Maiki Manfro Girardi

Avaliação de intermediadores de mensagens que suportam o protocolo *AMQP*

Caxias do Sul

2015

Maiki Manfro Girardi

Avaliação de intermediadores de mensagens que suportam o protocolo *AMQP*

Projeto de Diplomação submetido ao curso de Bacharelado em Ciência da Computação do Centro de Computação e Tecnologia da Informação da Universidade de Caxias do Sul, como requisito obrigatório para graduação.

UNIVERSIDADE DE CAXIAS DO SUL
CENTRO DE COMPUTAÇÃO E TECNOLOGIA DA INFORMAÇÃO
BACHARELADO EM CIÊNCIAS DA COMPUTAÇÃO

Orientador: André Luis Martinotto
Coorientador: Alexandre Erasmo Krohn Nascimento

Caxias do Sul
2015

Maiki Manfro Girardi
Avaliação de intermediadores de mensagens que
suportam o protocolo *AMQP*/ Maiki Manfro Girardi. – Caxias do Sul, 2015-
70 p. : il. (algumas color.), 30 cm.

Orientador: André Luis Martinotto

Projeto de Diplomação – UNIVERSIDADE DE CAXIAS DO SUL
CENTRO DE COMPUTAÇÃO E TECNOLOGIA DA INFORMAÇÃO
BACHARELADO EM CIÊNCIAS DA COMPUTAÇÃO, 2015.

1. AMQP. 2. *Analytic Hierarchy Process*. I. André Luis Martinotto. II. Universidade de Caxias do Sul. III. Faculdade de Ciências da Computação. IV. Avaliação de intermediadores de mensagem que implementam o protocolo *AMQP*

CDU 02:141:005.7

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

AMQP	*Advanced Message Queuing Protocol*
API	*Application Program Interface*
AHP	*Analytic Hierarchy Process*
CORBA	*Common Object Request Broker Architecture*
FIFO	*First In First Out*
HTTP	*Hypertext Transfer Protocol*
IIOP	*Internet Inter-Orb Protocol*
JMS	*Java Message Service*
JRMP	*Java Remote Method Protocol*
MOM	*Message Oriented Middleware*
MQTT	*Message Queue Telemetry Transport*
PaaS	*Platform as a Service*
OASIS	*Organization for the Advancement of Structured Information Standards*
QoS	*Quality of Service*
SSD	*Solid State Disk*
RAM	*Random Access Memory*
RPC	*Remote Procedure Call*
SASL	*Simple Authentication and Security Layer*
SaaS	*Software as a Service*
SOAP	*Simple Object Access Protocol*
SSD	*Solid State Disk*
STOMP	*Streaming Text Oriented Messaging Protocol*
TCC	*Trabalho de Conclusão de Curso*

Sumário

1 Introdução

Ao iniciarmos o desenvolvimento de uma aplicação que será hospedada em uma nuvem computacional baseada no modelo de licenciamento SaaS (*Software as a Service*), sempre temos como objetivo a hospedagem desta aplicação em nuvens computacionais comerciais conhecidas, como, por exemplo, a Amazon AWS (AMAZON, 2015), Microsoft Azure (MICROSOFT, 2015) e Google Cloud Services (GOOGLE, 2015).

Nuvens computacionais como as citadas, possuem recursos que permitem o escalonamento de máquinas virtuais, fazendo com que novos recursos sejam alocados de forma a atender picos de processamento (SOUSA; MOREIRA; MACHADO, 2009). Desta forma, é fundamental que a aplicação desenvolvida tenha condições de aproveitar ao máximo os novos recursos disponibilizados. Ou seja, que seja capaz de atender um número maior de requisições, com a menor quantidade de recursos possíveis e que utilize eficientemente os novos recursos que foram disponibilizados pela nuvem.

Para que a aplicação consiga utilizar eficientemente os novos recursos uma das alternativas é fazer o uso de um *middleware* que facilite a troca de informações entre os componentes da aplicação. A partir das informações coletadas, a aplicação poderá realizar uma melhor distribuição de carga entre os recursos disponíveis. De forma a facilitar essa comunicação foram desenvolvidos os *Middlewares* Orientados a Mensagem (MOM), que são aplicações responsáveis pela troca de mensagens entre sistemas distribuídos, e que reduzem a complexidade de desenvolvimento de aplicações que necessitam deste recurso [3].

Objetivando padronizar o desenvolvimento deste tipo de ferramentas, foi criada a Norma ISO/IEC 19464, que também é conhecida como *Advanced Message Queuing Protocol* (AMQP). O AMQP é um protocolo que tem como objetivo a padronização no desenvolvimento de *softwares* MOM, tratando de questões relacionadas com: o formato e a codificação das mensagens; camada de transporte; atomicidade de transações e as camadas de segurança. Existem diferentes ferramentas MOM que foram desenvolvidos baseadas na Norma AMQP, sendo que entre essas ferramentas destacam-se as de código aberto RabbitMQ (PIVOTAL, 2015), Apache ActiveMQ (ACTIVEMQ, 2015) e Apache Qpid (APACHE QPID, 2015).

Este trabalho tem por objetivo efetuar um estudo sobre os *middlewares* de código aberto que implementam a norma AMPQ. Para tal, foram definidas métricas e cenários para avaliar de forma consistente as ferramentas aqui estudadas. A partir desse estudo comparativo pretende-se auxiliar os desenvolvedores na escolha de ferramentas MOM que poderão ser utilizadas no desenvolvimento de aplicações para nuvens do tipo SaaS.

1.1 Problema de pesquisa

Nuvens computacionais são modelos de negócio que, entre outros benefícios, nos permitem contratar um serviço, onde você irá pagar somente pelos recursos utilizados. Devido a esta característica, muitas empresas de desenvolvimento de *software* estão migrando suas aplicações para a nuvem.

Para uma utilização mais eficiente dos recursos disponíveis em uma nuvem computacional uma das alternativas é fazer o uso de um *middleware* MOM, uma vez que este tipo de *software* facilita a troca de informações entre os módulos da aplicação, sendo que essas informações podem ser utilizadas para uma melhor distribuição da carga computacional entre os recursos disponíveis.

Este trabalho realizou uma avaliação em ferramentas MOM *open source*, que implementem a especificação *Advanced Message Queuing Protocol* (AMQP). A partir desta avaliação, foram disponibilizadas informações para auxiliar os desenvolvedores na escolha de uma ferramenta MOM.

1.2 Questão de pesquisa

Que *software open source* para gerenciamento de mensagens e que implemente a especificação *Advanced Message Queuing Protocol* (AMQP) oferece uma melhor performance e confiabilidade em ambientes de nuvens computacionais?

1.3 Objetivos

O principal objetivo deste trabalho consistiu em um estudo comparativo entre as ferramentas MOM que implementem a especificação *Advanced Message Queuing Protocol* (AMQP). Para que o objetivo deste trabalho seja atingido os seguintes objetivos específicos foram realizados:

1. Definição das ferramentas a serem avaliadas;
2. Definição do método de avaliação;
3. Definição dos critérios a serem avaliados;
4. Realização do estudo e análise dos resultados;

1.4 Estrutura do Trabalho

No Capítulo 2 é apresentado uma contextualização sobre o protocolo AMQP, onde são apresentadas suas principais funcionalidades. Neste capítulo são apresentadas também

algumas ferramentas que implementam este protocolo. Para maiores informações sobre o protocolo AMQP, sugere-se a leitura de (TRIELOFF et al., 2006).

No Capítulo 3 é abordado o Método Analítico Hierárquico que foi o método utilizado para a avaliação das ferramentas escolhidas. Maiores informações sobre o método podem ser encontradas em (SAATY, 1980; SAATY, 1999).

No Capítulo 4, são definidos os critérios que foram utilizados para a avaliação das ferramentas, sua hierarquia e definição das matrizes de comparação paritária. Além disso, serão apresentados seus cálculos de consistência.

No Capítulo 5, é descrita a implementação da aplicação de referência, as características do ambiente de teste e os resultado obtidoss.

No Capítulo 6, são apresentadas as conclusões do trabalho e algumas sugestões de trabalhos futuros.

2 PROTOCOLO AMQP

O Protocolo Avançado de Filas de Mensagem (AMQP - *Advanced Message Queuing Protocol*) é um protocolo ponto a ponto (*wire protocol*[1]) de padrão aberto para sistemas corporativos de troca de mensagens, que se caracteriza por ser multi-canal, assíncrono, seguro, portável e eficiente (TRIELOFF et al., 2006). Esse protocolo foi criado para se tornar um padrão e para garantir uma maior interoperabilidade entre todos os *Middlewares* Orientados a Mensagem (MOM), ou seja, permitir a troca de mensagens entre intermediadores de mensagem de qualquer vendedor que implemente este padrão.

O protocolo AMQP foi desenvolvido por John O'Hara para solucionar o problema do banco JPMorgan. Esse banco precisava de um serviço de mensagens que fosse confiável, interoperável e veloz, sendo capaz de transmitir mais de 500.000 mensagens por segundo (O'HARA, 2007; KRAMER, 2009). Devido a deficiência das ferramentas e padrões existentes John O'Hara, juntamente com a IMatrix, [2] criou este protocolo. Posteriormente, foi criado o AMQP *Working Group*, sendo que anos mais tarde este protocolo se tornou um padrão incorporado pela OASIS[3].

O AMQP não define somente o protocolo de rede mas também a semântica dos serviços da aplicação servidora. De fato, o AMQP define um protocolo binário bem como o comportamento para a transmissão de mensagens entre aplicações. Além disso, o AMQP define o armazenamento e o encaminhamento de mensagem (*store-and-forward*), publicação e assinatura de serviços (*publish-and-subscribe*), bem como os comportamentos para a transmissão de mensagens entre as aplicações (O'HARA, 2007).

De acordo com a especificação v0.10, o protocolo AMQP é dividido em três camadas, que são: modelo, sessão e transporte. A Figura 1 apresenta as três camadas do protocolo, sendo que nas próximas seções serão detalhadas as camadas do protocolo AMQP. A Seção 2.1 descreve a Camada de Modelo, que é a camada que contém o conjunto de objetos que serão avaliados neste trabalho. Na Seção 2.2 de forma breve a Camada de Sessão, que também será avaliada neste trabalho. A Seção 2.3 apresenta a camada de transporte, que será apresentada de forma sucinta já que essa não será avaliada.

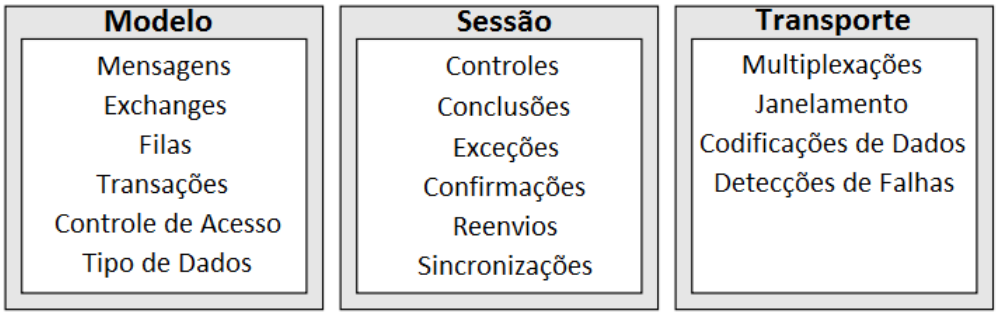
---

[1] *wire protocol* são as formas de obtenção de dados no formato ponto a ponto. Este tipo de protocolo se faz necessário quando uma aplicação precisa se comunicar com outra. Exemplos de *wire protocols* são *Internet Inter-Orb Protocol*(IIOP) para *Common Object Request Broker Architecture* (CORBA), *Java Remote Method Protocol* (JRMP) para *Remote Method Invocation*(RMI) e *Simple Object Access Protocol*(SOAP) para *Web Services*.

[2] http://www.imatix.com/. Disponível em 31/10/2015

[3] OASIS é um consórcio sem fins lucrativos que impulsiona o desenvolvimento, convergência e adoção de normas abertas para a sociedade da informação global (OASIS, 2015).

Figura 1: Camadas padrão AMQP v0.10



Fonte: AMQP: A General-Purpose Middleware Standard v0.10

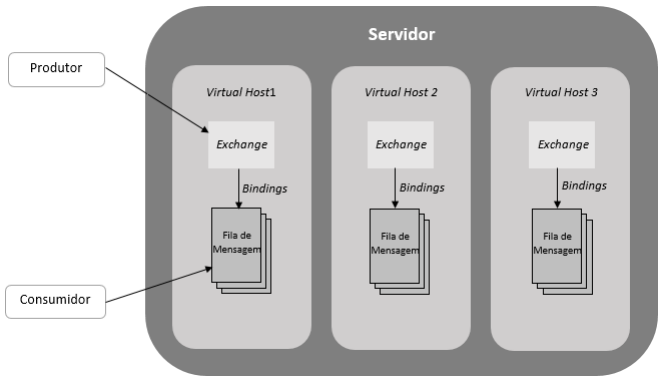
2.1 Camada de modelo

A camada de modelo define um conjunto de classes e de funcionalidades que podem ser utilizadas pela aplicação. De acordo com Trieloff et al. (2006) a camada de modelo é responsável por garantir:

> Interoperabilidade entre as implementações; prover um controle explícito sobre a qualidade do serviço (QoS - *Quality of Service*); suportar qualquer domínio de *middleware* (mensagem, transferência de arquivos, chamada remota de procedimentos, et.al); contemplar padrões de protocolos abertos de mensagem (como, por exemplo o *Java Message Service* (JMS); definir padrões de nomenclatura; permitir uma completa configuração do servidor a partir do protocolo; prover uma notação simples e de fácil integração por parte das aplicações.

A Figura 2 apresenta a estrutura da camada modelo AMQP onde, o produtor conecta-se a um *virtual host* enviando uma mensagem. Essa é direcionada a uma fila de mensagens a qual se encontra conectada a um consumidor que irá receber essa mensagem. Nas subseções seguintes serão detalhados os principais componentes desta camada.

Figura 2: Servidor AMQP



Fonte: baseado em Carmo (2012)

2.1.1 Servidores

Segundo Trieloff et al. (2006) os servidores, também conhecidos como *brokers*, são responsáveis por receber conexões dos clientes de mensagens (processos que enviam mensagens), e de processos que assinam filas (processos que recebem as mensagens). Além disso, é no servidor que ficam localizados os *virtual hosts*, os encaminhadores de mensagem (*exchanges*), as filas de mensagem e os *bindings*, que serão detalhados nas próximas subseções.

O servidor AMQP é responsável ainda por gerenciar a autenticação e permissões dos usuários (TRIELOFF et al., 2006). Portanto, antes do cliente ser direcionado a um *virtual host*, primeiro é necessário que ele se autentique no servidor AMQP. A autenticação é realizada utilizando o método de autenticação *Simple Authentication and Security Layer* (SASL)(TRIELOFF et al., 2006)

2.1.2 Virtual Hosts

Os *virtual hosts* são áreas semelhantes a *conteiners*[4], sendo que neles ficam armazenadas as filas de mensangem, os *exchanges* (responsáveis pelo encaminhamento de mensagens) e os *bindings* (TRIELOFF et al., 2006). Destaca-se, que quando um cliente deseja estabelecer uma conexão, esse pode se conectar apenas a um dos *virtuais hosts* disponíveis.

Um *virtual host* é uma área independente totalmente separada dos demais. Portanto não existe nenhuma forma de um *virtual host* se comunicar com outro. Além disso, quando um cliente se conecta a um *virtual host* não existe possibilidade de migrar para outro. Para tanto, é necessário encerrar a conexão e criar uma nova conexão.

O protocolo AMQP não define nenhum mecanismo para criar ou configurar um *virtual host*. A forma de criação e configuração é totalmente dependente da implementação da ferramenta que está sendo utilizada.

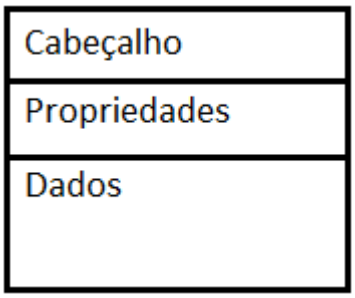
2.1.3 Mensagens

Uma mensagem é uma unidade atômica de encaminhamento e enfileiramento. As mensagens são compostas por um cabeçalho, propriedades e dados. Os campos de cabeçalho e de propriedades apresentam um conjunto de informações do tipo *chave/valor*. As entradas que estão no cabeçalho, apresentam informações definidas pelo padrão AMQP. Já as entradas, do bloco de propriedades, apresentam um conjunto de informações arbitrárias que podem ser definidas pela aplicação. O bloco de dados contém uma sequência de *bytes*

[4] *Conteiners* são áreas independentes que agrupam informações sobre um determinado processo ou programa. Em ambientes de desenvolvimento esse possui tudo que uma aplicação precisa para ser executada, ou seja, as aplicações ficam isoladas das demais aplicações, sendo essas totalmente independentes.

que corresponde ao conteúdo enviado pela aplicação (TRIELOFF et al., 2006). Na Figura 3 tem-se um exemplo da estrutura de uma mensagem AMQP.

Figura 3: Estrutura de uma mensagem AMQP



Fonte: baseado em Trieloff et al. (2006)

As mensagens podem ser persistentes, sendo armazenadas no disco rígido. Além disso, as mensagens possuem garantia de entrega, ou seja, essas serão reenviadas caso ocorra uma falha de rede, parada no servidor, *overflow* de uma fila, etc (TRIELOFF et al., 2006). As mensagens possuem ainda níveis de prioridade. Desta forma, mensagens que não necessitam ser processadas rapidamente podem ser definidas com uma prioridade baixa. Já as mensagens críticas, podem ser definidas com alta prioridade, sendo distribuídas rapidamente para os consumidores da fila (TRIELOFF et al., 2006).

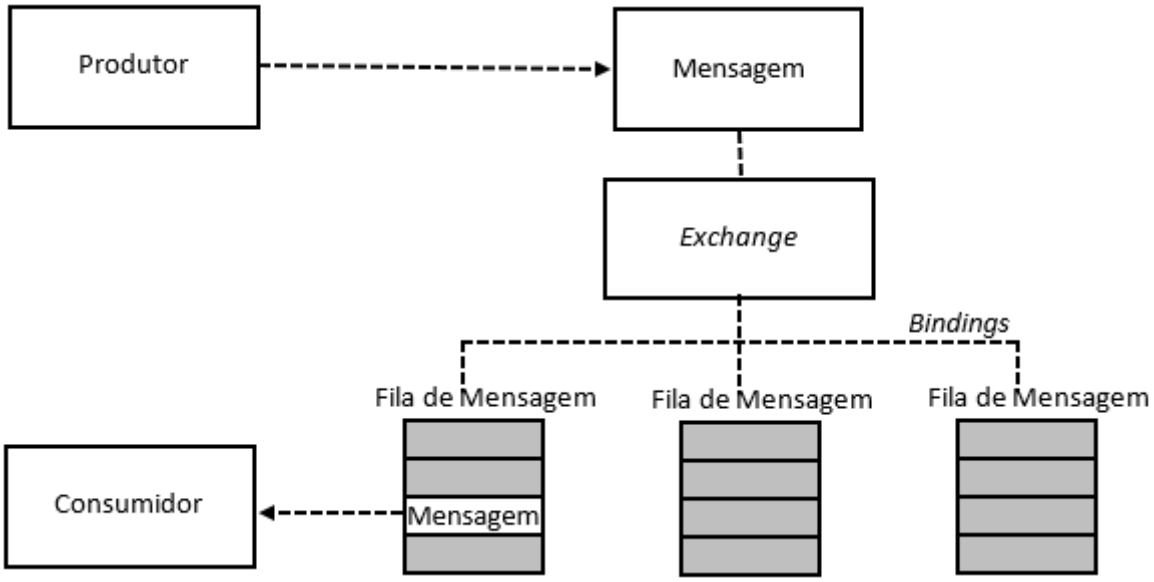
2.1.3.1 Fluxo de mensagens

O fluxo de mensagens, consiste no caminho que uma mensagem percorre dentro de um servidor AMQP e de seu *Virtual Host*. Conforme pode ser observado na Figura 4, o fluxo inicia a partir de um produtor que publica a mensagem e a envia a um servidor AMQP. Posteriormente, a mensagem é direcionada para o *exchange* que irá encaminhar a mensagem para uma das filas de mensagens. As filas de mensagens encontram-se conectadas ao *exchange* através de *bindings*. Por fim, o consumidor que estiver conectado a fila de mensagens irá receber a mensagem.

