# Avaliação Experimental dos *Brokers Apache Flume* e *Kafka* no Contexto de *Big Data*

Matheus Orlandi de Castro[1], Cristiano Bertolini[1], Edison Pignaton de Freitas[2], Evandro Preuss[1], Ricardo Tombesi Macedo[1]

[1]Universidade Federal de Santa Maria - UFSM
Campus Frederico Westphalen - Frederico Westphalen - RS

[2]Universidade Federal do Rio Grande do Sul - UFRGS
Caixa Postal 15064 Porto Alegre - RS

mo.castro@hotmail.com, cristiano.bertolini@ufsm.br,
epfreitas@inf.ufrgs.br, evandro.preuss@gmail.com, rmacedo@inf.ufsm.br

***Abstract.*** *Recently, many enterprises are using the Big Data technologies as a new market niche. However, there is no consensus about that is the most appropriated broker to employ in the Big Data context. In the literature, the main approaches are evaluating proprietary solutions, focusing only on processing, or they are targeting other contexts. This paper presents a experimental performance evaluation of the open sources Apache Flume and Kafka brokers. The evaluation considers the data extraction and data transfer processes on the big data context. The results reveal Apache Flume overcoming Kafka on 19% working with small files and on 10% handling big files. However, Kafka overcomes Apache Flume by handling real time collected streaming data.*

***Resumo.*** *Atualmente, muitas empresas vêm explorando tecnologias de Big Data como um novo nicho de mercado. No entanto, não existe um consenso sobre qual broker é mais indicado parar utilizar no contexto de big data. Na literatura, os principais trabalhos avaliam soluções proprietárias, se preocupam apenas com questões de processamento ou se concentram em contextos diferentes. Este trabalho apresenta uma avaliação de desempenho experimental do processo de extração e transferência de dados dos brokers de código aberto Apache Flume e Kafka no contexto de big data. Os resultados obtidos revelam que o broker Apache Flume supera seu concorrente em 19% ao trabalhar com arquivos pequenos e em 10% ao manipular arquivos grandes. Entretanto, o broker Kafka supera em 21,6% seu concorrente ao manipular dados de streaming coletados em tempo real.*

## 1. Introdução

*Big Data* surgiu como um nicho de mercado para muitas empresas que prestam serviços através da Internet [Pääkkönen and Pakkala 2015]. *Big Data* consiste em uma área de pesquisa para abordar o grande crescimento do volume de dados, desde a sua geração, passando pela aquisição, análise e armazenamento. De acordo com Gantz e Reinsel, o volume de dados criados e copiados aumentou nove vezes nos

cinco anos anteriores, representando um total de 1,8 ZB até o ano de 2011 (Zettabyte) [Gantz and Reinsel 2011]. Estudos recentes apontam que este volume dobrará a cada dois anos até 2020 [Gantz and Reinsel 2012]. Por meio do *big data* torna-se possível analisar grandes quantidades de dados, extraindo informações que podem trazer lucros para uma determinada corporação ou uma melhora significativa na gestão de recursos e o desenvolvimento de novas soluções explorando este conceito. Algumas das empresas mais conhecidas que exploram esta tendência consistem no *Twitter*, *LinkedIn* e *Facebook*.

No entanto, um desafio atual consiste na escolha de um *broker* de dados mais apropriado para implantação soluções baseadas em *big data*. Uma arquitetura *big data* geralmente compreende três fases principais, a fase de extração, transferência e processamento de dados [Pääkkönen and Pakkala 2015]. Um *broker* consiste em um componente de software responsável por coletar, transportar e negociar dados de uma base de dados origem para um mais clientes [Ramachandran et al. 2005]. Existem muitas soluções de *broker* disponíveis, no entanto, não existe um consenso sobre qual solução consiste na mais indicada para utilizar no contexto de *big data*.

