Vol.2022-IOT-58 No.7
2022/7/12

Pub/Sub メッセージングシステムの耐障害性検証ツールの設計と実装

轟木皓平[1]　近堂徹[1,2]　相原玲二[3]

概要：近年，広域に分散した IoT デバイスが生成するセンサデータの収集などに Pub/Sub メッセージングシステムが利用されるようになってきている．需要の増加に伴い，メッセージングシステムの性能評価ツールがいくつか開発されているが，これらは障害が発生していない状態での測定を目的としており，障害発生時の挙動を定量的に検証することは難しい．メッセージングシステムは分散システムの中核になることから，単一障害点を排除し可用性や耐障害性を向上させるための検証技術が重要になる．本研究では，コンテナ技術を利用してメッセージングシステムを任意のサーバ上に展開し，Chaos Engineering ツールを用いて指定した障害を発生させることで，メッセージングシステムの耐障害性を検証可能なツールの実装を行った．本ツールではクライアントを分散配置した上で，コンテナ停止によるインスタンス障害時の測定，ネットワークエミュレーションを利用したネットワーク障害時の測定，負荷ツールを利用したリソース障害時の測定を可能にする．実装したツールを用いて，複数のメッセージングシステムに対して障害を注入した場合の挙動について評価を行った．

キーワード：分散メッセージングシステム，Publish/Subscribe，耐障害性，コンテナ仮想化

Design and Implementation of a Fault Tolerance Verification Tool for Pub/Sub Messaging Systems

KOHEI TODOROKI[1]　TOHRU KONDO[1,2]
REIJI AIBARA[3]

1. はじめに

近年，広域に分散した IoT デバイスが生成するセンサデータの収集や，マイクロサービスアーキテクチャのアプリケーション間のデータ通信など，分散したエンティティ間の通信にメッセージングシステムが利用されるようになってきている．メッセージングシステムが採用する Publish/Subscribe モデル[1]は，データを生成する Publisher，データを分配する Broker，データを処理する Subscriber から構成され，トピックを介して Publisher から Subscriber にデータ配信を行うことで，データの生成や処理を行うアプリケーションを非同期かつ柔軟に増減することができる．

最近では多くのメッセージングシステムがオープンソースとして開発されており，実装方法の違いからそれぞれが異なる特徴を持つ．そのため，各メッセージングシステムに対して様々な比較や調査が行われている[2][3]．開発するシステムによってメッセージングシステムに求められる機能・性能はリアルタイム性や拡張性，耐障害性など様々であり，選択するメッセージングシステムが開発要件を満たしているかを判断することは重要になる．このような需要の増加に伴い，メッセージングシステムに対して定量的な評価を行うツールがいくつか開発されている[4][5]．しかし，これらのツールは障害が発生していない状態（以下，正常系と呼ぶ）での性能評価を目的としているため，サーバの停止やネットワーク障害などが発生した状態（以下，異常系と呼ぶ）における挙動を定量的に評価することは難しくなっている．メッセージングシステムは分散システムの中核になることから単一障害点を排除し可用性や耐障害性を向上させるための検証技術が重要となる．

本研究ではメッセージングシステムの耐障害性を検証可能なツールの設計と実装を行った．本ツールはコンテナ技術を利用して分散した実行環境にメッセージングシステムと任意のクライアントを自動展開し，ユーザが指定可能な様々な障害を測定中に発生させることで，障害発生時のシステムの挙動把握や性能解析などを支援することを目的とする．

本稿では 2 章で Pub/Sub メッセージングシステムの機能を実装したオープンソースと，既存の評価ツールについて説明する．3 章では本ツールの設計と実装について述べ，4 章では本ツールを利用して複数のメッセージングシステムの性能評価を行った結果について述べる．最後に 5 章で本研究のまとめと今後の課題について述べる．

2. 関連研究

2.1 Pub/Sub メッセージングシステム

Pub/Sub メッセージングシステムではデータ配信を行う Broker の役割をメッセージングシステムが担う．Pub/Sub

1 広島大学 大学院先進理工系科学研究科
Graduate School of Advanced Science and Engineering, Hiroshima University
2 広島大学 情報メディア教育研究センター
Information Media Center, Hiroshima University
3 広島大学 学長特命補佐（デジタル担当）
Executive Advisors to the President, Hiroshima University

メッセージングシステムで構成されるシステム例を図 1 に示す．データのやり取りはトピックと呼ばれる論理的なチャネルを介して行うため，データの生成や処理を行うアプリの増減が容易になっている．メッセージングシステムにはアプリケーション層での伝送保証が提供されている．伝送保証には最大一回の送信を保証する at-most-once（損失の可能性あり），送信先から受信応答（以下，ACK と呼ぶ）が返ってこなければ再送し最低一回の送信を保証する at-least-once（重複の可能性あり），重複なく確実に一回送信を保証する exactly-once の 3 つのレベルがある．また，メッセージングシステムには Pull 型と Push 型が存在する．Pull 型は Subscriber から Broker へのメッセージ要求に対して，Broker で永続化されたメッセージを配信する．Push 型は Broker がメッセージを受信した時点で Subscriber にメッセージを配信する．

