Análise Comparativa entre sistemas de mensageria: Apache Kafka vs RabbitMQ

Filipe Jessé de Castro Arruda[1], **Daves Martins**[2]
filipejesse5@gmail.com, daves.martins@ifsudestemg.edu.br

Abstract. *When one thinks of a distributed application, we soon wonder how their components will conversate, once it will traffic data over network. Considering this, it is said that a service responsible for managing this communication shall attend certain requirements, such as: High transfer ratio, low resource consume, easy management, among others. From this problematic, many messaging systems have been presented as a proposal to administer this communication with the best cost benefit. The present work aims to show two of these systems: Apache Kafka and RabbitMQ, both open source projects, in order to validate which one adapts best for certain scenarios, comparing the perfomance from one to the other. The results show that both technologies deal well with high volume of messages, in which, Apache Kafka contrasts on the management of resources for CPU use, while RabbitMQ stands out on memory and disk.*

Resumo. *Quando se pensa em uma aplicação distribuída, logo imaginamos como seus componentes irão se comunicar, uma vez que ela irá trafegar informações pela rede. Considerando isso pode-se dizer que um serviço responsável por gerenciar essa comunicação deverá atender certos requisitos, como: Alta taxa de transferência, baixo consumo de recursos, fácil gerenciamento, dentre outros. A partir desta problemática, diversos sistemas de mensagerias têm se apresentado com a proposta de gerir esta comunicação com o melhor custo-benefício. O presente trabalho visa apresentar dois destes sistemas: Apache Kafka e RabbitMQ, ambos de código aberto, a fim de validar qual se adapta melhor para determinados cenários, comparando o desempenho de um em relação ao outro. Os resultados mostram que ambas as tecnologias lidam bem com altos volumes de mensagens, sendo que, o Apache Kafka se destaca no gerenciamento de recursos para utilização de CPU enquanto o RabbitMQ se destaca em memória e disco.*

Palavras-chave: mensageria, *throughput*, sistemas distribuídos, análise de performance

1 Instituto Federal do Sudeste de Minas Gerais | Campus Juiz de Fora - MG - Brasil
2 Instituto Federal do Sudeste de Minas Gerais | Campus Juiz de Fora - MG - Brasil

1. Introdução

Com os constantes avanços tecnológicos ao decorrer dos anos é notável alterações nos hábitos da sociedade que os acompanham. Essas evoluções decorrem da necessidade das pessoas de consumirem informações em todos os seus aspectos, seja em redes sociais, canais de notícias ou blogs. O consumo constante e crescente de informação acarreta em uma grande massa de requisições simultâneas nos sistemas que gerenciam esses dados, o que pode ocasionar intermitências ou até fazer com que estes sistemas parem de funcionar por certo período. A partir desses impactos, foram iniciados estudos arquiteturais para que existissem formas de reduzi-los e gerar uma melhor experiência para os usuários.

Diversas abordagens vêm surgindo desde então, propondo novos designs e padrões arquiteturais para lidar com esse "Dilúvio de Dados" [TAN et al., 2013]. Dentre as principais, a abordagem utilizando sistemas distribuídos ganhou força e termos como "Arquitetura baseada em Microsserviços" e "Mensageria[3]" se tornaram comuns. Nesta pesquisa iremos abordar estas novas terminologias que ganharam espaço nos últimos anos e realizaremos experimentos para explorar, validar e comparar suas propostas.

Este artigo irá apresentar uma análise comparativa entre as tecnologias de mensageria *Apache Kafka* e *RabbitMQ*, estas que serão apresentadas de forma clara na próxima seção.

2. Conceituação e Trabalhos Relacionados

Nesta seção iremos apresentar conceitualmente as tecnologias que serão utilizadas nesta pesquisa, assim como os trabalhos relacionados usados como base teórica desta pesquisa.

2.1. Sistemas Distribuídos

Um Sistema Distribuído é definido como um conjunto de componentes, *hardware* e *software,* distribuído em computadores interligados em rede, comunicam-se e coordenam ações apenas enviando mensagens entre si [COULOURIS et al., 2013]. Os computadores que compõem este sistema podem estar separados por qualquer distância, podendo estar até mesmo em continentes diferentes.

Ainda Segundo Coulouris, a definição de sistemas distribuídos traz três principais consequências: concorrência, inexistência de relógio global e falhas independentes.