Trieloff et al. (2006, p.5) faz uma analogia do fluxo de mensagens a um servidor de *e-mail*:

> Segundo ela, um servidor AMQP é análogo a um servidor de *e-mails*, sendo que cada *exchange* age como um agente de transferência de mensagens, e cada fila de mensagens como uma caixa de correio. Os *bindings* definem as tabelas de roteamento em cada agente de transferência de mensagens. Clientes enviam mensagens para agentes de transferência que, em seguida, encaminham as mensagens para as caixas de correio. Já os consumidores coletam as mensagens destas caixas de correio.

Figura 4: Fluxo AMQP



Fonte: baseado em Trieloff et al. (2006)

Essa analogia mostra-se inadequada quando analisa-se o comportamento de um consumidor no padrão AMQP. Um consumidor AMQP pode ser uma aplicação que espera por uma mensagem vindo de uma fila de mensagem em particular, sendo que essa mensagem será processada fora da fila de mensagens. Outras particularidades do AMQP é a possibilidade dos clientes criarem ou destruírem filas de mensagem. Analogamente, seria como se no servidor de *e-mails* fosse possível criar ou excluir caixas de correio. Também é possível definir padrões de encaminhamento a partir de *bindings keys*. O uso de um *binding key* seria como se o servidor de *e-mail* permitisse que somente as mensagens com um cabeçalho específico fossem encaminhadas a uma determinada caixa de correio. A utilização de *bindind keys* será descrita na Seção 2.1.5.

2.1.4 Filas de mensagem

Uma fila de mensagem ou *message queue* é um *buffer* onde são armazenadas as mensagens que serão consumidas pelas aplicações. Estas aplicações podem criar, compartilhar, utilizar e destruir filas de mensagens, de acordo com as das permissões que possuem perante o servidor.

As filas de mensagens seguem o conceito *First In First Out* (FIFO) de forma limitada, uma vez que essas possuem um controle baseado em prioridades das mensagens (TRIELOFF et al., 2006). Para mensagens enviadas com uma mesma prioridade, o servidor irá manter a ordem das mensagens. Porém, no caso de um erro na entrega de uma mensagem, essa será colocada no final da fila e, consequentemente, ficará fora de ordem.

As filas de mensagem podem ser criar criadas com três estados de armazenamento,

que são (CARMO, 2012):

- Duráveis: são armazenadas até que sejam excluídas manualmente. Mesmo com essa característica, as filas duráveis não garantem que mensagens não persistentes sejam mantidas quando o servidor for reinicializado.
- Transientes: filas transientes permanecem disponíveis até que o servidor (*broker*) seja reiniciado.
- Auto excludentes: são filas que permanecem disponíveis até que não sejam mais utilizadas.

2.1.5 Bindings

Um *binding* representa a ligação entre uma fila de mensagens e um *exchange*. Um *binding* é uma conexão entre um único *exchange* e uma única fila de mensagens. É no *binding* que fica localizado a *binding key*, que é responsável por armazenar uma chave que é utilizada para o encaminhamento das mensagens. O *exchange* irá comparar o valor da *binding key* com a *routing key* de uma mensagem e encaminhará essa mensagem as filas de mensagem correspondentes. A *binding key* é definida quando é realizada a criação do *binding*. Já a *routing key* é definida no momento do envio da mensagem.

Aplicações podem criar e excluir os *bindings* conforme suas necessidades e, desta forma, o fluxo de mensagens pode ser alterado de acordo com as necessidades da aplicação. A durabilidade de um *binding* é definido pela durabilidade da fila de mensagens e do *exchange*. Caso uma fila de mensagens ou um *exchange* seja excluído, o *binding* será automaticamente removido.

2.1.6 Exchanges

Os *exchanges* são processos que ficam localizados dentro dos *virtual hosts* e são responsáveis por fazer o roteamento das mensagens entre as filas de mensagem (TRIELOFF et al., 2006). Ou seja, o *exchange* recebe as mensagens dos clientes e as encaminha para uma ou mais filas de mensagem.

Carmo (2012) destaca que os *exchanges* podem ser duráveis, transientes ou de exclusão automática. Os *exchanges* duráveis são aqueles que permanecem disponíveis até que sejam excluídos manualmente; os transientes são aqueles que duram até que o servidor seja reinicializado; e os de exclusão automática são removidos quando não são mais utilizados.

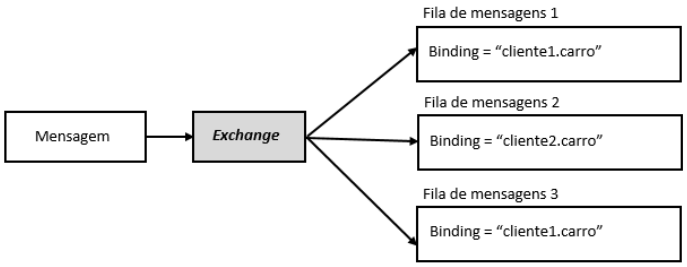
Um *exchange* pode encaminhar uma mensagem para múltiplas filas, sendo que neste caso são geradas novas instâncias da mensagem para que sejam consumidas de forma

independente. Nas subseções seguintes serão detalhados os métodos de encaminhamentos de mensagem definidos pelo padrão AMQP. Mais especificamente serão descritos o encaminhamento direto, o encaminhamento *fanout*, o encaminhamento por tópicos e o encaminhamento por cabeçalhos.

2.1.6.1 Encaminhamento direto (*direct exchange*)

O encaminhamento direto é uma forma de roteamento onde a mensagem é enviada para uma ou mais filas de mensagem, sendo que a *routing key* da mensagem deve ser idêntica a *binding key*. O funcionamento de uma fila de encaminhamento direto pode ser observado na Figura 5, onde existem três filas, sendo que foi definido para o *binding* da fila de mensagens 1 a chave "*cliente1.carro*". Já para o *binding* da fila de mensagens 2 a chave "*cliente2.carro*" e para o *binding* da fila de mensagens 3 a chave "*cliente1.carro*". Portanto, quando uma mensagem é enviada com um *routing key* "*cliente1.carro*", essa mensagem será encaminhada para as filas 1 e 3. É importante destacar que são enviadas mensagens idênticas as filas, porém com instâncias distintas, ou seja, é realizada uma cópia da mensagem para cada fila. Já, quando uma mensagem é enviada com um *routing key* "*cliente2.carro*", essa mensagem é encaminhada somente para a fila 2.

Figura 5: Encaminhamento direto



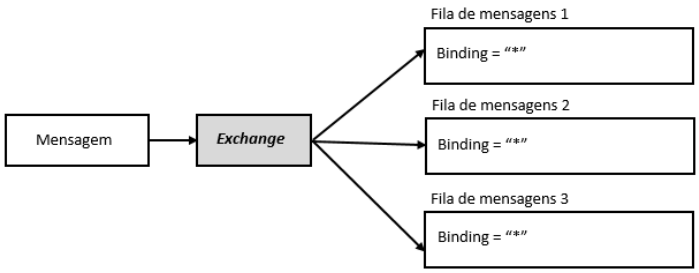
Fonte: elaborado pelo autor

2.1.6.2 Encaminhamento *fanout*

Encaminhamento *fanout* é uma forma de roteamento onde a mensagem é enviada para todas as filas de mensagem ligadas a um *exchange*, ou seja, de forma similar a um *broadcasting*[5]. O funcionamento de uma fila *fanout* pode ser observado na Figura 6, onde existem três filas, sendo que o *binding* para todas as filas encontra-se em branco. Uma vez que o *exchange* foi definido como *fanout*, a *binding key* não é verificada e mensagem é encaminhada para todas as filas. Desta forma, quando uma mensagem é enviada, não será realizada nenhuma comparação e uma instância da mensagem é encaminhada para cada uma das filas.

[5] *Broadcasting* em telecomunicações, é um método de transferência de mensagens, onde a mensagem é encaminhada para todas as estações que estão conectadas a mesma rede do computador que enviou a mensagem (Andrew Tanenbaum (2003). Computer Networks v.5. p. 380. ISBN-13: 978-0-13-212695-3).

Figura 6: Encaminhamento *fanout*



Fonte: elaborado pelo autor

2.1.6.3 Encaminhamento por tópicos (*topic exchange*)

O encaminhamento por tópicos é uma forma de roteamento onde a mensagem é enviada para uma ou mais filas assinadas, sendo que a *routing key* da mensagem deve corresponder a uma expressão que foi criada no *binding*.

Neste caso a *binding key* é gerada utilizando zero ou mais *tokens*, onde, cada token é delimitado pelo caractere ".". Caso seja necessário validar apenas uma parte da chave passada pela *routing key*, utiliza-se de caracteres coringa opcionais como, por exemplo, os caracteres "*" e "#". O caractere "*" faz com que seja aceito qualquer token na posição ocupada por esse. Já o caractere "#" faz com que seja aceito nenhum, um ou mais tokens (TRIELOFF et al., 2006). Um exemplo é a *binding key* "*.azul.#" que aceita as *routing keys* "carro.azul" e "caminhao.azul.6x6.mercedes", mas não aceita a *routing key* "azul.4x6".

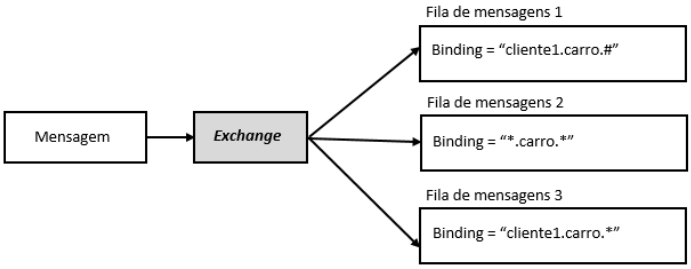
O funcionamento de uma fila de encaminhamento por tópicos pode ser observado na Figura 7. Nela existem três filas, sendo definido para o *binding* da fila de mensagens 1 a chave "*cliente1.carro.#*", para o *binding* da fila de mensagens 2 a chave "**.carro.**" e para o *binding* da fila de mensagens 3 a chave "*cliente1.carro.**". Portanto, quando uma mensagem é enviada com a *routing key* "*cliente1.carro.azul*", as filas de mensagem 1, 2 e 3 receberão a mensagem. Já, quando uma mensagem for enviada com uma *routing key* "*cliente1.carro.azul.4x4*", somente a fila 1 receberá a mensagem. Por fim, quando uma mensagem é enviada com a *routing key* "*cliente2.carro.preto*", somente a fila 2 irá receber a mensagem.

2.1.6.4 Encaminhamento por cabeçalhos (*headers exchange*)

No encaminhamento por cabeçalhos é definido uma lista de atributos e valores opcionais no cabeçalho da mensagem. Estes atributos serão avaliados pelo *exchange* para definir para qual fila de mensagens será enviada a mensagem. Neste caso não será informada nenhuma *routing key*.

Juntamente com a lista de valores é passado o atributo "*x-match*" que deve ser preenchido com os valores *any* ou *all*. O valor *any* define que uma mensagem será

Figura 7: Encaminhamento por tópicos

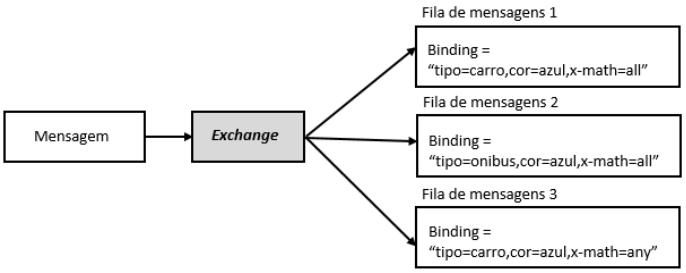


Fonte: elaborado pelo autor

encaminhada para uma fila se pelo menos um dos valores do cabeçalho combinar com os valores que foram definidos no *binding*. Esse comportamento é similar a operação lógica *OR*. Já a operação *all*, define que uma mensagem será enviada para a fila somente se todos os valores do cabeçalho da mensagem combinarem com os valores que foram definidos no *binding*. Esse comportamento é similar a operação lógica *AND*.

O funcionamento de uma fila de encaminhamento por cabeçalhos pode ser observado na Figura 8. Nela existem três filas de mensagem, sendo definido para o *binding* da fila de mensagens a lista de tópicos "*tipo=carro, cor=azul, x-match=all*", para o *binding* da fila de mensagens 2 a lista de tópicos "*tipo=onibus, cor=azul, x-match=all*" e para o *binding* da fila de mensagens 3 a lista de tópicos "*tipo=carro, cor=azul, x-match=any*". Assim, quando uma mensagem é enviada com o cabeçalho "*tipo=carro, cor=azul*", as filas de mensagem 1 e 3 receberão uma instância da mensagem. Já, quando uma mensagem é enviada com o cabeçalho "*tipo=onibus, cor=azul*", somente as filas de mensagem 2 e 3 receberão uma instância da mensagem. Por fim, quando uma mensagem é enviada com o cabeçalho "*tipo=carro, cor=*", somente a fila 3 receberá a mensagem.

Figura 8: Encaminhamento por cabeçalhos



Fonte: elaborado pelo autor

2.2 Camada de sessão

A camada de sessão foi introduzida a partir da versão 0.10 do protocolo AMQP, atuando como uma interface entre a camada modelo e camada de transporte. Essa tem

como principal objetivo proporcionar uma maior confiabilidade na transmissão de comandos para um servidor AMQP. Além disso, essa camada objetiva uma maior confiabilidade na transmissão de mensagens entre as aplicações que se comunicam com o servidor (TRIELOFF et al., 2006).

Assim, toda a operação de envio de mensagens, criação de filas de mensagem, etc deve ser feita dentro do contexto de uma sessão. De acordo com o padrão AMQP, a camada de sessão prove os seguintes serviços para a camada modelo:

- Identificação sequencial dos comandos: uma sessão garante que um comando é executado apenas uma vez. Para isto é utilizada uma numeração sequencial para uma identificação única dos comandos que foram enviados para um servidor. No caso de falhas, a partir dessa numeração, é possível identificar se um comando já foi executado ou não pelo servidor.

- Confirmação: para que o cliente seja capaz de descartar um comando, esse deve se certificar que o comando chegou a seu destino. A confirmação de entrega é utilizada para que um cliente saiba que o comando foi recebido pelo servidor. As confirmações podem ser enviadas em lotes ou encaminhadas individualmente. Além disso, uma notificação pode ser omitida, caso o cliente não necessite de uma confirmação urgente, sendo posteriormente sinalizada por uma notificação de comandos completados.

- Notificação de comandos completados: diferente da confirmação, a notificação de comandos completados tem como objetivo retornar a ordem em que os comandos foram executados pelo servidor. O cliente pode utilizar essa confirmação para verificar se uma transação foi totalmente executada. A notificação de comandos completados é um complemento a confirmação de entrega. Essas notificações podem ser enviadas em lotes ou encaminhadas individualmente.

- Reenvio e recuperação de falhas: para que o sistema se recupere de uma falha, a sessão deve ser capaz de reenviar um comando sem uma notificação de comando completado. Para tanto, a sessão utiliza-se dos recursos de confirmação e identificação sequencial dos comandos, permitindo que as mensagens, sejam reenviadas sem o risco de duplicidade.

2.3 Camada de transporte

A camada de transporte é baseada no *Stream Control Transmission Protocol* (SCTP) (KRAMER, 2009) e é responsável pelo encaminhamento da mensagem. Essa camada inclui os conceitos de portas, protocolo de cabeçalho, protocolo de negociação e *framing*, onde:

- O protocolo de cabeçalho: define o cabeçalho da mensagem e como os atributos devem ser informados.
- O protocolo de negociação: define os passos para o estabelecimento de uma conexão entre o cliente e o intermediador de mensagens. Além disso, o protocolo de negociação define como esses irão se comportar no caso de incompatibilidades.
- O *framing*: compreende a definição de como um mensagem deverá ser enviada entre um cliente e o intermediador de mensagens. É no *framing* que ficam as definições de segmentação da mensagem, tamanho máximo da mensagem por segmento, formato de envio, canais de envio, etc.

2.4 Implementações

Nesta seção serão abordadas as implementações do protocolo AMQP RabbitMQ (PIVOTAL, 2015), Apache Qpid (APACHE QPID, 2015) e Apache ActiveMQ (ACTIVEMQ, 2015), que foram as ferramentas avaliadas neste trabalho. Optou-se por essas ferramentas devido ao fato de serem, de código aberto e estarem de acordo com o protocolo AMQP.

2.4.1 RabbitMQ

O RabbitMQ é um intermediador de mensagens mantido pela *Pivotal Software,* que faz parte da empresa *EMC Corporation* e da sua subsidiária *VMware*. Esse é desenvolvido na linguagem Erlang[6] possuindo recursos que visam uma maior escalabilidade, bem como um sistema de *cluster* que garante uma alta disponibilidade do intermediador de mensagens. Além disso, esse possui um sistema de *plugins* que possibilita a adição de novos recursos ao intermediador de mensagens (PIVOTAL, 2015).

O RabbitMQ suporta o protocolo AMQP nas versões 0.9.1, 0.9 e 0.8 de forma nativa. A versão 1.0 do AMQP, por apresentar alguns conceitos distintos das demais versões (como por exemplo a camada de sessão), pode ser utilizada somente a partir da adição de um *plugin* experimental (PIVOTAL, 2015). Além disso, esse pode ser utilizado com os protocolos STOMP (*Streaming Text Oriented Messaging Protocol*) (STOMP, 2012), MQTT (*Message Queue Telemetry Transport*) (STANFORD-CLARK; HUNKELER, 1999) e HTTP (*Hypertext Transfer Protocol*) (FIELDING et al., 1999). Para tanto, basta adicionar o *plugin* adequado (PIVOTAL, 2015).

Por fim, o RabbitMQ possui implementações de sua API (*Application Program Interface*) em diversas linguagens, sendo que as mais utilizadas são: Java, .NET, C++, C, Perl, Go, Objective C, Java (Android), Ruby, Python, PHP, Java Script.

