Are data streaming platforms ready for a mission critical world?

Md. Monzurul Amin Ifath[1], Miguel Neves[1], Brandon Bremner[2], Jeff White[2], Tomas Szeredi[2], and Israat Haque[1]

[1]Dalhousie University
[2]General Dynamics Mission Systems Canada (GDMSC)

April 18, 2024

Abstract

Commodity-of-the-shelf (COTS) data streaming platforms, e.g., Apache Kafka, RabbitMQ, have been powering companies from several sectors to move and extract further insights from their data. Yet, little is known about the suitability of these platforms for mission critical applications such as disaster recovery, first-aid response and military coordination. This paper provides a provocative discussion on how prepared current data streaming platforms are for mission critical scenarios. We argue that, despite their intrinsic reliability and security features, these platforms are still hard to test/operate which ultimately hinders trust on their safety and security conditions. Moreover, they commonly assume a powerful underlying infrastructure, which makes it tough to keep a high performance in resource-constrained environments (e.g., satellite networks). We conclude by identifying relevant research directions to ameliorate the performance and suitability of data streaming platforms for mission-critical applications.

Are data streaming platforms ready for a mission critical world?

Md. Monzurul Amin Ifath⋆, Miguel Neves⋆, Brandon Bremner⋄, Jeff White⋄, Tomas Szeredi⋄, Israat Haque⋆
⋆Dalhousie University
⋄General Dynamics Mission Systems Canada (GDMSC)

Abstract—**Commodity-of-the-shelf (COTS) data streaming platforms, e.g., Apache Kafka, RabbitMQ, have been powering companies from several sectors to move and extract further insights from their data. Yet, little is known about the suitability of these platforms for mission critical applications such as disaster recovery, first-aid response and military coordination. This paper provides a provocative discussion on how prepared current data streaming platforms are for mission critical scenarios. We argue that, despite their intrinsic reliability and security features, these platforms are still hard to test/operate which ultimately hinders trust on their safety and security conditions. Moreover, they commonly assume a powerful underlying infrastructure, which makes it tough to keep a high performance in resource-constrained environments (e.g., satellite networks). We conclude by identifying relevant research directions to ameliorate the performance and suitability of data streaming platforms for mission-critical applications.**

I. Introduction

Data streaming platforms are software tools that enable the efficient and reliable transmission of data among nodes in distributed applications. These platforms can be mostly classified into four families: distributed logs, message queues, in-memory data stores, and data distribution services. According to a recent survey, more than 80% of the Fortune 100 companies use one or more of these platforms. Depending on the performance, scalability, availability, and security requirements, different platforms may be suitable for different scenarios. However, despite their wide adoption, it is not clear whether current data streaming platforms can fulfill the stringent requirements of mission-critical applications such as those powering the healthcare, defence, transport and energy sectors.

The literature on the use of data streaming platforms in mission-critical contexts is scarce. Some works (e.g., [1], [2]) focus on benchmarking different data streaming platforms, but they do not consider any mission-critical application. Other efforts (e.g., [3], [4]) apply data streaming platforms to solve problems in mission-critical domains such as shipping, avionics, and smart grids, but they are mostly limited to a single platform. To the best of our knowledge, our paper is the first to present a comprehensive analysis of various data streaming platforms for mission-critical scenarios.

In this paper, we compare the performance, reliability, and security of different data streaming platforms in a realistic mission-critical setup. We use a military coordination scenario as a case study and evaluate a couple of representative platforms. We identify the main challenges and trade-offs of using those platforms in a highly constrained network with low bandwidth, high latency, and high churn. In addition, we provide a few guidelines and suggestions for improving the design and configuration of data streaming platforms for mission-critical applications.

II. Background and motivation

A. Mission-critical applications

In this paper, we consider *mission-critical* application any application that either poses a hazard to human lives or can involve large and catastrophic losses. Sometimes these applications are also called *safety-critical* applications [5]. Examples of mission-critical applications include aircraft control, smart grid management, first-aid responding, and tactical coordination in military search and rescue operations. Modern mission-critical applications rely on complex systems for performing their tasks, normally combining several modules that exchange data through networked communication channels in a "system of systems" architecture.