- **Concorrência**: O sistema precisa saber lidar com a utilização simultânea de recursos para que nenhum módulo e/ou usuário seja afetado;
- **Inexistência de um relógio global**: Possuir componentes separados e independentes, além de significar que cada componente se mantém sozinho, também significa que os componentes não compartilham recursos como relógio físico, ou seja, a sincronização de horários pode ser comprometida;

---

3 Define que sistemas distribuídos podem se comunicar por meio de troca de mensagens (evento), sendo estas mensagens "gerenciadas" por um Message Broker.

- **Falhas independentes**: Como citaremos adiante, os sistemas isolados entre si tornam possível a tolerância de falhas, uma vez que cada componente do sistema pode ser reparado e substituído em tempo real sem impactar o resto da aplicação.

Em regra, um sistema distribuído deve ser facilmente escalável e extensível. Esta característica se dá graças a sua arquitetura ser composta por componentes (i.e., computadores) independentes e isolados [TANENBAUM et al., 2007], o que também é responsável pela resiliência deste modelo arquitetural, uma vez que caso um destes componentes falhe ou seja alterado, o sistema como um todo não deixa de funcionar e muitas vezes nem mesmo é afetado.

2.2. Message-Oriented Middleware (MOM)

No contexto de sistemas distribuídos um dos fatores cruciais para sua estruturação é a forma que sua comunicação é gerenciada. Um sistema do tipo *MOM* fornece um modelo distribuído de comunicação baseado em interações assíncronas. Em outras palavras, este modelo cria uma camada de comunicação onde cada módulo do sistema se conecta para enviar e receber as requisições durante sua execução.

A utilização deste *middleware* permite com que os componentes de um sistema possam se comunicar e não precisam necessariamente ficar parados esperando uma resposta síncrona, isso quer dizer que uma vez que uma requisição é enviada a aplicação pode continuar sua execução e a responsabilidade de entregar a mensagem passa a ser desta camada de comunicação, enquanto o receptor fica responsável por receber e executar a requisição, independentemente do requerente. Segundo [CURRY 2004], um *MOM* pode ser comparado ao serviço postal, onde mensagens são enviadas e a partir do momento que o serviço postal as recolhe, este é responsável pela entrega delas em segurança, e não necessariamente a pessoa que envia vai saber se foi entregue ou não.

2.3. Message Brokers - Mensageria

Como citado no tópico anterior, uma abordagem bem comum para a comunicação de sistemas distribuídos é a criação de uma camada responsável por gerenciar as mensagens enviadas pelos sistema, sejam elas eventos, requisições ou até mesmo *logs* de aplicação.

Uma camada de comunicação possui grande responsabilidade sobre o sistema no qual ela faz parte, uma vez que a mesma precisa lidar com a maior parte da comunicação do sistema, integrar cada componente e lidar com problemas como concorrência e indisponibilidade de algum módulo sem derrubar a aplicação como um todo. Visto isso diversas opções para esta camada surgiram, e estas recebem o nome de *Message Brokers* ou Gerenciadores de Mensageria, em português.

Um *Message Broker* é uma aplicação responsável por lidar com a comunicação e integração de sistemas distribuídos, recebendo mensagens e as distribuindo para as partes que fizerem sentido. Atualmente no mercado existem diversos exemplos de Messages Brokers, no presente trabalho iremos focar em apenas dois: Apache Kafka e RabbitMQ. Falaremos mais sobre eles adiante.

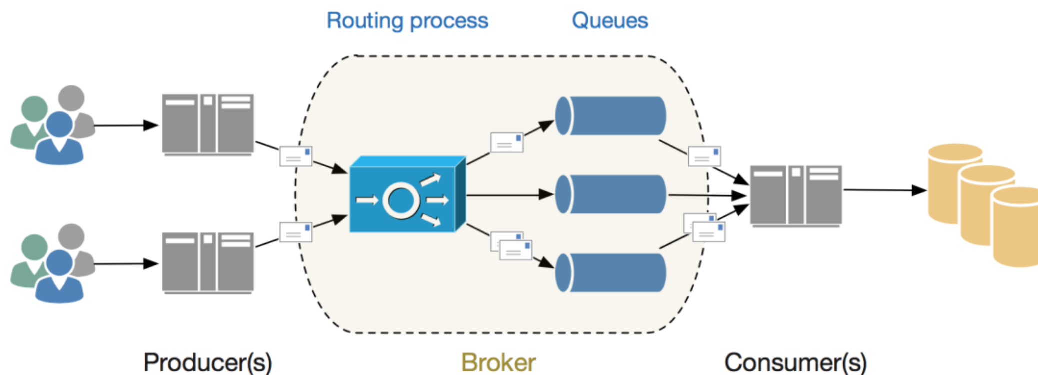

Figura 01: A figura representa a utilização do padrão Pub/Sub. Fonte: Nebbiolo Technologies