[6] http://www.erlang.org/

2.4.2 Qpid

O Qpid é um projeto da organização sem fins lucrativos *Apache Software Foundation*, que fornece recursos para o desenvolvimento de aplicativos sobre o protocolo AMQP. O Qpid oferece dois componentes, que são: uma API de mensagem para construção de aplicativos; e intermediadores de mensagens, que implementam diversas versões do protocolo (ver Tabela 1) (APACHE QPID, 2015).

Tabela 1: Compatibilidade Qpid

Componente	Linguagem	Plataformas	AMQP suportado
C++ broker	-	Linux, Windows	1.0, 0-10
Dispatch router	-	Linux	1.0
Java broker	-	JVM	1.0, 0-10, 0-9-1, 0-9, 0-8
Qpid JMS	Java	JVM	1.0, 0-10, 0-9-1, 0-9, 0-8
Qpid Messaging API	C++, Perl, Python, Ruby, .NET	Linux, Windows	1.0, 0-10
Qpid Proton	C, Java, JavaScript, Perl, PHP, Python, Ruby	JVM, Linux, OS X	1.0

fonte: http://qpid.apache.org

Os intermediadores de mensagem disponíveis no Qpid são totalmente baseados no padrão AMQP, provendo recursos para gerenciamento de filas, persistência de mensagens e gerenciamento. Esses intermediadores encontram-se disponíveis nas linguagens Java e C++, sendo que em ambas apresentam as mesmas funcionalidades (APACHE QPID, 2015). Além disso, existe um intermediar de mensagem chamado de *dispatch router*. Este é desenvolvido na linguagem C e se comunica com a aplicação a partir do Qpid Proton, fornecendo uma conexão flexível e escalável entre os clientes AMQP (APACHE QPID, 2015).

2.4.3 ActiveMQ

ActiveMQ é um intermediador de mensagem de código aberto muito utilizado pela comunidade. Esse foi desenvolvido pela organização sem fins lucrativos *Apache Software Foundation*, e seus objetivos são ser rápido e ter suporte a diversos protocolos de envio de mensagens (ACTIVEMQ, 2015). De acordo com a *Apache* esse é eficiente, confiável, e de fácil manutenção, suportando o protocolo AMQP versão 1.0 e os protocolos STOMP (STOMP, 2012), MQTT (STANFORD-CLARK; HUNKELER, 1999) e *OpenWire* (SNYDER; BOSNANAC; DAVIES, 2011).

O ActiveMQ não possui uma API de mensagens diretamente implementada, sendo necessário o uso de um cliente que implemente o protocolo AMQP versão 1.0. Um exemplo de cliente que pode ser utilizado é o *Apache Qpid Proton* (ACTIVEMQ, 2015).

2.4.4 Resumo das funcionalidades

Para a comunicação com os intermediadores de mensagem, será priorizado a versão do protocolo AMQP 0.9.1, pois essa é a definição que tem em sua especificação, a arquitetura da camada modelo dos intermediadores de mensagem. Destaca-se que as ferramentas RabbitMQ e Qpid implementam essa versão do protocolo por completo, já a ferramenta ActiveMQ implementa apenas a versão AMQP 1.0. Na Tabela 2 é apresentado um resumo das características das ferramentas que serão avaliadas neste trabalho.

Tabela 2: Comparação dos intermediadores de mensagem

	RabbitMQ	**Qpid**	**ActiveMQ**
Linguagem desenvolvida	Erlang	Java, C++	Java
Versões AMQP suportadas	0-8, 0-9, 0-9-1, 1.0	0-8, 0-9, 0-9-1, 0-10, 1.0	1.0
Outros protocolos suportados	STOMP, MQTT, HTTP	-	STOMP, MQTT, OpenWire
Possui clientes AMQP	Sim	Sim	Não
Principais linguagem dos clientes AMQP	Java, .NET, C++, C, Perl, Objective C, Java (Android), Ruby, Python, PHP, Go, Java Script, Node.js	C, C++, Java, Python, Perl, .NET, Ruby, Java Script	-

2.5 Considerações

Neste capítulo foi apresentado o protocolo AMQP, dando ênfase a camada de modelo e suas características. Na Seção 2.4 foram apresentados os intermediadores de mensagem que serão avaliados neste trabalho. No próximo capítulo será apresentado o método analítico hierárquico (AHP - *Analytic Hierarchy Process*), que será o método utilizado para a avaliação dos intermediadores de mensagens escolhidos.

[6] *Plugin* experimental.

3 MÉTODO DE AVALIAÇÃO

> "O tomador de decisões, quer esteja motivado pela necessidade de prever ou controlar, geralmente enfrenta um complexo sistema de componentes correlacionados, como recursos, resultados ou objetivos desejados, pessoas ou grupos de pessoas; ele está interessado na análise desse sistema. Presumivelmente, quanto melhor ele entender essa complexidade, melhor será sua decisão". (RIBEIRO, 2003).

A tomada de decisão é um desafio, uma vez que engloba um problema, um conjunto de variáveis e julgamentos de valores. Em problemas onde existem múltiplos critérios para serem analisados, a experiência e o conhecimento dos tomadores de decisão devem ser considerados. Porém, a experiência prévia e as preferências pessoais desses, podem acabar prejudicando nessa escolha, ou seja, essas experiências podem tornar a escolha subjetiva. Os métodos de análise multicritério, buscam modelar e resolver problemas desta natureza.

Dentre os métodos desse tipo, destaca-se o Método de Análise Hierárquica (AHP - *Analytic Hierarquic Process*). Neste capítulo será descrito o método AHP que foi o método utilizado neste trabalho. Optou-se por esse método de forma a diminuir a subjetividade na avaliação das ferramentas e por este já ter sido utilizado com sucesso em trabalhos anteriores (TROITIñO, 2012; SPOLTI, 2015). Maiores informações sobre o método AHP podem ser encontradas em (SAATY, 1980).

3.1 Método de Análise Hierárquica

O AHP é um método utilizado para tratar de critérios quantificáveis e/ou intangíveis, auxiliando as pessoas na tomada de decisões complexas. Além de auxiliar na escolha, o método AHP gera informações que justificam essa escolha. Para se tomar uma decisão de forma organizada e priorizando as necessidades, Chan, Kwok e Duffy (2004) destacam que o avaliador deve seguir os seguintes passos:

1. Definir o problema ou meta que se deseja alcançar. Nessa etapa, é fundamental que todos os critérios estejam bem definidos, proporcionando que a meta estipulada seja devidamente atingida.

2. Estruturar a decisão de forma hierárquica, onde, no topo da hierarquia tem-se o objetivo (Figura 9). Abaixo do objetivo, no nível intermediário, são definidos os critérios que podem ter mais de um nível (Figura 10). No último nível da árvore (nodo folha) ficam as alternativas que serão avaliadas. Nessa etapa, é importante se certificar que os níveis estejam consistentes e completos. Essa etapa será descrita na Seção 3.1.1.

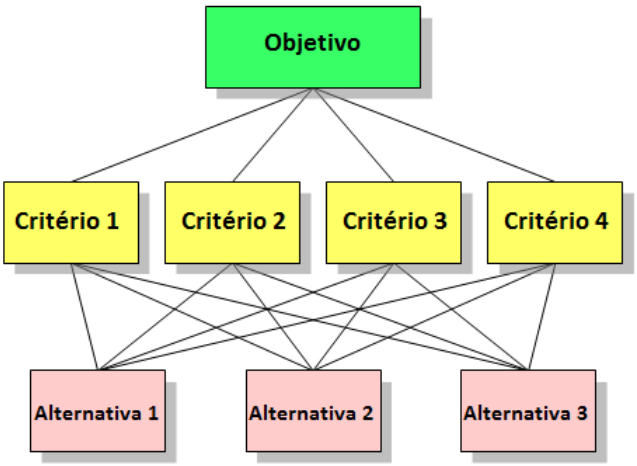
3. Construir matrizes de comparação paritária entre os elementos presentes em um mesmo nível e com um mesmo pai na hierarquia. Essas matrizes objetivam comparar dois a dois todos os critérios (e alternativas) que foram definidas. Nesta etapa devem ser feitos os julgamentos de prioridade de forma a avaliar os critérios, e as alternativas de acordo com sua importância. Esta etapa será descrita na Seção 3.1.2.

4. Calcular a consistência das matrizes de comparação paritária e avaliar sua consistência calculando a razão de consistência (RC). Esta etapa será descrita na Seção 3.1.3.

3.1.1 Estruturação Hierárquica

A estruturação hierárquica consiste em esquematizar os critérios e as alternativas priorizando os mesmos, de forma a alcançar o objetivo estipulado. A estrutura hierárquica é definida colocando no topo da hierarquia o objetivo. Abaixo do objetivo ficam os critérios e, por fim, as alternativas que serão avaliadas. Na Figura 9 pode ser observada a estrutura da hierarquia proposta pelo método AHP.

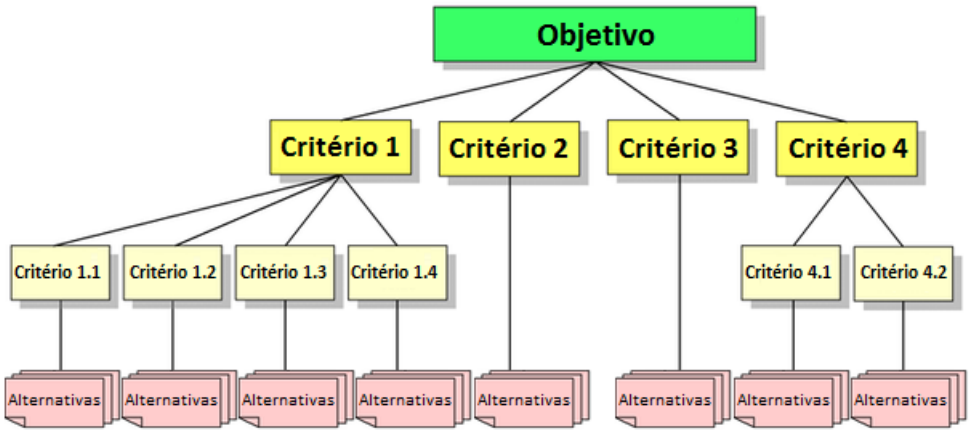
Figura 9: Hierarquia de um estudo em AHP com critérios de um nível



Fonte: baseado em Saaty (2008)

O método AHP permite que critérios sejam divididos em critérios de menor importância. Assim, quando uma situação é complexa, é mais fácil que alguns dos critérios sejam divididos em subcritérios que sejam mais simples. Mesmo divididos, a participação destes subcritérios continua fazendo parte da decisão global. Desta forma, quando for realizado o fechamento da avaliação, esses subcritérios irão compor o percentual do critério superior (critério pai da árvore hierárquica). Na Figura 10 é possível observar um exemplo de critérios subdivididos.

Figura 10: Hierarquia de um estudo em AHP com critérios com mais de um nível



Fonte: baseado em Saaty (2008)

3.1.2 Matrizes de comparação

A partir da hierarquia, são geradas as matrizes de comparação paritárias. Nesse momento, os critérios são comparados dois a dois e são definidos os pesos de acordo a importância que um critério possui sobre outro. A especificação do método AHP apresenta os pesos baseando-se em uma escala de 1 à 9, que é conhecida como a Escala Fundamental de Saaty (SILVA; BELDERAIN, 2005).

A Tabela 3 apresenta a escala fundamental de Saaty, onde observa-se que os valores ímpares são explicitamente descritos, sendo possível identificar a situação em que devem ser utilizados. Já os valores pares são considerados intermediários e devem ser utilizados em situações onde o avaliador possui dúvidas em relação a qual das faixas o critério se encontra.

Tabela 3: Escala fundamental de Saaty

Intensidade de importância	Definição	Explicação
1	Importância de mesmo nível	Duas atividades contribuem igualmente com o objetivo
3	Importância moderada	A experiência e o julgamento favorecem levemente uma atividade sobre outra
5	Importância forte	A experiência e julgamento favorecem fortemente uma atividade sobre outra
7	Muito forte ou de importância demonstrada	Uma atividade possui um favorecimento muito forte sobre outra
9	Importância extrema	A evidência favorece uma atividade com o maior grau de certeza
As intensidades 2, 4, 6 e 8 podem ser utilizadas para expressar valores intermediários.		
As intensidades 1.1, 1.2, 1.3, etc. podem ser utilizadas para elementos que estiverem muito próximos em importâncias.		

Fonte: Baseado em Saaty (2008)

Como já mencionado, para a criação da matriz de comparação paritária, todos os itens devem ser comparados dois a dois. Por exemplo, um critério X é comparado a um critério Y, e assim é definido o grau de importância que esse critério X possui perante o critério Y. De acordo com a Tabela 3 a análise deve ser feita da seguinte forma:

- Quando o critério X tiver um grau de importância maior que o critério Y, esse irá receber um valor inteiro do grau de importância (valor retirado da Tabela 3). Por exemplo, se o critério X for 5 vezes mais importante que o critério Y, esse receberá o peso 5.
- Uma vez que o critério Y possui um grau de importância menor que o critério X, esse irá receber um valor correspondente a *1/grau de importância*. No exemplo descrito o critério Y irá receber um peso $\frac{1}{5}$, uma vez que esse é 5 vezes menos importante que o critério X.
- Quando um critério é comparado com ele mesmo, o grau de importância será sempre 1.

Na Tabela 4 tem-se um exemplo de criação de uma matriz de comparação para uma família que deseja comprar um carro. Observa-se nesta tabela que o critério "*Preço*" é 6 vezes mais importante que o critério "*Manutenção*" e 4 vezes mais importante que o critério "*Consumo*". Observa-se ainda, que o critério "*Manutenção*" é 3 vezes menos importante que o critério "*Consumo*".

Tabela 4: Matriz com os graus de importância

	Preço	**Manutenção**	**Consumo**
Preço	1	6	4
Manutenção	1/6	1	1/3
Consumo	1/4	3	1
Total	**1,417**	**10**	**5,333**

Após a criação da matriz de grau de importância essa deve ser normalizada. A normalização é realizada através da divisão de cada elemento pelo somatório dos valores da coluna onde esse elemento se encontra. Após a normalização a soma de todos os elementos de uma coluna irá resultar em 1 (100%). Na Tabela 5 pode ser observada a normalização da matriz de comparação paritária que foi apresentada na Tabela 4.

Tabela 5: Normalização da matriz

	Preço	**Manutenção**	**Consumo**
Preço	1/1,417 = **0,706**	6/10 = **0,6**	4/5,333 = **0,750**
Manutenção	(1/6)/1,417 = **0,118**	1/10 = **0,1**	(1/3)/5,333 = **0,063**
Consumo	(1/4)/1,417 = **0,176**	3/10 = **0,3**	1/5,333 = **0,187**

A partir da matriz normalizada é calculado o vetor da prioridade média local (PML), que consiste no autovetor da matriz de comparação paritária normalizada. Esse vetor representa o percentual de importância que um critério possui sobre todos os demais critérios. Para o cálculo da PML é necessário somar os elementos da linha da matriz normalizada (Tabela 5), dividindo pela quantidade de critérios da matriz. A soma dos valores do vetor resultará sempre em 1 (100%). O processo de cálculo dos percentuais locais pode ser visto na Tabela 6. Desta forma, observa-se que o "*Preço*" é o critério mais importante, apresentando uma representatividade de 62,9%, enquanto o "*Consumo*" possui uma representatividade de 30,1%. Já a "*Manutenção*" possui uma representatividade de apenas 7%.

Tabela 6: Montagem dos percentuais locais

	Cálculo	**PML**
Preço	(0,706 + 0,6 + 0,750) / 3	**0,685 (68,5%)**
Manutenção	(0,118 + 0,1 + 0,063) / 3	**0,094 (9,4%)**
Consumo	(0,176 + 0,3 + 0,187) / 3	**0,221 (22,1)%**

3.1.3 Cálculo da Consistência

É difícil se obter um percentual de 100% de consistência, uma vez que existem fatores que podem afetar a análise. De fato, quando fazemos escolhas deste tipo, essas frequentemente consideram preferências, circunstâncias, etc. No método AHP existe um fator que é chamado de razão de consistência (RC) e é utilizada para avaliar a consistência dos resultados. O cálculo da razão de consistência é realizado a partir da equação:

$$RC = \frac{IC}{IR} \tag{3.1}$$

onde IC é o índice de consistência (descrito na Seção 3.1.3.1) e IR é o índice randômico de Saaty (descrito na Seção 3.1.3.2). A razão de consistência é apresentada com um valor de 0 e 1, onde 1 representa 100% de inconsistência. A razão de consistência deve assumir valores de:

- 5% ou menos para matrizes de ordem 3;
- 9% ou menos para matrizes de ordem 4;
- 10% ou menos para matrizes de ordem superior a 4.

Valores de RC superiores aos apresentados, demonstram que os resultados podem não ser confiáveis. Quando isto ocorre, deve-se revisar o processo e refazer os julgamentos a fim de evitar erros na comparação.

3.1.3.1 Índice de Consistência

O índice de consistência (IC) avalia o grau de inconsistência da matriz de comparação paritária e é obtido a partir da equação:

$$IC = \frac{|\lambda_{max} - n|}{n - 1} \tag{3.2}$$

onde, n representa a ordem da matriz. Já λ_{max} representa o maior autovalor da matriz de julgamentos paritários, sendo que para o cálculo de λ_{max} deve-se multiplicar a matriz de comparação paritária (MCP), pelo vetor de prioridades médias locais (PML). Abaixo é apresentado esse procedimento, sendo que os valores da matriz MCP e o do vetor PML foram retirados das Tabelas 4 e 6, respectivamente.

$$MCP * PML = \begin{bmatrix} 1 & 6 & 4 \\ \frac{1}{6} & 1 & \frac{1}{3} \\ \frac{1}{4} & 3 & 1 \end{bmatrix} * \begin{bmatrix} 0,685 \\ 0,094 \\ 0,221 \end{bmatrix} = \begin{bmatrix} 2,133 \\ 0,282 \\ 0,674 \end{bmatrix} \tag{3.3}$$

Após, deve-se dividir o vetor resultante pelo vetor de PML. Por fim, para obter o valor do λ_{max}, os elementos do vetor resultante da divisão devem ser somados e divididos pela quantidade de critérios. Esse procedimento pode ser observado na Equação 3.4, sendo que para o exemplo apresentado foi obtido um valor igual a λ_{max} de 3,058.

$$\begin{bmatrix} 2,133 \\ 0,282 \\ 0,674 \end{bmatrix} / \begin{bmatrix} 0,685 \\ 0,094 \\ 0,221 \end{bmatrix} = \begin{bmatrix} 3,113 \\ 3,014 \\ 3,046 \end{bmatrix} = \frac{3,113 + 3,014 + 3,046}{3} = \lambda_{max} = 3,058 \tag{3.4}$$

A partir do valor calculado para λ_{max} pode-se calcular o valor de IC. Neste caso, observa-se que o valor calculado para IC foi de 0,029 (Equação 3.5).

$$IC = \frac{3,058 - 3}{3 - 1} = 0,029 \tag{3.5}$$

3.1.3.2 Índice Randômico

O IR corresponde ao índice de randômico e é obtido para uma matriz aleatória recíproca, com elementos não-negativos e gerada de forma aleatória. Para matrizes de ordem n, os valores de IR foram aproximados por Saaty (SAATY, 1980) a partir de 500 amostras de matrizes que foram geradas aleatoriamente. Os índices aleatórios de Saaty são apresentados na Tabela 7.