近年のメッセージングシステムはトピックのメッセージを分散して Subscriber に配信する負荷分散や，複数の Broker でメッセージを複製し耐障害性を向上させるレプリケーションなど様々な機能を持つ．以下では本稿で評価を行った 3 つのメッセージングシステムの特徴について述べる．

2.1.1 Apache Kafka

Apache Kafka[6]は scala 言語で記述されたオープンソースのメッセージングシステムであり，Pull 型として実装されている．Kafka は高いスケーラビリティと耐障害性を提供する．Kafka のスケーラビリティは複数の Broker（Kafka サーバ）をクラスタ構成にすることで実現されている．クラスタで作成されたトピックはパーティションと呼ばれる単位で分割され，クラスタ内の Broker に分散配置される．メッセージはパーティション単位で送受信されるため複数の Broker での分散処理が可能になっている．Kafka の耐障害性はクラスタ内の Broker にパーティションの複製を作成するレプリケーション機能によって提供されている．レプリケーションではメッセージの送受信を行う Leader のパーティションと，複製のみを行う Follower のパーティションで構成される．Publisher の伝送保証はサーバからの ACK タイミングを複製数で指定することができる．0 を指定した場合は応答を必要としない at-most-once に，1 を指定した場合は Leader に書き込まれたタイミングで ACK を返す at-least-once になる．all を指定した場合，全ての Follower にメッセージが複製されたタイミングで ACK を返す．Subscriber は全ての Follower に複製されたメッセージのみを Leader から読み込むことでメッセージの整合性を保つ．

2.1.2 NATS

NATS[7]は Go 言語で記述されたオープンソースのメッセージングシステムであり，Push 型として実装されている．

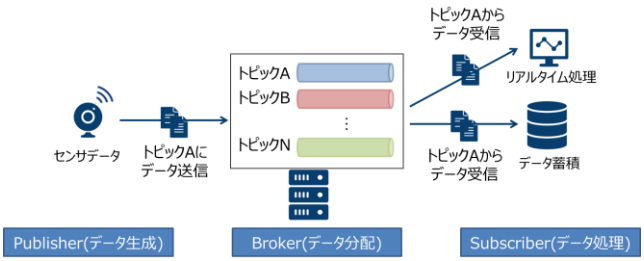

図 1 Pub/Sub メッセージングシステム

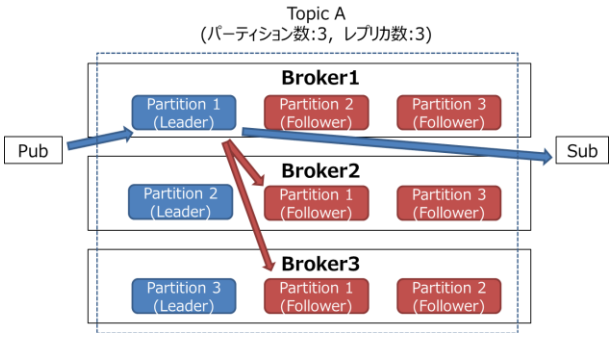

図 2 Kafka の動作概要

NATS は軽量なバイナリであり，大規模なクラウドインスタンスから IoT デバイスまで実行するサーバを限定しない．NATS は Subject と呼ばれる文字列を利用してメッセージのやり取りを行う．メッセージは永続化されず，送信時に Subscriber が存在しない場合，メッセージは破棄される．NATS は at-most-once にのみ対応している．NATS では耐障害性を提供するために複数の Broker（NATS サーバ）でクラスタを構築することが可能になっている，クラスタは全ての Broker が相互に接続されるフルメッシュ型になっており，リアルタイムに接続情報を通信することでトポロジの変更に自動で対応する．そのため，特定の Broker の Subject に送信されたメッセージが異なる Broker の Subscriber に自動でルーティングされる．NATS クライアントは接続している Broker がダウンした場合，自動でクラスタの異なる Broker に接続することで耐障害性を向上させている．

2.1.3 JetStream

JetStream は NATS の拡張機能として提供される，メッセージの永続化に対応したメッセージングシステムである．JetStream は指定した Subject に送信されてきたメッセージを永続化する Stream と，Stream のメッセージを Subscriber に配信する Consumer の 2 つのコンポーネントによって構成される．Consumer の設定を変更することで Push 型と Pull 型の両方が実装されている．JetStream では Stream にメッセージが保存されたタイミングで ACK を返す at-least-once がサポートされている．JetStream の耐障害性は Stream や Consumer をクラスタ内で複製するレプリケーション機能によって提供されている．レプリケーションは RAFT アル