2.3.1. Apache Kafka

O Apache Kafka é uma plataforma de código aberto de processamento de streaming desenvolvida pela *Apache Software Foundation* escrita em Scala e Java. O Kafka vem como uma opção resiliente e com uma alta capacidade de processamento de mensagens, já que sua arquitetura foi desenvolvida com este objetivo.

Principais características:

- **Plataforma**: Possui um core e você consegue plugar vários componentes criando um ecossistema em torno dele;
- **Armazenamento**: Todos os dados que entram no Kafka ficam armazenados por ele (existem regras para definir quanto tempo as informações ficam gravadas). Neste ponto o Kafka se destaca, pois pode reprocessar mensagens já enviadas, uma vez que fica armazenada, logo possui uma alta tolerância de falhas.;
- **Utilização de Disco**: Grava as informações em disco e não em memória, entretanto, consegue garantir uma alta performance devido à forma que seu processo de I/O é realizado.

Além destas características o Kafka possui alguns conceitos básicos que são importantes para compreensão dele, estes são *Producers, Consumers* e *Topics.*

- **Producer*:*** Responsável pelo envio de mensagens, no padrão *publisher/subscriber* podemos dizer que seria o *publisher;*
- **Consumer*:*** Responsável por receber as mensagens, no padrão *publisher/subscriber* podemos dizer que seria o *subscriber;*
- **Topic:** É o canal onde está o *streaming* de dados, ou seja, o lugar onde o *consumer* recebe mensagens enviadas pelo *producer.* Além disso, também é onde as mensagens ficam armazenadas para fins de consultas futuras.

2.3.2. RabbitMQ

O RabbitMQ, assim como o Kafka, possui seu código aberto, entretanto este não se apresenta como uma plataforma, seu foco é ser um *middleware* de mensagens e é nesta característica que iremos colocá-lo lado a lado ao Kafka.

O RabbitMQ não possui um protocolo próprio e se comunica utilizando o *Advanced Message Queuing Protocol (AMQP)*[4], provisionando casos de uso de filas com baixa latência.

A sua estrutura consiste em um conjuntos *brokers* responsáveis por gerenciar as filas que receberão as mensagens de cada componente. Diferente do kafka, as filas são *single-thread*, sendo assim a taxa de transferência das mesmas fica limitada pela capacidade das filas.

2.4. Revisão Sistemática

No contexto do presente trabalho, foi realizada uma busca por trabalhos que abordassem de forma qualitativa e quantitativa sistemas de gerenciamento de mensagens, com foco em RabbitMQ e Apache Kafka. Os principais trabalhos encontrados serão apresentados a seguir.

O escopo para a aplicação desta revisão se dá através da utilização de *benchmarkings*[5], técnicas comparativas de performance e *throughput*[6].

O método PICOC foi utilizado para definição dos objetivos e questionamentos do presente trabalho. "PICOC" é um acrônimo para: População, Intervenção, Comparação, Resultado e Contexto. A **Tabela 1** descreve este método.

A partir do PICOC foi definida a pergunta a qual o trabalho busca responder: Qual *Message Broker* é mais recomendado em um contexto de sistemas distribuídos a fim de minimizar o consumo de recursos e maximizar o *throughput*?

Uma vez definida a pergunta que o trabalho visa responder, a pesquisa foi realizada na plataforma Google Acadêmico buscando artigos que tivessem ligação ao tema da pesquisa que estivessem, exclusivamente, em português ou inglês. A *string* de busca utilizada foi: ((*Messaging* OR Mensageria) OR *throughput* OR ("*distributed system**" or "sistemas distribuídos")) AND (Kafka OR RabbitMQ OR ("*microservice**" OR "microserviço*")).

No total foram encontrados 428 resultados. Os artigos encontrados foram analisados e filtrados a fim de selecionar os mais relevantes para a pesquisa.

---

4 Protocolo de camada de aplicação padrão aberto para Message Oriented Middleware. As características definidoras do AMQP são Orientação mensagem, roteamento (incluindo o ponto-a-ponto e publicar e assinatura), confiabilidade e segurança.