Tabela 7: Tabela de índices aleatórios de Saaty

Ordem da Matriz	1	2	3	4	5	6	7	8	9	10
Valores de IR	0,00	0,00	0,58	0,90	1,12	1,24	1,32	1,41	1,45	1,49

Fonte: (SAATY, 1980)

Desta forma, para uma matriz de ordem 3, como a apresentada no exemplo proposto, tem-se um valor de IR igual a 0.58. Substituindo-se os valores de IC e IR na Equação 3.1 pelos valores 0,029 (calculado na Equação 3.5) e 0,58 (encontrado na Tabela 7), respectivamente, é possível se calcular RC (Equação 3.6), obtendo-se um valor de RC igual a 5%. Neste caso conclui-se que, a matriz de comparação encontra-se consistente, uma vez que para matrizes de ordem 3 espera-se um valor de RC inferior ou igual a 5%.

$$RC = \frac{0,029}{0,58} = 0,5 = 5\% \tag{3.6}$$

3.1.4 Considerações

Neste capítulo foi descrito o método AHP, apresentando seus elementos e como é feita a priorização dos critérios. Na Seção 3.1.1, foi apresentada a montagem da hierarquia proposta pelo método AHP. Já, na Seção 3.1.2, foi apresentada montagem das matrizes de comparação paritária e o cálculo do percentual médio local. Por fim, na Seção 3.1.3, foi apresentado o procedimento do cálculo de inconsistência, que tem por objetivo verificar a confiabilidade dos resultados. No próximo capítulo serão detalhados os critérios que foram utilizados para a avaliação dos intermediadores de mensagens escolhidos.

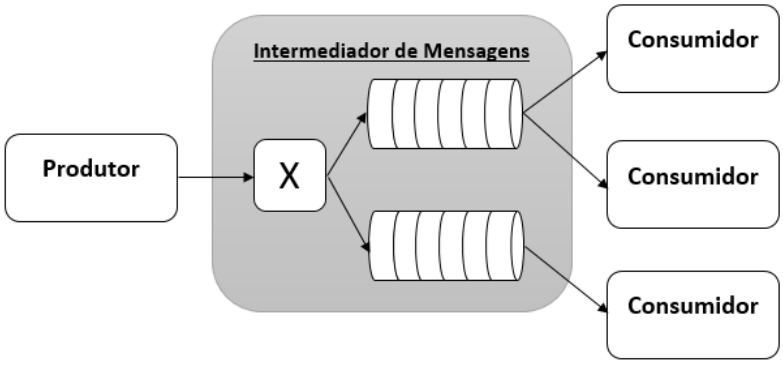
4 MODELO DE AVALIAÇÃO

De acordo com o método AHP, deve-se primeiramente definir um objetivo, que neste trabalho consiste em definir: "qual o intermediador de mensagens é o mais adequado para uso em aplicações que executam processos assíncronos". Nas seções seguintes será apresentada a aplicação de referência, bem como os critérios que foram utilizados para a avaliação dos intermediadores.

4.1 Aplicação de referência

Para a avaliação dos intermediadores de mensagens foi desenvolvida uma aplicação de referência. Esta aplicação foi desenvolvida na linguagem de programação C# (LIBERTY, 2005). Optou-se por esta linguagem de programação, uma vez que ela é suportada pelas ferramentas RabbitMQ, Qpid e ActiveMQ. Essa aplicação não apresenta uma interface gráfica e é composta por um produtor e um consumidor. Essa aplicação foi desenvolvida com funcionalidades que permitem utilizar todos os modelos de encaminhamento de mensagens. A Figura 11 apresenta a forma como a aplicação se comunica com o intermediador de mensagens.

Figura 11: Exemplo de aplicação produtora e consumidora



Fonte: elaborado pelo autor

O produtor apresenta por objetivo enviar mensagens ao intermediador de mensagens. Esse foi desenvolvido de forma flexível o suficiente para que seja possível enviar mensagens para todos os tipos de fila. Além disso, esse é tolerante a falhas de comunicação. Ou seja, no caso de uma falha na conexão, as mensagens serão reenviadas após o reestabelecimento da mesma. A partir de argumentos de entrada, passados por linha de comando, é possível definir: o IP, a porta, o *virtual host*, a quantidade de mensagens enviadas, o tipo de encaminhamento, a *routing key*, o cabeçalho (para encaminhamento por cabeçalho), o

envio persistido, o tamanho da mensagem e a quantidade de produtores (para teste de escalabilidade). Por fim, o produtor permite enviar múltiplas mensagens simultaneamente, possibilitando desta forma a realização de testes de escalabilidade.

Já o consumidor é uma aplicação que apresenta por objetivo se conectar a uma fila de mensagens e receber as mensagens que estão pendentes nesta fila. No caso de uma queda na conexão com o servidor, o consumidor reestabelece a conexão com o intermediador e continua recebendo as mensagens. Da mesma forma que no produtor, é possível definir: o IP, a porta, o *virtual host*, a fila de mensagem e a quantidade de consumidores (para teste de escalabilidade).

4.2 Critérios de Avaliação

Após a definição de uma meta, devem ser definidos os critérios que serão utilizados para a avaliação das alternativas. Desta forma, nas próximas seções serão apresentados os critérios que foram utilizados para avaliar as ferramentas que foram descritas na Seção 2.4. Esses critérios foram definidos considerando as principais características e necessidades de aplicações que utilizam processos assíncronos e que são executadas em nuvens computacionais do tipo SaaS.

4.2.1 Performance no produtor e consumidor

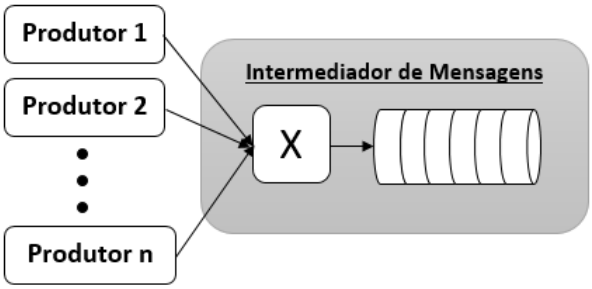
Este critério objetiva analisar o número de mensagens que são transmitidas (por segundo) por um produtor, e o número de mensagens que são recebidas (por segundo) por um consumidor. Para o produtor foram realizadas análises para todos os tipos de *exchange*, ou seja, foram realizados testes utilizando *exchanges* de encaminhamento direto, *fanout*, por tópicos e por cabeçalho. Para o consumidor foi realizada apenas uma análise, uma vez que o consumidor fica vinculado a uma fila e estas possuem o mesmo comportamento, independentemente do tipo de *exchange*. Ou seja, a performance do consumidor é independente do tipo de *exchange* utilizado.

Para realização dos testes foram consideradas mensagens de 64 bytes, 256 bytes, 1 KB e 4 KB. Esses tamanhos de mensagem foram baseados no trabalho de Marsh et al. (2008). Para cada um dos tamanhos de mensagem foram realizados 100 testes, sendo calculada a média e o desvio padrão. Em cada um dos 100 testes foram enviadas 50.000 mensagens.

4.2.2 Escalabilidade

A escalabilidade apresenta como principal objetivo avaliar o comportamento de um intermediador de mensagens perante o aumento no número de produtores e de consumidores. Nesta análise foram realizados os mesmos testes que foram descritos na Seção 4.2.1, porém com cenários que combinaram mais de um produtor e um consumidor. De fato, os testes foram realizados com 3, 5 e 10 produtores para cada tipo de *exchange* (Figura 12).

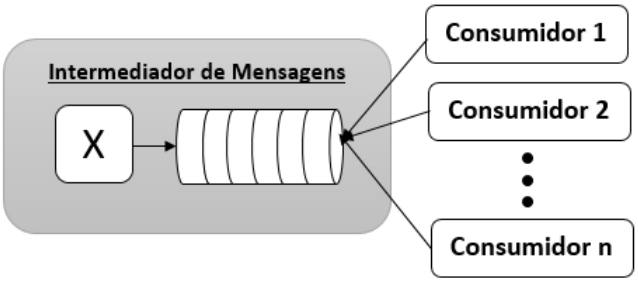
Figura 12: Exemplo da análise de escalabilidade no produtor



Fonte: elaborado pelo autor

Da mesma forma, foram realizados testes com 3, 5 e 10 consumidores, como pode ser observado na Figura 13. Os testes foram realizados utilizando um *exchange* do tipo direto, uma vez que o comportamento da fila independe do tipo do *exchange*.

Figura 13: Exemplo da analise de escalabilidade no consumidor

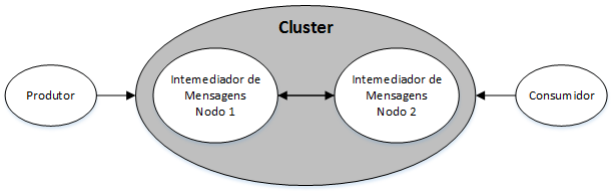


Fonte: elaborado pelo autor

4.2.3 Alta Disponibilidade

Garantir que o intermediador de mensagens se mantenha disponível e sem perda de mensagens é algo muito importante para grande parte das aplicações utilizadas em nuvens. Este critério tem como objetivo avaliar o comportamento do intermediador de mensagem perante a existência de falhas no servidor. Para tal, foi analisado os recursos existentes nos intermediadores para criação de um *cluster*[1] (Figura 14). Além disso, considerou-se as funcionalidades e o comportamento das ferramentas perante a perda de mensagens e uma parada do intermediador de mensagens.

Figura 14: Exemplo da analise de alta disponibilidade



Fonte: elaborado pelo autor

4.2.4 Disponibilidade de Documentação

Segundo Reis e Fortes (2003), somente 30% dos projetos de software livre apresentam uma documentação completa sendo que, somente 55% destes projetos revisam e mantém a documentação atualizada. Desta forma, nota-se que uma das maiores dificuldades na utilização de ferramentas de código aberto, consiste na existência de uma documentação atualizada e completa. Neste critério, foi considerada a quantidade de documentação disponível a partir de: livros publicados e disponíveis na Amazon.com (AMAZON, 2016); quantidade de tópicos existentes no fórum oficial das ferramentas; e a quantidade tópicos existentes no site *Stack Overflow* (STACKOVERFLOW, 2016). A partir desta avaliação será feita uma média para comparar tais resultados. Esse formato de avaliação foi baseado no trabalho de Barreto (2006).

[1] ***Cluster***: consiste em um conjunto de computadores conectados a partir de uma rede local que trabalham em conjunto e, muitas vezes, são vistos como um único sistema. Esses são utilizados para proporcionar alta disponibilidade e distribuição de carga computacional (BAKER, 2000).

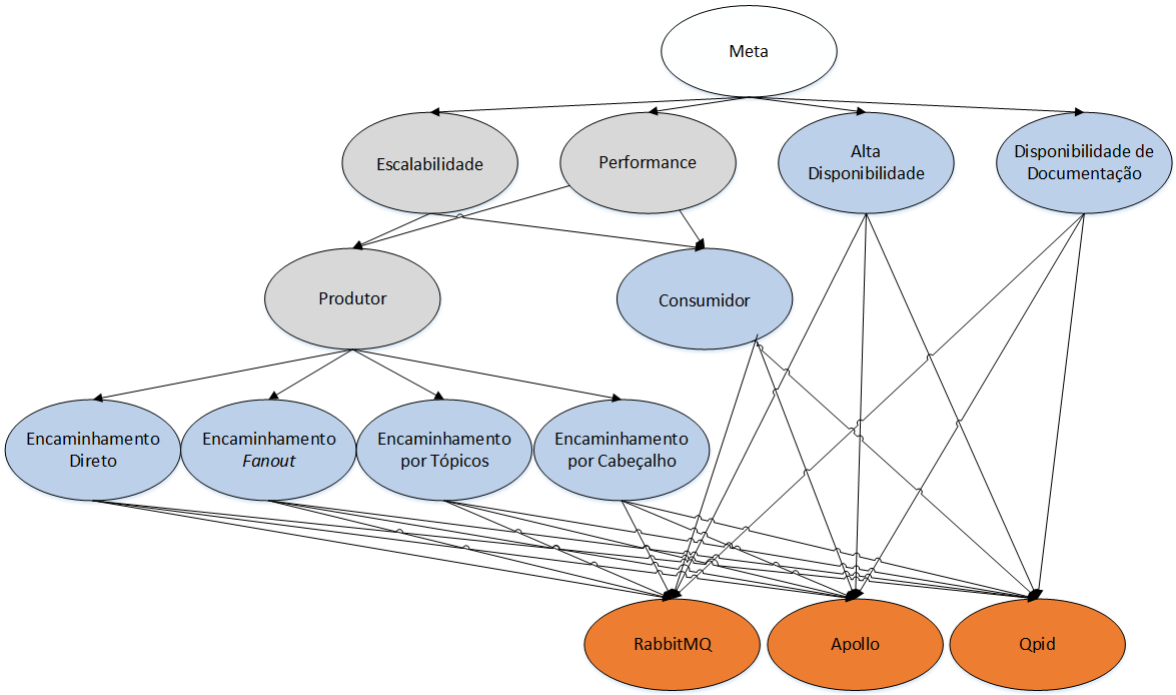
4.3 Aplicação do Método AHP no Problema Proposto

Para aplicação do método AHP, deve-se primeiramente definir a estrutura hierárquica dos critérios. Posteriormente, deve-se criar as matrizes de comparação paritária. Por fim, deve-se calcular o percentual médio local e a consistência para todas as matrizes. Nas próximas seções será apresentada a estrutura hierárquica criada neste trabalho. Além disso, será apresentado o processo de montagem das matrizes de comparação paritária, o cálculo do percentual médio local e a inconsistência das matrizes.

4.3.1 Estrutura Hierárquica Criada

Segundo Saaty (1980), uma hierarquia é uma abstração da estrutura de um sistema de forma a estudar as interações funcionais de seus componentes e seus impactos no sistema total. Na Figura 15, tem-se a hierarquia que foi desenvolvida a partir do objetivo, critérios e alternativas que foram definidas para avaliação dos intermediadores de mensagem. A partir da definição dos critérios, devem ser montadas as matrizes de comparação paritária. Nas próximas subseções serão calculados os pesos para os critérios e subcritérios dessa hierarquia. Por fim, serão calculados os percentuais médios locais das matrizes e o índice de consistência.

Figura 15: Hierarquia de critérios referente a meta proposta



Fonte: elaborado pelo autor

4.3.1.1 Matrizes de Comparação Paritária para os Critérios

Conforme a definição do método AHP, primeiramente devem ser definidas as importâncias dos critérios a partir da tabela de comparação paritária. Na Tabela 8 foram definidas as importâncias dos critérios. Como pode ser observado o critério "*Escalabilidade*" e o critério "*Performance*" apresentam uma maior representatividade. Isto deve-se ao comportamento de aplicações SaaS que concentram um grande número de usuários em uma mesma aplicação, possuindo assim, uma grande quantidade de requisições por minuto. Em seguida, o critério "*Alta Disponibilidade*" também se faz importante, uma vez que espera-se que o sistema não tenha seu funcionamento interrompido por falhas de hardware, software, conexão, etc.

Tabela 8: Tabela de avaliação dos critérios

	Escalabilidade	**Performance**	**Alta Disponibilidade**	**Disponibilidade de Documentação**
Escalabilidade	1	2	3	8
Performance	$\frac{1}{2}$	1	2	7
Alta Disponibilidade	$\frac{1}{3}$	$\frac{1}{2}$	1	5
Disponibilidade de Documentação	$\frac{1}{8}$	$\frac{1}{7}$	$\frac{1}{5}$	1

Após a criação da matriz de comparação paritária, esse deve ser normalizado calculando o vetor de prioridade local (PML) (Equação 4.1). Como pode ser observado, a "*Escalabilidade*" apresentou o maior percentual de representividade com 48,12%. Em seguida o critério "*Performance*" teve o percentual de representividade de 29,64%. Assim, esses critérios apresentarão uma maior representividade na comparação das ferramentas avaliadas. Já a "*Alta Disponibilidade*" apresenta um percentual de representatividade de 17,67% e a "*disponibilidade de documentação*" de 4,57%.

$$\begin{bmatrix} 0,511 & 0,549 & 0,474 & 0,421 \\ 0,255 & 0,275 & 0,316 & 0,368 \\ 0,170 & 0,137 & 0,158 & 0,158 \\ 0,064 & 0,039 & 0,053 & 0,053 \end{bmatrix} = \begin{bmatrix} \frac{0,511+0,549+0,484+0,381}{4} \\ \frac{0,255+0,275+0,323+0,333}{4} \\ \frac{0,170+0,137+0,161+0,238}{4} \\ \frac{0,064+0,039+0,032+0,048}{4} \end{bmatrix} = \overset{PMLValor}{\begin{bmatrix} 0,4812 \\ 0,2964 \\ 0,1767 \\ 0,0457 \end{bmatrix}} = \overset{PMLPercentual}{\begin{bmatrix} 48,12\% \\ 29,64\% \\ 17,67\% \\ 4,57\% \end{bmatrix}} \tag{4.1}$$

Após, foi calculada a consistência da matriz de comparação paritária. Observa-se na Equação 4.5, que a consistência foi de 2,1%. Portanto, uma vez que a matriz é de ordem 4 e a consistência obtida é inferior a 10% (Seção 3.1.3) pode-se concluir que a matriz de comparação paritária está consistente.

$$MCP * PML = \begin{bmatrix} 1 & 2 & 3 & 8 \\ \frac{1}{2} & 1 & 2 & 7 \\ \frac{1}{3} & \frac{1}{2} & 1 & 5 \\ \frac{1}{8} & \frac{1}{7} & \frac{1}{5} & 1 \end{bmatrix} * \begin{bmatrix} 0,4812 \\ 0,2964 \\ 0,1767 \\ 0,0457 \end{bmatrix} = \begin{bmatrix} 1,9696 \\ 1,2103 \\ 0,7138 \\ 0,1835 \end{bmatrix} \tag{4.2}$$

$$\lambda_{max} = \begin{bmatrix} 1,9696 \\ 1,2103 \\ 0,7138 \\ 0,1835 \end{bmatrix} / \begin{bmatrix} 0,4812 \\ 0,2964 \\ 0,1767 \\ 0,0457 \end{bmatrix} = \begin{bmatrix} 4,0940 \\ 4,0833 \\ 4,0396 \\ 4,0153 \end{bmatrix} = \frac{4,0940 + 4,0833 + 4,0396 + 4,0153}{4} = 4,058 \tag{4.3}$$

$$IC = \frac{4,058 - 4}{3} = 0,019 \tag{4.4}$$

$$RC = \frac{0,019}{0,9} = 0,021 = \mathbf{2,1\%} \tag{4.5}$$

4.3.1.2 Matrizes de Comparação Paritária para os Subcritérios Produtor e Consumidor

Como pode ser observado na Figura 15, os critérios "*Performance*" e "*Escalabilidade*" foram subdivididos nos subcritérios: "*Produtor*" e "*Consumidor*". Desta forma, é necessário criar a matriz de comparação paritária para esses subcritérios, bem como deve ser calculado o vetor de prioridade local. Neste caso não é necessário calcular a inconsistência, uma vez que a matriz apresenta uma ordem inferior a 3.

