On the Deployment Feasibility of Message Oriented Middlewares in Mission-critical Applications

Md. Monzurul Amin Ifath[1], Miguel Neves[2], Tommaso Melodia[3], Israat Haque[1]
[1]Dalhousie University, [2]Samsung Research, [3]Northeastern University

***Abstract*—Mission-critical applications (MCA) like smart grid management, first-aid response, and tactical coordination in military search and rescue operations refer to applications that can pose a risk to human lives or cause extensive and catastrophic losses. The deployment and management of these applications need careful consideration to meet the stringent performance demand of resource-constrained environments. One way to achieve such performance and dependability demand is to adopt Commodity-of-the-shelf (COTS) Message Oriented Middlewares (MOMs) (e.g., Apache Kafka, RabbitMQ) as they enable real-time data analytics and informed decision making. Despite their extensive usage in legacy business intelligent applications, little is known about their suitability for mission-critical applications. This paper fills that gap by first deploying and testing mission-critical applications on Apache Kafka and RabbitMQ. Then, we measure the performance, security, and reliability of the chosen MOMs to support MCA. The evaluation results confirm that Apache Kafka outperforms RabbitMQ, making it a potential candidate to deploy MCA. Specifically, Kafka requires 13x less bandwidth than RabbitMQ, which could be further reduced by 85% using effective parameter tuning. Our findings pave the way for MOMs to be adopted in mission-critical applications to meet their stringent performance and dependability demands.**

I. Introduction

Mission or *safety*-critical applications (MCA) are those that pose a risk to human lives or can cause extensive and catastrophic losses, e.g., aircraft control, smart grid management, first-aid response, and tactical coordination in military search and rescue operations [1]. These applications can leverage Message-Oriented Middleware (MOM) platforms (e.g., Apache Kafka, RabbitMQ) for efficient and reliable communications among distributed system entities, e.g., message brokers, producers, and consumers. More than 80% of Fortune 100 companies use one or more MOM platforms in their deployments [2]. Applications such as financial transaction monitoring, IoT sensor data processing, and content recommendations are actively using MOMs. The trend is also noticeable in mission-critical contexts where high reliability, fault tolerance, real-time performance, and security are important [3]. MOM simplifies the development process by promoting easier integration, enhancing component reuse and extension, and supporting improved overall scalability. However, it is essential to extensively evaluate the performance and dependability of the current MOM platforms to assess their readiness in mission-critical applications such as healthcare, defense, transport, and energy sectors.

Existing MOM platform evaluations, often conducted for traditional applications, may not adequately address the challenges posed by MCAs operating in constrained environments (e.g., limited bandwidth, limited computing power) [4]–[6]. Also, the evaluations in [7], [8] focus on MCA, such as shipping and tactical coordination on a single platform, limiting their findings. MCAs demand different configurations and selections of MOMs over an agile evaluation environment, but such setup and evaluations are missing in existing solutions. For example, in smart grid management, high availability and rapid response times are essential to prevent cascading failures in power distribution. In tactical situations, systems may prioritize quick data access over completeness. These diverse requirements underscore the need for careful consideration when selecting and configuring MOMs for MCAs.

This work fills that gap by first developing a realistic and extensible testbed to assess MOMs for MCA with performance and resource requirements. Then, we select two MOM platforms, Apache Kafka and RabbitMQ (rMQ), for mission-critical scenarios. Kafka's disk-based storage model provides stronger guarantees for synchronization and consistency essential in MCA. Unlike other MOMs (e.g., Redis, DDS), which keep history in memory and are therefore practically limited in the number of items they can retain, Kafka's history is persisted to disk, limiting it only by available storage capacity [9]. Also, Kafka and rMQ streams maintain extensive history logs, ensuring *reliable* data access even after reconnections. Kafka's pulling mechanism provides immediate awareness of link loss or restoration despite potential latency and increased network traffic. Finally, rMQ offers flexible routing and message delivery options, making it suitable for complex messaging patterns, albeit with a trade-off in throughput. These robust features render both platforms well-suited for meeting the stringent requirements of MCA. We choose two MCAs: i) smart grid management (wired mesh communications) and ii) military coordination (wireless satellite communications). These two applications have unique environmental and performance demands, e.g., the bandwidth demand of smart grid management and military coordination is at most 12.5 and 0.625 Mbytes/s, respectively, while response time must be at most 100 ms and 20 s, respectively.

