T; N, \lambda)\right)\end{aligned} \quad (9)$$

where constants $\log(\lambda)$, $\log(\Gamma(N))$, and $\log(F_{Erl}(T; N, \lambda))$ can be precomputed once per run for efficiency.

B. Optimization of Proposal Parameters

To maximize the efficiency of the rejection sampling method, we tune the parameters of the Kumaraswamy proposal distribution to best match the truncated Erlang target. The goal is to find the optimal shape parameters a and overscaling factor p such that the acceptance probability is maximized while ensuring that the proposal dominates the target over the interval of interest.

Given the desired mode of the proposal Kumaraswamy function, we determine the corresponding b parameter using the following relation:

$$b = \frac{\text{mode}^{-a}(-1 + a + \text{mode}^a)}{a}. \quad (10)$$

The proposal is then fully defined by (a, mode, p).

a) Feasibility Domain and Penalty Term: To ensure that the proposal remains above the target density in the most relevant region, we numerically identify an interval $[x_1, x_2] \subset [0, 1]$ where the truncated Erlang PDF is significantly non-zero. The loss function to be minimized is defined as:

$$\mathcal{L}(a, p) = -\mathbb{E}[\text{acceptance rate}] + \text{penalty}, \quad (11)$$

where the penalty term imposes a large cost when the proposal falls below the target on $[x_1, x_2]$. This guarantees the correctness of the rejection sampler.

b) Optimization Strategy: Optimization uses the `differential_evolution` algorithm (in the SciPy implementation), a global evolutionary method for non-convex landscapes [14], [15]. To further improve robustness, we scan across small shifts in the mode of the Erlang target, adjusting the Kumaraswamy shape accordingly. For each candidate mode shift, the optimizer explores the parameter space, based on the shape of the target function: $p \in [1.0, 2.0]$, $a \in [1, 30]$. The best configuration is selected as the one that maximizes the acceptance rate while maintaining full coverage of the target. The resulting proposal shows valuable overlap with the truncated Erlang and ensures high sampling efficiency.

C. Symmetric Proposal: A Simplified Strategy

An alternative to optimizing a Kumaraswamy proposal over the entire significant interval $[x_{\text{left}}, x_{\text{right}}]$ is to focus solely on the subinterval $[x_{\text{mode}}, x_{\text{right}}]$. The goal is to construct a one-sided proposal $g(x)$ that dominates $f_{TrErl}(x; N, \lambda)$ in this region, and then reflect it around the mode to obtain a symmetric proposal $g_{sym}(x)$ over the whole subinterval $[x_{\text{left}}, x_{\text{right}}]$.

The symmetric proposal can be built as follows:

1) Optimize via the `differential_evolution` algorithm the Kumaraswamy parameters (a, p) such that:

$$g(x) \geq f_{TrErl}(x; N, \lambda), \quad \forall x \in [x_{\text{mode}}, x_{\text{right}}].$$

2) Define the symmetric proposal by reflecting $g(x)$ around x_{mode}:

$$g_{sym}(x) = \begin{cases} g(x), & \text{if } x \geq x_{\text{mode}}, \\ g(2x_{\text{mode}} - x), & \text{if } x < x_{\text{mode}}. \end{cases}$$

3) Verify that $g_{sym}(x) \geq f_{TrErl}(x; N, \lambda)$ for all $x \in [x_{\text{left}}, x_{\text{right}}]$.

This strategy reduces the complexity of the optimization, guarantees symmetry by design, and provides a valid proposal over the entire interval of interest.

V. Performance Evaluation

We evaluate feasibility and efficiency of the proposed method under different implementation choices, with reference to deployment settings identified in the literature [6], [7]. The evaluation is structured around four research questions:

- **Q1:** Does the proposed aggregated simulation correctly reproduce the expected distributions under full batch and timeout limiting conditions?
- **Q2:** How much does the proposed aggregated simulation (enabled by Section III) improve efficiency compared to a message-level simulation?
- **Q3:** How do different samplers of truncated Erlang compare in terms of accuracy and computational efficiency?
- **Q4:** In the implementation of the novel rejection sampling based on Kumaraswamy proposal function, how effectively does the optimization algorithm adapt the parameters of the proposal distribution as the batch size increases, and how do different construction strategies, i.e., range-based optimization versus symmetric reflection, affect overall sampling performance?