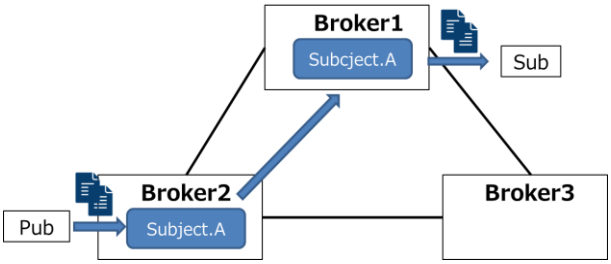

図 3 NATS の動作概要

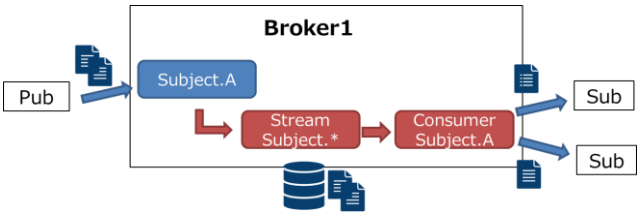

図 4 JetStream の動作概要

ゴリズム[8]によって実装されており，可用性を向上させるためには奇数台でクラスタを構築する必要がある．クラスタでは(複製数-1)/2 の台数までのサーバ停止に対応することができる．Stream のレプリケーションでは 1 台が Leader として選出され，Publisher に対する ACK 処理などは Leader によって行われる．

2.2 既存の評価ツール

2.2.1 MQTTLoader

MQTTLoader[4]は Java で実装された，MQTT によるメッセージングシステムに対する負荷テストを行うクライアントツールである．MQTT は固定ヘッダが最小で 2 バイトの軽量なプロトコルであり，トラフィックの削減を行いたい IoT デバイス等に適している．MQTTLoader では MQTT の最新の MQTTv5 で追加された shared-subscription 機能に焦点を当てた測定が可能になっている．

MQTTLoader は任意のサーバで Publisher や Subscriber を実行することができる．設定ファイルを利用して接続するサーバ，実行時間，メッセージサイズや QoS など様々なパラメータを制御することができる．実行時に得られる結果は Publisher・Subscriber のスループットの平均や最大，メッセージのレイテンシである．また，output パラメータによって Subscriber が受信したメッセージを保存可能であり，レイテンシの時間変化を分析可能になっている．

2.2.2 THE OPENMESSAGING BENCHMARK FRAMEWORK

THE OPENMESSAGING BENCHMARK FRAMEWORK [5]はクラウド上で分散メッセージングシステムを展開しベンチマークを実行するツール群である．AWS もしくは AlibabaCloud 上に数回の CLI コマンドで実行環境を構築可能な容易さが特徴になっている．Apache Kafka や NATS，Apache Pulsar など 7 つのメッセージングシステムがサポートされている．ベンチマークでは極端に小さいメッセージサイズや短時間の測定のようなエッジケースではなく，メッセージ生成のバーストや大量のトピック，長時間の実行などの本番環境のようなユースケースを想定している．実行時に得られる結果は Publisher・Subscriber の平均スループット，メッセージの平均レイテンシである．

2.2.3 Testing the Reliability of Apache Kafka（TRAK）

TRAK[9]は Apache Kafka の信頼性を調査するために開発されたツールである．Docker Compose を使用して 1 台のホスト上で Kafka クラスタの起動, 再起動, スケールアップ, スケールダウンを行うことができる．構築されたクラスタの Kafka コンテナに対して，障害注入フレームワークである Pumba を利用することでコンテナ間の通信遅延やパケットロスを発生させることが可能になっており，Kafka のネットワーク状態がメッセージの重複や損失に及ぼす影響を調査することが可能になっている．

2.3 既存ツールの課題

2.2 節で説明した既存ツール群には 3 つの課題が存在する．1 点目は実行する環境が限定されている点である．1 ホスト上でのクラスタ構築とクライアントの同時実行はマシンリソースの競合が発生する可能性があるため，任意の分散した環境での測定を行えることが望ましい．2 点目は耐障害性に着目した測定が困難な点である．メッセージングシステムは複数のサーバでデータを複製し耐障害性を向上させている．この機能を有効にした場合のスループットに対する検証は行われているが[10]，Broker 間でのネットワーク障害が発生した場合におけるシステム全体のスループットやレイテンシへの影響は検証されていない．分散システムにおけるネットワーク障害やインスタンス障害時の性能低下や動作把握が重要となる．3 点目はネットワークの地理的距離やネットワーク環境を考慮した測定が困難な点である．近年ではマルチ AZ 環境など，地理的に離れた環境に分散配置し可用性を向上させることが推奨されている．サーバを地理的に離れた場所に分散配置した場合，ネットワークのレイテンシが増加するため，新たな検証が必要になるが，そのような測定環境の構築はコストが高くなる．本研究では上記の 3 つの課題を解決可能なシミュレーションツールの開発を目的とした．