Then, we comprehensively analyze the performance, reliability and security features of the chosen MOM platforms. The evaluation results reveal that Kafka outperforms rMQ in performance and security. For example, it requires 13x less bandwidth than rMQ, which could be further reduced by 85% using effective parameter tuning. Furthermore, Kafka adds only a negligible bandwidth demand for secure authentication compared to rMQ. Thus, we further investigate Kafka's relia-

bility and observe that around 25% of messages can be lost in the presence of network partitions, which requires developing resilience applications in the presence of failures. We will share the software artifact as a GitHub repository [10] if the paper is accepted.

II. Background

This section presents the necessary background on MOM platform to understand the proposed work.

MOM platforms facilitate communication between distributed components via asynchronous message passing. They can be broker-based or brokerless. *Broker-based* designs assume the existence of a central intelligence (a.k.a., the message broker) responsible for managing the message delivery to all interested parties. Most existing brokers adopt a *publish/subscribe* model, where messages are categorized according to their topics/routing keys. This design adds a decoupling layer between data senders (publishers/producers) and receivers (subscribers/consumers), as their communications can be asynchronous. *Brokerless* designs, on the other hand, realize the data communication by establishing point-to-point connections directly between senders and receivers. Messages are delivered using push or pull mechanisms. *Push-based* models are suitable for low-latency communication but may overwhelm slow consumers, whereas *pull-based* models offer more control at the expense of potential delay.

MOM platforms can offer different levels of data persistence, which refers to the ability to store messages for future consumption and/or recovery. Data persistence can be useful for several reasons, such as delivering messages to applications that join the system at a later time and keeping the data available even when a publishing application has terminated. Current platforms can use different strategies to persist data, including in-memory, disk, or database persistence, each with its own trade-offs between performance, durability, and consistency. In terms of consistency models, MOMs support mostly two: strong or weak. *Strongly consistent* platforms require data to be the same in all nodes at any time. *Weakly consistent* platforms, on the other hand, can tolerate delays and keep forwarding messages while updating the state between a leader broker and its replicas. We may choose the consistency model as per the applications need. Table I summarizes common MOM platforms categorized by their design philosophies and consistency guarantees. We present four representative platforms relevant to MCAs: distributed logs, message queues, in-memory data stores, and data distribution services.

Distributed logs (e.g., Apache Kafka) store all messages in persistent logs, allowing replay and strong synchronization across nodes which is key for fault-tolerant systems. Message queues (e.g., RabbitMQ) typically discard messages after delivery but offer efficient memory use and low latency; recent extensions (e.g., RabbitMQ Streams) add persistent and replayable messaging options. In-memory data stores (e.g., Redis) focus on high-speed data exchange using volatile storage, suited for applications with low latency tolerance but less reliability need. Data distribution services (DDS) adopt a brokerless architecture with peer-to-peer communication, supporting QoS and real-time guarantees but often requiring complex configuration.

These design choices of MOM platforms directly affect their suitability for MCAs. For instance, the choice of persistence not only influences replication overhead and bandwidth requirements but also impacts latency during network recovery. So, careful consideration of these design choices is essential to align MOM selection and configuration with the performance demands of mission-critical applications.