Unless otherwise noted, experiments are conducted using an Erlang(N, λ) distribution with rate λ adjusted to set the mode $(N-1)/\lambda$ to 10, timeout equal to $T = 12$, and a batch size of $N = 1000$. Results are averaged over 500 independent iterations to ensure statistical robustness. The maximum coefficient of variation obtained for the Kumaraswamy total time is 0.02066, and the coefficient of variation of the same method for the accuracy is 0.0094. Both values are based on the Kumaraswamy-based rejection sampling with dominant proposal. Where not explicitly specified, the term Kumaraswamy used in plots and in the discussion of Research Questions Q2 and Q3 refers interchangeably to the Kumaraswamy-based rejection sampling method with dominant proposal.

To answer **Q1**, we compare results obtained through fine-grained stochastic simulation reproducing the arrival of single messages against theoretical closed forms derived in the analytical characterization of Section III. In particular, we evaluate distributions under the two limiting conditions: batch flush triggered by reaching the maximum size N, and batch flush triggered by a timeout T. Results are illustrated for visual inspection in Fig. 1, and developed in Tables II and III with quantitative metrics of accuracy expressed in terms of Mean Absolute Error (MAE) and Kullback-Leibler divergence (KL). Results indicate that time behavior emerging from fine-grained stochastic simulation converges to theoretical forms.

For **Q2**, we use the *Exponential summation* sampler as baseline solution that reproduces the number of elementary samples that would be needed in a fine-grained simulation at the level of individual messages, and we compare its execution time against those of the methods that fully exploit aggregated simulation by NumPy Gamma, RGamma, PyTorch Gamma, and Kumaraswamy with dominant proposal samplers. As shown in Fig. 2-(top), all the samplers NumPy Gamma, RGamma, PyTorch Gamma, and Kumaraswamy are by far more efficient than the exponential summation, with a gain increasing with the batch size.

For **Q3**, in Fig. 2-(bottom), we compare the time per valid sample for NumPy Gamma, RGamma, PyTorch Gamma, and Kumaraswamy samplers, for different values of the batch size. To evaluate the potential error introduced by focusing the optimization of the proposal distribution only on the significant region of the truncated Erlang target, we compute the KL divergence between the optimized Kumaraswamy proposal and the target distribution. KL divergence values for other benchmark methods are also reported in Table IV.

For **Q4**, we analyze how the optimization procedure adjusts the parameters of the Kumaraswamy distribution for variable values of the batch size. Results, not reported here for space limitations, indicate that the algorithm successfully adjusts these parameters to maintain a stable acceptance rate around $59-60\%$. To highlight the difference between the dominant and the symmetric forms of the proposal function, Fig. 3 illustrates their different optimized shapes with respect to a target Erlang distribution: the symmetric proposal effectively reduces the deviation on the left side of the curve, i.e., the proposal optimized over the full interval $[x_{left}, x_{right}]$. This improvement is mainly consequent to the asymmetric nature of the Kumaraswamy distribution in the dominant strategy, which results in a looser fit on the left side of the mode, which in turn hinders the acceptance rate in rejection sampling. This difference results in a significant improvement of the acceptance rate during rejection sampling, with values ranging with the batch size (i.e. the number of phases) in 0.76-0.78 for the symmetric version and in 0.66-0.67 for the dominant implementation. The symmetric approach yields consistently higher acceptance due to its tighter adherence to the target distribution, particularly on the left side, where the dominant proposal exhibits a noticeable mismatch.

Finally, Fig. 4 compares the total sampling time of the library methods and the two Kumaraswamy-based rejection strategies as a function of the timeout. The symmetric Kumaraswamy-based proposal achieves performance comparable to PyTorch, while the version adopting the dominant proposal shows slightly higher time. Both approaches outperform the R-based `rgamma` method, and the NumPy Gamma implementation remains the most efficient across all timeout values. For the sake of readability, the method based on the exponential sampler is not included, as its execution time is significantly higher and would have distorted the visual representation. All timings were measured under the same number of accepted samples (1000).