Na Tabela 9 pode-se observar que o critério "*Produtor*" apresenta uma maior representatividade sobre o critério "*Consumidor*", pois em aplicações de grande escala, frequentemente, tem-se a necessidade de uma maior eficiência no envio de mensagens. Já o recebimento não apresenta a mesma necessidade devido a característica do uso assíncrono de mensagens.

Tabela 9: Tabela de avaliação dos subcritérios

	Consumidor	**Produtor**
Consumidor	1	$\frac{1}{2}$
Produtor	2	1

Em seguida, a matriz de comparações paritárias foi normalizada e calculado o vetor de prioridade local (PML) (Equação 4.6). Como pode ser observado, o "*Produtor*" teve o percentual de representatividade de 66,67% e o "*Consumidor*" o percentual de 33,33%.

$$\begin{bmatrix} 0,333 & 0,333 \\ 0,667 & 0,667 \end{bmatrix} = \begin{bmatrix} \frac{0,333+0,333}{2} \\ \frac{0,667+0,667}{2} \end{bmatrix} = \overset{PMLValor}{\begin{bmatrix} 0,3333 \\ 0,6667 \end{bmatrix}} = \overset{PMLPercentual}{\begin{bmatrix} 33,33\% 66,67\% \end{bmatrix}} \tag{4.6}$$

4.3.1.3 Matrizes de Comparação Paritária para os roteamentos do produtor

Os critérios "*Performance*" e "*Escalabilidade*" quando analisados na parte do produtor (Figura 15), foram subdivididos nos subcritérios: "*Encaminhamento Direto*", "*Encaminhamento Fanout*", "*Encaminhamento por Tópicos*" e "*Encaminhamento por Cabeçalho*". Desta forma, é necessário criar a matriz de comparação paritárias para esses subcritérios, bem como deve ser calculado o vetor de prioridade local e sua inconsistência.

Para esses subcritérios, foi gerada a Tabela 10 onde foram definidas as importâncias de cada tipo de encaminhamento. Como pode-se observar, os critérios "*Encaminhamento direto*" e "*Encaminhamento por Fanout*" apresentam uma importância maior perante os demais formatos de encaminhamento. Isso deve-se ao fato das aplicações SaaS necessitarem, na maior parte dos casos, enviar dados a somente uma fila de mensagens ou enviar para todas as filas (*broadcasting*). Já o uso de "*Encaminhamento por Tópicos*" e "*Encaminhamento por cabeçalhos*" não é tão comum neste tipo de aplicações.

Tabela 10: Tabela de avaliação dos subcritérios

	Direto	**Fanout**	**Tópicos**	**Cabeçalho**
Direto	1	3	6	6
Fanout	$\frac{1}{3}$	1	3	3
Tópicos	$\frac{1}{6}$	$\frac{1}{3}$	1	1
Cabeçalho	$\frac{1}{6}$	$\frac{1}{3}$	1	1

Posteriormente, a matriz de comparações paritárias foi normalizada e calculado o vetor de prioridade local (PML) (Equação 4.7). Como pode ser observado, o encaminhamento direto teve o percentual de representatividade de 58,34%, o encaminhamento *fanout* de 23,99%, o encaminhamento por tópicos de 8,83% e o encaminhamento por cabeçalho de 8,83%.

$$\begin{bmatrix} 0,600 & 0,643 & 0,545 & 0,545 \\ 0,200 & 0,214 & 0,273 & 0,273 \\ 0,100 & 0,071 & 0,091 & 0,091 \\ 0,100 & 0,071 & 0,091 & 0,091 \end{bmatrix} = \begin{bmatrix} \frac{0,600+0,643+0,545+0,545}{4} \\ \frac{0,200+0,214+0,273+0,273}{4} \\ \frac{0,100+0,071+0,091+0,091}{4} \\ \frac{0,100+0,071+0,091+0,091}{4} \end{bmatrix} = \overset{PMLValor}{\begin{bmatrix} 0,5834 \\ 0,2399 \\ 0,0883 \\ 0,0883 \end{bmatrix}} = \overset{PMLPercentual}{\begin{bmatrix} 58,34\% \\ 23,99\% \\ 8,83\% \\ 8,83\% \end{bmatrix}} \tag{4.7}$$

Por fim, foi calculada a consistência da matriz de comparação paritária. Observa-se na Equação 4.11, que a consistência foi de 0,8%. Portanto, uma vez que a matriz de comparação paritária é de ordem 4 e a consistência obtida foi inferior a 10% (Seção 3.1.3), pode-se concluir que a matriz está consistente.

$$MCP * PML = \begin{bmatrix} 1 & 3 & 6 & 6 \\ \frac{1}{3} & 1 & 3 & 3 \\ \frac{1}{6} & \frac{1}{3} & 1 & 1 \\ \frac{1}{6} & \frac{1}{3} & 1 & 1 \end{bmatrix} * \begin{bmatrix} 0,5834 \\ 0,2399 \\ 0,0883 \\ 0,0883 \end{bmatrix} = \begin{bmatrix} 2,363 \\ 0,964 \\ 0,354 \\ 0,354 \end{bmatrix} \tag{4.8}$$

$$\lambda_{max} = \begin{bmatrix} 2,363 \\ 0,964 \\ 0,354 \\ 0,354 \end{bmatrix} / \begin{bmatrix} 0,5834 \\ 0,2399 \\ 0,0883 \\ 0,0883 \end{bmatrix} = \begin{bmatrix} 4,050 \\ 4,018 \\ 4,009 \\ 4,009 \end{bmatrix} = \frac{4,050 + 4,018 + 4,009 + 4,009}{4} = 4,022 \tag{4.9}$$

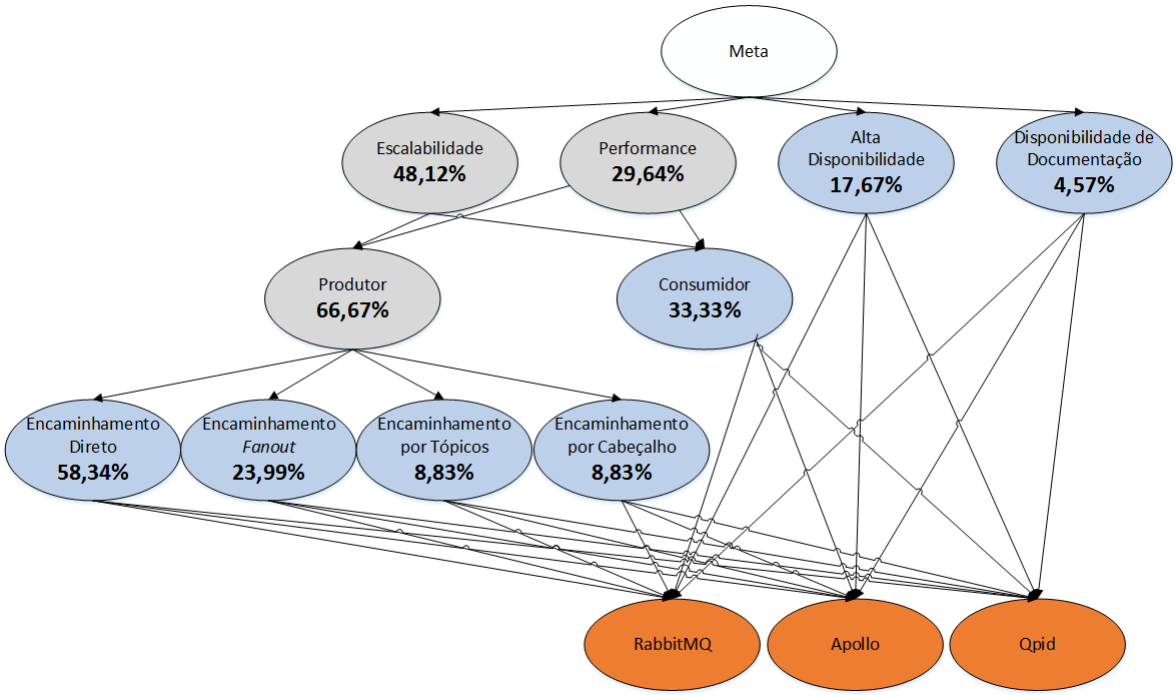
$$IC = \frac{4,022 - 4}{3} = 0,007 \tag{4.10}$$

$$RC = \frac{0,007}{0,9} = 0,008 = \mathbf{0,8\%} \tag{4.11}$$

4.3.2 Estrutura Hierárquica com os Pesos

Na Figura 16 tem-se a estrutura hierárquica com os pesos distribuídos de acordo com o percentual local de importância de cada um dos critérios.

Figura 16: Hierarquia de critérios com pesos definidos



Fonte: elaborado pelo autor

A seguir foi calculado o percentual médio global para os valores, onde pode-se observar que:

- Os subcritérios "*Produtor*" e "*Consumidor*" localizados abaixo do critério "*Escalabilidade*", possuem uma representatividade de 32,08% e 16,04%, respectivamente. Esses percentuais correspondem a representatividade desses subcritérios sobre o total, uma vez que esses representam 66,67% e 33,33% do seu critério superior.

- Os subcritérios "*Produtor*" e "*Consumidor*", localizados abaixo do critério "*Performance*", possuem uma representatividade global de 19,76% e 9,88%, respectivamente, uma vez que esses representam 66,67% e 33,33% do seu critério superior.

- Os subcritérios "*Encaminhamento Direto*", "*Encaminhamento Fanout*", "*Encaminhamento por Tópicos*"e "*Encaminhamento por Cabeçalho*", possuem um vetor de prioridade local de 58,34%, 23,99%, 8,83% e 8,83%, respectivamente. Considerando o total, as suas representatividades globais são de 18,73%, 7,70%, 2,83% e 2,83%. Quando consideramos o critério *Produtor*" que está abaixo do critério "*Performance*" suas representatividades globais são de 11,53%, 4,74%, 1,74% e 1,74%.

- O subcritério "*Consumidor*", que encontra-se abaixo dos critérios "*escalabilidade*" e "*performance*", possui uma representatividade global de 16,04% e 9,88%, respectivamente, sobre o total.

A Tabela 11 apresenta um resumo com os percentuais globais dos critérios que serão avaliados nos testes dos intermediadores.

Tabela 11: Tabela com os percentuais globais para os critérios avaliados

		Percentual Global
Performance	Encaminhamento Direto	11,53%
	Encaminhamento Fanout	4,74%
	Encaminhamento por Tópicos	1,74%
	Encaminhamento por Cabeçalhos	1,74%
	Consumidor	9,88%
Escalabilidade	Encaminhamento Direto	18,73%
	Encaminhamento Fanout	7,70%
	Encaminhamento por Tópicos	2,83%
	Encaminhamento por Cabeçalhos	2,83%
	Consumidor	16,04%
Documentação		4,57%
Alta Disponibilidade		17,67%

4.4 Considerações

Neste capítulo, foi apresentada a aplicação do método AHP que será utilizado para avaliar os intermediadores de mensagem. Na Seção 4.2 foram descritos os critérios que foram utilizados para a avaliação dos intermediadores. Já na Seção 4.3 foi apresentada a estrutura hierárquica e foram definidos os pesos de cada um dos critérios. No próximo capítulo serão apresentados os testes e os resultados obtidos.

5 APLICAÇÃO DESENVOLVIDA E TESTES REALIZADOS

Neste capítulo será apresentada uma breve descrição sobre a aplicação desenvolvida. Além disso, serão apresentados os resultados obtidos com os intermediadores.

5.1 Aplicação desenvolvida

A aplicação foi desenvolvida utilizando a linguagem de programação C#, na versão 4.5 do *framework .NET* (MICROSOFT, 2016b). Para a realização dos testes de escalabilidade, ou seja, para a execução de mais de um produtor e um consumidor, foram utilizadas *threads*[1]. Para a utilização dos intermediadores de mensagem RabbitMQ e QPID foi utilizada a biblioteca *RabbitMQ.Client* versão 3.6.1.0 (PIVOTAL, 2016), que implementa a versão 0.9.1 do protocolo AMQP. Já para a utilização do intermediador ActiveMQ, foi utilizada a biblioteca *AMQPNetLite* na versão 1.1.8 (AMQP.NET LITE, 2016). Não foi possível utilizar a biblioteca *RabbitMQ.Client*, pois esse intermediador não possui suporte a versão 0.9.1 do protocolo AMQP. O código fonte completo da aplicação desenvolvida, encontra-se no CD-ROM anexado a este trabalho.

A aplicação desenvolvida é formada basicamente por 3 partes, que são: conexão, envio, recebimento. Observa-se na Figura 17 o estabelecimento da conexão com o intermediador de mensagens. Essa etapa é composta pelo método responsável por abrir a conexão (*CreateConnection* - linha 7) e o método responsável por estabelecer uma sessão (*CreateModel* - linha 8).

Figura 17: Trecho da aplicação responsável pelo estabelecimento da conexão

```
1 var factory = new ConnectionFactory();
2 factory.VirtualHost = _virtualHost;
3 factory.Protocol = Protocols.DefaultProtocol;
4 factory.HostName = _ipAddress;
5 factory.Port = _port;
6 factory.ContinuationTimeout = new TimeSpan(1, 0, 0);
7 this._connection = factory.CreateConnection();
8 this._channel = _connection.CreateModel();
```

Fonte: elaborado pelo autor

[1] ***Threads***: são fluxos de execução que executam dentro de um processo. Essas são utilizadas para que uma aplicação tenha condições de executar trechos de código de forma concorrente (TANENBAUM; BOS, 2014).

Na Figura 18 é apresentado o processo de envio de uma mensagem para todos os tipos de *exchanges*. Na linha 6 tem-se o método *BasicPublish* que é responsável pelo envio de mensagens. O parâmetro *exchange* corresponde ao *exchange* responsável por encaminhar uma mensagem para a fila desejada. O parâmetro *routingKey*, será avaliado pelo *exchange* no envio de mensagens do tipo: direto, *fanout* e por tópicos. Já o atributo *Headers*, localizado na linha 5, será avaliado pelo *exchange* no envio de mensagens do tipo de encaminhamento por cabeçalho. Para maiores informações sobre o envio de mensagens sugere-se a documentação da biblioteca *RabbitMQ.Client* (PIVOTAL, 2016).

Figura 18: Trecho da aplicação responsável pelo envio de mensagens

```
1 byte[] body = Encoding.UTF8.GetBytes("Mensagem");
2 var properties = _channel.CreateBasicProperties();
3 properties.DeliveryMode = 2; //Persistente
4 properties.ContentType = "text/plain";
5 properties.Headers = headerAttributes;
6 _channel.BasicPublish(exchange: exchange, routingKey: routingkey, basicProperties: properties, body: body);
```

Fonte: elaborado pelo autor

Por fim, na Figura 19, é apresentado o recebimento de mensagens. O recebimento de uma mensagem é efetuado através do método *BasicGet* (linha 2) e a confirmação do recebimento da mensagem através do método *BasicAck* (linha 5).

Figura 19: Trecho da aplicação responsável pelo recebimento de mensagens

```
1 _channel.BasicQos(prefetchSize: 0, prefetchCount: 1, global: false);
2 BasicGetResult result = _channel.BasicGet(queueName, false);
3 var message = Encoding.UTF8.GetString(result.Body);
4 //Confirma o recebimento da mensagem
5 _channel.BasicAck(result.DeliveryTag, false);
```

Fonte: elaborado pelo autor

5.2 Testes Realizados

Os testes foram realizados em um computador com processador Intel I7-3610QM com 8 núcleos de 3.3 GHz. Esse computador possui ainda, 8 GB de memória RAM (DDR3 de 800Mhz), um disco rígido *Solid State Disk* (SSD) de 250 GB e o sistema operacional Microsoft Windows 10.

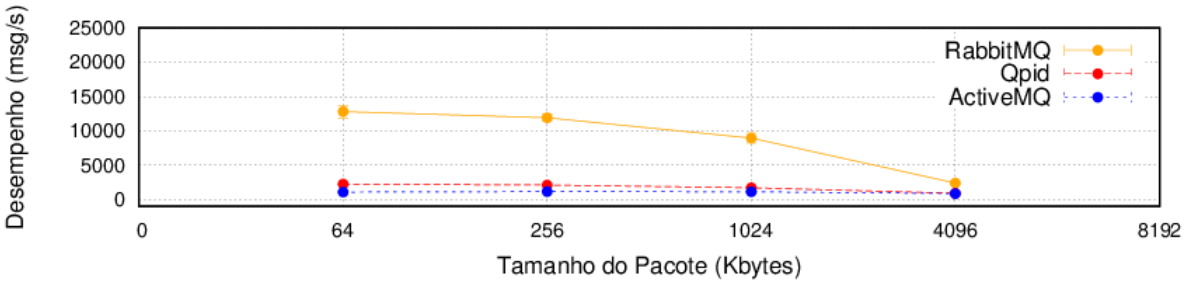
Para a execução dos intermediadores de mensagem e da aplicação de referência não foi utilizado o sistema operacional da máquina hospedeira. De fato, para a instalação e execução destes foram criadas máquinas virtuais utilizando o software de virtualização VMWare Player (VMWARE, 2016).

A máquina virtual utilizada para o intermediador de mensagens foi configurada com um processador Intel I7-3610QM com 3 núcleos de 3.3 GHz, 4 GB de memória RAM e disco rígido de 40 GB. O sistema operacional utilizado foi o Linux na distribuição Fedora 23 com arquitetura de 64 bits (*Kernel* versão 4.2). Já a aplicação de referência foi executada em uma máquina virtual com um processador Intel I7-3610QM com 3 núcleos de 3.3 GHz, 3 GB de memória RAM, disco rígido de 40 GB e o sistema operacional Windows 7 com uma arquitetura de 32 bits.

5.2.1 Teste de performance no encaminhamento direto

Na Figura 20 tem-se um gráfico apresentando o número de pacotes enviados por segundo em função do tamanho do pacote para os intermediadores RabbitMQ, QPID e ActiveMQ, utilizando um encaminhamento direto.

Figura 20: Teste de performance para o encaminhamento direto

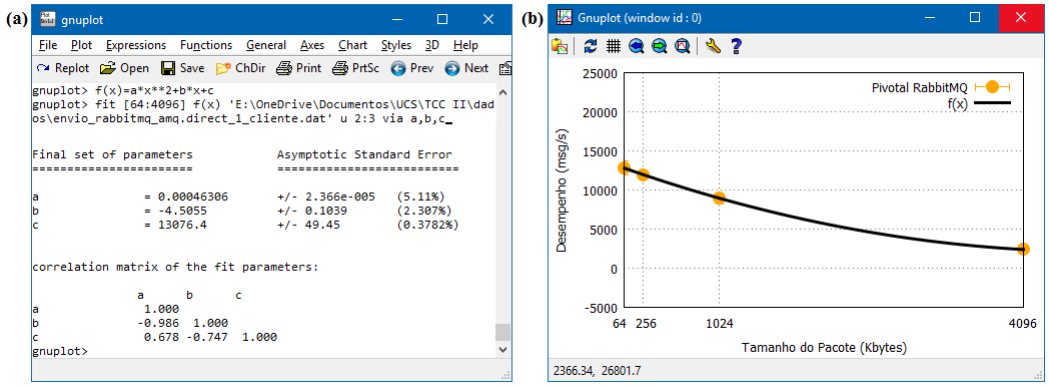


Fonte: elaborado pelo autor

Para a comparação da performance dos intermediadores utilizando o método AHP, torna-se necessário converter esses valores em um valor único numérico. Para tanto, optou-se por calcular a área da função obtida a partir do ajuste de um polinômio de segundo grau aos valores de número de pacotes enviados em função do tamanho. Para esse ajuste foi utilizada a função "*fit*" implementada no software Gnuplot versão 4.6 (GNUPLOT, 2016). Esta função utiliza o algoritmo de *Levenberg–Marquardt* (KELLEY, 1999) para o ajuste.