5 "Processo de avaliação da empresa em relação à concorrência, por meio do qual incorpora os melhores desempenhos de outras firmas e/ou aperfeiçoa os seus próprios métodos", por Oxford Languages.

6 Métrica responsável por mensurar taxa de transferência, ou seja, quantidade de dados que são trafegados em determinado espaço de tempo

Tabela 1: **PICOC**

PICOC	Palavra-Chave
População	Mensageria, Sistemas Distribuídos, Apache Kafka, RabbitMQ, Pub/Sub
Intervenção	Apache Kafka, RabbitMQ
Comparação	Apache Kafka x RabbitMQ
Resultado	Reduzir consumo de recursos e aumentar fluxo de dados.
Contexto	Sistemas Distribuídos

2.5. Trabalhos Relacionados

[Dobbelaere 2017] em seu trabalho sobre Kafka e RabbitMQ nos apresenta uma visão crítica e comparativa entre ambos os sistemas, conceituando ambas as tecnologias de um ponto de vista qualitativo além de propor diversos cenários que estressam a solução para gerar uma análise quantitativa, observando principalmente características como latência e *throughput*.

[Rabiee 2018] apresenta em sua tese de mestrado o trabalho intitulado "*Analyzing Parameter Sets For RabbitMQ and Apache Kafka On A Cloud Platform*". Neste a abordagem é parecida com a de Dobbelaere, entretanto com uma atenção especial para os parâmetros de configuração de ambos os sistemas, além de trazer o foco para outras características como CPU e memória ao realizar a comparação quantitativa entre ambas as tecnologias.

[Wang 2015] em seu trabalho "*Kafka and Its Using in High-throughput and Reliable Message Distribution*", apresenta o Apache Kafka como uma solução resiliente e de alta performance para distribuição de mensagens. Em seu trabalho tem como principal foco defender como o Apache Kafka tem se tornado uma solução madura para substituir mensagerias tradicionais, apesar de também trazer à tona o fato de boa parte de sua configuração ser realizada de forma manual, o que aumenta a complexidade desta solução.

Em resumo diversos trabalhos vêm trazendo abordagens diferentes para a escolha de qual gerenciador de mensagens se adapta melhor para determinado cenário e aplicação(ões). Dito isto algumas características aparecem frequentemente nestas comparações. Estas são: *throughput* ou taxa de transferência, disco, CPU e memória ram. Neste trabalho, estas quatro características serão nosso foco.

3. Metodologia

A presente pesquisa foi pautada em uma abordagem quantitativa e foram analisadas as métricas: *throughput*, disco, CPU e memória.

3.1. Tecnologias e Implementações

O cenário proposto para simular a utilização das tecnologias de mensageria selecionadas propõe uma abordagem simples e foram implementadas duas *APIs*[7] para atuar como *consumer/subscriber* e uma terceira como *producers/publishers*, além da camada de mensageria que será composta por contêineres contendo instâncias do Apache Kafka e RabbitMQ. A linguagem utilizada para a solução foi o C# e o framework ASP.Net 5. A linguagem e framework utilizados não são o foco da presente pesquisa e foram escolhidos por conta da familiaridade com a tecnologia.

Além da implementação da solução, as ferramentas escolhidas para mensurar os resultados foram o Prometheus em conjunto com o Grafana, ambas de código aberto. Para a preparação do ambiente de monitoramento também foram criados containers com o Prometheus e o Grafana.

A **Figura 2** mostra como será a aplicação após levantamento da infra.

A proposta das *APIs* responsáveis por consumir as filas não é nada complexa. Em resumo, quando a aplicação é iniciada ela se inscreve no tópico ou fila para qual está responsável. Uma vez que se inscreveu ela irá consumir todas as mensagens enviadas para o mesmo. Já a *API* de envio de mensagens possui certa complexidade, uma vez que recebe, em sua requisição, as configurações para criar e enviar a(s) mensagem(s). Esta *API* recebe o nome de *ProducerAPI*.

A *ProducerAPI* possui os seguintes parâmetros em sua requisição:

- **MessagesCount**: Recebe um valor inteiro determinando a quantidade de mensagens que será enviada para a camada de mensageria;
- **MessageSize**: Recebe o tamanho de cada mensagem enviada em bytes;
- **UseParallelism**: Recebe um boolean que definirá se a API irá enviar as mensagens em paralelo ou uma a uma;
- **ParallelismLimit**: Caso a opção anterior seja verdadeira, esta define o grau de paralelismo utilizado no envio das mensagens;
- **QueueType**: Neste definimos para qual mensageria as mensagens serão enviadas: 0 para Kafka e 1 para RabbitMQ.

