MMEliot: Um Modelo para Internet das Coisas Explorando a Elasticidade da Computação em Nuvem

Vinicius F. Rodrigues[1], **Thiago Y. R. da Silva**[1]
Rodrigo da R. Righi[1], **Cristiano A. da Costa**[1]

[1] Programa Interdisciplinar de Pós-Graduação em Computação (PIPCA)
Universidade do Vale do Rio dos Sinos (UNISINOS)
Av. Unisinos – 950 – Cristo Rei – São Leopoldo – RS – Brazil

{vfrodrigues,tyrsilva,rrrighi,cac}@unisinos.br

Abstract. *The adoption of Internet of Things (IoT) technologies has been growing in the last few years significantly. For this reason, there is an increasing necessity of systems capable of processing high amount of data with a certain performance level. Some strategies adopt the use of Cloud Computing as promising solution to address this problem. In this context, this paper proposes MMEliot as an extension of the Eliot cloud computing elasticity model for IoT. MMEliot model combines CPU load data and response time from IoT requests to overpass the problem related to lack of reactivity. Our experiments show that MMEliot is able to anticipate allocation and release of resources from the cloud, generating better performance.*

Resumo. *Atualmente, a adoção de tecnologias relacionadas à Internet das Coisas vem aumentando significantemente. Devido a isso, cada vez mais são exigidas altas demandas de computação para tratar volumes grandes de dados. Como solução, algumas abordagens empregam a Computação em Nuvem para suprir tais necessidades. Neste contexto, este artigo apresenta um modelo de elasticidade em nuvem MMEliot que estende o modelo Eliot. O modelo combina dados de carga de CPU e de tempo de resposta de requisições IoT para sobrepassar problemas relacionados à falta de reatividade. Os resultados obtidos nas avaliações demonstram que o MMeliot é capaz de antecipar a alocação e liberação de recursos na nuvem, resultando em ganhos de desempenho.*

1. Introdução

A Internet das Coisas, também conhecida em Inglês como *Internet of Things* (IoT), permite que objetos, pessoas ou coisas possam transferir dados através da rede sem ação de um ser humano [Atzori et al. 2010, Gubbi et al. 2013]. De acordo com a consultoria IDC[1], o universo digital de dados proveniente de sensores e sistemas corresponde a 2% do universo digital e esse número tende a aumentar em 10% até 2020. Além do impressionante volume de dados envolvido em IoT, em um período de apenas 7 anos a quantidade de "coisas" conectadas tende a aumentar em 50% para 30 bilhões de dispositivos conectados em 2020. Dentre as diversas tecnologias disponíveis para viabilizar sistemas IoT, o RFID (*Radio Frequency Identification*) (padrão EPCglobal[2]) destaca-se como sendo uma

[1]http://www.emc.com/leadership/digital-universe/2014iview/index.htm
[2]http://www.gs1.org/epcglobal

das principais [Welbourne et al. 2009]. Através do RFID, é possível identificar objetos e pessoas através de tags que, além de transmitir dados de identificação, podem transmitir dados adicionais incluindo condições do ambiente e dados provenientes de sensores. Em tais sistemas, torna-se necessário um *middleware* que possa realizar a intermediação entre a comunicação entre sistemas de uma organização e a infraestrutura de *hardware* de um sistema RFID. Além do RFID, outra tecnologia que tem sido muito utilizada por conta dos seus diversos benefícios é a Computação na Nuvem, a qual oferece alocação de recursos sob demanda [Galante and d. Bona 2012]. Aliado aos benefícios da Computação na Nuvem está a elasticidade de recursos, em que recursos computacionais podem ser adicionados ou removidos do ambiente a qualquer momento transparentemente para o usuário final [Mell and Grance 2011]. Dentre as principais estratégias de elasticidade, destacam-se a propostas de elasticidade automática reativa e proativa diferindo na abordagem adotada para o lançamento de operações [Nikravesh et al. 2015].

Plataformas tradicionais de IoT geralmente apresentam problemas relacionados à escalabilidade em momentos de alta demanda em que grandes massas de dados gerados simultaneamente podem ocasionar quedas de desempenho do sistema [Gomes et al. 2014]. Algumas abordagens da literatura buscam atacar este problema propondo estratégias de elasticidade em nuvem para sistemas IoT [Righi et al. 2016]. Em particular, o modelo proposto por Righi et al. [Righi et al. 2016], chamado Eliot, foca especificamente na elasticidade reativa de recursos. A motivação para a criação do *Eliot* foram os resultados coletados a partir do trabalho conduzido por Gomes et al. [Gomes et al. 2014], em que os componentes do *middleware* se mostraram instáveis durante os testes realizados. Porém, soluções como *Eliot* apresentam limitações relacionadas às métricas utilizadas para avaliação da elasticidade. Em particular, *Eliot* considera apenas uma métrica simples em sua análise de carga, além de empregar *thresholds* de elasticidade fixos. Esta estratégia apresenta problemas relacionados à perda de reatividade conforme demonstrado na Figura 1. Conforme a figura, a caixa em amarelo indica uma situação em que não ocorrem alterações na quantidade de CPU até atingir-se o *threshold* máximo, mesmo a carga crescendo de forma constante. Considerando tais limitações, este artigo apresenta o modelo MMEliot como extensão do modelo *Eliot*. MMEliot provê elasticidade reativa multimétrica, além de aplicar cálculo de séries temporais para a previsão de comportamento futuro da carga de CPU da nuvem. O modelo procura demonstrar uma solução que combine múltiplas métricas da nuvem proporcionando reatividade em situações de grande ou pouca demanda de forma automática e transparente para usuários.

Este artigo está organizado em 6 seções. Na seção 2, observa-se a literatura e o estado da arte nos trabalhos relacionados. O modelo MMEliot é detalhado na seção 3, explorando os detalhes do modelo proposto e suas funcionalidades. Na seção 4 descreve-se a metodologia de avaliação e os cenários para avaliação do protótipo. Em seguida, na seção 5 são discutidos os resultados obtidos. Por fim, na seção 6 apresenta-se as contribuições e sugere-se oportunidades para trabalhos futuros.

2. Trabalhos Relacionados

Soluções encontradas na literatura concentram-se em encontrar maneiras de lidar com a escalabilidade e balanceamento de carga de *middlewares* IoT. Oliot (*Open Language for Internet of Things*) é uma plataforma de código aberto voltada para coleta e compartilhamento de dados provenientes de eventos para aplicações IoT [Byun and Kim 2015].