III. Related work

A few efforts have compared the performance and reliability functionalities of different MOM platforms. Fu et al. [4] compare the latency and throughput of five MOM platforms, namely Apache Kafka, RabbitMQ, RocketMQ, ActiveMQ and Apache Pulsar, on a customized testing framework. The authors vary features such as the message size, number of producers/consumers, and number of partitions in their analyses. Maharjan et al. [5] perform a similar study, but also consider the Redis platform in their comparisons and use a standard benchmarking tool (OpenMessaging framework) as part of their methodology. Alongside throughput and latency, Dobbelaere et al. [6] evaluate the reliability features of COTS MOM platforms. In particular, the authors show that throughput can drop more than 50% in both Apache Kafka and RabbitMQ when replication is in place. In common, none of the above studies consider analyzing MOM platforms in MCAs.

Recent research has proposed the adoption of COTS MOM platforms in MCAs. For example, Liu and Jiang [7] developed a MOM framework for the communication between shipboard information system modules based on Apache Pulsar. Rosa et al. [8] deployed the standard Data Distribution Service (DDS) API on top of the Derecho distributed coordination library and analyzed it in the context of an avionics application. Albano et al. [11] performed a qualitative comparison between the Remote Procedure Call (RPC), message passing, and publish/subscribe architectures when applied to smart grids. Unlike these efforts, we provide a comprehensive analysis of COTS MOM platforms when operating on a highly constrained infrastructure, i.e., subject to high link delays, low bandwidth capacity, and frequent node disconnections, all common characteristics of modern mission-critical systems.

IV. Deployment and Implementation

This section presents the mapping of mission-critical applications onto the selected MOM platforms. It details the deployment setups and design rationale tailored to meet the specific requirements of each application. Table II summarizes the key requirements for the respective applications.

Military coordination: Figure 1 depicts the military coordination application scenario. We assume several battle units on land, air, and sea communicating over a tactical network [8]. Each unit comprises multiple members, such as soldiers and aircraft, that must share information (e.g., position reports) within and between units. Each unit is also organized into a

TABLE I: Example message oriented middlewares.

Platform	Family	Communication pattern	Forwarding model	Persistence	Consistency model	Advantages	Disadvantages
Kafka	Distributed log	Broker-based	Pull	Yes	Weak / Strong	⊙, ↑	λ
RabbitMQ	Message queue	Broker-based	Push	Yes	Strong	∼	↓
ZeroMQ	Message queue	Broker-based / brokerless	Push	No	Weak / Strong	⋄, ↑	♯
Redis	In-memory data store	Broker-based	Pull / push	Yes	Weak	D⃗	♯
OpenDDS	Data distribution service	Brokerless	Push	Yes	Weak / Strong	∞, ✓	✠

↑ High throughput ⊙ Scalability ∼ Flexible routing and delivery ⋄ Lightweight D⃗ Fast data access and processing ∞ Real-time
✓ Reliable λ Lack of complete security measures ↓ Lower throughput ♯ Limited / no persistence ✠ Complex configuration

star topology, i.e., the communication always flows through a local command point (red dashed boxes). Different units connect through a satellite network, and the whole system requires: i) ordered, secure, and authenticated delivery of every message at least once and ii) tolerance to disconnections at the unit level, i.e., the log of messages from within and in-between units must be synchronized upon a unit reconnection. Battle unit disconnection is a common situation in the military domain, where the entire unit disconnects from a tactical network to move to a different place on the battlefield and then reconnects from the new place.

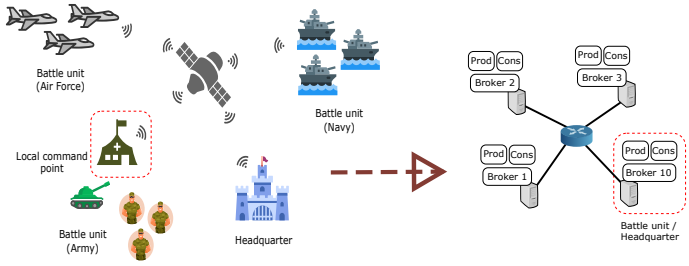

Fig. 1: Military coordination application reference deployment.