Para auxiliar no monitoramento foram utilizadas as ferramentas Grafana e Prometheus , e com isso conseguimos retirar diversos gráficos e métricas das aplicações.

---

7 *Application Programming Interface*

No caso do Kafka, por não possuir uma dashboard nativa, também foi utilizada a dashboard fornecida pela Confluent a fim de proporcionar um melhor gerenciamento.

Uma vez que toda a infra estava de pé, foi desenvolvida uma aplicação de console para popular as filas a fim de gerar nossos cenários de teste, estes que apresentaremos adiante. Esta aplicação tem uma simples missão, a partir das configurações passadas ela irá enviar mensagens constantes para a ProducerAPI com os dados para envio das mensagens para a fila.

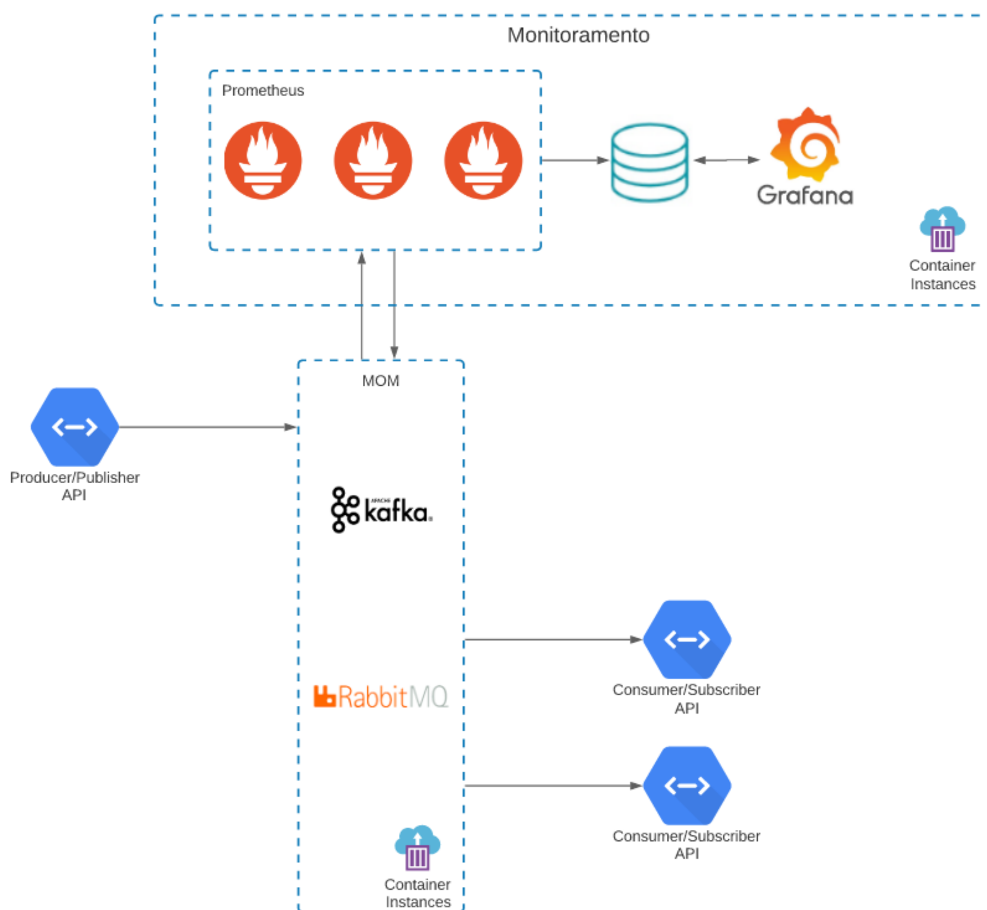

Figura 2: Modelagem da Aplicação utilizando RabbitMQ e Apache Kafka.
Fonte: Do autor

3.2. Cenários Propostos

Quando falamos sobre envio de mensagens em grande escala, existem dois fatores que podem definir se uma mensageria irá ou não atender a demanda do projeto: quantidade de mensagens e tamanho das mesmas. Por conta disso, na presente pesquisa iremos propor cenários em que estes atributos serão validados.