3. 提案ツール

3.1 ツール設計

本研究ではメッセージングシステムの実行環境を柔軟かつ自動で展開・削除でき，ユーザが指定した障害を任意

のタイミングで発生させることで障害発生時の動作把握や性能の定量評価を行うことができるシミュレーションツールの開発を行った．シミュレーション環境の構築には，プラットフォームへの依存を小さくし，かつ展開と削除を容易に行うことができる Docker を利用した．また，実際のネットワークを仮想化し任意のコンテナ間の通信に障害や疑似的な遅延を注入できるよう Docker のオーケストレーションツールである DockerSwarm[11]を利用した．2 つの技術を導入することでメッセージングシステムとクライアントを分散した環境で自動展開した測定シミュレーションを可能にしている．ツールの特徴であるメッセージングシステムの耐障害性の検証を行うために，Docker コンテナレベルでの障害を注入できる Chaos Engineering ツールの Pumba[12]を利用した．Pumba を利用することで，測定中にコンテナに対して様々な障害を注入できる．ツールでは任意の Pub/Sub クライアントの展開を行えるようにし，同一環境で複数のメッセージングシステムを変更しながらのスループットやレイテンシなどの性能比較，パラメータや Broker のトポロジを変更した時の性能変化に対する評価，サーバの停止やネットワーク間の遅延・パケットロスの増加といった障害を注入した場合における性能変化やメッセージの損失・重複率に対する評価などを可能にする．

本ツールではユーザが使用するサーバと構築したい環境，発生させたい障害をテンプレートファイルとして YAML 形式で定義する．また，ツールを実行するサーバを管理ノード，シミュレーション環境を構築するサーバを実行ノードとして定義する．

3.2 ツール実装

3.2.1 テンプレートファイル

テンプレートファイルは実行環境を構築するサーバ，使用するメッセージングシステムやクライアントとパラメータ，発生させる障害を YAML 形式で記述したファイルであり，ツールはこのファイルを基にシミュレーションを実行する．図 5 に Kafka，Publisher，Subscriber を 1 台ずつ展開する際のテンプレートファイルの例を示す．

base には ssh 接続するユーザと実行ノードの情報を記載している．sudo のパスワードや ssh のキーファイルなどは環境変数を介した設定が可能である．

systems には Kafka と Zookeeper，Publisher，Subscriber の情報や設定を記載している．各コンテナの展開先の指定には base で設定した実行ノードの名前を node_name に設定する．コンテナの動作変更には環境変数か設定ファイルを利用する．環境変数を利用する場合はテンプレートファイルの environment に記述が可能である．設定ファイルを利用する場合，ツールはテンプレートファイルに記載したコンテナ名（例では kafka-1 や kafka-pub-1）の設定ファイルを格納するディレクトリを管理ノード上に作成することができ，ユーザはそのディレクトリ内にコンテナ用の設定ファイルを配置する．ツールはシミュレーション実行時にそのディレクトリ内のファイルを実行ノードへ転送し，コンテナへのマウントを自動で行うことでコンテナの動作を変更できる．デフォルトでは実行ノードの/tmp/<コンテナ名>/configs に転送し，コンテナの/tmp/configs にマウントする．

```
base:
  user: todoroki                          ┐
  keyfile: $KEYFILE                       │ sshを行う
  sudopass: $SUDOPASS                     ┘ ユーザー情報
  nodes:
    node:                                 ┐
      address: <実行ノードIP>             │ 実行ノードの指定
    manager:                              │ と名前付け
      address: <実行ノードIP>             │
      is_manager: true                    ┘
systems:
  type: kafka                             ┐ メッセージングシステム
  duration: 120s                          ┘ の指定と実行時間
  broker:                                 ┐
    zookeeper:                            │
      zoo-1:                              │
        node_name: node                   │
        environment:                      │
          <環境変数で設定を記述>          │ Broker設定
    kafka:                                │
      kafka-1:                            │
        node_name: node                   │
        environment:                      │
          <環境変数で設定を記述>          ┘
  publisher:                              ┐
    kafka-pub-1:                          │
      image: <コンテナイメージ名>         │
      node_name: node                     │
      command: <コンテナの実行コマンド>   │ Publisherと
  subscriber:                             │ Subscriber設定
    kafka-sub-1:                          │
      image: <コンテナイメージ名>         │
      node_name: node                     │
      command: <コンテナの実行コマンド>   ┘
actions:                                  ┐
  fault1:                                 │
    target_containers:                    │
      - kafka-1                           │
    start: 30s                            │ 障害の設定
    duration 20s                          │
    mode: delay                           │
    params:                               │
      time: 10                            ┘
```

図 5 テンプレートファイルの記述例

actions には注入する障害の情報を記載している．対象のコンテナや障害の種類，発生時刻や継続時間などを指定することができる．

3.2.2 実装と動作概要

本ツールは管理ノードから ssh や sftp を利用して実行ノード上にシミュレーション環境の展開を行う．実行ノードには事前に管理ノードから ssh 可能なユーザの作成と，Docker のインストールが必要になる．ツールは python3.8