A Figura 21 (a) apresenta os valores dos coeficientes a,b e c, que foram obtidos através do ajuste do polinômio de segundo grau aos valores de números de pacotes em função do tamanho do pacote para o intermediador RabbitMQ. Como pode ser observado os valores de a,b e c obtidos foram de 0,00046, -4,45 e 13076,4, respectivamente. Já na Figura 21 (b) tem-se o gráfico da função obtida a partir desse ajuste.

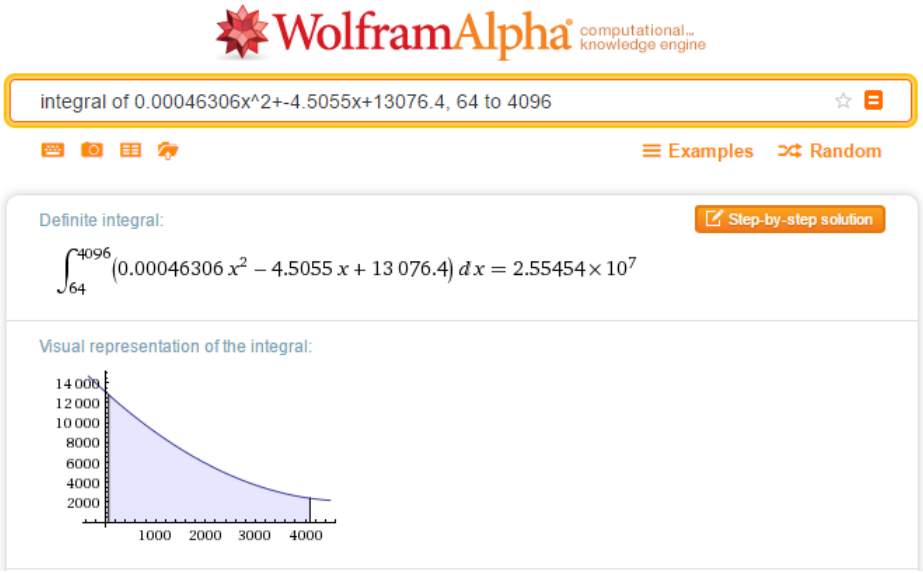
Figura 21: Ajuste de um polinômio de segundo grau aos valores de número de pacotes enviados em função do tamanho do pacote para o intermediador RabbitMQ



Fonte: elaborado pelo autor

Após, foi calculada a área total abaixo da curva gerada pela equação através do cálculo da integral definida (GUIDORIZZI, 2000). O cálculo da integral foi realizado no intervalo de 64 bytes a 4 KB, que representam o menor e o maior tamanho de pacote, respectivamente. Para o cálculo da integral utilizou-se o *website* WolframAlpha (WOLFRAMALPHA, 2016). Na Figura 22 tem-se o exemplo do cálculo da integral utilizado o *website* WolframAlpha. Como pode ser observado, para o intermediador RabbitMQ foi obtida uma área de $2,55454 * 10^7$.

Figura 22: Cálculo da integral definida no intervalo de 64 bytes até 4 KB



Fonte: elaborado pelo autor

Esse mesmo procedimento foi realizado para os intermediadores RabbitMQ, Qpid e ActiveMQ, sendo que os valores das áreas calculadas foram de $2,55454 * 10^7$, $5,5372 * 10^6$ e $4,3202 * 10^6$, respectivamente. Posteriormente, esses valores foram normalizados a partir da divisão entre o valor da área para cada intermediador, pela soma total das áreas. Esse procedimento foi adotado baseando-se nos trabalhos de Saaty e Vargas (2012), Yang, Chu e Chouhuang (2004), Besteiro et al. (2009) e Whitaker (2007). De acordo com esses trabalhos quando temos um valor numérico não se faz necessário utilizar a escala de Saaty, uma vez que esses já possuem uma representatividade perante aos demais. Na Tabela 12 pode-se observar que o intermediador RabbitMQ possui uma representatividade de 72,16%. Já o intermediador Qpid possui uma representatividade de 15,64% e o intermediador ActiveMQ de 12,20%.

Tabela 12: Matriz de representatividade do teste de performance para um encaminhamento direto

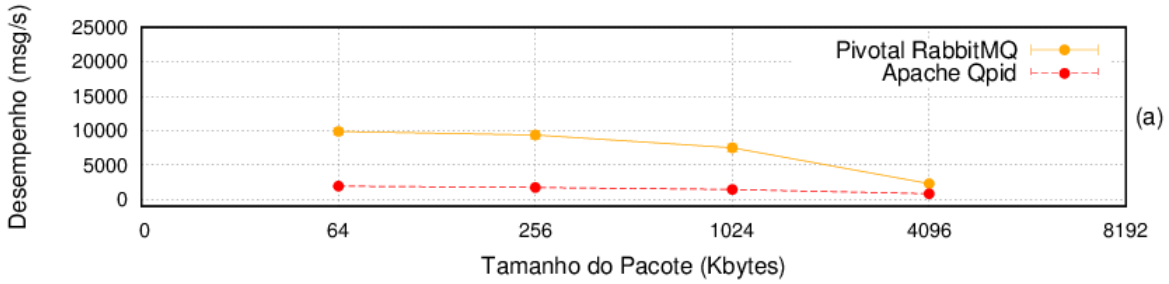
Intermediador	Área	Perc
RabbitMQ	$2,5545 * 10^7$	**72,16%**
Qpid	$5,5372 * 10^6$	**15,64%**
ActiveMQ	$4,3202 * 10^6$	**12,20%**

Esse procedimento foi adotado nos demais testes de performance. Desta forma nas próximas seções serão apresentados somente o gráfico com o número de pacotes em função do tamanho de pacote e as matrizes de representatividade.

5.2.2 Teste de performance no encaminhamento *fanout*

Na Figura 23 tem-se um gráfico apresentando o número de pacotes enviados em função do tamanho do pacote para os intermediadores RabbitMQ e QPID, utilizando um encaminhamento *fanout*. Não foram realizados testes com o intermediador ActiveMQ, uma vez que essa não possui suporte para este tipo de encaminhamento (ACTIVEMQ, 2015).

Figura 23: Teste de performance para o encaminhamento *fanout*



Fonte: elaborado pelo autor

As áreas calculadas foram de $2,2433 * 10^7$ e $4,7419 * 10^6$ para os intermediadores RabbitMQ e Qpid, respectivamente. Para o intermediador ActiveMQ assumiu-se uma área de 0, uma vez que não foram realizados testes com o mesmo. Como pode ser observado na Tabela 13, os percentuais de representatividade obtidos foram de 83,14%, 16,86% e 0%, para os intermediadores RabbitMQ, Qpid e ActiveMQ, respectivamente.

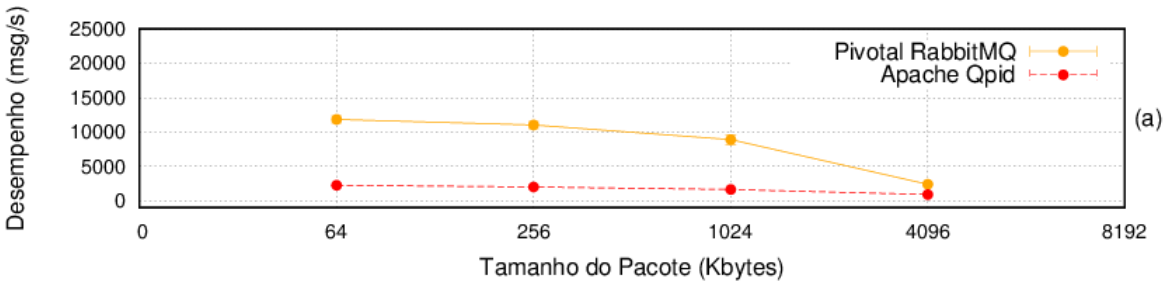
Tabela 13: Matriz de representatividade do teste de performance para um encaminhamento *fanout*

Intermediador	Área	Perc
RabbitMQ	$2,2433 * 10^7$	**83,14%**
Qpid	$4,7419 * 10^6$	**16,86%**
ActiveMQ	0	**0%**

5.2.3 Teste de performance no encaminhamento por tópicos

Na Figura 24 tem-se um gráfico apresentando o número de pacotes enviados em função do tamanho do pacote para os intermediadores RabbitMQ e QPID, utilizando um encaminhamento por tópicos. Não foram realizados teste com o intermediador ActiveMQ, uma vez que esse também não possui suporte para este tipo de encaminhamento (ACTIVEMQ, 2015).

Figura 24: Teste de performance para o encaminhamento por tópicos



Fonte: elaborado pelo autor

Neste caso, as áreas calculadas foram de $2,6132 * 10^7$ e $5,2976 * 10^6$ para os intermediadores RabbitMQ e Qpid, respectivamente. Para o intermediador ActiveMQ assumiu-se uma área de 0. Como pode ser observado na Tabela 14, os percentuais de representatividade obtidos foram de 82,55%, 17,45% e 0%, para os intermediadores RabbitMQ, Qpid e ActiveMQ, respectivamente.

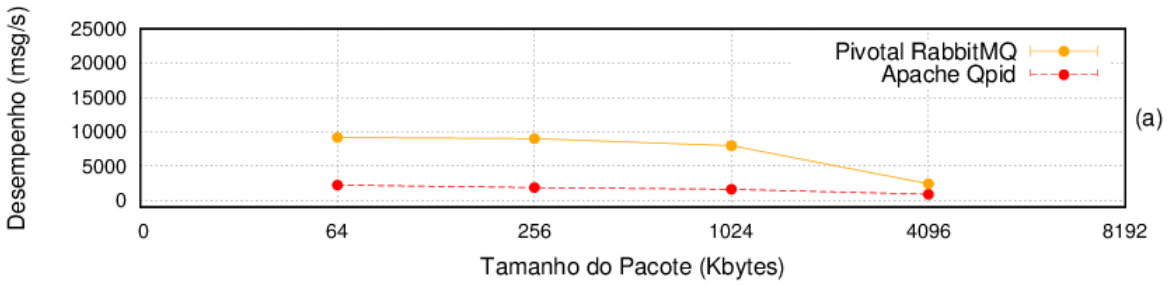
Tabela 14: Matriz de representatividade do teste de performance para um encaminhamento por tópicos

Intermediador	Área	Perc
RabbitMQ	$2,6132 * 10^7$	**82,55%**
Qpid	$5,2976 * 10^6$	**17,45%**
ActiveMQ	0	**0%**

5.2.4 Teste de performance no encaminhamento por cabeçalho

Na Figura 25 tem-se um gráfico apresentando o número de pacotes enviados em função do tamanho do pacote para os intermediadores RabbitMQ e QPID, utilizando um encaminhamento por cabeçalho. Não foram realizados teste com o intermediador ActiveMQ, uma vez que essa não possui suporte para este tipo de encaminhamento (ACTIVEMQ, 2015).

Figura 25: Teste de performance para o encaminhamento por cabeçalhos



Fonte: elaborado pelo autor

As áreas calculadas foram de $2,4901 * 10^7$ e $5,1175 * 10^6$ para os intermediadores RabbitMQ e Qpid, respectivamente. Para o intermediador ActiveMQ assumiu-se uma área de 0. Como pode ser observado na Tabela 15, os percentuais de representatividade obtidos foram de 82,95%, 17,05% e 0%, para os intermediadores RabbitMQ, Qpid e ActiveMQ, respectivamente.

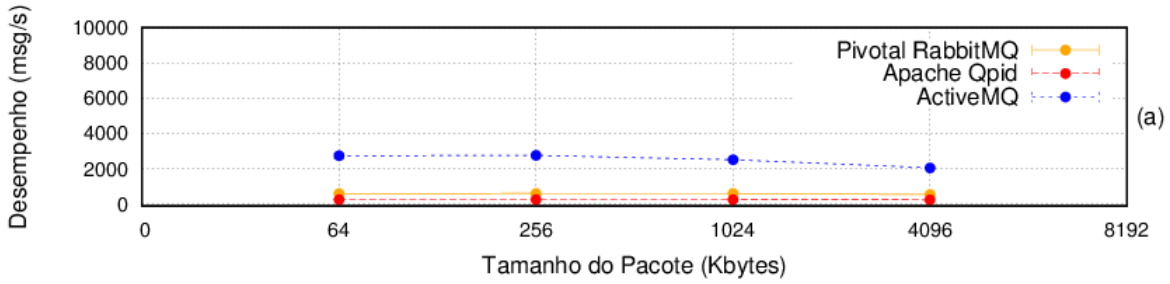
Tabela 15: Matriz de representatividade do teste de performance para um encaminhamento por cabeçalhos

Intermediador	Área	Perc
RabbitMQ	$2,4901 * 10^7$	**82,95%**
Qpid	$5,1175 * 10^6$	**17,05%**
ActiveMQ	0	**0%**

5.2.5 Teste de performance no consumidor

Na Figura 26 tem-se um gráfico apresentando o número de pacotes recebidos em função do tamanho do pacote para os intermediadores RabbitMQ, QPID e ActiveMQ.

Figura 26: Teste de performance para o recebimento



Fonte: elaborado pelo autor

Para esses testes, as áreas calculadas foram de $2,4061 * 10^6$, $1,0309 * 10^6$, $9,4759 * 10^6$ para os intermediadores RabbitMQ, Qpid e ActiveMQ, respectivamente. Como pode ser observado na Tabela 16, os percentuais de representatividade obtidos foram de 18,64%, 7,98% e 73,38%, para os intermediadores RabbitMQ, Qpid e ActiveMQ.

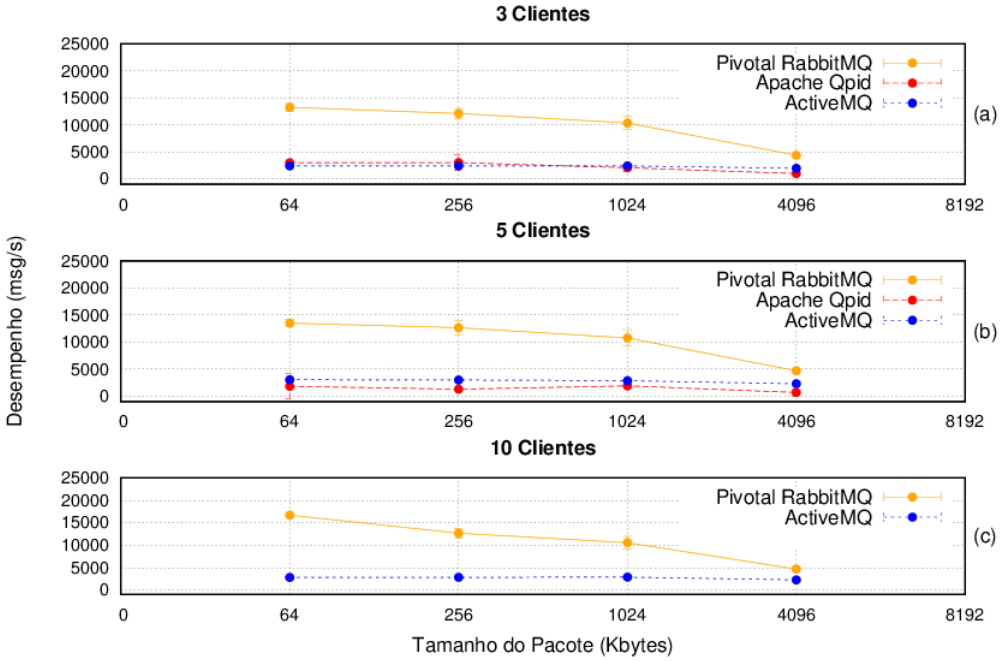
Tabela 16: Matriz de representatividade do teste de performance para o recebimento de mensagens

Intermediador	Área	Perc
RabbitMQ	$2,4061 * 10^6$	**18,64%**
Qpid	$1,0309 * 10^6$	**7,98%**
ActiveMQ	$9,4759 * 10^6$	**73,38%**

5.2.6 Teste de escalabilidade no encaminhamento direto

Na Figura 27 tem-se os gráficos que apresentam os resultados dos testes de escalabilidade que foram realizados com 3, 5 e 10 produtores para um encaminhamento direto. Ou seja, esta figura apresenta o número de pacotes enviados em função do tamanho do pacote para os intermediadores RabbitMQ, QPID e ActiveMQ nos três casos. Não foi possível realizar os testes com 10 produtores para o intermediador Qpid, pois esse apresentou uma sobrecarga e falhou.

Figura 27: Teste de escalabilidade para o encaminhamento direto



Fonte: elaborado pelo autor

Para a comparação de escalabilidade dos intermediadores utilizando o método AHP, foi utilizado o mesmo procedimento adotado para as análises anteriores. Porém neste caso, pela existência de 3 gráficos, a normalização foi realizada para cada caso e ao final foi realizada a média dos mesmos. Essa abordagem foi baseada nos trabalhos de Whitaker (2007) e Saaty e Vargas (2012).

Na Tabela 17 (a), observa-se as áreas obtidas para cada um dos testes e o percentual resultante da normalização. Para o Qpid, no teste com 10 produtores, assumiu-se uma área de 0. Na Tabela 17 (b), tem-se os percentuais de representatividade finais que foram de 66,47%, 8,52% e 22,01%, para os intermediadores RabbitMQ, Qpid e ActiveMQ, respectivamente.