Application requirements. Current mission-critical applications can have quite distinct requirements. Table I shows three representative examples that reflect customer demands from our industry partner. The examples also illustrate existing design trade-offs regarding performance, fault tolerance, and the capacity of the underlying infrastructure. The first application (business intelligence) depicts a comparison with the standard usage of current commodity data streaming platforms. Unlike mission-critical systems, business intelligence ones have plenty of resources (beyond 100 Gbps Ethernet links), soft latency constraints (up to hours for the processing of big data sets), and usually no lives at risk to deal with. As a result, these systems can either tolerate small periods of unavailability (CP) or work in a weakly consistent manner (AP). CP and AP refer to the CAP theorem, a.k.a. Brewer's theorem, which states that a distributed system can simultaneously provide only two of the consistency (C), availability (A), and partition tolerance (P) properties.

Data streaming as an intrinsic component. Data streaming platforms are becoming an integral part of mission critical systems. Early efforts include integrating RTI DDS into emergency vehicle tracking [6] and Apache Kafka into spacecraft [7] control systems. The reasons why COTS data streaming platforms have proven to be advantageous are twofold: first, they can abstract away several low level details associated with message sending such as connection management, data

TABLE I: Comparison between business and mission critical application requirements.

Application	Link type	Risk	Available bandwidth	Tolerated latency	Fault tolerance*
Business intelligence	High-speed Ethernet, Optical fiber	No	$>$ 100 Gbps	hours	CP, AP
First-aid responding	LTE, 5G	High	$<$ 10 Gbps	$<$ 10ms for a connected ambulance	CA
Military coordination	SATCOM	Medium	$<$ 5 Mbps	$<$ 20s	AP
Aircraft control	LTE, SATCOM	High	$<$ 1 Mbps (cockpit)	$<$ 2s for position reports	CA

*C = Consistency, A = Availability, P = Partition tolerance.

buffering, and re-transmissions, therefore leaving engineers free to focus on high-level communication patterns (e.g., who should get a given message, should senders push or receivers pull it). Second, COTS tools tend to be more robust and well-maintained compared to custom (or domain-specific) solutions thanks to large testing campaigns and frequent code reviews [8].

B. Related work

A few efforts have compared different data streaming platforms with respect to their performance, reliability and security functionalities. Fu et al. [1] compare the latency and throughput of five data streaming 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. [2] perform a similar study, but also consider the Redis platform in their comparisons and use a standard benchmarking tool, i.e., the OpenMessaging framework, as part of their methodology. In addition to throughput and latency, Dobbelaere and Esmaili [9] evaluate the reliability features of COTS data streaming 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 data streaming platforms in mission-critical scenarios.

Recent research has proposed the adoption of COTS data streaming platforms on mission-critical applications. For example, Liu and Jiang [3] developed a data streaming framework for the communication between shipboard information system modules based on Apache Pulsar. Rosa et al. [4] 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. [10] 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 (both qualitative and quantitative) of COTS data streaming 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.

III. Data streaming platforms

A. Design choices

Data streaming platforms can be characterized as a combination of several design choices. Next, we discuss some of the most important ones.

Communication pattern. Data streaming platforms can be either broker-based or brokerless. *Broker-based* tools assume the existence of a central intelligence (a.k.a., the message broker) which is responsible for managing the message delivery to all interested parties. Most existing brokers adopt a publish/subscribe model where messages are categorized (or routed) according to topics/routing keys. This design adds a decoupling layer between data senders (or publishers/producers) and receivers (or subscribers/consumers) as they don't have to be simultaneously connected for a message to be delivered. In other words, communication happens asynchronously. *Brokerless* tools, on the other hand, realize the data communication by establishing point-to-point connections directly between senders and receivers. In both cases, data streaming platforms may support different messaging protocols to forward data, including AMQP, MQTT, XMPP, WebSockets and custom protocols (e.g., the Kafka Wire Protocol).