The link capacity should be up to 0.625 Mbytes/s with a maximum latency of 20 seconds. The system must handle a network size of up to 20 nodes, with a path length of up to 4 hops with a link delay between 10 ms and 2 seconds. These non-functional requirements ensure timely and reliable communication, prioritizing high availability to accommodate dynamic and unpredictable environments, with control traffic overhead not exceeding 50% of the link capacity, and the synchronization latency for a single message must be at most 5 seconds [12].

We implement and deploy this application on the chosen MOM platforms and compare to which extent each platform can meet application requirements. We assume a cluster-based setup where each battle unit hosts a message broker (i.e., cluster node) at its local command point. We replicate message queues (or logs) among all nodes participating in the cluster to ensure they remain operational upon a battle unit disconnection. In other words, each broker should be able to i) keep forwarding messages from/to members within a unit when this unit is disconnected and ii) automatically synchronize intra-unit messages with other units upon a re-connection. Each node in our reference architecture also hosts a data producer/consumer that mimics the battle unit members (e.g., a ship, soldier, or aircraft). Each producer continuously generates data at 30 Kbytes/s, simulating real-world actors (e.g., soldiers) in a battle unit.

Smart grid management: Figure 2 illustrates the reference deployment of the smart grid management application [13], which coordinates data from sensors and monitoring devices across a large-scale power grid. The grid is divided into five zones, each managed by a local control center. These centers are interconnected through a nationwide communication network characterized by high data rate and strict latency requirements. Each control center comprises multiple components that generate and process time-sensitive telemetry (e.g., voltage, frequency, load) for real-time state estimation, anomaly detection, and corrective actions.

The communication network supports a maximum throughput of 12.5 Mbytes/s with end-to-end latencies not exceeding 100 ms. The intra-zone link delay ranges between 1 and 10 ms. The system must ensure: i) strict message ordering for accurate sequential event analysis, ii) consistent and available delivery of messages under failures, and iii) at-least-once delivery semantics to prevent data loss in critical operations such as load shedding and grid restoration.

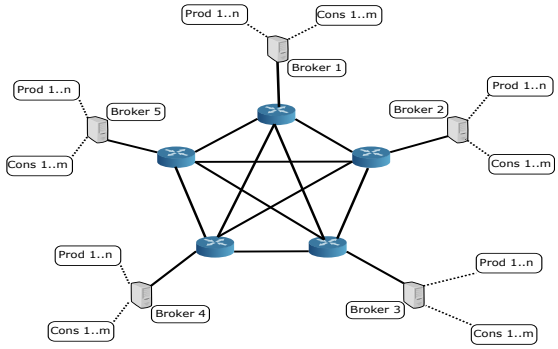

Fig. 2: Smart grid management reference deployment (inspired from [13]).

We deploy MOM clusters within each control center, with brokers replicating data across zones to maintain resiliency against node or link failures. Producers (sensors and meters) and consumers (controllers and analytics engines) are deployed within the same zone but are logically decoupled via the MOM layer. Each producer emits data at a rate consistent with modern phasor measurement units (PMUs), approximated as 100 Kbytes/s. This architecture ensures that each zone can

operate autonomously while preserving a global view of the grid-state via synchronized broker replication and message persistence.

TABLE II: Summary of military coordination and smart grid management application requirements.