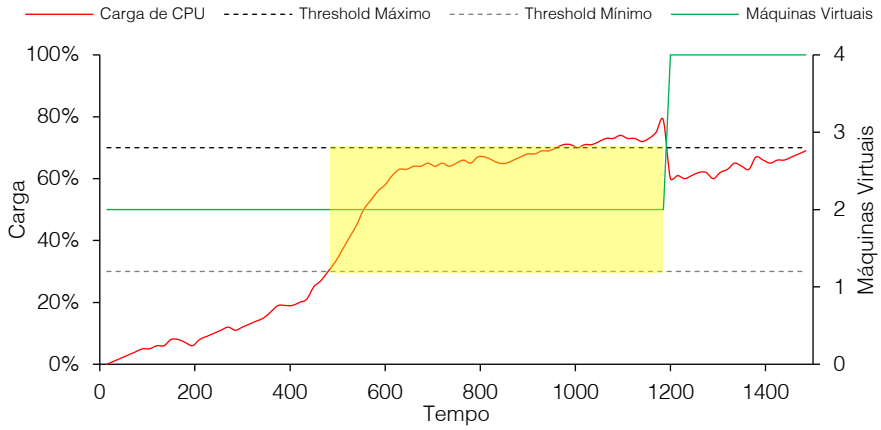


Figura 1. Elasticidade reativa baseada em thresholds fixosa.

A solução apresenta os seguintes eventos: (i) *document-based model*; (ii) *SensorEvent*. Para tratar problemas relacionados à performance, escalabilidade e confiabilidade no EPCIS, os autores desenvolveram o sistema *message queue*, fazendo uso do *Active Message Queue Protocol* (AMQP) e a sua implementação RabbitMQ[3]. No *Capture Service*, o modelo *Work queue* é utilizado para uma entrega confiável dos eventos e balanceamento de carga. Já o *Query Service*, utiliza o modelo *Publish/Subscribe*, que é composto por *subscription scheduler*, um *fanout exchange* e *binded queues*, em que o AMQP recebe mensagens e as adiciona na fila [Byun and Kim 2015].

A solução proposta por Lui et al. [Liu et al. 2008] é uma arquitetura voltada para a distribuição da especificação *Application Level Events* (ALE). A plataforma é composta por três componentes: (i) *Global-ALE*; (ii) *Sub-ALE*; (iii) *Connection Pool*. O principal objetivo desta arquitetura é permitir que o *middleware* RFID possa executar a filtragem de dados EPC sem que ocorra sobrecarga ou atrasos na entrega destes dados filtrados e compilados pelo componente ALE. Os três componentes da arquitetura proposta operam de maneira integrada, em que o componente *Global-ALE* é responsável por responder as requisições de aplicações de cliente e pela coordenação da execução dos diversos *Sub-ALEs*. O componente *Sub-ALE*, por sua vez, recebe dados de etiquetas do *Connection Pool*, realiza a filtragem e o agrupamento destes dados e, após issso, o envio para o componente *Global-ALE*. O *Connection Pool* recebe dados de etiquetas de leitores RFID, armazena-os no EPC *Pool* e por fim os transmite para o componente *Sub-ALE* de acordo com as informações de agendamento. Uma proposta diferente é apresentada pot Schmidt et al. [Schmidt et al. 2011]. Os autores propõem uma solução, chamada DHT-based *distributed* ALE, focando questões de escalabilidade em sistemas IoT. A arquitetura proposta é baseada em tabelas *hash* distribuídas e utiliza o mecanismo *peer-to-peer* (P2P) para evitar gargalos referentes a escalabilidade do componente ALE. A motivação para criação desta solução foi a falta de um tratamento adequado por parte da especificação ALE padrão envolvendo escalabilidade, o que o torna instável em situações que envolvem uma grande quantidade de leituras de etiquetas RFID. Neste sistema, qualquer ALE é traduzido em um nó no sistema P2P. Quando especificações são recebidas, o sistema as divide e as distribui entre todos os ALEs.

[3]https://www.rabbitmq.com/

Focando elasticidade em nuvem para sistemas IoT, Righi et al. [Righi et al. 2016] propõem uma arquitetura chamada Eliot implementada sobre o *middleware* RFID padrão EPCglobal, executado em uma estrutura de nuvem computacional. O principal objetivo da arquitetura proposta é oferecer escalabilidade e elasticidade ao *middleware* IoT de forma transparente do ponto de vista do usuário final. As seguintes modificações no padrão EPCglobal foram realizadas: (i) divisão do módulo EPCIS; (ii) disponibilização de *templates* em máquinas virtuais; (iii) escalabilidade e elasticidade do EPCIS. No mesmo contexto de nuvem, Nguyen et al. desenvolveram uma arquitetura chamada *EPCloudFlow* cujo objetivo é tratar o problema relacionado a balanceamento de carga em máquinas virtuais que armazenam o componente ALE [Nguyen et al. 2015a]. Para tratar o problema mencionado a estratégia prediz os efeitos do atraso na migração de dados, a futura carga das máquinas virtuais e o tempo que leva para os dados migrados estabilizarem. Uma vez que os resultados da predição são gerados estes são utilizados para decidir se alguma máquina virtual está sobrecarregada e se é necessário iniciar o processo de migração de dados. As decisões de projeto mencionadas foram implantadas a partir dos componentes *Load Predictor* e *Migration Manager*, respectivamente responsáveis pela predição e gerenciamento das operações de migração.

Outra estratégia utilizada para garantir maior desempenho em nuvem é através da adição de réplicas de máquinas virtuais. Ziekow et al. [Ziekow et al. 2010] apresentam um serviço de computação em nuvem para rápido acesso a dados de RFID. A abordagem é composto por um conjunto de servidores distribuídos em máquinas virtuais na nuvem chamados de *Cloud-EPCIS*. Esses servidores recebem requisições de clientes e redirecionam a requisição para um dos EPCISs locais, chamados *Master-EPCIS*, conforme requisitos de localidade. O principal objetivo da solução é diminuir o tempo de uma consulta através da distribuição de carga entre os *Cloud-EPCIS*s e a também através da escolha de um *Master-EPCIS* próximo. Também adotando réplicas de máquinas virtuais, Nguyen et al. [Nguyen et al. 2015b] focam no balanceamento de carga entre diversos módulos ALE. A arquitetura proposta é composta por diversas réplicas do serviço ALE. O balanceamento de carga dos fluxos de dados gerados pelos *Readers* é realizada através predição de carga e migração dos fluxos de dados entre as máquinas virtuais. Essas tarefas são realizadas por um Controlador que monitora as estatísticas da métrica CPU para a tomada de decisão de migrações.