Cada cenário conta com um período de amostragem de 15 minutos, sendo o intervalo de envio de cada lote de mensagens de 5 segundos.

No que tange as configurações de ambiente, para esta pesquisa a aplicação foi publicada em ambiente isolado *on premise,* cujas configurações eram: Intel(R) Core(TM) i5-10210U CPU @ 1.60Hz 2.11 GHz, RAM: 8.00 GB (7.76 Usable).

Na **Tabela 2** demonstramos os cenários propostos assim como suas configurações:

Tabela 2:Cenários propostos para levantamento de dados referentes às mensagerias.

Cenário	Qtd de Mensagens/5seg	Tamanho mensagens (bytes)	Paralelismo
1	1000	100	N/A
2	1000	1000	N/A
3	10000	100	2
4	10000	1000	2
5	10000	1000	N/A

4. Análise e Discussão de Resultados

Como já apresentado anteriormente, este trabalho tem como foco as seguintes características: *throughput,* uso de disco, memória e CPU. A seguir apresentaremos os resultados em cada um dos nossos cenários para estas métricas.

4.1. Throughput

Em relação à taxa de transferência, o RabbitMQ e o Apache Kafka se mostraram bem semelhantes, e mantiveram seus resultados próximos em todos os cenários como é apresentado na **Figura 3**.

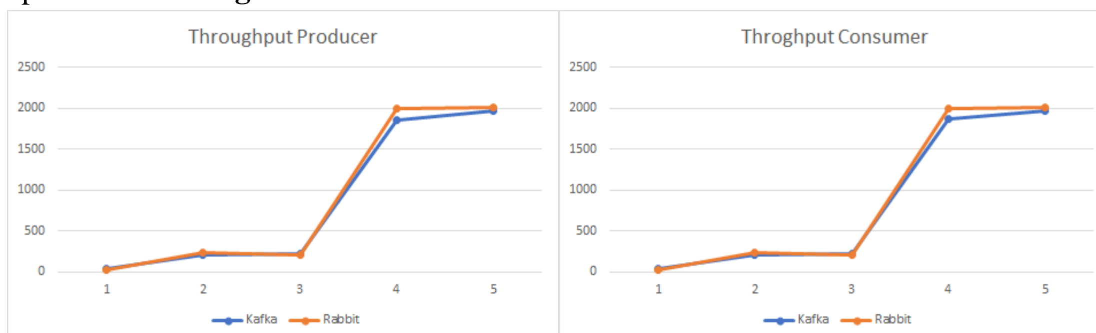

Figura 3: Comparação das métricas de *throughput* entre Kafka e Rabbit, escala em Bytes.

Apesar das alterações no tamanho das mensagens e quantidade, tanto o Kafka quanto o RabbitMQ se adaptaram ao cenário e atenderam a demanda sem dificuldades e de forma automática.

4.2. Disco

Em relação ao uso de Disco é onde percebemos uma das maiores diferenças entre estas mensagerias. Enquanto o RabbitMQ não possui a utilização de disco como uma de suas características, a arquitetura do Apache Kafka propõe justamente o contrário, uma vez que suas mensagens são gravadas em disco e lidas diretamente do mesmo utilizando uma abordagem que torna esta leitura relativamente rápida até mesmo comparada à utilização de memória ram. O resultado dessa diferença de abordagem sobre o Disco pode ser visto na **Figura 4.**