Application: Military coordination **Critical requirements:** Low-latency awareness, Network resilience **Link type:** SATCOM **Available bandwidth:** $<$ 0.625 Mbytes/s **Tolerated latency:** $<$ 20 s **Fault tolerance:** Availability, Partition tolerance
Application: Smart grid management **Critical requirements:** High availability, Rapid response **Link type:** Wired mesh **Available bandwidth:** $<$ 12.5 Mbytes/s **Tolerated latency:** $<$ 100 ms **Fault tolerance:** Consistency, Availability

Setup and Evaluation: We ran our experiments on a 10-core Intel Xeon Silver 4210R @ 2.40 GHz server with 32 GB of memory and 2 TB of storage. The server, running Ubuntu 22.04 (kernel version 5.15), had hyper-threading disabled. Our reference deployment architecture was implemented on Mininet version 2.3.0, which uses containers to emulate real applications, providing a more realistic evaluation environment compared to traditional simulators like ns-3 and OMNeT++. Our testbed setup is adaptable and supports integrating new MOMs through plug-ins (details in [10]). The testbed also allows one to configure standard communication channel parameters on emulated links, including delay, bandwidth, and packet loss. The dependability of the testbed is assessed by comparing its performance with the hardware baselines [12].

We deployed two MOM platforms: Apache Kafka 3.8.0 and RabbitMQ 3.9.3 (rMQ) in the testbed to evaluate the performance of the chosen applications. We set 10 brokers in the military coordination application while 5 in the smart grid management. These brokers can support a varying number of producers and consumers depending on available capacity and application need. This tightly coupled and small-scale clusters of brokers are chosen to ensure deterministic performance (e.g., ultra low latency) and rapid fault recovery. In the case of a large number of brokers demand (e.g., 100 nodes), a hierarchical and adaptive deployment strategies are recommended to avoid challenges like an increased synchronization overhead, a highly complex configuration tuning, and a higher susceptibility to network partitions [14]. Our study thus establishes a robust baseline for constrained deployments and provides a foundation for future extensions to larger scales.

We adopted three key metrics to evaluate the MOM platforms: bandwidth demand, message latency, and loss ratio. Bandwidth demand, measuring the total bandwidth required for message forwarding, is a critical indicator of performance and security overhead, particularly when security mechanisms like authentication are enabled. Message latency defined as the elapsed time between a producer sending a message and the reception of the acknowledgment from the consumer, directly assesses the performance and responsiveness of the MOM platforms. The loss ratio, representing the proportion of messages lost, is a direct measure of reliability. We collect these measurements after a 60-second warm-up interval to ensure the system reaches a steady state. We conducted ten iterations of each evaluation to measure their mean and confidence intervals.

V. Evaluations Results

This section shares the key findings of the performance, security, and reliability of the chosen platforms.

A. Performance

While the performance of MOMs has been studied in local and wide area networks, little is known about their response in highly constrained infrastructures, e.g., networks presenting low-bandwidth, high link delay, and high probability of data loss, which are encountered in mission-critical domains. Thus, we consider such constrained environments for the chosen MCA and evaluate the bandwidth and latency of Kafka and rMQ.

We consider the military coordination application in the bandwidth evaluation due to its bandwidth-constrained links (0.625 Mbytes/s). Figure 3 shows the total bandwidth demand from Kafka and rMQ for different numbers of broker nodes. MOMs generally require a high replication level in mission-critical setups, resulting in high communication overhead. The higher bandwidth demand of rMQ compared to Kafka (approximately 13x) can be attributed to several factors. First, rMQ does not compress messages by default during inter-broker communication, leading to more data being transferred, especially in scenarios with high replication for fault tolerance. Second, rMQ's message delivery mechanism involves sending messages one at a time, which increases the overhead of control traffic. In contrast, Kafka's architecture allows for message batching, reducing the relative overhead of control traffic and more efficient data transfer. These architectural differences contribute to Kafka's lower bandwidth consumption, making it more suitable for bandwidth-constrained MCAs.

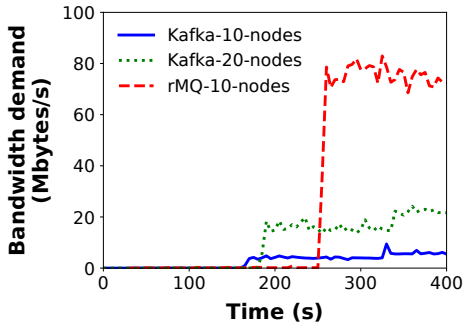

Fig. 3: Bandwidth demand of Kafka and rMQ with varying number of brokers.