Tabela 17: Matrizes de representatividade dos testes de escalabilidade para um encaminhamento direto

	3 Clientes		5 Clientes		10 Clientes	
Intermediador	Área	Perc.	Área	Perc.	Área	Perc.
RabbitMQ	$3,2628 * 10^7$	67,29%	$3,4262 * 10^7$	67,01%	$3,0537 * 10^7$	74,12%
Qpid	$6,5454 * 10^6$	13,50%	$6,1735 * 10^6$	12,07%	0	0%
ActiveMQ	$9,3154 * 10^6$	19,21%	$1,0697 * 10^7$	20,92%	$1,0665 * 10^7$	25,88%

(a)

Intermediador	Resultado
RabbitMQ	**69,47%**
Qpid	**8,52%**
ActiveMQ	**22,01%**

(b)

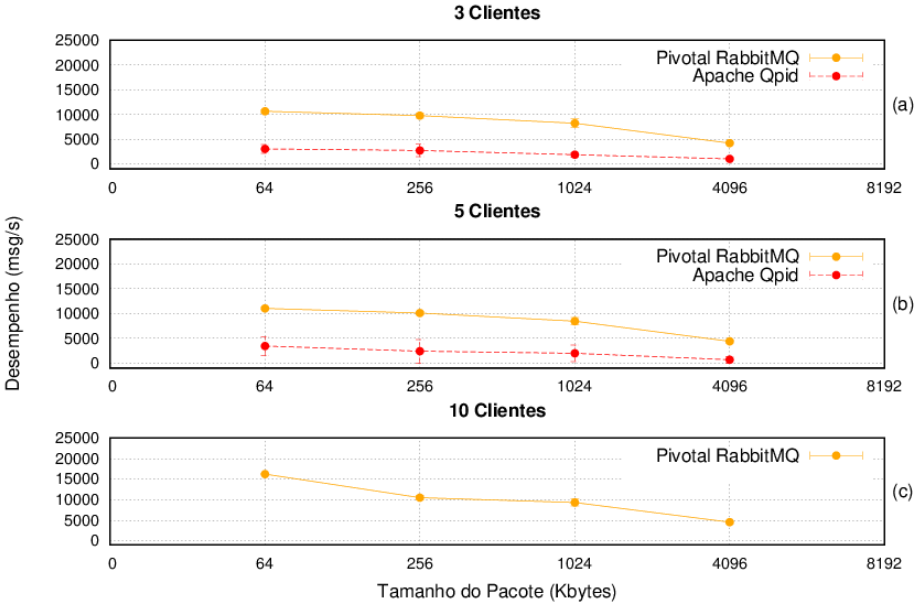
Destaca-se que esse procedimento foi adotado em todos os testes de escalabilidade. Desta forma, nas próxima seções serão apresentados somente os gráficos com o número

de pacotes enviados (ou recebidos) em função do tamanho de pacote e as matrizes de representatividade.

5.2.7 Teste de escalabilidade no encaminhamento *fanout*

Na Figura 20 tem-se um gráfico apresentando o número de pacotes enviados em função do tamanho do pacote para os intermediadores RabbitMQ e QPID, para um encaminhamento *fanout*. Essa figura apresenta os gráficos com os resultados obtidos considerando 3, 5 e 10 produtores. Não foram realizados teste com o intermediador ActiveMQ, uma vez que essa não possui suporte para este tipo de encaminhamento (ACTIVEMQ, 2015). Além disso, não foi possível realizar os testes com 10 produtores para o intermediador Qpid, uma vez que esse apresentou uma sobrecarga e falhou.

Figura 28: Teste de escalabilidade para o encaminhamento *fanout*



Fonte: elaborado pelo autor

Na Tabela 18 (a), observa-se as áreas obtidas para cada um dos testes e o percentual resultante da normalização. Para o intermediador ActiveMQ e para o Qpid (com 10 produtores), assumiu-se uma área de 0. Na Tabela 18 (b), tem-se as representatividades calculadas, sendo que os valores obtidos foram de 88,84%, 11,16% e 0%, para os intermediadores RabbitMQ, Qpid e ActiveMQ, respectivamente.

Tabela 18: Matrizes de representatividade dos testes de escalabilidade para um encaminhamento *fanout*

	3 Clientes		5 Clientes		10 Clientes	
Intermediador	Área	Perc.	Área	Perc.	Área	Perc.
RabbitMQ	$2,6707 * 10^7$	82,47%	$2,7366 * 10^7$	84,05%	$2,5818 * 10^7$	100,00%
Qpid	$5,6779 * 10^6$	17,53%	$5,1935 * 10^6$	15,95%	0	0%
ActiveMQ	0	0%	0	0%	0	0%

(a)

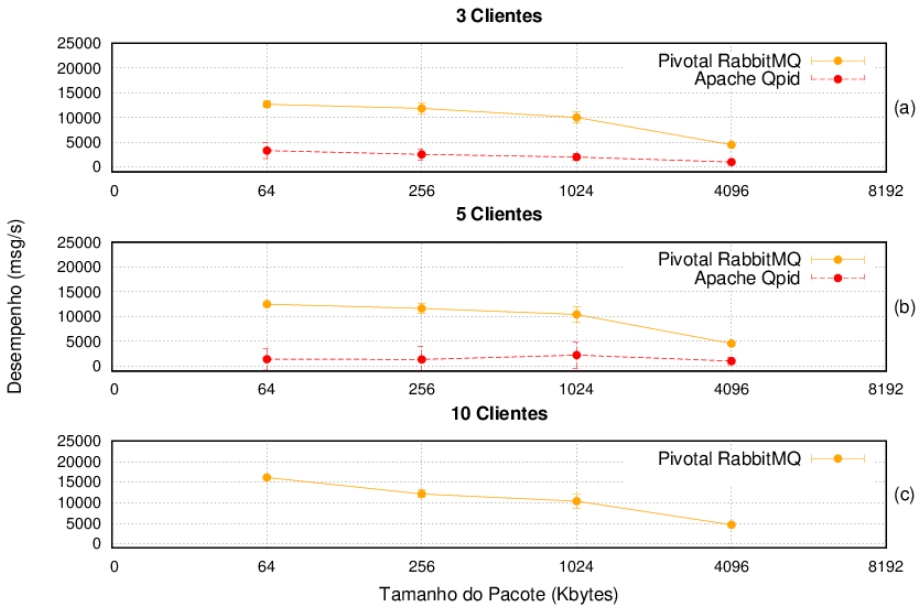
Intermediador	Resultado
RabbitMQ	**88,84%**
Qpid	**11,16%**
ActiveMQ	**0%**

(b)

5.2.8 Teste de escalabilidade no encaminhamento por tópicos

Na Figura 20 tem-se um gráfico apresentando o número de pacotes enviados em função do tamanho do pacote para os intermediadores RabbitMQ e QPID, utilizando um encaminhamento por tópicos. Essa figura apresenta os resultados considerando 3, 5 e 10 produtores. Não foram realizados testes com o intermediador ActiveMQ, uma vez que essa não possui suporte para este tipo de encaminhamento (ACTIVEMQ, 2015). Além disso, não foi possível realizar testes com 10 produtores para o intermediador Qpid. De fato, nestes testes ocorreu uma sobrecarga no intermediador e esse voltou a falhar.

Figura 29: Teste de escalabilidade para o encaminhamento por tópicos



Fonte: elaborado pelo autor

Na Tabela 19 (a), observa-se as áreas obtidas para cada um dos testes e o percentual resultante da normalização. Para o intermediador ActiveMQ e para os testes com o Qpid (com 10 produtores) assumiu-se uma área de 0. Tabela 19 (b), tem-se as representatividades calculadas, sendo que os valores obtidos foram de 88,24%, 11,76% e 0%, para os

intermediadores RabbitMQ, Qpid e ActiveMQ, respectivamente.

Tabela 19: Matrizes de representatividade dos testes de escalabilidade para um encaminhamento por tópicos

	3 Clientes		5 Clientes		10 Clientes	
Intermediador	Área	Perc.	Área	Perc.	Área	Perc.
RabbitMQ	$3,1946 * 10^7$	84,29%	$3,3595 * 10^7$	80,45%	$3,0104 * 10^7$	100,00%
Qpid	$5,9559 * 10^6$	15,71%	$8,1655 * 10^6$	19,55%	0	0%
ActiveMQ	0	0%	0	0%	0	0%

(a)

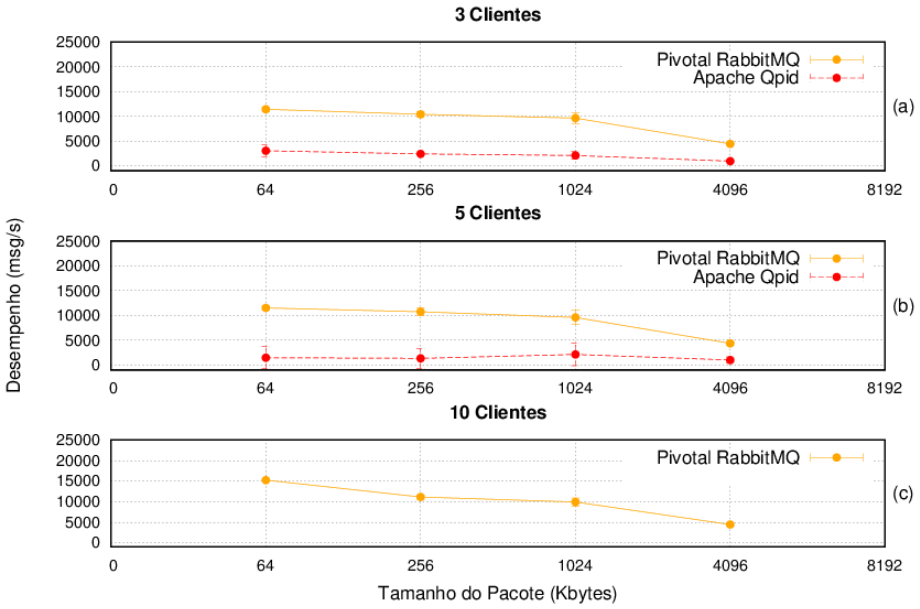
Intermediador	Resultado
RabbitMQ	**88,24%**
Qpid	**11,76%**
ActiveMQ	**0%**

(b)

5.2.9 Teste de escalabilidade no encaminhamento por cabeçalhos

Na Figura 20 tem-se um gráfico apresentando o número de pacotes enviados em função do tamanho do pacote para os intermediadores RabbitMQ e QPID, utilizando um encaminhamento por cabeçalhos. A figura mostra o desempenho da ferramenta considerando 3, 5 e 10 produtores. Não foram realizados teste com o intermediador ActiveMQ, uma vez que essa não possui suporte para este tipo de encaminhamento (ACTIVEMQ, 2015). Além disso, não foi possível realizar os testes com 10 produtores para o intermediador Qpid, pois o intermediador apresentou uma sobrecarga e falhou.

Figura 30: Teste de escalabilidade para o encaminhamento por cabeçalho



Fonte: elaborado pelo autor

Na Tabela 20 (a), observa-se as áreas obtidas para cada um dos testes e o percentual resultante da normalização. Para o intermediador ActiveMQ e para os testes realizados com o Qpid (com 10 produtores), assumiu-se uma área de 0. Na Tabela 20 (b), tem-se as representatividades calculadas, sendo que os valores foram de 88,81%, 11,19% e 0%, para os intermediadores RabbitMQ, Qpid e ActiveMQ, respectivamente.

Tabela 20: Matrizes de representatividade dos testes de escalabilidade para um encaminhamento por por cabeçalho

	3 Clientes		5 Clientes		10 Clientes	
Intermediador	Área	Perc.	Área	Perc.	Área	Perc.
RabbitMQ	$3,1454 * 10^7$	83,24%	$3,1371 * 10^7$	83,18%	$2,7564 * 10^7$	100,00%
Qpid	$6,3315 * 10^6$	16,76%	$6,3433 * 10^6$	16,82%	0	0%
ActiveMQ	0	0%	0	0%	0	0%

(a)

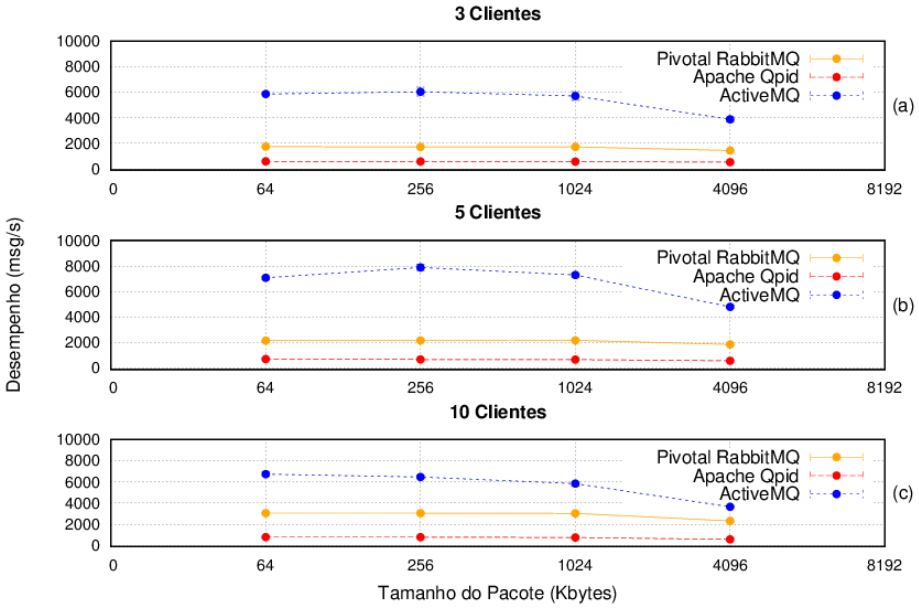
Intermediador	Resultado
RabbitMQ	**88,81%**
Qpid	**11,19%**
ActiveMQ	**0%**

(b)

5.2.10 Teste de escalabilidade no recebimento de mensagens

Na Figura 20 tem-se um gráfico apresentando o número de pacotes recebidos em função do tamanho do pacote para os intermediadores RabbitMQ, QPID e ActiveMQ, considerando 3, 5 e 10 consumidores.

Figura 31: Teste de escalabilidade para o recebimento de mensagens



Fonte: elaborado pelo autor

Na Tabela 21 (a), observa-se as áreas obtidas para cada um dos testes e o percentual resultante da normalização. Na Tabela 21 (b), tem-se as representatividades calculadas, sendo que os valores obtidos foram de 25,84%, 7,52% e 66,64%, para os intermediadores RabbitMQ, Qpid e ActiveMQ, respectivamente.

Tabela 21: Matrizes de representatividade dos testes de escalabilidade no recebimento de mensagens

	3 Clientes		5 Clientes		10 Clientes	
Intermediador	Área	Perc.	Área	Perc.	Área	Perc.
RabbitMQ	$6,6048 * 10^6$	22,23%	$8,4395 * 10^6$	22,27%	$1,1416 * 10^7$	33,01%
Qpid	$2,2537 * 10^6$	7,58%	$2,5693 * 10^6$	6,78%	$2,8302 * 10^6$	8,18%
ActiveMQ	$2,0858 * 10^7$	70,19%	$2,6890 * 10^7$	70,95%	$2,0333 * 10^7$	58,80%

(a)

Intermediador	Resultado
RabbitMQ	**25,84%**
Qpid	**7,52%**
ActiveMQ	**66,64%**

(b)

5.2.11 Alta Disponibilidade

O primeiro intermediador de mensagens avaliado foi o Pivotal RabbitMQ. Nesse é possível realizar uma configuração de alta disponibilidade chamada de *cluster* ativo-ativo. Nesse tipo de configuração temos mais de um nodo do intermediador ativo com um balanceador de carga (*load balancer*[2]), ou seja, o *load balancer* distribui as conexões entre os nodos ativos. No caso de uma falha em um dos nodos do *cluster*, as conexões são encaminhadas para os outros nodos ativos. É importante destacar que as mensagens são replicadas entre todos os nodos ativos do *cluster*. Desta forma, todos os eles possuem o estado atual das filas de mensagem.

Para o intermediador de mensagens Apache QPID, a opção disponível é a configuração de um *cluster* ativo-passivo. Neste caso, tem-se apenas um nodo ativo (mestre), sendo que todas as mensagens são replicadas aos outros nodos. Porém, esses não recebem conexões até que não sejam transformados em nodos mestres, ou seja, não forem ativados. É importante destacar que caso o nodo mestre falhe, é necessário ativar manualmente o nodo passivo. Assim, todas as conexões que estavam ativas com o intermediador de mensagem irão falhar. Desta forma, é necessário a implementação de um *failover*[3] nos produtores e consumidores.

Por fim, para o intermediador de mensagens Apache ActiveMQ, existe uma implementação de alta disponibilidade similar a do intermediador QPID, porém com a ativação automática do nodo passivo. Desta forma, quando o nodo mestre falhar, o nodo passivo

[2] **Load Balancer**: são responsáveis por distribuir requisições entre os nodos ativos. Esses são utilizados para garantir um melhor uso dos recursos computacionais.

[3] ***failover***: é um mecanismo da aplicação que provê uma troca de servidor para um outro no caso de uma falha.

é ativado automaticamente. É importante destacar que como não existe um sistema de cluster ativo-ativo, é necessário que o produtor e consumidor implementem um *failover*.

De modo a comparar os intermediadores, foi utilizada a escala fundamental de Saaty (ver Seção 3.1.2, Tabela 3), sendo atribuído peso 1 para o intermediador RabbitMQ, uma vez que esse implementa um sistema de cluster ativo-ativo. Já para o intermediador ActiveMQ foi atribuído um peso igual a 5, uma vez que esse possui um sistema de alta disponibilidade baseado no modelo cluster ativo-passivo automático. Desta forma, exigindo uma implementação de um mecanismo *failover* nos produtores e consumidores.

Por fim, atribuiu-se um peso 6 para o intermediador Qpid que também implementa um sistema de cluster ativo-passivo, porém com acionamento manual. Considerou-se um peso 9 para um sistema que não apresenta um modelo de alta disponibilidade. Os percentuais de representatividade são apresentados na Tabela 22, sendo que os valores obtidos foram de 72,0%, 10,0% e 18,0%, para os intermediadores RabbitMQ, Qpid e ActiveMQ, respectivamente. Já a inconsistência calculada foi de 2,5% para os valores obtidos.

Tabela 22: Tabela de pesos para o critério alta disponibilidade

	RabbitMQ	Qpid	ActiveMQ	PML
RabbitMQ	1	6	5	72,0%
Qpid	$\frac{1}{6}$	1	$\frac{1}{2}$	10,0%
ActiveMQ	$\frac{1}{5}$	2	1	18,0%

Inconsistência: 2,5%

5.2.12 Disponibilidade de Documentação

O intermediador RabbitMQ apresenta aproximadamente 9300 tópicos publicados em seu fórum, aproximadamente 15000 perguntas no fórum *Stack Overflow* e 9 livros publicados. Já o intermediador Apache Qpid, apresenta aproximadamente 3000 tópicos publicados em seu fórum, aproximadamente 700 perguntas no fórum *Stack Overflow* e nenhum livro publicado. Por fim, o intermediador Apache ActiveMQ apresenta aproximadamente 14500 tópicos publicados em seu fórum, aproximadamente 10500 perguntas no fórum *Stack Overflow* e 5 livros publicados.

Na Tabela 23 (a), tem-se um resumo desses valores e a normalização dos mesmos. Na Tabela 23 (b) tem-se que os valores obtidos foram de 52,08%, 4,62% e 43,30%, para os intermediadores RabbitMQ, Qpid e ActiveMQ, respectivamente.

Tabela 23: Tabela de pesos para o critério disponibilidade de documentação

	Forum Oficial		*Stack Overflow*		Livros	
Intermediador	Tópicos	Perc.	Tópicos	Perc.	Qtde.	Perc.
RabbitMQ	9300	34,70%	15000	57,25%	9	64,29%
Qpid	3000	11,20%	700	2,67%	0	0%
ActiveMQ	14500	54,10%	10500	40,08%	5	35,71%

(a)

Intermediador	Resultado
RabbitMQ	**52,08%**
Qpid	**4,62%**
ActiveMQ	**43,30%**

(b)

5.2.13 Resultados Finais

Ao final, os resultados obtidos para cada critério, devem ser compilados de forma a obter-se um resultado único para cada intermediador. Na Tabela 24 tem-se a compilação dos resultados obtidos em cada um dos testes para cada um dos intermediadores (ver Seção 4.3.2). Além disso, na última coluna da tabela, tem-se o percentual médio global. Esses foram apresentados na Seção 4.3.2 (ver Tabela 11).