Com base no estudo realizado, a Tabela 1 apresenta os principais detalhes das 7 soluções descritas nesta seção. Apenas os trabalhos *EPCloudFlow* e *Eliot* fazem uso da métrica de CPU em uma técnica de elasticidade reativa. Esta métrica é coletada a partir de um ambiente computacional na nuvem com o objetivo de ser utilizada no processo momento de decidir se uma determinada máquina virtual está sobrecarregada ou sub-utilizada. Entretanto, constatou-se a lacuna de pesquisa relacionada ao emprego da elasticidade em nuvem e principalmente na falta de múltiplas métricas nas técnicas de elasticidade estudadas. Considerando os trabalhos que foram implantados na nuvem, é interessante do ponto de vista de qualidade de serviço que tais sistemas monitorem de tempo em tempo diversos recursos computacionais para que estes dados possam ser utilizados em uma técnica de elasticidade e a partir disto permita que o sistema possa tomar decisões inteligentes com base nos dados coletados.

Tabela 1. Tabela comparativa entre os trabalhos relacionados.

Solução	Confiabilidade	Tipo de Elasticidade	Balanceamento de Carga	Gerenciamento de Dados	Segurança
Oliot [Byun and Kim 2015]	EPCglobal reader protocol	Não abordado	Active Message Queuing Protocol (AMQP)	Especificações da EPCglobal	Assinatura Publicação
Distributed ALE in RFID Middleware [Liu et al. 2008]	EPCglobal reader protocol	Reativa, métrica simples	Load predictor e Migration manager	Especificações da EPCglobal	Não abordado
DHT-based distributed ALE [Schmidt et al. 2011]	Global-ALE	Não abordado	Sub-ALE	Especificações da EPCglobal	Global-ALE interfaces
Eliot [Righi et al. 2016]	EPCglobal reader protocol	Reativa, métrica simples	EPCIS load balancer	Especificações da EPCglobal	Assinatura Publicação
EPCloudFlow [Nguyen et al. 2015a]	Chord protocol	Não abordado	Sistema P2P Chord	Especificações da EPCglobal	Não abordado
Ziekow et al. [Ziekow et al. 2010]	EPCglobal reader protocol	Não abordado	EPCIS load balancer	Especificações da EPCglobal	Não abordado
Nguyen et al. [Nguyen et al. 2015b]	EPCglobal reader protocol	Não abordado	Load predictor e Migration manager	Especificações da EPCglobal	Não abordado

3. Modelo MMELIOT (*Multi Metric Eliot*)

Esta seção apresenta o modelo MMEliot - Multi Metric Eliot. MMEliot é um modelo que utiliza elasticidade reativa combinando múltiplas métricas em um ambiente computacional na nuvem. Este modelo procura explorar os benefícios da utilização de múltiplas métricas e as vantagens que a nuvem tem a oferecer. Dentre os benefícios da combinação de múltiplas métricas destaca-se a antecipação e assertividade na alocação ou liberação de recursos nas ações de elasticidade, algo que não é possível alcançar com apenas uma métrica de avaliação. Ainda, prevê a utilização da métrica tempo de resposta como recurso para ser utilizado na técnica de elasticidade. De forma breve, esta métrica é monitorada e coletada a cada requisição feita pelas aplicações cliente e a partir disto um tempo de predição é calculado para esta métrica que por sua vez é utilizado na técnica de elasticidade para decidir o melhor momento para alocar ou liberar máquinas virtuais na nuvem. Assim, procura-se um melhor gerenciamento dos recursos computacionais para evitar que máquinas virtuais fiquem sobrecarregadas ou sub-utilizadas.

3.1. Decisões de Projeto

Como base para o modelo MMEliot, definiu-se estender um *middleware* RFID que utiliza apenas uma métrica na técnica de elasticidade em um modelo que combina múltiplas métricas com o objetivo de ganhar performance e reduzir a perda de pacotes. Do ponto de vista do usuário, o MMEliot foi desenvolvido com o objetivo de ser transparente para o usuário final, no sentido de que o mesmo não terá a necessidade de alterar regras relacionadas a técnica de elasticidade multi métrica para que o sistema se comporte de maneira adequada. Desta forma, parte-se do pressuposto que o usuário já possui uma aplicação para ser executada com quantidade de recursos computacionais definida em um ambiente computacional na nuvem dinâmico e sem a necessidade de alterar o código fonte.

A partir do estudo realizado por Righi et al. [Righi et al. 2016], foi constado o gargalo em torno da métrica de CPU em situações de grande volume de dados por aplicações cliente. Portanto, o modelo MMEliot segue a estratégia de elasticidade reativa utilizando

thresholds superior e inferior para previsão de carga calculada em cima da métrica de CPU. A Figura 2 demonstra a utilização de *thresholds* em que em um certo momento o *threshold* superior é alcançado e recursos devem ser alocados na nuvem. Em seguida, o *threshold* inferior é atingido demonstrando o momento em que recursos devem ser liberados. A métrica de CPU é obtida através do ambiente na nuvem e calculada a partir da técnica *aging* com o objetivo de evitar falsos-positivos e falsos-negativos nas ações de alocar e liberar recursos computacionais. A técnica *aging* faz parte do método de séries temporais *Moving Average*, cujo objetivo é calcular a média de dados considerando uma janela das últimas observações mais recentes [Lorido-Botran et al. 2014].