In terms of latency evaluation, we assess the performance of Kakfa and rMQ for the smart grid application due to its strict latency requirement (100 ms), which is not the

case for the military coordination application. Figure 4 shows the evaluation results. Overall, the latency increases significantly (more than 200% and 75% for Kafka and rMQ, respectively) in a highly constrained network compared to a baseline scenario with negligible link delay (1 ms). However, despite this sharper degradation, Kafka still delivers lower absolute latency than rMQ across most percentiles. This is primarily because rMQ relies on the AMQP protocol, which introduces multiple round-trip delays due to connection setup and acknowledgment requirements. Furthermore, rMQ must wait for consumer acknowledgments before sending the next message, limiting throughput under constrained conditions. Kafka, with its asynchronous, batched communication and disk-based buffering, is better suited for sustained message flow even when link quality degrades.

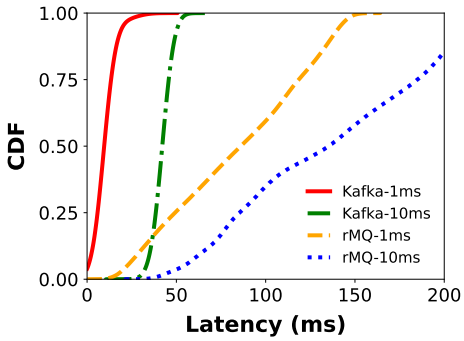

Fig. 4: Cumulative distribution function (CDF) of the message latency from Kafka and rMQ with different link delays.

B. Security

Mission-critical systems are increasingly being targeted by attackers. As a result, there has been an uptick in novel defenses for hardening these systems (e.g., [15], [16]). Although security is still not a primary concern on most MOMs, the support for different security mechanisms on them is expanding quickly. In this evaluation, we first present the security settings that operators must configure in MOM platforms. Then, we present the added overhead that MOMs may introduce to offer required security services in MCA.

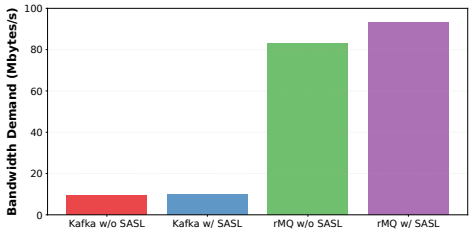

Fig. 5: Bandwidth overhead comparison for Kafka and rMQ with and without SASL authentication enabled in the military coordination application.

We first extensively analyze the level of security that the chosen MOMs can offer. We observe that current platforms are not secure by default, i.e., users must explicitly set a security mechanism to enforce policies. For example, Kafka's data encryption approach must explicitly set TLS listeners on each broker to avoid plaintext traffic flow. Also, operators must delete any existing plaintext listener to avoid automatic initialization. The second issue is the manual ACL configuration due to not having any high-level API, which can be time consuming and error-prone in large scale MCA. Next, we assess the performance overhead introduced by enabling the Simple Authentication and Security Layer (SASL) authentication framework for both Kafka and rMQ to check their applicability in MCA. We consider the military coordination application due to its stringent resource requirements. The evaluation results are shown in Figure 5, which indicates that Kafka and rMQ add around 2% and 12% bandwidth overhead, respectively. The difference in overhead between the two platforms is due to a combination of factors, primarily the architectural and protocol-level distinctions between Kafka and rMQ. Kafka's native binary protocol handles SASL with greater efficiency, while rMQ's reliance on the AMQP protocol, mandates SASL for secure connections, contributes to a higher relative overhead.