Data forwarding model. Data streaming platforms can forward messages to receivers in two modes: push and pull. *Push-based* tools push messages to receivers as soon as they can, which is good for low latency messaging. As a downside, this design choice can easily overwhelm receivers when the latter is not able to process messages as fast as they come. A common solution to this problem is to configure a prefetch limit in the broker which bounds the number of unacknowledged messages that a receiver may have at any given time. Pushing messages to receivers may also cause issues when receivers are no longer processing messages, e.g., due to a subtle disconnection. *Pull-based* tools, on their turn, wait for consumers to ask for data. In this case, consumers can request a message (or batch of messages) from a given offset. Tight loops may happen when there is no message past the offset which can be minimized by setting longer pooling intervals.

Data persistence. Data streaming 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

TABLE II: Example data streaming platforms.

Platform	Family	Communication pattern	Forwarding model	Persistence	Consistency model
Kafka	Distributed log	Broker-based	Pull	Yes	Weak / Strong
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
OpenDDS	Data distribution service	Brokerless	Push	Yes*	Weak / Strong

*Optional persistence support.

use different strategies to persist data, including in-memory, disk, or database persistence, each with its own trade-offs between performance, durability, and consistency.

Consistency model. Broadly speaking, data streaming platforms follow one of two consistency models: strong or weak consistency. *Strongly consistent* platforms require data to be the same in all of its nodes at any time. In other words, a message will not be forwarded to a receiver before all broker replicas (assuming a broker-based system) commit it to their local logs. *Weakly consistent* platforms, on the other hand, can tolerate delays and keep forwarding messages while updating state between a leader broker and its replicas. Weakly consistent platforms can have a broad range of operational modes. For example, a message can be considered ready to be delivered after either the leader or a quorum of nodes have committed its data. The leader election strategy (e.g., user-defined, randomly selected) and the behavior upon a network partition/re-connection (e.g., delegate/return leadership to a preferred node) are also flexible parameters.

B. *Example tools*

Table II summarizes some of the most common data streaming platforms currently available. Our goal is not to be exhaustive, but rather to offer a glimpse into the different types of existing platforms. The "Family" column defines broader groups of platforms that have several features in common. The remaining columns follow the discussion in Section III-A. Based on our investigation, we could identify four main platform families which are further described below.

Distributed logs. Distributed logging solutions provide messaging services while storing a copy of each message into a log file (or topic) by default. This feature enables new consumers to access the log file and read messages from any point in time (or message offset). Exemplars of this family include tools such as Apache Kafka and Apache Pulsar.

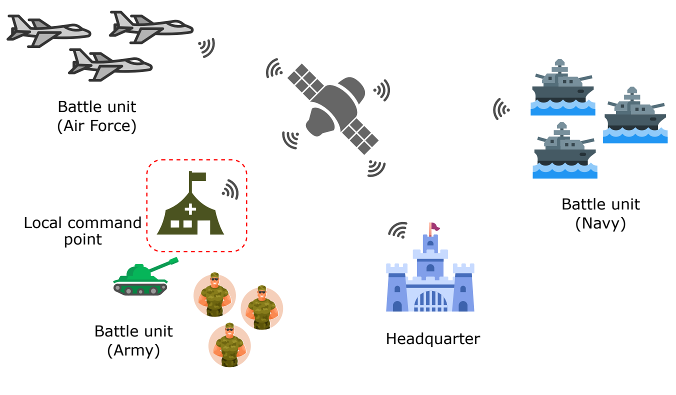
Message queues. Unlike distributed logs, message queues usually do *not* store a copy of a message after it is delivered, making it impossible for different consumers to get the same message. While allowing several optimizations such as using efficient in-memory data structures to hold data until its delivery, current message queuing systems frequently present worse performance compared to distributed logs, e.g., due to the need to forward messages individually. Modern message queues started offering support for non-destructive message consumption as separate plug-ins such as the RabbitMQ streams. Other tools in this family include ActiveMQ, ZeroMQ, and RocketMQ.