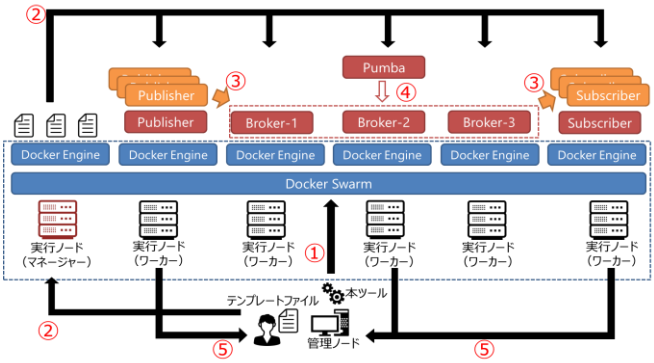

図 6 ツールの動作概要

を用いて実装を行った．

本ツールの動作概要を図 6 に示す．まず，ツールは実行ノード上で Docker Swarm を利用したクラスタとオーバーレイネットワークを構築する（図 6 ①）．次にテンプレートファイルに記述されたメッセージングシステム（Broker）の情報から Docker コンテナを展開する定義ファイルを作成し，実行ノードに sftp で転送して Swarm 上に Broker を展開する（図 6 ②）．この展開処理には，設定ファイルの転送やコンテナとホストが共有するディレクトリの作成処理なども含まれる．Broker の展開後，同様に Publisher と Subscriber を展開し測定を開始する（図 6 ③）．もしテンプレートファイルに発生させたい障害の記述があれば，ツールは Broker や Publisher，Subscriber と同様に Pumba コンテナを対象のコンテナが起動している実行ノードに展開することで，ユーザが指定したコンテナの停止を行ったり，コンテナ間のネットワークに遅延やパケットロスを発生させたりすることができる（図 6 ④）．計測が完了後，ツールは Publisher･Subscriber が出力した実行結果のファイル，Pumba，Publisher，Subscriber コンテナが標準出力したログを管理ノードに収集する（図 6 ⑤）．デフォルトでは実行ノードの /tmp/<コンテナ名>/results がコンテナの /tmp/results にマウントされ，ツールは実行ノードの/tmp/<コンテナ名>/results から結果を収集する．結果の収集が完了後，コンテナがホストと共有していたディレクトリをすべて削除し，DockerSwarm のクラスタを削除することで実行前の環境に戻す．この処理により，次の計測が前回の計測の影響を受けないようになっている．

4. 評価

本章ではツールの実用性を示すために，メッセージの永続化の有無，伝送保証やレプリケーション方法が異なる Kafka，NATS，JetStream の 3 つのメッセージングシステムに対して Publisher と Subscriber のスループット，メッセージのレイテンシの 3 つの指標の定量評価を行った．スループットは 1 秒間に処理されたメッセージ数とバイト数であり，レイテンシは Publisher がメッセージを送信してから Subscriber が受信するまでの時間である．評価は正常系と異常系の 2 つの観点から行い，正常系における測定ではメッセージングシステムの性能把握を支援できることを示し，異常系における測定では障害発生時におけるメッセージングシステムの性能を定量的に評価できることを示す．評価環境を表 1 に，使用したコンテナイメージを表 2 に示す．クライアントは Java と各メッセージングシステムのライブラリを用いて実装した．

表 1 評価環境

ホスト名	スペック
ホストマシン	CPU: Intel(R) Xeon(R) CPU E 5620 @ 2.40GHz Mem:80GB OS:VMware ESXi 6.5U3
Publisher， Subscriber，Broker×3	vCPU：4 Mem：8GB OS:Ubuntu Server 20.04

表 2 使用したコンテナイメージ

コンポーネント	イメージ：タグ
Publisher，Subscriber	measurement-client
Zookeeper	cp-zookeeper:6.2.4
Kafka	cp-kafka:6.2.4
JetStream	nats:2.8.1

4.1 正常系における評価

正常系における評価ではメッセージングシステムがスタンドアロン構成の場合とクラスタ構成でレプリケーションを有効にした場合の 2 つの測定を行った．

4.1.1 スタンドアロン構成でのスループット性能

本項では Kafka，NATS，JetStream のスタンドアロン構成における正常系の性能を比較するために，メッセージングシステムをスタンドアロン（1 台）構成で展開し，Publisher と Subscriber を 1 台ずつ異なる VM に展開して 3 つの評価指標を 2 分間計測した．JetStream の Subscriber には Pull 型を採用し，Kafka は ACK を返さない場合と返す場合とを計測対象とした．IoT では小さいデータサイズでの利用も考えられることから，メッセージサイズを 100B から 64KB まで変化させて計測を行った．

測定結果の Publisher のスループットを図 7，図 8 に示す．Kafka は ACK の数を kafka-acks-<0 or 1>でラベルに表記している．スタンドアロン構成では，メッセージの永続化と ACK を必要としない NATS は他の 2 つのメッセージングシステムに比べて 3~10 倍のスループット性能を持つ

ことが分かる．永続化を行う Kafka と JetStream を比較すると，Publisher がバッチ機能を標準で持つ Kafka はメッセージサイズが小さい場合でも高いスループットでの実行が可能になっている．ACK が 1 台になってもスループットの低下が見られないのは ACK 待機によって Publisher のバッファに溜まったメッセージをバッチ送信しているためだと考えられる．JetStream は ACK 待機と送信のオーバヘッドによってメッセージサイズが小さい場合のスループットは低くなっているが，サイズの増加に伴って上記のボトルネックが解消されるため，高いスループットが期待できる．