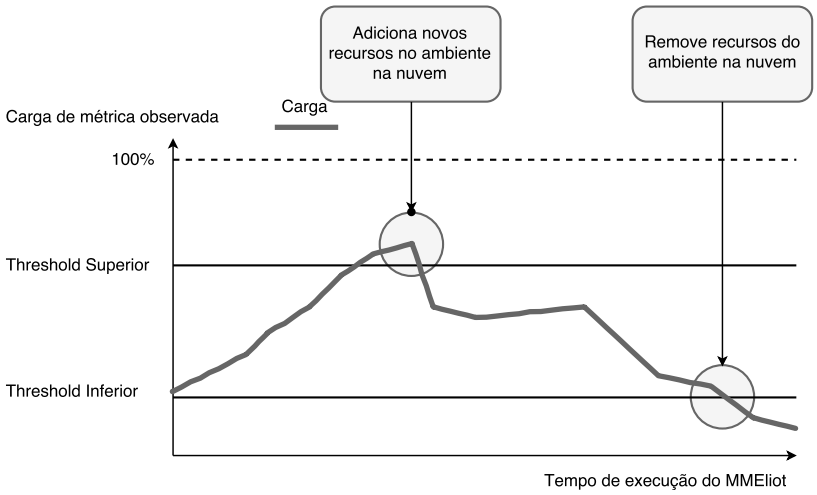


Figura 2. Modelo de elasticidade reativa baseado em *thresholds* da CPU adotado pelo MMEliot.

Para a avaliação do modelo, os testes compreendem a utilização de uma aplicação sintética para a geração de carga. A aplicação utilizada pode ser encontrada no artigo apresentado por Márcio et. al [Gomes et al. 2014]. Através dessa aplicação é possível variar a quantidade de tags e leitores RFID para a geração de dados. Além disso, para a avaliação é possível variar a quantidade *threads* que geram requisições de clientes simulando situações com baixa ou alta carga. A principal ideia é testar diferentes configurações de tags e leitores, além de variar a quantidade de clientes aumentando ou diminuindo a concorrência de acesso ao EPCIS.

O modelo MMEliot utiliza a métrica tempo de resposta como base para tomadas de decisão no mecanismo de elasticidade. Essa métrica indica o tempo médio levado entre o envio das requisições para a *EPCIS Query Interface* e o retorno das respostas. Ainda, MMEliot utiliza a técnica *aging* para definição desta métrica. O modelo segue a estratégia de calcular um percentual de aumento ou diminuição da previsão de tempo entre ciclos de requisições. Caso o aumento em percentual da quantidade de requisições seja igual ou inferior a um determinado percentual isto torna-se um indicativo que deve-se alocar ou liberar recursos. A Figura 3 demonstra uma situação em que ocorre um aumento constante na carga de dados e como consequência a métrica previsão de tempo atinge um percentual de aumento pré-estipulado dando início a alocação de recursos. Logo em seguida, conforme o decréscimo acentuado na carga de dados atingiu-se um percentual

pré-determinado e iniciou-se a liberação de recursos na nuvem. O comportamento apresentado na Figura 3 representa o objetivo do presente trabalho que é antecipar a alocação e liberação de recursos na nuvem. Este modelo ao ser utilizado no âmbito de consultas ao componente *EPCIS Query Interface* pode-se gerar benefícios onde destaca-se: (i) melhor gerenciamento de recursos computacionais; (ii) diminuição da ocorrência de máquinas virtuais saturadas ou sub-utilizadas; (iii) aumento na quantidade de requisições processadas com sucesso.

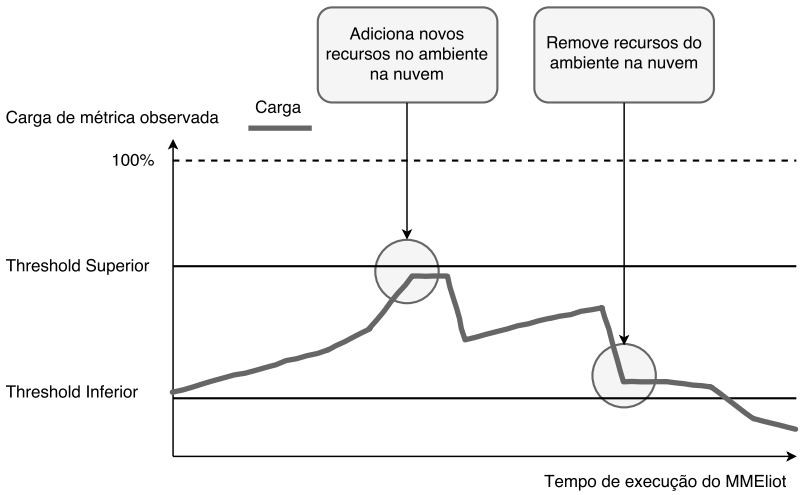


Figura 3. Modelo de elasticidade reativa baseado no tempo de resposta adotado pelo MMEliot.

3.2. Arquitetura

MMEliot é um *middleware* RFID padrão EPCGlobal que permite que aplicações não elásticas possam tirar proveito dos benefícios da elasticidade em nuvem sem necessidade de alterações no código por parte do usuário. Para oferecer elasticidade, o modelo atua com a operação de alocação de instâncias de máquinas virtuais. A proposta consiste em alterações no *middleware* padrão EPCGlobal chamado *Eliot*[4]. A Figura 4 demonstra a arquitetura proposta por este trabalho como extensão ao modelo Eliot. As principais contribuições do modelo MMEliot são representadas pelas caixas cinzas do Gerente de Elasticidade. Enquanto o algoritmo de elasticidade do modelo Eliot considera apenas a predição de carga, MMEliot adicionalmente leva em consideração a previsão de tempo. Os itens relacionados ao *Gerenciador de Elasticidade MMEliot* apresentados no diagrama são detalhados na subseção 3.3.