Na literatura, os principais trabalhos avaliam o desempenho de soluções proprietárias, focam na fase de processamento de dados dos *brokers* ou se concentram em contextos diferentes do *big data*. Henjes *et. al.* avaliaram o desempenho da vazão do *Message Broker WebSphereMQ* [Henjes et al. 2006]. Todavia, apenas uma solução foi avaliada, a qual possui o código fechado. Córdova avaliou duas das plataformas de código aberto na área de processamento de *big data*, *Storm* e *Spark* [Córdova 2014]. Todavia, o estudo foi conduzido para analisar apenas a fase de processamento de dados dos *broker*, ignorando as etapas de extração e transferência de dados. Ionescu avaliou os *brokers* de código aberto *RabbitMQ* e *ActiveMQ* em situações de aumento do tamanho da mensagem [Ionescu 2015]. Todavia, o escopo da pesquisa foi restrito ao contexto de tecnologias de *middleware* orientados a mensagens, não compreendendo o contexto de *big data*.

Este trabalho apresenta uma avaliação de desempenho experimental dos *brokers Apache Flume* e *Kafka* no contexto de *big data*. Para conduzir a avaliação foi desenvolvido um protótipo de um *software* capaz de simular um fluxo grande de dados e também realizar trocas de mensagens entre *brokers*. Diferentemente dos trabalhos da literatura, este trabalho avalia a fase de extração e transferência de dados. Além disto, este trabalho considera apenas soluções de código aberto que podem ser livremente adaptadas para fins específicos e são amplamente utilizados em grandes provedores de serviços, como o *LinkedIn*, *Amazon*, *Netflix*.

Os resultados obtidos mostram que a escolha de qual destes softwares depende do tipo de arquivo a ser manipulado. O *Apache Flume* superou em 19% o *Apache Kafka* ao trabalhar com arquivos pequenos (tamanho médio de 11,8 KB). Além disso, o *Apache Flume* também foi 10% melhor que seu concorrente ao manipular arquivos grandes (com tamanho médio de 1009 MB). Entretanto, quando se trata de *streaming* de dados em tempo real, a situação se inverte, sendo o *Apache Kafka* 21,6% mais rápido que seu rival.

Este artigo está organizado como segue. A Seção 2 apresenta os trabalhos relacionados. A Seção 3 descreve a avaliação experimental de desempenho dos *brokers*. A Seção 4 detalha os experimentos. A Seção 5 conclui o artigo.

## 2. Trabalhos Relacionados

Uma arquitetura *big data* geralmente compreende três fases principais, a fase de extração, transferência e processamento de dados [Pääkkönen and Pakkala 2015]. A fase de extração emprega técnicas de coleta de dados em bases de dados. A fase de transferência é responsável por enviar os dados coletados para tratamento e a fase de processamento realiza as análises dos dados com um determinado fim. Os principais trabalhos da literatura sobre avaliação de desempenho de *brokers* se concentram em soluções proprietárias, na fase de processamento de dados ou se concentram em contextos diferentes do *big data*.

Henjes *et. al.* investigaram o desempenho de vazão da solução proprietária *Message Broker WebSphereMQ* [Henjes et al. 2006]. O estudo considerou diferentes números de *subscribers*, *publishers*, mensagens de diferentes tamanhos, diferentes tipos de filtros e estes filtros de diferentes complexidades. Os resultados obtidos revelaram que o tamanho da mensagem tem um impacto significativo sobre a vazão de dados no servidor. Entretanto, a avaliação foi conduzida com apenas um *broker*, o *WebsphereMQ* da IBM, o qual consiste em um software de código fechado.

Córdova avaliou duas das plataformas de código aberto na área de processamento de *big data*, *Storm* e *Spark* [Córdova 2014]. O objetivo deste estudo consistiu em fornecer uma visão geral de muitas características de alto nível de ambos os sistemas, para poder compreender e avaliar as suas velocidades de processamento. Os resultados obtidos revelaram que o *Storm* foi em torno de 40% mais rápido em relação ao *Spark* ao manipular registros pequenos. No entanto, com o aumento do tamanho dos registros o *Spark* conseguiu melhorar seu desempenho, chegando a superar o *Storm*. Todavia, o estudo foi conduzido para analisar a fase de processamento de dados dos *broker*, ignorando as etapas de extração e transferência de dados.