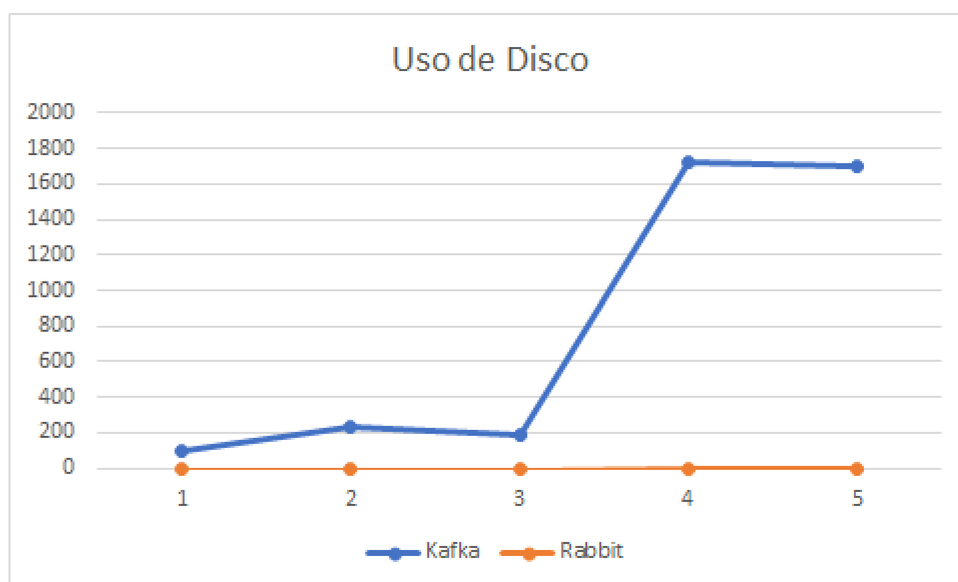

Figura 4: Comparação das métricas de utilização de disco entre Kafka e Rabbit, escala em Megabytes

O uso de disco do Kafka possui um crescimento constante conforme mais mensagens são enviadas, este fato se dá devido a suas configurações de retenção de mensagens assim como o seu fator de replicação de tópicos.

4.3. Memória

O Kafka também se destaca do ponto de vista de utilização de memória, enquanto a utilização do Rabbit não passava de alguns megabytes, o kafka chegou a atingir a ordem de grandeza de gigabyte, como podemos ver na **Figura 5.**

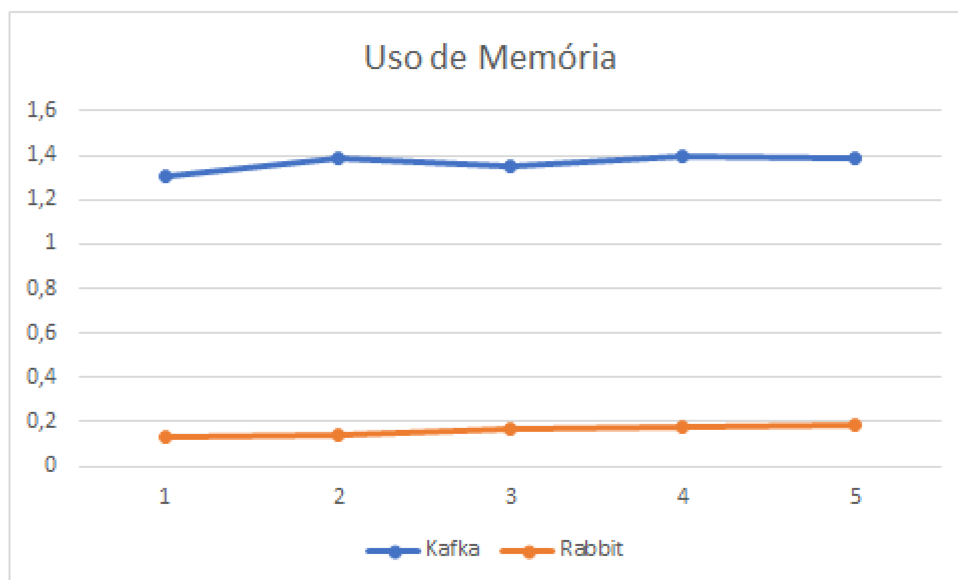

Figura 5: Comparação das métricas de utilização de memória entre Kafka e Rabbit, escala em Gigabytes

Apesar de o Kafka possuir uma infraestrutura mais robusta devido à grande quantidade de componentes e recursos de sua plataforma, o consumo de memória foi bem constante e com pouca variação entre os diferentes cenários propostos, semelhante ao RabbitMQ, com isso pode-se dizer que ambos os *Message Brokers* mantém seu uso de memória estável.

4.4. CPU

A respeito do monitoramento da utilização de CPU encontramos algumas curiosidades em ambas as abordagens, enquanto a utilização de CPU está diretamente ligada à quantidade de mensagens para o RabbitMQ, a utilização ou não do paralelismo no envio das mensagens é o que faz diferença para o Kafka, como vemos na **Figura 6**.