As principais funções de monitoramento e gerenciamento da elasticidade são realizadas pelo *Gerenciador de Elasticidade MMEliot*. Ele é responsável pela coleta de dados dos recursos do ambiente, aplicar avaliação e gerenciar operações de elasticidade. A cada intervalo de tempo o *Gerenciador* verifica se existem recursos inicializados anteriormente que podem estar disponíveis para utilização. Isto ocorre devido ao tempo de inicialização das máquinas virtuais. Após essa verificação, o *Gerenciador* coleta dados de monitoramento e aplica uma técnica de avaliação para se obter dados relacionados

[4]https://github.com/eliot-project/eliot

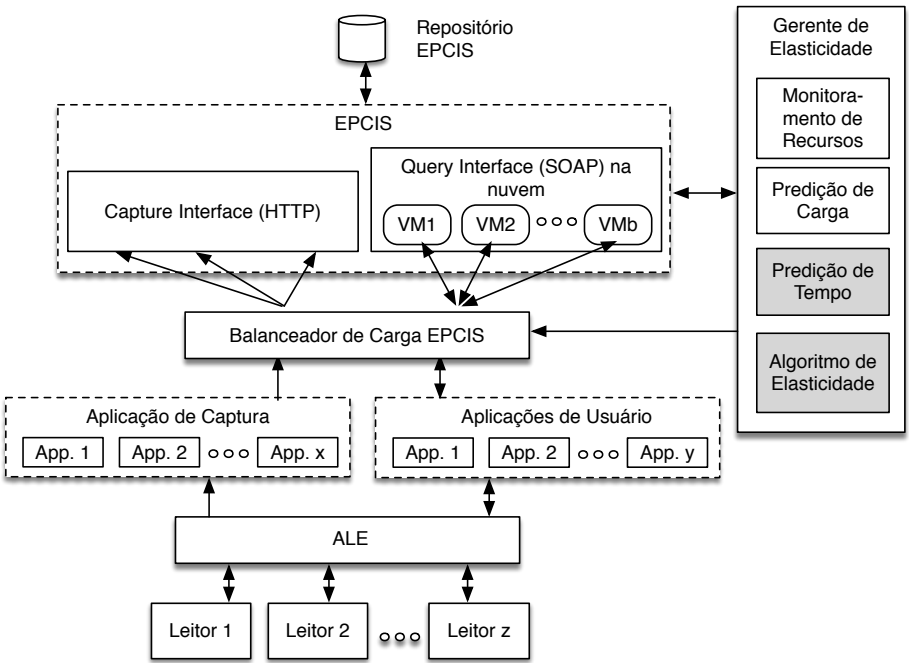


Figura 4. Arquitetura do modelo MMEliot. As caixas cinzas representam as contribuições em relação ao modelo Eliot.

a carga do ambiente. Estes resultados são utilizados posteriormente para definição de *thresholds* superior e inferior. Após a realização dos procedimentos mencionados, o *Gerenciador* aguarda um período de tempo até reiniciar a verificação, coleta e cálculo de valores. O *Gerenciador de Escalabilidade MMEliot* considera dados coletados de máquinas que compõem a infraestrutura de nuvem como entrada para análise de necessidade de alocação ou liberação de recursos computacionais do modelo mestre-escravo. O Gerenciador é composto por 4 submódulos, sendo eles:

- *Monitoramento de recursos*: responsável pela coleta de dados das métricas de CPU e tempo de resposta da nuvem;
- *Previsão de tempo*: responsável por calcular a previsão de tempo do sistema baseando-se nos dados da métrica de tempo de resposta coletada pelo monitor de recursos;
- *Previsão de carga*: responsável por calcular a previsão de carga do sistema baseando-se nos dados da métrica de CPU coletada pelo monitor de recursos;
- *Algoritmo de elasticidade*: responsável por decidir se a alocação ou liberação de recursos da nuvem devem ocorrer tendo como base os valores das predições de tempo e carga.

3.3. Gerenciador de Elasticidade MMEliot

Para aplicar elasticidade reativa multi métrica ao módulo EPCIS foi implementado um *Gerente de Elasticidade* para identificar sobrecargas e sub-utilizações no sistema. A partir da análise das métricas coletadas toma-se a decisão de alocar ou liberar máquinas virtuais na nuvem, em que estas métricas são coletadas a partir do ambiente enquanto o sistema estiver sendo executado. Para gerenciamento da elasticidade foram utilizadas instruções do tipo *regra-condição-ação* e *thresholds* mínimos e máximos pré-estabelecidos

para a previsão de carga (*load prediction*) baseada na métrica de CPU. Diferentemente da previsão de carga, não é definido um *threshold* para a previsão de tempo (*time prediction*) pois é difícil mensurar um valor para o tempo médio de respostas das requisições realizadas. Portanto, decidiu-se utilizar um percentual indicativo de aumento ou diminuição da previsão de tempo (baseada no tempo de resposta) em conjunto com instruções do tipo *regra-condição-ação*.

3.3.1. Monitoramento de recursos

O monitoramento de recursos é realizado periodicamente através da coleta de dados estatísticos do uso de CPU e do tempo de resposta das requisições. Os dados de CPU são coletados por meio da API fornecida pelo provedor de nuvem, enquanto que o tempo de resposta é calculado a cada requisição enviada com sucesso para a *EPCIS Query Interface*. Posteriormente, estes dados são utilizados para o cálculo de previsão de carga e de tempo para a tomada de decisão de elasticidade dos recursos da nuvem. Para coletar o tempo de resposta entre o envio e recebimento das requisições uma série de passos é realizada. Primeiramente, é identificada qual máquina virtual vai receber a próxima requisição. Em seguida, o tempo anterior ao envio e o tempo posterior ao processamento da requisição são coletados. Através desses tempos é possível calcular o tempo de resposta atual para aquela determinada máquina virtual. Por fim, esse tempo é armazenado na máquina virtual correspondente para posterior consulta.

O monitoramento dos recursos em uso nas máquinas virtuais ativas e a tomada de decisão para alocar ou liberar máquinas virtuais na nuvem são realizados periodicamente. Inicialmente, é criado um vetor de máquinas virtuais que irá receber a máquina virtual que receberá a próxima requisição cliente. Em seguida, é recebida uma *string* através do protocolo UDP contendo informações sobre CPU e tráfego de rede em uso por uma máquina virtual e é armazenado em *buffer*. No passo seguinte, é extraído deste *buffer* o índice correspondente a máquina virtual que será utilizado no vetor que armazena dados sobre todas máquinas virtuais. Dados de CPU, tráfego de rede de entrada e de saída da máquina virtual informada previamente também são coletados e armazenados. Particularmente, a previsão de carga e de tempo é calculado através do método *aging*, conforme demonstrado nos Algoritmos 1 e 2, respectivamente. Por fim, é realizada a chamada do Algoritmo 3 descrito na subseção 3.3.4, que avalia as métricas coletadas e decide se aloca ou libera máquinas virtuais quando necessário.