C. Reliability

Network partitioning and disconnections may occur in certain applications due to the nature of their communications. For example, military coordination application uses intermittent wireless communications prone to failure along with dynamic addition and deletion of nodes. Thus, we mimic such anomaly conditions in this applicaiton by randomly disconnecting a node for approximately 20% of the experiment duration. Specifically, we randomly disconnect the node hosting the leader broker for one of the topics. The disconnection time follows a uniform distribution between 100 and 140 seconds, with a mean disconnection time of 120 seconds. The node disconnects after the system reaches its steady state and re-connects to the cluster after the disconnection period expires. Finally, we evaluate the reliability of Kafka as it already outperforms rMQ.

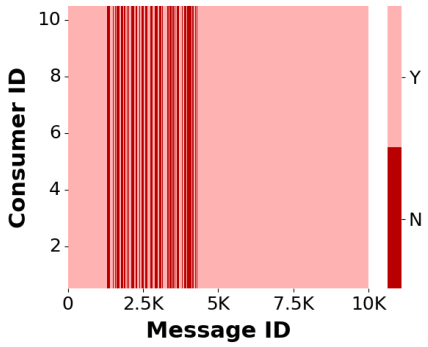

Fig. 6: Message delivery matrix for the co-located producer with a disconnected broker.

We examine the data delivery performance of a Kafka producer co-located with the disconnected broker by recording the message reception status of each message for all consumers. As in Figure 6, we observe that around 25%

of all messages are lost during the network partition period. Moreover, these failures only affect the topic/log whose leader broker is disconnected. This result aligns with previous studies [17] that attribute such behavior to the data reconciliation process of Kafka. When the disconnected broker rejoins the cluster, Kafka may discard data or retrieve it from a stale log. Thus, operators must consider a reliable design if uses Kafka as their MOM platform.

D. Suggested Configuration Improvement

Most MOMs ship with a large number of configurable parameters (e.g., replication factor, batch size, acknowledgment type). These parameters can be fine tuned for the best configuration for a chosen MCA. Since Kafka outperforms rMQ, we further investigate its capability to further improve resource requirement, e.g., bandwidth, to support applications like military coordination, which has the most scarce bandwidth. We apply a grid search approach with more than 20 Kafka parameters, including producer, broker, and consumer. Note that these parameters are chosen following Kafka manual and prior benchmarks [2], [18]. The list of key Kafka parameters of our evaluations is shown in Table III. The complete list of parameters is available in our public repository [10].

Parameters	Placement	Default	Range Utilized
Linger	Producer	0 s	0 - 1 s
Batch Size	Producer	16 KB	4 - 256 KB
Compression	Producer	None	gzip, lz4, zstd
Acknowledgment	Producer	1	0, 1, 2
Buffer Memory	Producer	32 MB	2 - 64 MB
Replica Max Wait Time	Broker	0.5 s	0 - 5 s
Replica Min Fetch Bytes	Broker	1 B	1 - 200 KB
Max Wait Time	Consumer	0.5 s	0 - 5 s
Min Fetch Bytes	Consumer	1 B	1 - 200 KB
Session Timeout	Consumer	10 s	1 - 20 s

TABLE III: Configuration of key Kafka parameters to optimize bandwidth usage.

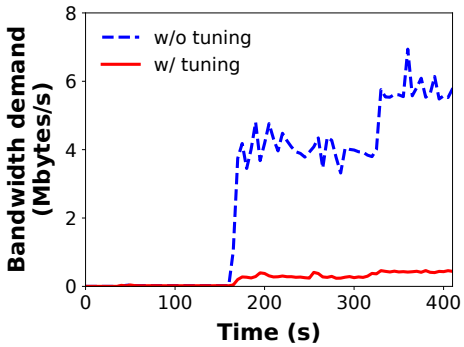

Fig. 7: Impact of parameter tuning on Apache Kafka total bandwidth demand.