The proposed method offers a favorable balance between computational efficiency and statistical accuracy. It maintains a stable acceptance rate, adapts well to different system configurations, with performance comparable to well-established

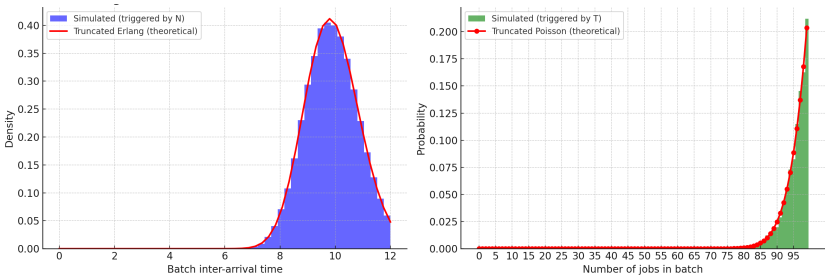

Fig. 1: Example histograms obtained by the aggregated sampling method for truncated Erlang (left) and (discrete) truncated Poisson distributions (right)

TABLE II: Accuracy Metrics for Erlang Interarrival Time

Simulated Batches	KL Mean	KL Std	MAE Mean	MAE Std
1000	1.0923e-02	1.46e-03	8.0031e-03	1.83e-03
5000	4.6801e-03	5.95e-04	5.4591e-03	4.37e-04
10000	3.7706e-03	5.60e-04	5.6346e-03	6.07e-04
20000	3.1827e-03	6.97e-04	5.0524e-03	5.32e-04
50000	2.7479e-03	2.28e-04	4.9234e-03	3.80e-04

TABLE III: Accuracy Metrics for Poisson Batch Size

Simulated Batches	KL Mean	KL Std	MAE Mean	MAE Std
1000	3.5304e-01	1.68e-01	5.6574e-03	1.66e-03
5000	8.2434e-02	3.16e-02	2.5084e-03	6.37e-04
10000	4.8381e-02	1.23e-02	1.8770e-03	3.00e-04
20000	2.8218e-02	5.39e-03	1.5364e-03	1.90e-04
50000	1.1594e-02	1.95e-03	8.2649e-04	1.05e-04

TABLE IV: KL Divergence vs. Truncated Erlang

Method	KL Divergence for N=1500
Kumaraswamy	0.002340
Exponential Sum	0.000175
NumPy Gamma	0.000100
R Gamma	0.000224
PyTorch Gamma	0.000135

implementations such as `rgamma` and PyTorch, making it a practical and promising solution for real-world sampling tasks.

VI. Conclusion

We introduced a theoretical and computational framework for modeling the message generation process in Kafka-like systems. Assuming Poisson arrivals, we formally characterized the dual-trigger batching mechanism, which depends on both a maximum batch size and a timeout. This results in a mixed interarrival time distribution with a truncated Erlang component with a fixed timeout threshold. To achieve efficiency in sampling from Erlang distributions with large shape parameters, we proposed a lightweight rejection sampling method. This approach exploits analytical invertibility and shape flexibility of the Kumaraswamy distribution, here used as proposal function, with parameters optimized by a multi-parameter differential evolution algorithm. Two optimization strategies are considered: a dominant proposal, designed to upper-bound the target over the entire region of interest; and a symmetric proposal, built by optimizing the right side of the mode and reflecting it to construct a tighter overall fit. Our experimental evaluation confirmed that the proposed method matches the accuracy of state-of-the-art implementations in NumPy, R, and PyTorch, while significantly outperforming the classic exponential sampler in terms of computational efficiency. Notably, the symmetric formulation attains sampling times that are on par with PyTorch and clearly outperforms the `rgamma` function from R.

References

[1] J. Kreps, "Kafka: A distributed messaging system for log processing," in *NetDB Workshop*, 2011.

[2] S. Vyas, R. K. Tyagi, C. Jain, and S. Sahu, "Performance evaluation of Apache Kafka – A modern platform for real time data streaming," in *Proceedings of ICIPTM*, vol. 2, 2022, pp. 465–470.

[3] G. van Dongen and D. Van den Poel, "Evaluation of stream processing frameworks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 8, pp. 1845–1858, 2020.

[4] B. Bejeck, *Kafka Streams in Action: Real-time Apps and Microservices with the Kafka Streams API*. Manning, 2018.

[5] S. Park and J.-H. Huh, "A study on big data collecting and utilizing smart factory based grid networking big data using Apache Kafka," *IEEE Access*, vol. 11, pp. 96 131–96 142, 2023.