3.3.2. Previsão de carga

A métrica de previsão de carga utiliza a técnica *aging* com o objetivo de evitar falsos-positivos e falsos-negativos nas ações de alocação e liberação de máquinas virtuais. De forma breve, a técnica *aging* atribui um peso maior para observações mais recentes, dividindo por uma potência de 2 para cada elemento subsequente da série histórica. Por exemplo, considerando um *threshold* máximo seja de 81% e observações como 60, 68, 72, 70 e 81, sendo este último a observação mais recente, a previsão de carga *lp* irá informar o valor 73,12 como resultado da seguinte expressão: $lp = \frac{81}{2} + \frac{70}{4} + \frac{72}{8} + \frac{68}{16} + \frac{60}{32}$. Desta forma, o valor "81"será considerado como falso-positivo e não irá disparar qual-

quer ação de elasticidade. Essa abordagem permite suavizar picos, uma vez que a alocação errônea de máquina virtual não irá ocorrer caso sejam analisados os dados históricos [Tanenbaum 2010].

Para tratar o cálculo de previsão de tempo foi desenvolvido o Algoritmo 1, composto por duas variáveis de controle: (i) *occurrence[]*; (ii) *lp_epcis*. O vetor *occurrence[]* armazena valores relacionados a CPU das máquinas virtuais e deve ser inicializado com zero em todos os seus elementos e a cada período de tempo preestabelecido deve ser atualizado com o valor de CPU analisado. A variável *lp_epcis* armazena a previsão de carga momentânea do parâmetro analisado e é atualizada a cada período de tempo preestabelecido aplicando-se a técnica *aging* no vetor *occurrence[]*.

Algoritmo 1: Cálculo de previsão de carga com método *aging*.

Entrada: Vetor occurrence[] contendo o valor dos recursos computacionais
Saída: Previsão de carga com método *aging*

```
1 lp_epcis ← 0;
2 for cada elemento de occurrence[] variando i do
3     lp_epcis ← lp_epcis + occurrence[i] / 2^(i+1);
4 end
```

3.3.3. Previsão de tempo

A técnica de elasticidade também deve ser executada periodicamente considerando a análise de uma previsão de tempo para a métrica de tempo de resposta. Esta métrica utiliza a técnica *aging* com o objetivo de evitar falsos-positivos e falsos-negativos nas ações de alocação e liberação de máquinas virtuais. Para tratar o cálculo de previsão de tempo foi desenvolvido o Algoritmo 2, composto por três variáveis de controle: (i) *occurrence_resp_time[]*; (ii) *tp_epcis*; (iii) *previous_tp_epcis*. A variável *previous_tp_epcis* armazena o valor da previsão de tempo anterior para que posteriormente seja calculado a previsão de tempo atual. O vetor *occurrence_resp_time[]* armazena valores relacionados ao tempo de resposta das máquinas virtuais e deve ser inicializado com zero em todos os seus elementos e a cada período de tempo preestabelecido deve ser atualizado com o valor do tempo de resposta analisado. A variável *tp_epcis* armazena a previsão de tempo momentânea do parâmetro analisado e é atualizada a cada período de tempo preestabelecido aplicando-se a técnica *aging* no vetor *occurrence_resp_time[]*.

3.3.4. Algoritmo de elasticidade

A partir do estudo conduzido por Righi et al. [Righi et al. 2016], optou-se por criar uma técnica de elasticidade que priorizasse a métrica de CPU em relação ao tempo de resposta por causa dos resultados obtidos. Estes resultados demonstraram que o principal gargalo no sistema avaliado foi a métrica de CPU. Ainda sobre os resultados obtidos por Righi et al. [Righi et al. 2016], decidiu-se utilizar a métrica tempo de resposta em conjunto com a métrica de CPU apenas em situações aonde os *thresholds* superior e inferior estavam

Algoritmo 2: Cálculo de previsão de tempo com método *aging*.

Entrada: Vetor occurrence_resp_time[] contendo o valor dos tempos de respostas coletados

Saída: Previsão de tempo com método *aging*

1 $previous\_tp\_epcis \leftarrow current\_tp\_epcis$;
2 **for** $cada\ elemento\ de\ occurrence\_resp\_time[]\ variando\ i$ **do**
3 $\quad current\_tp\_epcis \leftarrow current\_tp\_epcis + \frac{occurrence\_resp\_time[i]}{2^{(i+1)}}$;
4 **end**

próximos de serem atingidos. Para tal, decidiu-se aumentar em 20% o *threshold* inferior e diminuir em 20% o *threhold* superior. O objetivo de alterar estes dois *thresholds* foi proporcionar antecipação da alocação ou liberação de recursos na nuvem.

Para tratar a técnica de elasticidade foi desenvolvido o Algoritmo 3 composto por instruções do tipo *regra-condição-ação*. O cálculo sobre o percentual de aumento ou decréscimo da previsão de tempo atual em relação a previsão de tempo anterior é realizado na linha 1, sendo o resultado desta operação um valor absoluto utilizado nas condições das linhas 3 e 7. É dado prioridade para a previsão de carga *lp_epcis* em relação ao percentual de previsão de tempo *percent_tp_epcis*, conforme exemplificado nas linhas 2 e 6, aonde em ambos casos a primeira condição avaliada verifica se a previsão de carga *lp_epcis* atingiu os *thresholds* superior e inferior. Logo, uma vez que o *threshold* superior ou inferior seja atingido, o *Gerenciador* deve agir de acordo, alocando ou liberando recursos na nuvem. Por outro lado, a previsão de carga *lp_epcis* trabalha em conjunto com o percentual de previsão de tempo *percent_tp_epcis* em situações aonde não foram atingidos os *thresholds* superior ou inferior, conforme demonstrado nas linhas 3 e 7. Na linha 3 é verificado se o percentual de aumento da previsão de tempo *percent_tp_epcis* ultrapassou um percentual preestabelecido e se a previsão de carga *lp_epcis* está próximo do *threshold* superior no caso de alocação de recursos. Já a linha 7 verifica se a previsão de carga *lp_epcis* está próximo *threshold* inferior no caso de liberar recursos na nuvem.

Ainda no Algoritmo 3, caso a previsão de carga *lp_epcis* seja menor que o *threshold* inferior e caso exista mais de uma máquina virtual EPCIS alocada, a máquina virtual com menor carga é desalocada. A mesma regra da quantidade de máquinas virtuais ativas se aplica para a previsão de tempo *percent_tp_epcis* em ambas condições de alocação ou liberação de recursos da nuvem. Para finalizar o Algoritmo, os processos de alocação ou liberação de recursos se comunicam de forma síncrona com o módulo balanceador de carga para que não ocorra de uma máquina virtual que esteja sendo alocada fique ociosa ou então uma requisição seja direcionada para uma máquina recém liberada, gerando falha.