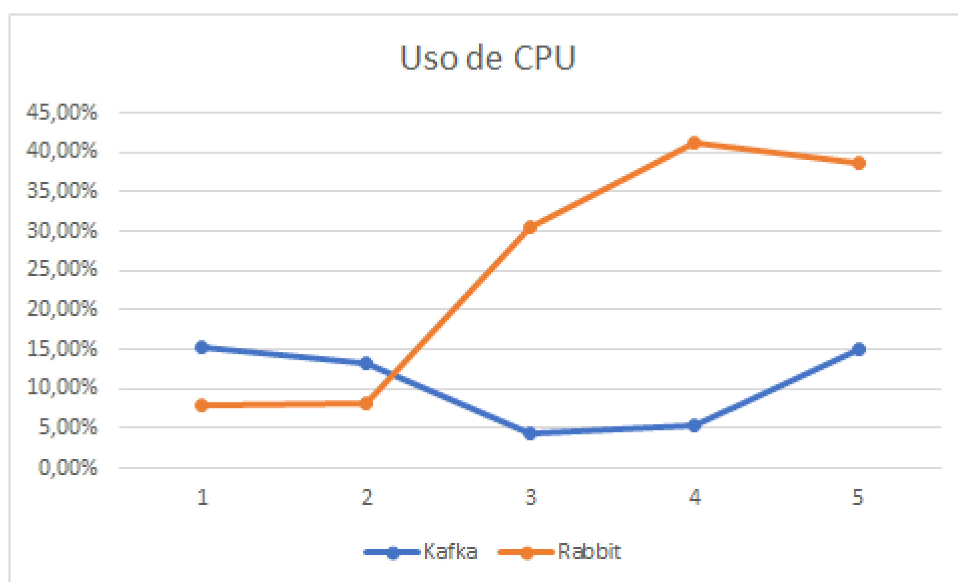

Figura 6: Comparação das métricas de utilização de CPU entre Kafka e Rabbit.

Neste ponto podemos perceber que em cenários onde as mensagens foram enviadas utilizando paralelismo, ou seja, mais de uma mensagem sendo enviadas simultaneamente para o *Broker,* o Apache Kafka manteve a utilização de CPU reduzida em relação ao RabbitMQ, mostrando um melhor gerenciamento dos recursos referentes a esta métrica.

5. Conclusões e Trabalhos Futuros

Este projeto buscou encontrar e trazer uma comparação da utilização de dois sistemas de mensageria a fim de gerar evidências para escolha de qual mensageria se adapta melhor em cada cenário, assim como os *trade-offs* de escolher uma ou outra.

Pode-se dizer que a escolha de uma melhor mensageria vai depender de qual cenário sua aplicação irá atuar, por exemplo, em um cenários onde o número de mensagens enviadas é baixo ambas as mensagerias cumprem o proposto e o RabbitMQ se destaca por fazer isso utilizando menos recursos da máquina que o Kafka, entretanto quando o número de mensagens cresce, apesar de o Rabbit manter um *throughput* elevado, a utilização de CPU cresce consideravelmente, enquanto a do Kafka se mantém bem abaixo.

Em resumo, ambas as mensagerias atendem e são competitivas em relação à taxa de transferência e velocidade de consumo/produção de mensagens, já do ponto de vista de Memória e Disco o RabbitMQ possui vantagens sobre o Kafka enquanto a respeito de utilização de CPU o Kafka se mantém à frente do Rabbit.

Referências

TANENBAUM, Andrew S.; VAN STEEN, Maarten. **Distributed systems: principles and paradigms**. Prentice-hall, 2007.

COULOURIS, George et al. **Sistemas Distribuídos: Conceitos e Projeto**. Bookman Editora, 2013.

TAN, Wei et al. Social**-network-sourced big data analytics**. IEEE Internet Computing, v. 17, n. 5, p. 62-69, 2013. Disponível em: <https://ieeexplore.ieee.org/abstract/document/6596496>

CURRY, Edward. Message**-oriented middleware.** Middleware for communications, p. 1-28, 2004.

DOBBELAERE, P. and K. S. Esmaili. **Kafka versus RabbitMQ**. *ArXiv* abs/1709.00333, 2017.

RABIEE, Amir. Analyzing **Parameter Sets For Apache Kafka and RabbitMQ On A Cloud Platform**. 2018.

WANG, Zhenghe et al. **Kafka and its using in high-throughput and reliable message distribution**. In: 2015 8th International Conference on Intelligent Networks and Intelligent Systems (ICINIS). IEEE, 2015. p. 117-120.

LU, Ruirui et al. **Stream bench: Towards benchmarking modern distributed stream computing frameworks**. In: 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing. IEEE, 2014. p. 69-78.