次にレイテンシの結果を図 9 に示す．Push 型の NATS のレイテンシが Pull 型の 2 つと同程度になっているのは Publisher の生成速度に Subscriber の処理が追いついておらず，Subscriber 側のバッファでの滞留時間やバッファから溢れたメッセージの破棄が原因だと考えられる．また，メッセージサイズが 1KB と 4KB の JetStream のレイテンシが他に比べて 2 桁小さくなっている．これは JetStream の Pull 処理で 1 度に取得するメッセージ数として指定したパラメータ（150 個）が Publisher のスループットに対して効率良く受信できるパラメータであったと考えられる．

以上のことからスタンドアロン構成のメッセージングシステムでは，厳密なメッセージ保証を必要とせず高いスループットが必要になる場合は NATS が適している．一方で，メッセージの永続化と非同期受信が必要な場合において，Kafka はどのメッセージサイズにおいても安定した性能を提供しており，メッセージサイズが大きい場合は JetStream の適用も考えられる．また，JetStream はデフォルトで一定時間内のメッセージ重複を削除する機能を有しているため，確実に 1 度の配信が必要なアプリケーションでの活用が有効だといえる．レイテンシを小さくする場合，Publisher の生成速度を考慮した Subscriber のパラメータ設定や分散処理が必要になる．

4.1.2 クラスタ構成でのスループット性能

本項では Kafka，JetStream の耐障害性を向上させるレプリケーション機能が正常系の性能に及ぼす影響を調査するために，メッセージングシステムを 3 台のクラスタ構成にし，レプリケーション機能を有効にした状態でスタンドアロン構成の時と同様の測定を行った．Kafka では ACK が 1 台の場合と 3 台（all）の場合でそれぞれ行った．

スタンドアロン構成とクラスタ構成での Publisher のスループット比較を図 10 に示す．ラベルの single はスタンドアロン構成を，replication はクラスタ構成の結果を表している．この結果から，いずれの場合もクラスタ構成にすることでスループットの低下が見られ，特に Kafka において ACK が 3 台の場合は 90%程度の性能低下となった．これは全ての Broker でメッセージの複製が完了する前の処理が大きく影響しているためである．一方，ACK が 1 台の

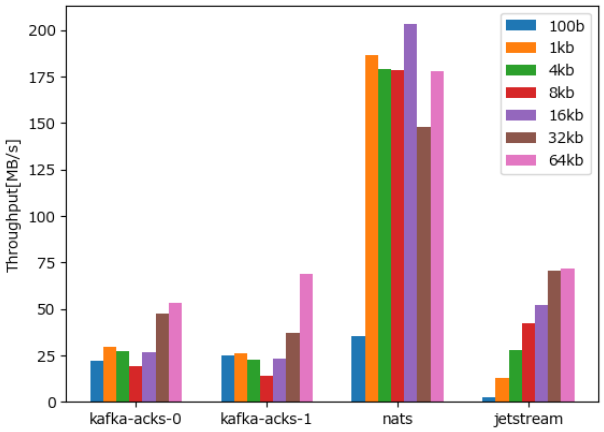

図 7 スタンドアロン構成時の Publisher スループット（秒間あたりのバイト数）

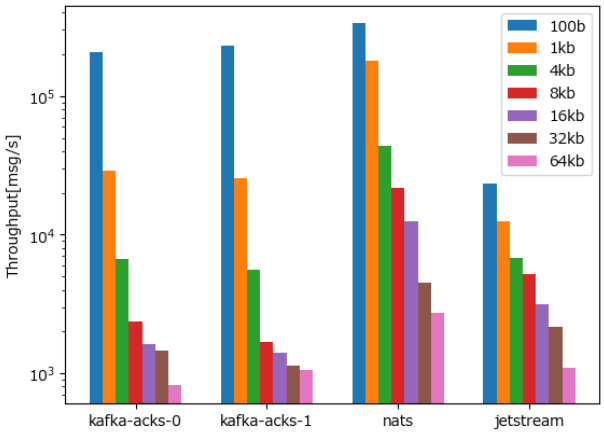

図 8 スタンドアロン構成時の Publisher スループット（秒間あたりのメッセージ数）

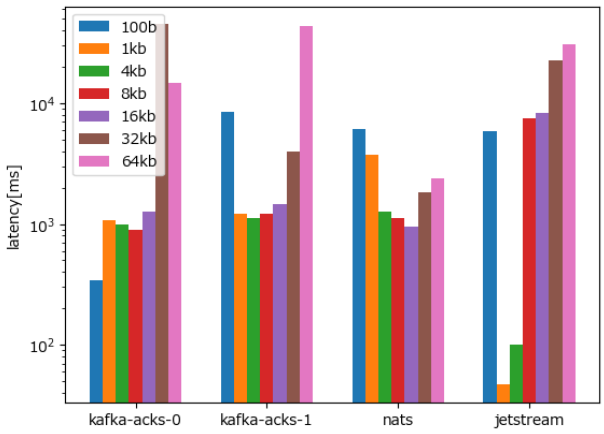

図 9 スタンドアロン構成時のメッセージレイテンシ

場合でも 50%程度の性能低下が確認された．

今回の計測では Kafka，JetStream ともに 30MB/sec 前後でスループット限界がみられた．これは，全ての VM が同じ HDD 上で実行されているため，HDD への書き込みがボトルネックになっていると考えられる．物理的に分散した環境での評価やスケーラビリティに対する評価は今後の課題である．