[6] H. Wu, Z. Shang, and K. Wolter, "Performance prediction for the Apache Kafka messaging system," in *Proceedings of HPCC/SmartCity/DSS*, 2019, pp. 154–161.

[7] H. Wu, Z. Shang, G. Peng, and K. Wolter, "A reactive batching strategy of Apache Kafka for reliable stream processing in real-time," in *Proceedings of ISSRE*, 2020, pp. 207–217.

[8] K. Goodhope, J. Koshy, J. Kreps, N. Narkhede, R. Park, J. Rao, and V. Y. Ye, "Building LinkedIn's real-time activity data pipeline," *IEEE Data Eng. Bull.*, vol. 35, no. 2, pp. 33–45, 2012.

[9] Google, "Pub/Sub: Batch Messaging," https://cloud.google.com/pubsub/docs/batch-messaging, 2025.

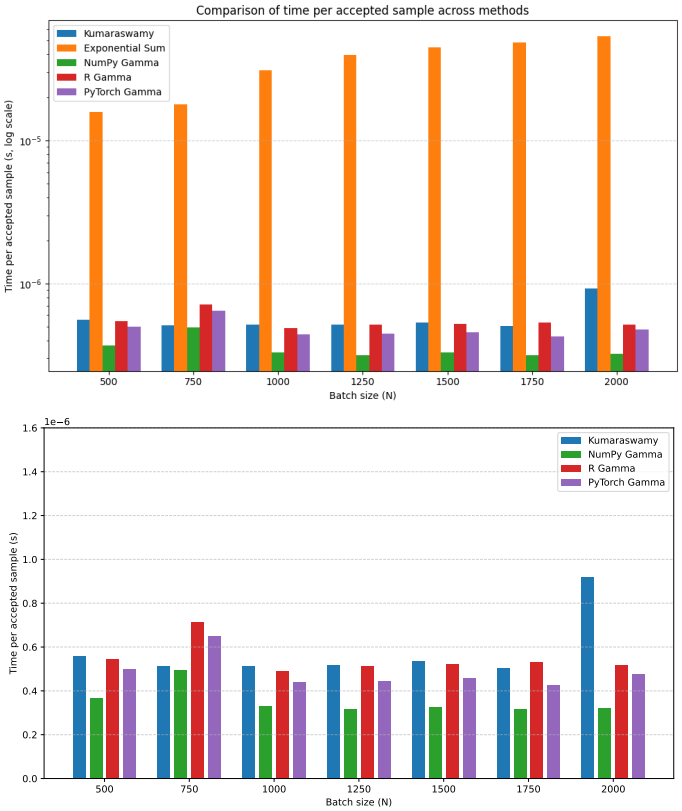

Fig. 2: Time per accepted sample as a function of the batch size, for all the methods including summation of Exponentials (top) and highlighted for the other ones (bottom).

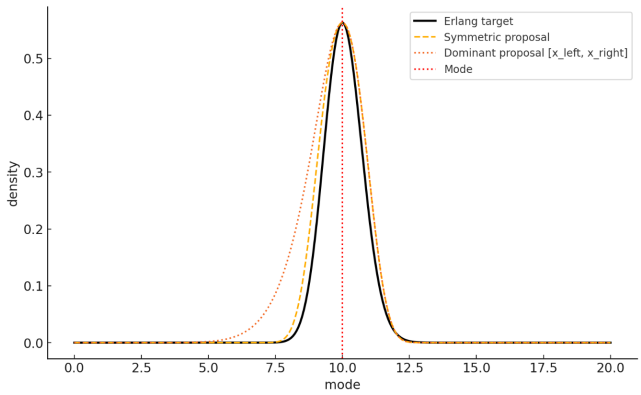

Fig. 3: Comparison between the Erlang target distribution and two proposal functions used in rejection sampling (dominant and symmetric).