4. Metodologia de Avaliação

Esta Seção apresenta a metodologia de avaliação para o modelo proposto. A subseção 4.1 apresenta os perfis de teste ascendente e descendente utilizados na avaliação. A subseção 4.2 apresenta as métricas utilizadas para geração dos gráficos e para comparação de desempenho entre o modelo proposto e o *Eliot*. Por fim, a subseção 4.3 apresenta os parâmetros de teste utilizados na técnica de elasticidade, sendo os *threholds* e percentual de

Algoritmo 3: Alocação e liberação de máquina virtual.

Entrada: Previsão de carga *lp_epcis*, seus limites superior e inferior, previsão de tempo atual *current_tp_epcis*, previsão de tempo anterior *previous_tp_epcis* e *percent_tp_target*

Saída: Alocação ou liberação de máquina virtual

```
1  percent_tp_epcis ←
   calc_percent_inc_dec_tp(current_tp_epcis, previous_tp_epcis);
2  if ((lp_epcis > high_threshold_epcis) or
3  ((lp_epcis > high_threshold_epcis * 0.8) and (count_allocated_vm >= 1)
   and (percent_tp_epcis > percent_tp_target)) then
4      vm ← cria uma nova máquina virtual EPCIS
5      notifica balanceador de carga EPCIS (adiciona vm)
6  else if ((lp_epcis < low_threshold_epcis) or
7  ((lp_epcis < low_threshold_epcis * 1.2) and
   (percent_tp_epcis < percent_tp_target)) and
   (count_allocated_vm > 1)) then
8      vm ← seleciona máquina virtual com menor carga
9      notifica balanceador de carga EPCIS (remove vm)
10     libera máquina virtual EPCIS (vm)
11 end
```

aumento ou diminuição do tempo de resposta.

4.1. Cenários de Teste

Para realizar a avaliação da estrutura escalável implementada na nuvem, foi posto em prática o fluxo de leitura de dados EPC, com o objetivo de gerar diferentes demandas de consultas por dados EPC, refletindo cenários de uso dos recursos computacionais do *cluster* de máquinas *EPCIS Query Interface* e consequentemente alocação e liberação dessas máquinas virtuais de forma automática, eficiente e transparente para aplicações cliente. Nós baseamos nossa estratégia de avaliação na avaliação realizada pelo *Eliot*. Porém, o fluxo de leitura de dados foi alterado para oferecer escalabilidade e elasticidade ao componente *EPCIS Query Interface*. Os passos para geração de carga de requisições consistem em consultas ao *EPCIS Query Interface* através de dois perfis de teste:

- **Ascendente**: Gera carga de requisições de forma crescente e simultaneamente. Iniciando em uma requisição e terminando com o limite máximo de requisições definido pelo usuário. O objetivo deste perfil de teste é simular um crescimento constante da demanda de recursos computacionais. O resultado esperado é a alocação automática de máquinas *EPCIS Query Interface* conforme a quantidade de requisições cresce;
- **Descendente**: Gera carga decrescente de requisições simultâneas. Inicia-se em uma quantidade máxima de requisições definida pelo usuário e encerra sem nenhuma requisição. O objetivo é avaliar o comportamento do sistema em situações de queda constante da demanda de recursos computacionais. O resultado esperado é a queda pela demanda de requisições e como consequência seja reduzido

a utilização de recursos computacionais, acarretando em liberação de máquinas virtuais *EPCIS Query Interface* da nuvem, conforme a quantidade de requisições diminui.

4.2. Métricas de avaliação

Os resultados obtidos através dos perfis de teste ascendente e descendente para a arquitetura escalável são apresentados primeiramente para o *middleware Eliot* e posteriormente para o modelo MMEliot, em que as seguintes métricas foram utilizados:

- **Quantidade média de requisições por segundo**: representa o total de requisições executadas dividido pelo tempo de execução;
- **Quantidade de máquinas virtuais ativas**: representa a quantidade de máquinas virtuais ativas no *cluster* de máquinas virtuais *EPCIS Query Interface*;
- **Previsão de carga média (%)**: Indica a média de previsão de carga do *cluster EPCIS Query Interface* durante os testes;
- **Previsão de tempo de resposta média (%)**: Indica o momento em que a técnica de elasticidade proposta pelo MMEliot é ativada para situações de alocação ou liberação de recursos da nuvem;
- **Tempo de resposta médio por requisição (milissegundos)**: indica o tempo médio de resposta entre o envio das requisições para o *EPCIS Query Interface* e o retorno da resposta.

Para medir os recursos consumidos da nuvem durante a execução do modelo proposto de forma mais fácil, decidiu-se utilizar a métrica chamada *Energia* (Equação 1). Esta métrica baseia-se na relação próxima entre consumo energético e consumo de recursos. A Equação 1 é composta por duas variáveis: *I* significa a quantidade de máquinas virtuais ativas e *T* denota o tempo em que estas máquinas virtuais ficaram ativas. Por exemplo, considerando uma unidade de tempo em minutos e a execução total da aplicação em 3 minutos, o seguinte resultado pode ser obtido: 1 minuto (1 máquina virtual), 2 minutos (2 máquinas virtuais), 3 minutos (1 máquina virtual), totalizando 8 para a Equação abaixo. Sendo assim, a *Energia* é útil para comparações entre aplicações elásticas.

$$Energia = \sum_{i=1}^{I}(i \ \times \ T(i)) \tag{1}$$

4.3. Parâmetros de teste

A arquitetura com suporte a elasticidade do componente *EPCIS Query Interface* é o objetivo principal deste trabalho. Com a realização dos testes, espera-se avaliar o modelo proposto e identificar vantagens em relação a arquitetura sem suporte a elasticidade. Para o teste com elasticidade do *EPCIS Query Interface*, o MMEliot foi configurado com o *threshold* inferior igual a 30 e o *threshold* superior igual a 70. Estes valores foram escolhidos para que o *middleware* IoT trabalhasse dentro de uma faixa de operação que evitasse perda de pacotes. Em relação ao percentual de aumento ou decréscimo do tempo de resposta foi definido o valor de 20%. Sempre que a previsão de carga *lp_epcis* ultrapasse o *threshold* superior ou inferior será alocada ou liberada uma máquina virtual no cluster de máquinas *EPCIS Query Interface*. Em relação ao percentual de previsão de tempo *percent_tp_epcis*, caso seja atingido o valor de 20% de aumento ou decréscimo da

previsão de tempo, será avaliado se o cenário é propício para alocação ou liberação de máquinas no cluster de *EPCIS Query Interface*. O cluster de máquinas virtuais *EPCIS Query Interfac* possui um limite máximo de 4 máquinas virtuais ativas e limite mínimo de 1 máquina virtual ativa. O monitoramento periódico foi configurado para 10 segundos tento em vista o tempo de atualização da plataforma de nuvem utilizada. Um tempo de monitoramento muito pequeno pode resultar em leituras de valores repetidos dos recursos da nuvem.