Figure 7 shows the improvement in the bandwidth demand of Apache Kafka after tuning the required parameters. Specifically, we measure the total bandwidth demand of Kafka before (with default setup) and after the tuning process and show that efficient parameter tuning can reduce the demand by 85% compared to the default configurations. However, there are two main challenges associated with the tuning process: first, it takes a considerable amount of time to find a sweet spot among the configuration parameters, and second, it may be necessary to accommodate several goals (e.g., minimize both bandwidth demand and latency) during the tuning process, which makes the problem even more complicated. We plan to explore automated parameter tuning as future research.

VI. Conclusion

This paper explored how existing MOM platforms could support mission-critical applications. We chose military coordination and smart grid applications as representative use cases and assessed the performance, reliability and security of Apache Kafka and rMQ. Extensive evaluations confirm that Kafka outperforms rMQ in resource demand and overhead for security services. Then, we have shown that Kafka's resource demand can be further reduced by judicious configuration parameter tuning, e.g., we could reduce the bandwidth demand by 85% applying parameter tuning. As part of future work, we will investigate additional mission-critical applications with in-depth performance evaluations of varying degrees of network size and workload. We will also explore the automatic parameter tuning and ACL configurations.

References

[1] K. Fowler, "Mission-critical and safety-critical development," *IEEE Instrumentation & Measurement Magazine*, 2004.

[2] Apache kafka. Accessed: 2025-04-09. [Online]. Available: https://kafka.apache.org/.

[3] B. Al-Madani *et al.*, "Integrating data distribution service (dds) in smart traffic systems: A comprehensive review," *Computer*, 2025.

[4] G. Fu *et al.*, "A fair comparison of message queuing systems," *IEEE Access*, 2021.

[5] R. Maharjan *et al.*, "Benchmarking message queues," *Telecom*, 2023.

[6] P. Dobbelaere *et al.*, "Kafka versus rabbitmq: A comparative study of two industry reference publish/subscribe implementations: Industry paper," in *Proc. of the ACM DEBS*, 2017.

[7] K. Liu *et al.*, "High performance shipborne message queuing service prototype system based on apache pulsar," in *IEEE ICIBA*, 2021.

[8] L. Rosa *et al.*, "Derechodds: Strongly consistent data distribution for mission-critical applications," in *IEEE MILCOM*, 2021.

[9] J. Zhang *et al.*, "Comparison of middlewares in edge-to-edge and edge-to-cloud communication for distributed ros 2 systems," *Journal of Intelligent & Robotic Systems*, 2024.

[10] Pinetdalhousie/mission-critical-messaging-platforms. Accessed: 2024-04-10. [Online]. Available: https://github.com/PINetDalhousie/mission-critical-messaging-platforms

[11] M. Albano *et al.*, "Message-oriented middleware for smart grids," *Elsevier Computer Standards & Interfaces*, 2015.

[12] M. M. A. Ifath *et al.*, "Fast prototyping of distributed stream processing applications with stream2gym," in *Proc. of the IEEE ICDCS*, 2023.

[13] P. Kansal *et al.*, "Bandwidth and latency requirements for smart transmission grid applications," *IEEE Transactions on Smart Grid*, 2012.

[14] X. Guo *et al.*, "Towards scalable, secure, and smart mission-critical iot systems: review and vision," in *Proc. of the EMSOFT*, 2021.

[15] N. Burow *et al.*, "Moving target defense considerations in real-time safety- and mission-critical systems," in *Proc. of the ACM Workshop on MTD*, 2020.

[16] J. Wang *et al.*, "Rt-tee: Real-time system availability for cyber-physical systems using arm trustzone," in *IEEE Symposium on SP*, 2022.

[17] A. Alquraan *et al.*, "An analysis of network-partitioning failures in cloud systems," in *Proc. of the USENIX OSDI Conf.*, 2018.

[18] Z. Kang *et al.*, "Dmsconfig: Automated configuration tuning for distributed iot message systems using deep reinforcement learning," *arXiv preprint arXiv:2302.09146*, 2023.