In-memory data stores. Although originally designed as a high-throughput and low-latency alternative to traditional (i.e., disk-oriented) database systems, in-memory data stores have also gained visibility for their messaging capabilities in the latest years. In particular, the ability to simultaneously store, retrieve, process, and forward data of almost any type to subscribers has proven to be largely useful for high-performance applications (e.g., gaming, trading systems). Most in-memory data stores organize data as key-value pairs and use message keys to identify interested subscribers. Example tools from this family are Redis and Hazelcast.

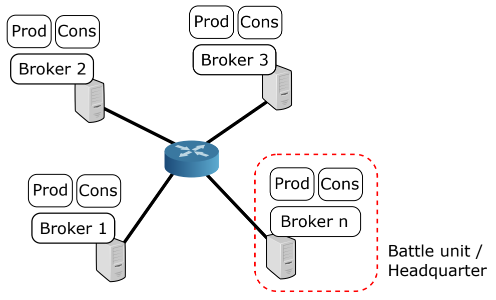
Data distribution services. Data Distribution Services are a family of tools that implement the DDS protocol from the Object Management Group (OMG) for streaming data among connected nodes. While abstracting away many low-level networking tasks (e.g., message addessing, serialization and delivery) from users, the latter is still responsible for configuring features such as data persistence and QoS policies. Data distribution services adopt a brokerless (or peer-to-peer) communication model by design, and example tools from this family include OpenDDS, Eclipse Cyclone DDS, RTI Connext DDS, and eProsima Fast DDS.

IV. Methodology

We compare the suitability of different data streaming platforms to a mission-critical deployment by evaluating their capabilities in a military coordination scenario. In addition to the long know-how of our industry partner, this application enables us to easily push the platforms under test to their limit due to the need for simultaneously stressing several requirements. Figure 1(a) depicts our application scenario. We assume several battle units on land, air, and sea communicating over a tactical network. Each unit comprises multiple members such as soldiers and air crafts that must share information (e.g., position reports) both within and in between units. Each unit is also organized into a star topology meaning that the communication always flows through a local command point (red dashed boxes). Different units connect to each other through a satellite network, and the whole system requires: i) ordered, secret, authenticated, and guaranteed delivery for every message; and ii) tolerance to disconnections at the unit level, meaning the log of messages from within and in-between units must be synchronized upon a unit re-connection. Battle unit disconnection is a common situation in the military



(a)



(b)

Fig. 1: *a)* Military coordination scenario adopted in our evaluation; *b)* Reference data streaming platform deployment.

domain where the whole unit disconnects from a tactical network to move to a different place on the battlefield and then re-connects from the new place.

We deploy our example application using different data streaming platforms and compare (both quali- and quantitatively) to which extent each platform can meet every application requirement. Figure 1(b) presents our reference deployment architecture. We assume a cluster based setup where each battle unit hosts a message broker (i.e., cluster node) located 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. A similar situation appears, e.g., when an ambulance equipped with various health sensors must reach out to a patient in a remote location. 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).

We implement our reference deployment architecture on top of Mininet version 2.3.0 and consider two data streaming platforms in our experiments: Apache Kafka (version 2.8.0) and RabbitMQ (version 3.9.3). The latter is henceforth called rMQ for simplicity. We focus on these two platforms for two main reasons: i) they are broker-based tools and thus in line with our reference deployment architecture; and ii) they are well documented tools and widely deployed in several different domains. The source code, configuration files, and generated logs from our experiments are publicly available [11]. We run our experiments in a 10 core Intel Xeon Silver 4210R @ 2.40 GHz server with 32 GB of memory and 2 TB of storage. The server is equipped with Ubuntu 22.04 (kernel version 5.15) and has hyper-threading disabled. Unless stated otherwise, each producer in our deployment constantly generates data at 30 Kbytes/s, mimicking real actors (e.g., soldiers) in a battle unit. Finally, we adopt three metrics to evaluate the data streaming platforms: message latency, loss ratio, and the total throughput (or bandwidth demand) required by the whole cluster to forward messages.