Ionescu avaliou os *brokers* de código aberto *RabbitMQ* e *ActiveMQ* em situações de aumento do tamanho da mensagem no escopo de tecnologias de *middleware* orientados a mensagens [Ionescu 2015]. As métricas analisadas consistiram na velocidade de processamento para envio/recebimento de mensagens e na carga de memória. Os resultados obtidos revelam que o *broker ActiveMQ* possui um desempenho superior quando se trata de receber mensagens, em contrapartida o *RabbitMQ* é mais rápido quando a tarefa a ser executada é a de enviar mensagens para uma aplicação cliente. Todavia, o escopo da pesquisa foi restrito ao contexto de tecnologias de *middleware* orientados a mensagens, não compreendendo o contexto de *big data*.

Este trabalho apresenta uma avaliação experimental de desempenho dos *brokers Apache Flume* e *Kafka* no contexto de *big data*. Assim como em [Henjes et al. 2006], a avaliação proposta analisa a métrica de vazão de dados. No entanto, este trabalho considera apenas soluções de código aberto que podem ser livremente adaptadas para fins específicos. A decisão de avaliar soluções livres corrobora com os objetivos apresentados em [Córdova 2014] e [Ionescu 2015]. Todavia, diferentemente de [Córdova 2014], a avaliação apresentada analisa a fase de extração e transferência de dados e ao contrário de [Ionescu 2015], o trabalho proposto se concentra no contexto de *big data*.

## 3. Avaliação Experimental de Desempenho dos *Brokers*

Esta seção descreve o projeto de avaliação experimental de desempenho dos *brokers*. A Subseção 3.1 apresenta o projeto do protótipo. A Subseção 3.2 detalha as carga de

trabalho de trabalho utilizadas.

### 3.1. Projeto do Protótipo

Esta subseção descreve o projeto de um protótipo para interagir com os *broker Apache Flume* e *Kafka*. A Figura 1 mostra como o protótipo interage com o *broker Apache Flume*. Os dados gerados são coletados por um ou mais agentes do *Flume*, sendo que cada agente pode possuir uma ou mais *sources, channels e sinks*. O *source* é responsável por coletar os dados gerados pelos *scripts* e pela aplicação e armazenar no canal configurado. O canal utilizado atua como a memória principal, por ser mais rápida que as outras possibilidades oferecidas pelo *broker*. Quando terminado o transporte ao longo do flume estes dados foram armazenados no sistema de arquivos do *Hadoop* (uma estrutura voltada para ambientes distribuídos que permite realizar análises e armazenamento de grandes conjuntos de dados estruturados e não estruturados), através da configuração do *Sink*. O *Hadoop* é necessário para que se consiga implementar o ambiente esperado para a realização da análise de desempenho entre os *brokers*.

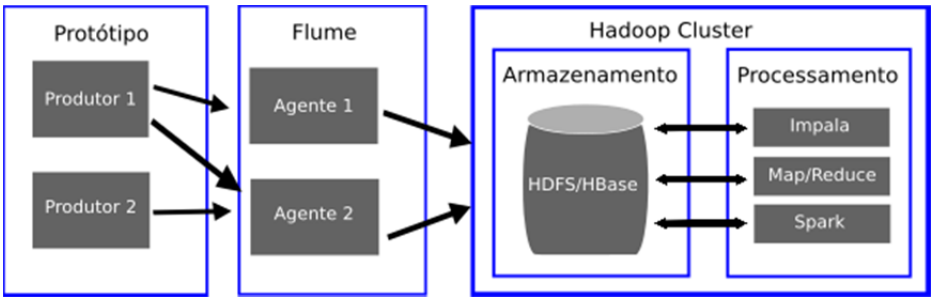


**Figura 1. Esquema de interação entre o protótipo, Apache Flume e Hadoop**

A Figura 2 apresenta como o protótipo interage com o *Kafka*. O *broker Apache Kafka* não é capaz de realizar a coleta dos dados gerados pelos *scripts* e pela aplicação. Para contornar esta dificuldade, o protótipo produz os dados e alimenta o *broker* enviando mensagens para uma ou mais partições definidas previamente.