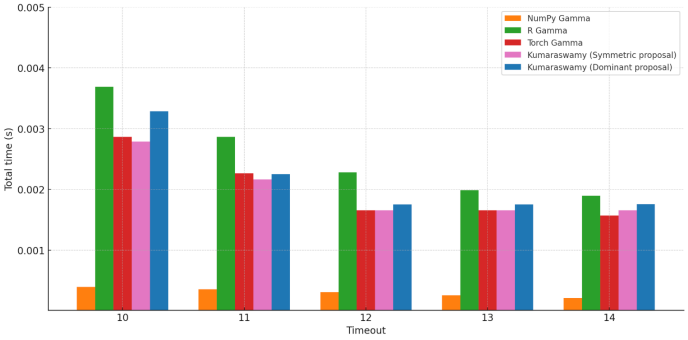

Fig. 4: Total sampling time required by each method as a function of the timeout.

[10] Amazon, "Batching for Managed Streaming," https://aws.amazon.com/blogs/compute/introducing-aws-lambda-batching-controls-for-message-broker-services/, 2025.

[11] L. Devroye, "Nonuniform random variate generation," *Handbooks in operations research and management science*, vol. 13, pp. 83–121, 2006.

[12] A. Horváth and M. Telek, *Phase Type Distributions: Theory and Application*. John Wiley & Sons, 2024.

[13] P. Reinecke, L. Bodrog, and A. Danilkina, "Phase-type distributions," in *Resilience Assessment and Evaluation of Computing Systems*. Springer, 2012, pp. 85–113.

[14] R. Storn and K. Price, "Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces," *Journal of Global Optimization*, vol. 11, no. 4, pp. 341–359, 1997.

[15] S. Das and P. N. Suganthan, "Differential evolution: A survey of the state-of-the-art," *IEEE Transactions on Evolutionary Computation*, vol. 15, no. 1, pp. 4–31, 2011.

[16] T. Das, Y. Zhong, I. Stoica, and S. Shenker, "Adaptive stream processing using dynamic batch sizing," in *Proceedings of SOCC*. ACM, 2014, pp. 16:1–16:13.

[17] E. A. Luengo, "Gamma pseudo random number generators," *ACM Computing Surveys*, vol. 55, no. 4, pp. 1–33, 2022.

[18] J. H. Ahrens and U. Dieter, "Computer methods for sampling from gamma, beta, Poisson and binomial distributions," *Computing*, vol. 12, no. 3, pp. 223–246, 1974.

[19] D. Fiems, "Efficient sampling from phase-type distributions," *Operations Research Letters*, vol. 57, p. 107184, 2024.

[20] F. Bause, A. Blume, P. Buchholz, A. Puzicha, and A. Timmermann, "Adaption of stochastic models (ASMo) - A tool for input modeling," in *Proceedings of QEST*. Springer, 2024, pp. 72–89.

[21] L. Martino and D. Luengo, "Extremely efficient generation of gamma random variables for $\alpha \geq 1$," *arXiv preprint arXiv:1304.3800*, 2013.

[22] M. Jerrum, "Fundamentals of partial rejection sampling," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 66, pp. 123–135, 2004.

[23] M. Castro, C. Azevedo, and J. Nobre, "A robust quantile regression for bounded variables based on the Kumaraswamy rectangular distribution," *Journal of Stat. Computation and Simulation*, vol. 93, 2023.

[24] M. Wasserman and G. Mateos, "Stabilizing the Kumaraswamy distribution," *arXiv preprint arXiv:2410.00660*, 2024.

[25] S. Nadarajah, G. Cordeiro, and E. Ortega, "General results for the Kumaraswamy-G distribution," *Journal of Statistical Theory and Applications*, vol. 11, pp. 265–290, 2012.

[26] G. Marsaglia and W. W. Tsang, "A simple method for generating gamma variables," *ACM Trans. Math. Softw.*, vol. 26, no. 3, pp. 363–372, 2000.

[27] ——, "The ziggurat method for generating random variables," *Journal of Statistical Software*, vol. 5, no. 8, p. 1–7, 2000.

[28] J. H. Ahrens and U. Dieter, "Generating gamma variates by a modified rejection technique," *Communications of the ACM*, vol. 25, no. 1, p. 47–54, Jan. 1982.

[29] ——, "Computer methods for sampling from the exponential and normal distributions," *Communications of the ACM*, vol. 15, no. 10, p. 873–882, Oct. 1972.

[30] G. E. P. Box and M. E. Muller, "A note on the generation of random normal deviates," *The Annals of Mathematical Statistics*, vol. 29, pp. 610–611, 1958.