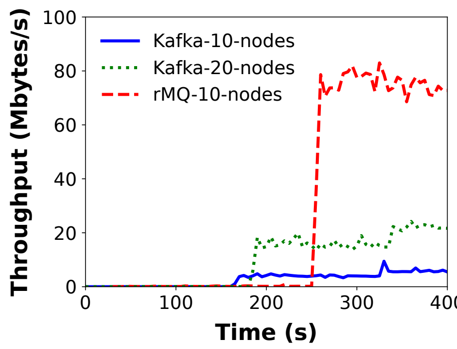


Fig. 2: Total bandwidth demand from both Kafka and rMQ for different cluster sizes.

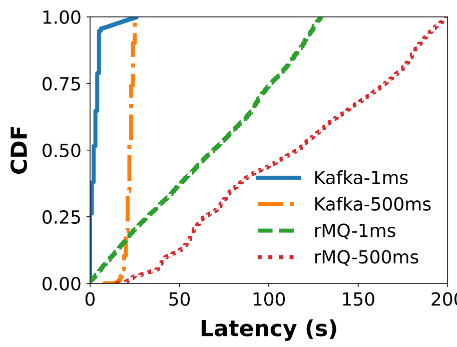


Fig. 3: Cumulative distribution function (CDF) of the message latency from both Kafka and rMQ for different link delays.

V. Main findings

A. Performance

(R1) Do data streaming platforms scale in a mission critical setup? Figure 2 shows the aggregated throughput (or total bandwidth demand) from both Kafka and rMQ for different cluster sizes. As can be seen, data streaming platforms in general require a high replication level to work on mission-critical setups which results in high communication overhead. Also, rMQ requires significantly more bandwidth than Kafka – approximately 13x for a 10-node topology. This is mainly due to two reasons: i) unlike Kafka, rMQ does not compress messages exchanged between cluster members, e.g., for replicating data; and ii) rMQ brokers deliver messages one at a time, which forbids batching and increases the control traffic overhead.

(R2) Can parameter tuning improve performance? Most data streaming platforms ship with a large number of configurable parameters (e.g., replication factor, batch size, acknowledgment type). While this flexibility is essential for the proper operation of the data streaming platform on different domains, the resulting high-dimensional configuration space makes it difficult for users to determine the best configuration option (if any) for their application. In this sense, we evaluate the extent to which parameter tuning can improve the performance of data streaming platforms in a mission-critical setup. We use the same 10-node topology described in (R1) for this experiment and apply a grid search approach on a reasonable set of Kafka parameters. We consider more than 15 parameters, including broker, sender and receiver parameters. Please refer to our public repository [11] for an exact list and plots depicting our experimental results.

We measure the total bandwidth demand of Kafka before and after the tuning process using a network monitoring tool, and find that tuning Kafka significantly reduces the demand by up to an order of magnitude compared to the default configuration. Despite its great potential, there are two main challenges associated with the data streaming platform tuning process: first, it takes a considerable amount of time (a few days in our case) 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 discuss these issues in further detail and point to alternative solutions in Section VI.

(R3) How do data streaming platforms perform in a highly constrained network? While the performance of data streaming platforms has been widely studied in local and wide area networks (LANs and WANs respectively), 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, such as the ones we encounter in mission-critical domains. In this experiment, we shed light on this question by emulating a highly constrained network on top of Mininet and investigating the data streaming platform performance when forwarding messages. As in the previous experiments, we use the topology described in Figure 1(b), but set the link delay to 500 ms for all links (i.e., the round-trip time – RTT is approximately 2 seconds). According to our industry partners, this is in line with current SATCOM military networks. Moreover, we adjust both Kafka and rMQ connection timeouts accordingly so that the tools can work appropriately with such high link delays.