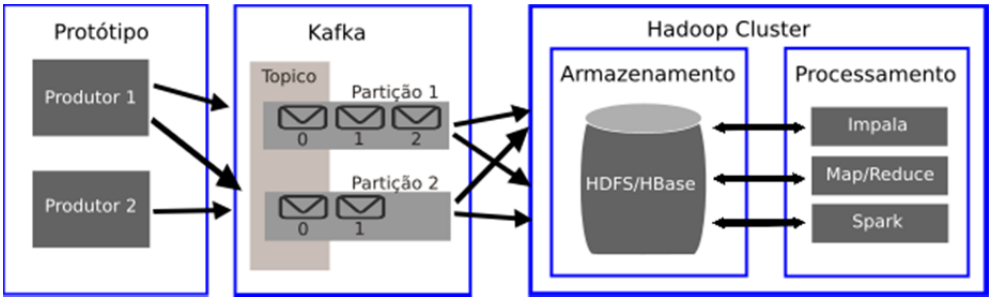


**Figura 2. Esquema de interação entre o protótipo, Apache Kafka e Hadoop**

### 3.2. Carga de Trabalho

No contexto de *Big Data*, os dados podem ser adquiridos pro meio de arquivos de *log*, base de dados ou *streaming*, sendo fundamental o uso de *brokers* para a coleta destes dados, para posterior processamento, análise e armazenamento. Para alcançar esta características, a avaliação foi conduzida com dois tipos de carga, uma carga de trabalho artificial e uma carga de trabalho real. A carga de trabalho artificial é produzida através de *shell scripts* desenvolvidos para criar os seguintes conjuntos de dados: $(i)$ muitos

arquivos pequenos; $(ii)$ poucos arquivos extremamente grandes e $(iii)$ uma mescla de muitos arquivos pequenos e grandes. A geração dos arquivos é feita a partir do arquivo *urandom*, o qual é capaz de fornecer caracteres de forma aleatória e ilimitada, sendo nativo nas distribuição Linux.

A carga de trabalho real consistiu em fluxos de dados *streaming*. Para utilizar esta carga de trabalho uma aplicação foi desenvolvida para usar dados reais do *Twitter*. Esta aplicação recupera *tweets* públicos, filtra os *tweets* de acordo com parâmetros informados, agrega essas informações e as formata de acordo com um padrão aceito pelos *brokers*. A Figura 3 apresenta o esquema de funcionamento da aplicação desenvolvida.

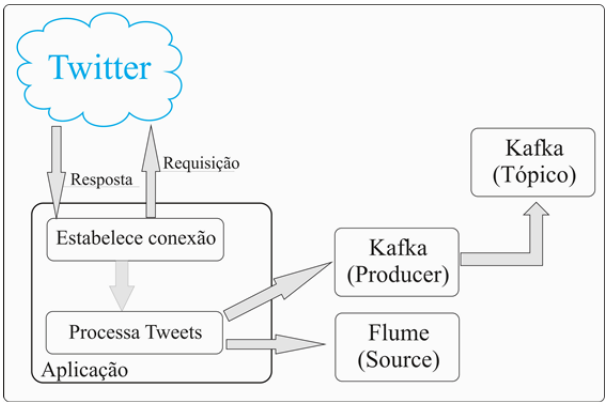


**Figura 3. Esquema de funcionamento da aplicação**

A aplicação utiliza a API (Interface de Programação de Aplicação) *Twitter streaming API*, a qual possibilita recuperar qualquer *tweet* público. Esta mesma API foi utilizada por [Córdova 2014] para realizar uma análise de desempenho entre dois serviços pertencentes a fase de processamento de dados. Além disso, a aplicação gera duas *Consumer Key*, sendo uma delas secreta e dois *Tokens* de acesso, sendo um deles secreto também. A *Consumer Key* é utilizada para a autenticação do usuário cadastrado no *Twitter*, para conseguir acesso a API *Twitter Streaming*. Os *Tokens* possui a função de autenticar as requisições enviadas ao *Twiter* para a coleta das mensagens [1].