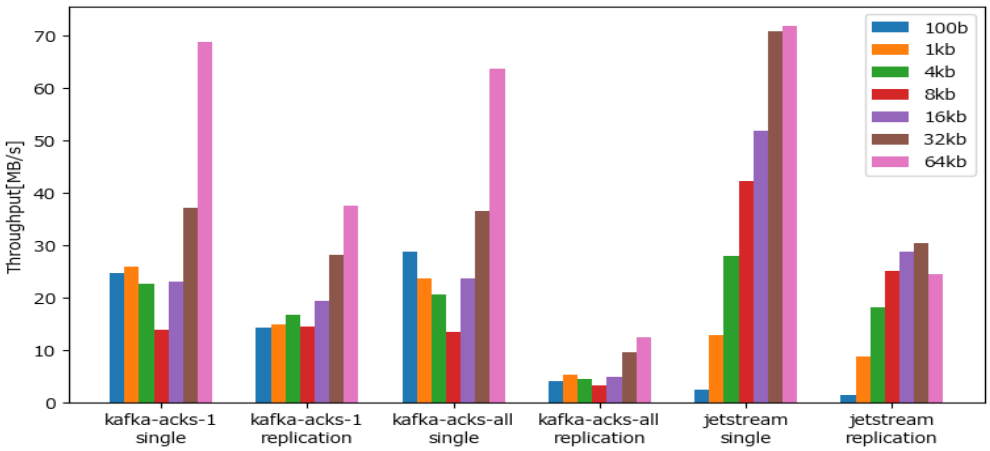

図 10 スタンドアロン構成とクラスタ構成での Publisher スループット比較（秒間あたりのバイト数）

4.2 異常系における評価

異常系における評価ではメッセージングシステム間のネットワーク障害がレプリケーション機能有効時の性能に及ぼす影響を定量的に評価するために，4.1.2 項と同様のクラスタ環境を構築し測定開始から 40 秒後に 20 秒間，Leader の Broker から複製の Broker 方向のネットワークに遅延を注入した．4.1.2 項の結果から，メッセージサイズは Kafka と JetStream とで同程度のスループットが確認された 4KB に固定，Publisher におけるスループットは安定性を考慮し，正常系での平均スループットの 7 割程度の送信レートでメッセージを送信するように設定した．測定は Kafka の ACK が 1 台の場合と 3 台の場合と JetStream に対して行った．ネットワーク障害には 10msec の遅延を注入した．

測定結果を図 11 に示す．赤点線内の区間が Pumba のログから障害が発生していたと推定される時間である．障害発生前の 20 秒間の安定した性能を本測定の正常時とし，障害発生中の性能と比較した結果を表 3 に示す．Kafka の性能は ACK がどちらの場合でも遅延の影響を大きく受けている．ACK が 1 台の場合，Subscriber の性能が 50% 低下しレイテンシは 100 倍以上に増加した．また，ACK が 1 台の場合は Broker 間の遅延に関わらず Leader への書き込みは継続されるため，障害回復後に Broker に溜まったメッセージの配信により Subscriber 側で急激な負荷の増加が発生することが分かる．ACK が 3 台の場合，Publisher 側で 70% 程度の性能低下が確認された．JetStream は注入した遅延分のレイテンシの増加は見られたが大きな性能の低下は見られなかった．

以上からレプリケーション手法によってネットワーク遅延に対する耐性が大きく異なっていることがわかる．Publisher からの送信負荷が同程度の場合，Kafka のレプリケーションはネットワークの性能低下に対して大きく影響を受け，JetStream のレプリケーションはネットワークの性能低下に対する耐性が高くなっていることが分かる．

5. おわりに

本研究ではメッセージングシステムの耐障害性を検証するためのシミュレーションツールの開発を行った．本ツールでは複数のメッセージングシステムを任意の環境に展開することで正常系における性能評価や特性把握の支援と異常系における定量評価を行うことができる．

評価ではスタンドアロン構成の Kafka，NATS，JetStream における正常系での評価と，3 台構成でレプリケーション機能を有効にした Kafka と JetStream の正常系と異常系における評価を行うことで，本ツールがメッセージングシステムの性能把握を支援できることを示した．