Figure 3 shows the cumulative distribution function (CDF) of the message latency in Kafka and rMQ for different link delays. Overall, the latency increases significantly (more than 300% and 85% at the median for Kafka and rMQ, respectively) in a highly constrained network compared to a baseline scenario with negligible link delay. Moreover, we can see that rMQ performs consistently worse than Kafka. That is mainly due to two reasons: i) AMQP connections require several RTTs before messages can be forwarded; and ii) rMQ brokers must wait for consumers to acknowledge that they have received a message (or group of messages) before sending the next one.

B. Reliability

(R4) Do data streaming platforms experience message loss upon a network partition? A common characteristic on mission-critical networks is the high probability of network partitions, either because of the existence of unstable links (e.g., a distant cellular tower) or because of sudden node disconnections such as when a battle unit moves. In this sense, it is important to assess the reliability of current data streaming platforms upon network partitions. For that, we repeat the experiment described in (R3) while randomly disconnecting a node for approximately 20% of the experiment duration. The node disconnects after the system has reached its steady state and re-connects to the cluster after the disconnection period expires.

We examine the data delivery performance of a Kafka producer co-located with the disconnected broker by recording message reception status of each message for all consumers. We observe a loss ratio of approximately 0.25 (i.e., around 25% of all messages were lost) during the network partition period (plot can be found in the public repository). Moreover, these failures only affect the topic/log whose leader broker is disconnected. This result confirms the findings of previous studies [8] that attribute such behavior to the data reconciliation process of ZooKeeper (the distributed coordination service used by Apache Kafka). When the disconnected broker rejoins the cluster, ZooKeeper may discard data or retrieve it from a stale log. We do not observe similar behavior in the newer Raft-based Kafka.

C. Security

(R5) Which security mechanisms are available on data streaming platforms? 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., [12], [13]). Although security is still not a primary concern on most data streaming platforms, the support for different security mechanisms on them is expanding quickly. As part of our analysis, we briefly summarize the security mechanisms available on COTS data streaming platforms as a way to identify: i) common flaws not covered by the deployed solutions; and ii) potential misuses of the available security modules.

Most existing data streaming platforms center their security efforts around three main services: authentication, authorization, and data encryption. For authentication, both Kafka and rMQ support the Simple Authentication and Security Layer (SASL) framework [14]. Although they differ on the set of natively supported SASL authentication mechanisms, a large group of authentication protocols including LDAP, Kerberos and OAuth2 is available through external plug-ins. Regarding authorization, both Kafka and rMQ use access control lists (ACLs) to control which users are allowed to perform certain operations on given resources. However, none of the tools have native support for specifying complex policies such as the ones used to manage user groups and require external modules to deploy, e.g., role- or attribute-based access control (a.k.a. RBAC and ABAC policies, respectively). Finally, Kafka and rMQ provide data encryption by means of the TLS protocol where both tools support version 1.3. None of the security mechanisms mentioned above are enabled on the platforms we studied by default. Therefore, users must explicitly configure and activate them according to their needs and preferences.

(R6) What kind of misuses are data streaming platform security mechanisms mostly susceptible to? Given our experience deploying different data streaming platforms on a highly secure environment, we decide to share some of the lessons we learned in terms of configuration mistakes. The most common issue is the fact that current platforms are not secure by default, meaning the user must explicitly set a security mechanism to enforce policies other than "no action". For example, Kafka's data encryption approach requires platform operators to explicitly set TLS listeners on each broker otherwise traffic will flow in plaintext. Moreover, operators must *also* remember to delete any existing plaintext listener, which are automatically initialized. A second main challenge we faced was properly configuring an ACL. Since none of the analyzed tools provide a high-level description language for specifying access control policies, operators may end up having to set permissions at an extremely fine granularity, which makes the process cumbersome and error-prone, especially in large-scale scenarios. Furthermore, there is no way to validate a deployed configuration other than exhaustive (i.e., brute force) testing.