Para o *Kafka* a aplicação foi integrada diretamente com o *driver* produtor do *broker* e do produtor para o tópico configurado. O tópico pode ser definido como uma categoria de mensagens, que possuirá apenas um configurado para os sete *brokers* do *cluster*. Já para o *Flume* basta a correta configuração dos três parâmetros principais, o *source*, *channel* e o *sink* para que seja possível seu funcionamento. Esta aplicação foi responsável por gerar o *streaming* de dados, em tempo real, para a realização dos experimentos.

## 4. Experimentos

Esta seção apresenta os experimentos realizados e os resultados obtidos. Os experimentos foram conduzidos como objetivo de mensurar o desempenho dos dois *brokers*, o *Apache Kafka* e o *Apache Flume*. A Subseção 4.1 descreve o ambiente utilizado, os cenários de avaliação e as métricas utilizadas. A Subseção 4.2 detalha os resultados obtidos.

[1]https://dev.twitter.com/docs/

### 4.1. Ambiente, Cenários e Métricas

Esta subseção descreve o ambiente, os cenários e métricas da avaliação de desempenho. Os experimentos foram conduzidos em um ambiente de *cluster* composto por sete computadores *Dell Optiplex GX270*. Cada computador foi equipado com um processador *Pentium 4 HT* 2.80 GHz, 3 GB de memória RAM e 160 GB de armazenamento. O sistema operacional utilizado nestes computadores consistiu na distribuição Linux Rocks 6.1 *Emerald Boa*, projetada para utilização de em *clusters* de alto desempenho. A conexão entre os computadores do *cluster* foi feita por meio de interfaces de rede de 1000 Mbps. A utilização de um *cluster* foi necessária para minimizar qualquer limitação física de processamento. Os experimentos foram conduzidos com as versões 0.8.2.2 do *Apache Kafka*, 3.4.8 do *Apache Zookeeper* e 1.6.0 do *Apache Flume*.

Quatro cenários foram construídos para avaliar os *brokers* em diferentes situações. O primeiro cenário (*Experimento 1*) simula a transferência de arquivos de *logs* de servidores, sendo configurado com uma grande quantidade de arquivos criados artificialmente com tamanho médio de 11,8 KB. O segundo cenário (*Experimento 2*) considera uma situação onde os *brokers* precisariam manipular grandes arquivos, onde foram utilizados arquivos criados artificialmente com tamanho médio de 1009 MB. O terceiro cenário (*Experimento 3*) simula a transferência de arquivos comuns gerados por usuários, com tamanho similar a arquivos de fotos, músicas, mensagens de *email* e documentos de texto. Para alcançar este objetivo, o *Experimento 3* foi projetado para manipular uma mescla de arquivos pequenos e grandes artificialmente criados com tamanho médio 4,6 MB, variando de 10 KB até 10 MB. O quarto cenário (*Experimento 4*) considera uma situação de carga de trabalho real coletada em tempo real do *Twitter*.

**Tabela 1. Configuração dos Experimentos**

| Cenário | Nº de Arquivos | Nº de Brokers | Tam. Total |
|---|---|---|---|
| *Experimento 1* | 2.650 | 7 | 31,32 GB |
| *Experimento 2* | 28 | 7 | 28,26 GB |
| *Experimento 3* | 8323 | 7 | 38,31 GB |
| *Experimento 4* | N/A | 3 | N/A |

Como mostra a tabela, o experimento 4 não possui um número fixo de arquivos, isso se dá pelo fato que a aplicação coleta fluxo de dados, por isso também, o tamanho total transportado varia para cada um dos *brokers* após as quatro horas de execução. Outro ponto à ser observado está ligado aos três primeiros experimentos, estes utilizando os arquivos artificialmente gerados pelo script, não possuem um tempo de execução fixo, pois variam para cada um dos *brokers*.