今回は全ての Broker を同一ホストの VM 上で行ったが，機器が物理的に分散した環境での評価や，スケーラビリティの評価も行う必要がある．また，ネットワーク障害だけでなく，サーバを停止するインスタンス障害や負荷を注入した場合のリソース障害などに対する評価も行っていく必要がある．加えて，メッセージスループットとレイテンシ以外の指標による評価についても検討していく．メッセージングシステムのメトリックスなどを収集可能にすることでより詳細な分析が可能になると考えているため，これらの機能をツールに組み込むことが今後の課題である．

謝辞

本研究の一部は JSPS 科研費 21K11846 および 2022 年度国立情報学研究所公募型共同研究 22S0207 の助成を受けて実施している．

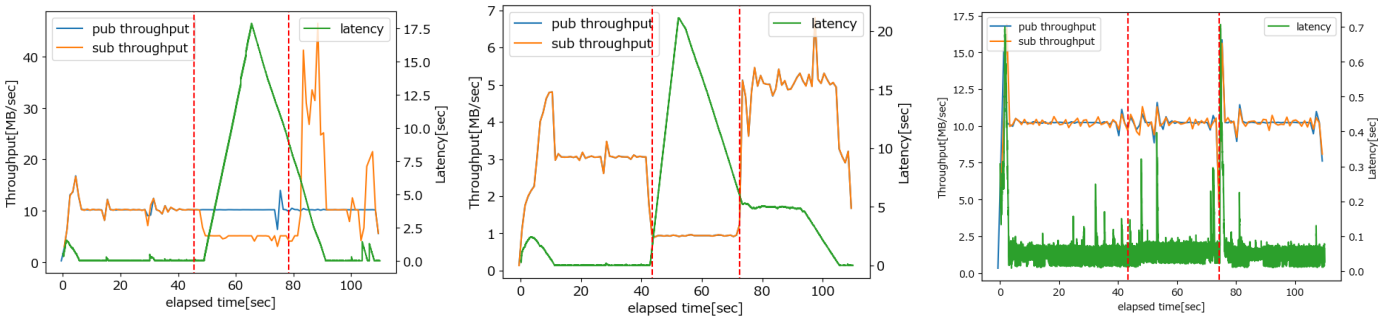

図 11 Kafka（ACK 数 1）（左）と Kafka（ACK 数 3）（中央）と JetStream（右）の
Broker 間ネットワーク障害（遅延 10msec）発生時の性能変化

表 3 正常時と Broker 間に遅延 10msec を注入した異常時の性能比較

メッセージングシステム	Publisher スループット[MB/sec]		Subscriber スループット[MB/sec]		レイテンシ[sec]	
	正常時	異常時	正常時	異常時	正常時	異常時
Kafka（ACK 数 1）	10.216	10.218	10.217	5.362	0.034	9.58
Kafka（ACK 数 3）	2.991	0.94	2.991	0.94	0.043	13.276
JetStream	10.227	10.236	10.233	10.249	0.051	0.062

参考文献

[1] P. T. Eugster, P.A. Felber, R.Guerraoui, and A.-M. Kermarrec, "The Many Faces of Publish/Subscribe", ACM Computing Surveys, vol. 35, no. 2, pp. 114-141, 2003
[2] JhonVineet, XiaLiu. "A Survey of Distributed Message Broker Queues", 3 Apr 2017
[3] Sharvari T, Sowmya Nag K. "A study on Modern Messaging Systems- Kafka, RabbitMQ and NATS Streaming", 8 Dec 2019
[4] Ryohei Bannno, Koki Ohsawa, Yusuke Kitagawa, Takumu Takada, Toshinori Yoshizawa. "Measuring Performance of MQTT v5.0 Brokers with MQTTLoader" 2021 IEEE 18th Annual Consumer Communications & Networking Conference(CCNC), pp.1-2, 9 Jan 2021
[5] THE OPENMESSAGING BENCHMARK FRAMEWORK, https://openmessaging.cloud/docs/benchmarks/ (Accessed Jun. 6, 2022)
[6] Apache Kafka, https://kafka.apache.org/(Accessed Jun. 6, 2022)
[7] NATS, https://docs.nats.io/ (Accessed Jun. 6, 2022)
[8] .Deigo Ongaro, John Oustenrhout. "In Search of an Understandable Consensus Algorithm", 2014 USENIX Annual Technical Conference, June 2014, pp. 305-320
[9] Han Wu, Zhihao Shang, Katinka Wolter. "TRAK: A Testing Tool for Studying the Reliability of Data Delivery in Apache Kafka", 2019 IEEE International Symposium on Software Resitance Engineering Workshops(ISSREW), pp. 394-397, Oct. 2019
[10] 一瀬 絢衣, 竹房あつ子, 中田 秀基, 小口 正人. “Kafka を利用したリアルタイム動画像解析フレームワークの レプリケーションによる性能変化の考察”, xSIG2018, 28-30 May 2018
[11] Docker Swarm, https://matsuand.github.io/docs.docker.jp.onthefly/engine/swarm/ (Accessed Jun. 6, 2022)
[12] Pumba, https://github.com/alexei-led/pumba (Accessed Jun. 6, 2022)