VI. Research directions

Automatic parameter tuning. As we observed in Section V-A, proper parameter tuning can play a crucial role in the performance of current data streaming platforms. Unfortunately, the large number of parameters and the complex and non-linear interactions among them make tuning the configuration of data streaming platforms labor-intensive and time-consuming. While automatic configuration tuning tools exist (e.g., [15]), they normally: i) focus on performance-related parameters without paying attention to their side effects on, e.g., reliability, security, and execution cost; and ii) require a large number of training samples to create accurate models to suggest new configurations. An interesting direction for future investigation is to apply more recent data-driven approaches (e.g., reinforcement learning from human feedback) to explore the large configuration space while considering other aspects besides performance. For example, a reinforcement learning agent can learn and receive rewards based on the human preferences for the system's behavior.

High-coverage testing. Modern distributed systems, including data streaming platforms, usually adopt complex designs that involve intricate communication protocols, non-determinism, and the combination of several interdependent modules. As a result, it becomes extremely hard to test all possible states that these systems can reach out which causes bugs to be hidden until production. Despite recent efforts to extensively test distributed systems while in-house, few initiatives have targeted (or been adopted on) data streaming platforms. In this sense, an important open research challenge is to devise automated and systematic methods and tools for high-coverage testing of data streaming platforms. This includes a framework for test specification as well as tracing.

VII. Conclusion

In this paper, we explore how different data streaming platforms can support mission-critical applications. We use a military coordination scenario as a representative use case and assess the platforms based on their performance, reliability, and security. We find that existing platforms have different trade-offs depending on their design and configuration and that they need to cope with varying network quality and security threats. We also suggest some research directions for enhancing the platforms in this domain, such as automatic parameter tuning, high-coverage testing, and policy-based access control. We hope that our work can inspire further research and development in this domain.

References

[1] G. Fu, Y. Zhang, and G. Yu, "A fair comparison of message queuing systems," *IEEE Access*, vol. 9, pp. 421–432, 2021.

[2] R. Maharjan, M. S. H. Chy, M. A. Arju, and T. Cerny, "Benchmarking message queues," *Telecom*, vol. 4, no. 2, pp. 298–312, 2023.

[3] K. Liu and Y. Jiang, "High performance shipborne message queuing service prototype system based on apache pulsar," in *2021 IEEE 2nd International Conference on Information Technology, Big Data and Artificial Intelligence (ICIBA)*, vol. 2, 2021, pp. 865–870.

[4] L. Rosa, W. Song, L. Foschini, A. Corradi, and K. Birman, "Derechodds: Strongly consistent data distribution for mission-critical applications," in *MILCOM 2021 - 2021 IEEE Military Communications Conference (MILCOM)*, 2021, pp. 684–689.

[5] K. Fowler, "Mission-critical and safety-critical development," *IEEE Instrumentation & Measurement Magazine*, vol. 7, no. 4, pp. 52–59, 2004.

[6] Track and monitor vehicles and assets. Accessed: 2023-03-29. [Online]. Available: https://www.rti.com/developers/case-code/vehicle-tracking.

[7] Mission-critical, real-time fault-detection for nasa's deep space network using apache kafka. Accessed: 2023-03-29. [Online]. Available: https://videos.confluent.io/watch/xRmEDYjAhiVu56xcmMy6Yr

[8] A. Alquraan, H. Takruri, M. Alfatafta, and S. Al-Kiswany, "An analysis of network-partitioning failures in cloud systems," in *Proceedings of the 13th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'18. USA: USENIX Association, 2018, p. 51–68.

[9] P. Dobbelaere and K. S. Esmaili, "Kafka versus rabbitmq: A comparative study of two industry reference publish/subscribe implementations: Industry paper," in *Proceedings of the 11th ACM International Conference on Distributed and Event-Based Systems*, ser. DEBS '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 227–238.