A avaliação de desempenho foi conduzida observando duas métricas, a vazão de dados e a taxa de processamento. A análise da vazão de dados possibilita avaliar a quantidade de *bytes* por segundo que um *broker* consegue transportar, quanto maior for a vazão, melhor será o desempenho *broker*. A taxa de processamento apresenta o custo de execução do *broker* para o processador da máquina em que ele está sendo executado, quanto menor o valor desta métrica, melhor o desempenho do *broker*.

### 4.2. Resultados

Esta subseção detalha os resultados dos experimentos em termos das métricas de vazão e consumo de processamento. Os resultados de vazão mostram o *broker Apache Flume*

superando o *broker Apache Kafka* ao tratar apenas arquivos pequenos e apenas arquivos grandes. A Figura 4, mostra a vazão de dados utilizando arquivos sintéticos.

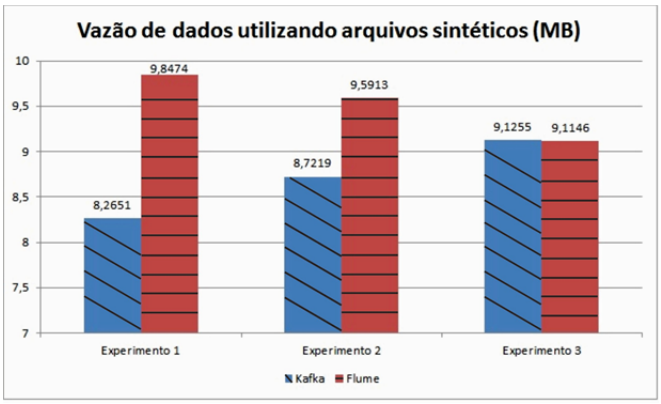


**Figura 4. Vazão de Dados Utilizando Arquivos Sintéticos**

Os resultados obtidos mostram uma vantagem do *Apache Flume* em dois dos cenários e um resultado próximo ao seu concorrente em apenas uma das execuções. No cenário *experimento 1*, o *Apache Flume* processou 2650578 arquivos apresentando a vantagem de de 19% em relação ao *Apache Kafka*. No cenário *experimento 2*, o *Apache Flume* apresentou uma vazão de dados de 870 KB/s, superando em 10% o seu concorrente. No cenário *experimento 3* ambos os *brokers* tiveram desempenho praticamente igual, sendo o *Apache Kafka* apenas 0,11% ou 10,9 KB/s.

A Figura 5 apresenta os resultados da métrica de vazão do cenário *experimento 4*, a qual utiliza a aplicação desenvolvida. Esta aplicação coletou postagens no *Twitter* em tempo real e entregou aos dois *brokers* sem nenhum filtro de consulta, ou seja, foi coletado qualquer *tweet* público disponível. O experimento teve duração de quatro horas para cada um dos *brokers*. Os resultados coletados apresentam uma vazão de 21,6% ou 126,72 KB/s superior do *Apache Kafka* em relação ao *Apache Flume*.