Tabela 24: Tabela com o resultado dos testes realizados para os intermediadores e o percentual médio global dos critérios

	Percentual Médio Local			PMG dos
	RabbitMQ	Qpid	ActiveMQ	Critérios
Performance Direto	72,16%	15,64%	12,20%	11,53%
Performance Fanout	83,14%	16,86%	0,00%	4,74%
Performance Topicos	82,55%	17,45%	0,00%	1,74%
Performance Cabeçalhos	82,95%	17,05%	0,00%	1,74%
Escalabilidade Direto	69,47%	8,52%	22,01%	18,73%
Escalabilidade Fanout	88,84%	11,16%	0,00%	7,70%
Escalabilidade Tópicos	88,24%	11,76%	0,00%	2,83%
Escalabilidade Cabeçalho	88,81%	11,19%	0,00%	2,83%
Performance Consumidor	18,64%	7,98%	73,38%	9,88%
Escalabilidade Consumidor	25,84%	7,52%	66,64%	16,04%
Alta Disponibilidade	72,00%	10,00%	18,00%	17,67%
Documentação	52,08%	4,62%	43,30%	4,57%

Para que seja possível obter o resultado final da análise, os valores do percentual médio local devem ser convertidos para uma representatividade global. Assim, deve ser feita uma multiplicação da representatividade média local do intermediador pela representatividade global do critério (SAATY, 1980). Ao final, os valores obtidos deverão ser somados obtendo-se o resultado para cada intermediador. A Tabela 25 apresenta o resultado deste e o percentual resultante para os intermediadores avaliados.

Como pode ser observado, o intermediador RabbitMQ apresentou uma representatividade de 61,08% e, portanto, apresenta superioridade sobre os demais intermediadores. Essa superioridade foi evidenciada em quase todos os critérios avaliados. De fato, este apresentou um desempenho inferior somente nos testes de recebimento de mensagens com a performance e escalabilidade do consumidor inferior a ferramenta ActiveMQ.

Já o intermediador ActiveMQ apresentou uma representatividade de 28,63% onde, mesmo não implementando todos os tipos de encaminhamento de mensagens propostos pela

Tabela 25: Tabela com os valores dos testes e o resultado final obtido na avaliação deste trabalho

	Cálculo da Normalização			Resultado		
	RabbitMQ	Qpid	ActiveMQ	RabbitMQ	Qpid	ActiveMQ
Performance Direto	(72,16%*11,53%)	(15,64%*11,53%)	(12,20%*11,53%)	8,32%	1,80%	1,41%
Performance Fanout	(83,14%*4,74%)	(16,86%*4,74%)	(0,00%*4,74%)	3,94%	0,80%	0,00%
Performance Topicos	(82,55%*1,74%)	(17,45%*1,74%)	(0,00%*1,74%)	1,44%	0,30%	0,00%
Performance Cabeçalhos	(82,95%*1,74%)	(17,05%*1,74%)	(0,00%*1,74%)	1,44%	0,30%	0,00%
Escalabilidade Direto	(69,47%*18,73%)	(8,52%*18,73%)	(22,01%*18,73%)	13,01%	1,60%	4,12%
Escalabilidade Fanout	(88,84%*7,70%)	(11,16%*7,70%)	(0,00%*7,70%)	6,84%	0,86%	0,00%
Escalabilidade Tópicos	(88,24%*2,83%)	(11,76%*2,83%)	(0,00%*2,83%)	2,50%	0,33%	0,00%
Escalabilidade Cabeçalho	(88,81%*2,83%)	(11,19%*2,83%)	(0,00%*2,83%)	2,51%	0,32%	0,00%
Performance Consumidor	(18,64%*9,88%)	(7,98%*9,88%)	(73,38%*9,88%)	1,84%	0,79%	7,25%
Escalabilidade Consumidor	(25,84%*16,04%)	(7,52%*16,04%)	(66,64%*16,04%)	4,14%	1,21%	10,69%
Alta Disponibilidade	(72,00%*17,67%)	(10,00%*17,67%)	(18,00%*17,67%)	12,72%	1,77%	3,18%
Documentação	(52,08%*4,57%)	(4,62%*4,57%)	(43,30%*4,57%)	2,38%	0,21%	1,98%
				61,08%	**10,29%**	**28,63%**

norma AMQP de versão 0.9.1. Esse desempenho deve-se principalmente a sua performance no recebimento de mensagens. De fato, esse foi superior aos demais intermediadores neste critério. A documentação também teve bons resultados, uma vez que os foruns e os posts no *Stack Overflow* demonstram uma comunidade de desenvolvimento ativa.

Por fim, o intermediador Qpid obteve uma representatividade de apenas 10,29%. Essa apresentou os piores resultados em todos os critérios.

6 CONSIDERACOES FINAIS

Os estudos realizados neste trabalho objetivaram prover informações que auxiliassem na escolha de um intermediador de mensagens que implemente o padrão AMQP, e que seja mais adequado para ser utilizado como *middleware* na execução de processos assíncronos em ambientes de nuvens SaaS.

Para entendimento do funcionamento do intermediador de mensagens, foi apresentado o protocolo AMQP, que tem por objetivo padronizar o desenvolvimento de intermediadores de mensagem. Esse protocolo define entre outras coisas, a arquitetura do intermediador, o formato da mensagem, os tipos de encaminhamento, os procedimentos de tolerância a falhas, os mecanismos de segurança, os tipos de dados, etc. Foram escolhidas para esta avaliação os intermediadores de código aberto RabbitMQ, Qpid, ActiveMQ que implementam esse protocolo.

Para realizar os testes comparativos, foi utilizado o método analítico hierárquico que auxilia na avaliação diminuindo a subjetividade do avaliador. Baseando-se nesse método foi definido um modelo de avaliação, que procurou avaliar a performance e escalabilidade dos intermediadores. Além disso, procurou-se avaliar a alta disponibilidade dos intermediadores e sua documentação. Como ferramenta de apoio a avaliação dos intermediadores, definiu-se uma aplicação de referência, que tinha por objetivo produzir e consumir mensagens.

Como resultados, obteve-se que o intermediador RabbitMQ é a mais adequado para o desenvolvimento de aplicações que executem processos assíncronos em nuvens do tipo SaaS. Este apresentou uma representatividade de 61,08%,sendo superior em quase todos os critérios avaliados. Além disso, observou-se durante os testes, que este intermediador apresenta uma alta confiabilidade e um baixo consumo de recursos computacionais.

Já o intermediador ActiveMQ, apresentou um percentual de representatividade de 28,63%. Essa ferramenta mostrou-se adequada em situações onde o número de mensagens a ser recebidas é crítico. De fato esta apresentou um melhor desempenho em todos os testes envolvendo o recebimento de mensagens. Além disso, esse intermediador apresentou um baixo consumo de recursos computacionais na execução.

Por fim, o intermediador Qpid, apresentou um percentual de representatividade de apenas 10,29%. Esse foi o que apresentou mais problemas durante a execução, apresentando instabilidade no envio e recebimento de mensagens, necessitando o desenvolvimento de mecanismos de *failover*. Destaca-se também, seu alto consumo de recursos computacionais, o que inclusive inviabilizou os testes em cenários de escalabilidade de 10 clientes.

6.1 Trabalhos Futuros

Como sugestões de trabalhos futuros tem-se:

- Avaliar a qualidade do serviço (QoS - *Quality of Service*) dos intermediadores de mensagens que já encontram-se disponíveis em nuvens comerciais como, por exemplo, os intermediadores Windows Azure Service Bus (MICROSOFT, 2016a), Google Cloud Pub/Sub (GOOGLE, 2016) e Amazon Simple Queue Service (AMAZON, 2016).
- Avaliar intermediadores de mensagem comerciais que implementam o protocolo AMQP. Como exemplo de intermediadores pode-se citar: SwiftMQ (IIT, 2016), Service Bus for Windows Server (MICROSOFT, 2016c) e Red Hat JBoss A-MQ (RED HAT, 2016).
- Avaliar os intermediadores de mensagem em testes de performance e escalabilidade com mais tamanhos de mensagem, utilizando também mensagens de 8 KB, 16 KB, 32 KB e 64 KB.

Referências

AMAZON WEB SERVICES, INC. *Amazon Simple Queue Service (SQS)*. [S.l.], 2016. Disponível em: <https://aws.amazon.com/pt/sqs/>. Acesso em: 15 jun. 2016. Citado na página 66.

AMAZON.COM INC. *Amazon Web Services*. [S.l.], 2015. Disponível em: <https://aws.amazon.com>. Acesso em: 10 nov. 2015. Citado na página 11.

AMAZON.COM, INC. *Amazon.com: Online Shopping for Electronics, Apparel, Computers, Books, DVDs & more*. [S.l.], 2016. Disponível em: <https://www.amazon.com/>. Acesso em: 03 jun. 2016. Citado na página 40.

BAKER, M. Cluster computing white paper. *arXiv preprint cs/0004014*, 2000. Citado na página 40.

BARRETO, C. G. *Agregando Frameworks de Infra-Estrutura em uma Arquitetura Baseada em Componentes: Um Estudo de Caso no Ambiente AulaNet*. Tese (Doutorado) — PUC-Rio, 2006. Citado na página 40.

BESTEIRO, A. M. et al. A utilização do método ahp para traçar, como ferramenta para o auxilio a decisão de um candidato, a escolha de um curso de engenharia. 2009. Citado na página 51.

CARMO, T. d. R. *Uso do padrão AMQP para transporte de mensagens entre atores remotos*. Dissertação (Mestrado) — Universidade de São Paulo, 2012. Citado 2 vezes nas páginas 16 e 20.

CHAN, A. H.; KWOK, W.; DUFFY, V. G. Using ahp for determining priority in a safety management system. *Industrial Management & Data Systems*, Emerald Group Publishing Limited, v. 104, n. 5, p. 430–445, 2004. Citado na página 29.

FIELDING, R. et al. *Hypertext transfer protocol–HTTP/1.1*. [S.l.], 1999. Citado na página 25.

GNU PROJECT. *Gnuplot*. [S.l.], 2016. Disponível em: <http://gnuplot.info/>. Acesso em: 03 jun. 2016. Citado na página 49.

GOOGLE INC. *Google Cloud Pub/Sub*. [S.l.], 2016. Disponível em: <https://cloud.google.com/pubsub/>. Acesso em: 15 jun. 2016. Citado na página 66.

GOOGLE TECHNOLOGY COMPANY. *Google Cloud Computing, Hosting Services & Cloud Support*. [S.l.], 2015. Disponível em: <https://cloud.google.com/>. Acesso em: 10 nov. 2015. Citado na página 11.

GUIDORIZZI, H. L. *Um curso de cálculo, vol. 2*. [S.l.]: Grupo Gen-LTC, 2000. Citado na página 50.

IIT SOFTWARE GMBH. *SwiftMQ Enterprise Messaging Platform*. [S.l.], 2016. Disponível em: <http://www.swiftmq.com/>. Acesso em: 15 jun. 2016. Citado na página 66.

KELLEY, C. T. *Iterative methods for optimization*. [S.l.]: Siam, 1999. v. 18. Citado na página 49.

KRAMER, J. Advanced message queuing protocol (amqp). *Linux J.*, Belltown Media, Houston, TX, v. 2009, n. 187, nov. 2009. ISSN 1075-3583. Disponível em: <http://dl.acm.org/citation.cfm?id=1653247.1653250>. Citado 2 vezes nas páginas 15 e 24.

LIBERTY, J. *Programming C#: Building. NET Applications with C#*. [S.l.]: "O'Reilly Media, Inc.", 2005. Citado na página 37.

MARSH, G. et al. Scaling advanced message queuing protocol (amqp) architecture with broker federation and infiniband. *Ohio State University, Tech. Rep. OSU-CISRC-5/09-TR17*, 2008. Citado na página 38.

MICROSOFT CORPORATION. *Microsoft Azure: Plataforma e serviços de computação em nuvem*. [S.l.], 2015. Disponível em: <https://azure.microsoft.com>. Acesso em: 10 nov. 2015. Citado na página 11.

MICROSOFT CORPORATION. *AMQP.Net Lite: AMQP 1.0 library for .Net and WinRT*. [S.l.], 2016. Disponível em: <https://www.nuget.org/packages/AMQPNetLite/1.1.8>. Acesso em: 10 jun. 2016. Citado na página 47.

MICROSOFT CORPORATION. *Azure Service Bus, a cloud-based messaging system for connecting apps and devices across public and private clouds*. [S.l.], 2016. Disponível em: <https://azure.microsoft.com/en-us/services/service-bus/>. Acesso em: 15 jun. 2016. Citado na página 66.

MICROSOFT CORPORATION. *.NET - Powerful Open Source Cross Platform Development*. [S.l.], 2016. Disponível em: <https://www.microsoft.com/net>. Acesso em: 10 jun. 2016. Citado na página 47.

MICROSOFT CORPORATION. *Service Bus for Windows Server (Service Bus 1.1)*. [S.l.], 2016. Disponível em: <https://msdn.microsoft.com/en-us/library/dn282144.aspx>. Acesso em: 15 jun. 2016. Citado na página 66.

OASIS, O. Organization for the advancement of structured information standards. *Oasis Open*, v. 2, n. 4, 2015. Citado na página 15.

O'HARA, J. Toward a commodity enterprise middleware. *Queue*, ACM, v. 5, n. 4, p. 48–55, 2007. Disponível em: <http://dl.acm.org/citation.cfm?id=1255424/>. Acesso em: 15 out. 2015. Citado na página 15.

PIVOTAL. *.NET/C# RabbitMQ client library*. [S.l.], 2016. Disponível em: <https://www.rabbitmq.com/dotnet.html>. Acesso em: 10 jun. 2016. Citado 2 vezes nas páginas 47 e 48.

PIVOTAL SOFTWARE. *RabbitMQ - Messaging that just works*. [S.l.], 2015. Disponível em: <https://www.rabbitmq.com/>. Acesso em: 01 nov. 2015. Citado 2 vezes nas páginas 11 e 25.

RED HAT, INC. *Red Hat JBoss A-MQ*. [S.l.], 2016. Disponível em: <https://www.redhat.com/en/technologies/jboss-middleware/amq>. Acesso em: 15 jun. 2016. Citado na página 66.

REIS, C. R.; FORTES, R. P. de M. *Caracterizaç ao de um Processo de Software para Projetos de Software Livre*. Tese (Doutorado) — PhD thesis, University of Sao Paulo, Brazil, 2003. Citado na página 40.

RIBEIRO, A. de L. *Teorias da administração*. [S.l.]: Saraiva, 2003. Citado na página 29.

SAATY, T. L. *The Analytic Hierarchy Process*. New york: McGraw-Hill, 1980. Citado 6 vezes nas páginas 13, 29, 34, 35, 41 e 62.

SAATY, T. L. *Decision making for leaders: the analytic hierarchy process for decisions in a complex world*. [S.l.]: RWS publications, 1999. v. 2. Citado na página 13.

SAATY, T. L. Decision making with the analytic hierarchy process. *International journal of services sciences*, Inderscience Publishers, v. 1, n. 1, p. 83–98, 2008. Citado 2 vezes nas páginas 30 e 31.

SAATY, T. L.; VARGAS, L. G. The seven pillars of the analytic hierarchy process. In: *Models, Methods, Concepts & Applications of the Analytic Hierarchy Process*. [S.l.]: Springer, 2012. p. 23–40. Citado 2 vezes nas páginas 51 e 55.

SILVA, R. M.; BELDERAIN, M. C. N. Considerações sobre métodos de decisão multicritério. *International journal of services sciences*, Instituto Tecnológico de Aeronáutica, ITA, São Paulo, 2005. Citado na página 31.

SNYDER, B.; BOSNANAC, D.; DAVIES, R. *ActiveMQ in action*. [S.l.]: Manning, 2011. v. 47. Citado na página 26.

SOUSA, F. R.; MOREIRA, L. O.; MACHADO, J. C. Computação em nuvem: Conceitos, tecnologias, aplicações e desafios. *II Escola Regional de Computação Ceará, Maranhão e Piauí (ERCEMAPI)*, p. 150–175, 2009. Citado na página 11.

SPOLTI, E. *Avaliação de frameworks de desenvolvimento de aplicações móveis multiplataforma*. 2015. Trabalho de Conclusão de Curso - Ciência da Computação, Universidade de Caxias do Sul, Caxias do Sul, 2015. Citado na página 29.

STACK EXCHANGE INC. *Stack Overflow*. [S.l.], 2016. Disponível em: <http://stackoverflow.com/>. Acesso em: 03 jun. 2016. Citado na página 40.

STANFORD-CLARK, A.; HUNKELER, U. Mq telemetry transport (mqtt). 1999. Disponível em: <http://mqtt.org>. Citado 2 vezes nas páginas 25 e 26.

STOMP GROUP. *The Simple Text Oriented Messaging Protocol*. [S.l.], 2012. Disponível em: <https://stomp.github.io/>. Acesso em: 21 nov. 2015. Citado 2 vezes nas páginas 25 e 26.

TANENBAUM, A. S.; BOS, H. *Modern Operating Systems*. 4th. ed. Upper Saddle River, NJ, USA: Prentice Hall Press, 2014. ISBN 013359162X, 9780133591620. Citado na página 47.

THE APACHE SOFTWARE FOUNDATION. *Apache ActiveMQ*. [S.l.], 2015. Disponível em: <http://activemq.apache.org/>. Acesso em: 01 mar. 2016. Citado 9 vezes nas páginas 11, 25, 26, 51, 52, 53, 56, 57 e 58.

THE APACHE SOFTWARE FOUNDATION. *Apache Qpid, Messaging built on AMQP*. [S.l.], 2015. Disponível em: <http://qpid.apache.org/>. Acesso em: 01 nov. 2015. Citado 3 vezes nas páginas 11, 25 e 26.

TRIELOFF, C. et al. Advanced message queuing protocol protocol specification. amq-spec. *AMQP. org*, v. 0.10, 2006. Citado 9 vezes nas páginas 13, 15, 16, 17, 18, 19, 20, 22 e 24.

TROITIñO, D. *Aplicações Ricas na Web*. 2012. Trabalho de Conclusão de Curso - Sistemas de Informação, Universidade de Caxias do Sul, Caxias do Sul, 2012. Citado na página 29.

VMWARE, INC. *VMware Virtualization for Desktop & Server, Application, Public & Hybrid Clouds*. [S.l.], 2016. Disponível em: <http://www.vmware.com/>. Acesso em: 03 jun. 2016. Citado na página 48.

WHITAKER, R. Criticisms of the analytic hierarchy process: Why they often make no sense. *Mathematical and Computer Modelling*, Elsevier, v. 46, n. 7, p. 948–961, 2007. Citado 2 vezes nas páginas 51 e 55.

WOLFRAM ALPHA LLC. *Wolfram|Alpha: Computational Knowledge Engine*. [S.l.], 2016. Disponível em: <https://www.wolframalpha.com/>. Acesso em: 02 jun. 2016. Citado na página 50.

YANG, K.-L.; CHU, P.; CHOUHUANG, W.-T. Note on incremental benefit/cost ratios in analytic hierarchy process. *Mathematical and computer modelling*, Elsevier, v. 39, n. 2, p. 279–286, 2004. Citado na página 51.