[10] M. Albano, L. L. Ferreira, L. M. Pinho, and A. R. Alkhawaja, "Message-oriented middleware for smart grids," *Elsevier Computer Standards & Interfaces*, vol. 38, pp. 133–143, 2015.

[11] Pinetdalhousie/mission-critical-messaging-platforms. Accessed: 2024-01-15. [Online]. Available: https://github.com/PINetDalhousie/mission-critical-messaging-platforms

[12] N. Burow, R. Burrow, R. Khazan, H. Shrobe, and B. C. Ward, "Moving target defense considerations in real-time safety- and mission-critical systems," in *Proceedings of the 7th ACM Workshop on Moving Target Defense*, ser. MTD'20. New York, NY, USA: Association for Computing Machinery, 2020, p. 81–89.

[13] J. Wang, A. Li, H. Li, C. Lu, and N. Zhang, "Rt-tee: Real-time system availability for cyber-physical systems using arm trustzone," in *2022 IEEE Symposium on Security and Privacy (SP)*, 2022, pp. 352–369.

[14] Sasl overview (gnu simple authentication and security layer 2.2.0). Accessed: 2023-09-12. [Online]. Available: https://www.gnu.org/software/gsasl/manual/html_node/SASL-Overview.html

[15] M. Bilal and M. Canini, "Towards automatic parameter tuning of stream processing systems," in *Proceedings of the 2017 Symposium on Cloud Computing*, ser. SoCC '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 189–200.

BIOGRAPHIES

Md. Monzurul Amin Ifath is a Doctoral student at Dalhousie University, where he conducts research on distributed systems, computer networks, and machine learning. Monzurul holds a Bachelor's degree from BUET, Bangladesh, and has three years of experience working with DRDC and GDMS-C on collaborative projects. His research aims to develop novel solutions for challenging problems in the domain of data streaming platforms.

Miguel Neves received his PhD in computer science from UFRGS, Brazil, in 2020. He is currently an AI Researcher at Samsung Research. Previously, he worked as a Postdoctoral Researcher at Dalhousie University. Miguel has over four years of experience with research and development projects in collaboration with companies such as Dell, Intel, and GDMS-C. His current research interests lie on the interplay of artificial intelligence, security, networking, and distributed systems.

Brandon Bremner is a Software Engineer with over six years of experience in the industry. Brandon has worked on several innovative research and development projects in the cyber and acoustic domains. He currently works at General Dynamics Mission Systems Canada (GDMS-C) as a Software Engineering Developer. Brandon has a Bachelor of Computer Science with a Specialization in Communications Technologies and Cyber Security from Dalhousie University.

Jeff White received his BSc in Computational Science from the University of Saskatchewan in 1984. He has worked at the University of Saskatchewan, Alberta Research Council, Alta Vista and for over the last 20 years with General Dynamics Mission Systems Canada (GDMS-C) as a lead systems architect. He has been the technical lead on several research and development projects, including future tactical networks, distributed infrastructure services and applications frameworks, and efficient data distribution mechanisms.

Tomas Szeredi received his PhD in physics from McMaster University in 1993. He has worked for IBM, MacDonald Dettwiler, and for over 20 years at General Dynamics Mission Systems Canada (GDMS-C). Tomas has extensive experience designing, developing, and deploying large scale, complex, distributed systems. Tomas is currently a Senior Software Systems Architect at GDMS-C focusing on Army land systems and Innovation.

Israat Haque is an Associate Professor in the Faculty of Computer Science at Dalhousie University and works in the areas of Software-Defined Networking (SDN) and the Internet of Things (IoT), focusing on the performance, security, and reliability aspects of wired and wireless networked and distributed systems. She received her Ph.D. from the Department of Computing Science at the University of Alberta. Then, she held an NSERC post-doctoral position at the University of California, Riverside, before joining Dalhousie University.