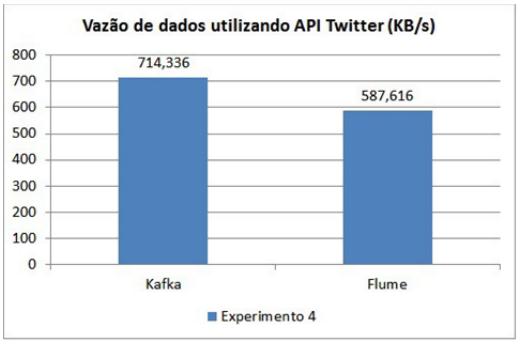


**Figura 5. Vazão de dados utilizando a aplicação desenvolvida**

Os resultados da métrica de consumo de processamento mostram que os dois *brokers* obtiveram resultados semelhantes. Nos experimentos utilizando a aplicação, a porcentagem de uso do processador, para cada um dos nós, se manteve próximo de 5%, já para os testes utilizando arquivos sintéticos, o *Apache Flume* manteve uma média de 10% durante a execução, já o *Kafka*, demonstrou uma economia de 5% em relação ao seu concorrente. Durante a execução dos experimentos, não foi detectada nenhuma falha no decorrer do transporte dos dados.

Por fim, observando os resultados obtidos, é possível relacioná-los a algumas características de cada *broker*, por exemplo, o *Kafka* tem como foco o processamento de *streaming* de dados em tempo real. A API *producer* contida nele, permite que aplicações desenvolvidas enviem fluxo de dados para os seus tópicos de maneira eficiente. Isso explica o seu melhor desempenho no quarto experimento, utilizando a aplicação desenvolvida. Já nos experimentos anteriores, o Flume obteve um desempenho superior. O principal motivo para isso, se dá pelo fato de que sua arquitetura mais simples ter como foco o processamento de arquivos de *logs* de servidores, através da coleta e agregação destes dados. Por isso este *broker* não depende de nenhum outro *software* para realizar esta coleta, bastando apenas a correta configuração do *source*.

## 5. Conclusões

Este artigo apresentou uma avaliação de desempenho dos *brokers Apache Flume* e *Kafka* no contexto de *big data*. Foram definidos quatro cenários de avaliação utilizando dados gerados artificialmente e dados reais coletados em tempo real. Duas métricas foram consideradas, a vazão de dados e a taxa de processamento. Os resultados obtidos mostram o *Apache Flume* superando em 19% o *Apache Kafka* ao trabalhar com arquivos pequenos (tamanho médio de 11,8 KB). Além disso, o *Apache Flume* também foi 10% melhor que seu concorrente ao manipular arquivos grandes (com tamanho médio de 1009 MB). Entretanto, quando se trata de *streaming* de dados em tempo real, a situação se inverte, sendo o *Apache Kafka* 21,6% mais rápido que seu rival. Estes resultados vão de encontro com o foco de aplicação de cada *broker*, onde o *Apache Flume* tem sua arquitetura desenvolvida visando um melhor desempenho na coleta e agregação de *logs* de servidores enquanto o *Apache Kafka* possui seu desenvolvimento voltado para o transporte de *streaming* de dados em tempo real. Como trabalho futuro, pretende-se realizar uma análise de desempenho em serviços responsáveis por fazer a análise dos dados coletados.

## Referências

Córdova, P. (2014). Analysis of real time stream processing systems considering latency.

Gantz, J. and Reinsel, D. (2011). Extracting value from chaos. *IDC iview*, 1142:1–12.

Gantz, J. and Reinsel, D. (2012). The digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the far east. *IDC iView*, 2007:1–16.

Henjes, R., Menth, M., and Zepfel, C. (2006). Throughput performance of java messaging services using WebsphereMQ. In *IEEE International Conference on Distributed Computing Systems Workshops*, pages 26–26.

Ionescu, V. M. (2015). The Analysis of the Performance of RabbitMQ and ActiveMQ. In *IEEE International Conference-Networking in Education and Research*, pages 132–137.

Pääkkönen, P. and Pakkala, D. (2015). Reference Architecture and Classification of Technologies, Products and Services for Big Data Systems. *Big Data Research*, 2(4):166 – 186.

Ramachandran, U., Modahl, M., Bagrak, I., Wolenetz, M., Lillethun, D., Liu, B., Kim, J., Hutto, P., and Jain, R. (2005). MediaBroker: A pervasive computing infrastructure for adaptive transformation and sharing of stream data. *Pervasive and Mobile Computing*, 1(2):257 – 276.