4.4. Protótipo e Ambiente de Testes

O *middleware* RFID escolhido como base para o desenvolvimento do MMEliot foi o Fosstrak[5] por seguir o padrão EPCGlobal e por ser *open source*. Entre os diversos Sistemas Gerenciadores de Banco de Dados (SGBD), foi escolhido para implementação o MySQL[6]. A escolha pelo MySQL se deu por conta dos benefícios associados ao uso de SQL, sendo a Atomicidade, Consistência, Isolamento e Durabilidade em suas transações, garantido assim a integridade e confiabilidade dos dados [Bartholomew 2010]. Outro ponto que levou a escolha do MySQL como SGBD se deu por conta de sua compatibilidade com o *middleware* Fosstrak e compatibilidade com a linguagem de programação Java escolhida para o desenvolvimento do MMEliot, sendo uma linguagem fortemente tipada, que segue padrões de orientação a objetos.

O ambiente selecionado para testes foi o laboratório C01 413 localizado no Programa de Pós-Graduação em Computação Aplicada (PIPCA) da UNISINOS. Este laboratório possui 18 computadores homogêneos com processadores de dois núcleos de 2.9 GHz e 4GB de memória, interconectados através de uma rede 100 Mbps. A ferramenta escolhida para gerenciar os recursos computacionais na nuvem foi o OpenNebula[7] por conta de seus benefícios relacionados a simplicidade, confiabilidade, flexibilidade e por ser *open source* [OpenNebula 2016]. Dentre todas máquinas disponíveis, foram utilizadas 9 máquinas para a configuração da plataforma na nuvem OpenNebula, dos quais um deles foi instalado o servidor OpenNebula servindo de *front-end* da nuvem. Já o restante das máquinas foram utilizados como nós para executar máquinas virtuais. Dentre os oito nós OpenNebula, sete foram alocados para máquinas virtuais *EPCIS Query Interface* e uma delas foi adicionado o repositório *EPCIS* contendo o MySQL.

5. Resultados

Após a estrutura computacional ter sido implantada na nuvem, foram realizados diversos testes para garantir que os componentes do *middleware Eliot* estavam funcionando de acordo com o esperado. Dentre os testes realizados, foi dado foco na leitura de dados EPC através da aplicação cliente utilizada para testes acessando de forma transparente o *cluster* de máquinas virtuais que executam o *EPCIS Query Interface*. Este teste foi concluído com sucesso uma vez que dados EPC armazenados no repositório *EPCIS* foram retornados com sucesso. A partir da conclusão dos testes realizados, partiu-se para a execução dos cenários de testes mencionados na seção 4.

[5]http://fosstrak.github.io/
[6]https://www.mysql.com/
[7]http://opennebula.org/

5.1. Carga Ascendente

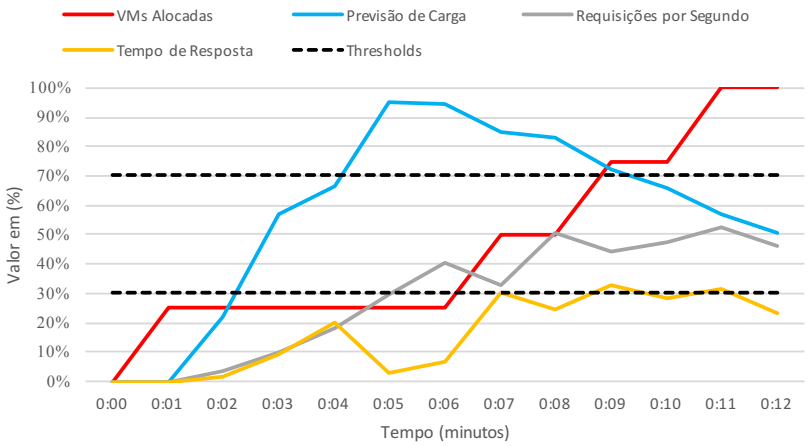
A Figura 5 apresenta o comportamento dos testes relacionados ao perfil de teste ascendente para a arquitetura escalável com no máximo 210 requisições simultâneas. É possível verificar que na Figura 5 (b), os indicadores do tempo de resposta e da quantidade de requisições simultâneas se desprende, ao contrário do apresentado na Figura 5 (a). Por outro lado, ambas execuções demonstradas nas figuras apresentam comportamento semelhante em relação aos indicadores de quantidade de máquinas virtuais ativas e requisições por segundo, apresentando dependência. É possível verificar que o indicador de quantidade de máquinas virtuais ativas na Figura 5 (a) apenas instancia uma nova máquina virtual no minuto 6, ao contrário do comportamento apresentado na Figura 5 (b) que instancia uma nova máquina virtual no minuto 4. Esta diferença de 2 minutos ocorre por conta da técnica de elasticidade proposta pelo MMEliot apresentada na subseção 3.3.4. Através desta técnica foi possível verificar um aumento no tempo de resposta durante os 2 primeiros minutos do teste e aliado a previsão de carga próxima do *threshold* superior foi constatado a necessidade de alocação de recursos na nuvem, com isso, no minuto 4 o indicador de VMs alocadas atingiu 50%, ou 2 máquinas virtuais ativas. Apesar dos resultados apresentados na Figura 5 (b) demonstrarem a eficiência do modelo proposto, o mesmo não se aplica ao comportamento esperado dos indicadores requisições por segundo e tempo de resposta, pois era esperado que a linha de ambos ficassem próximas.

5.2. Carga Descendente

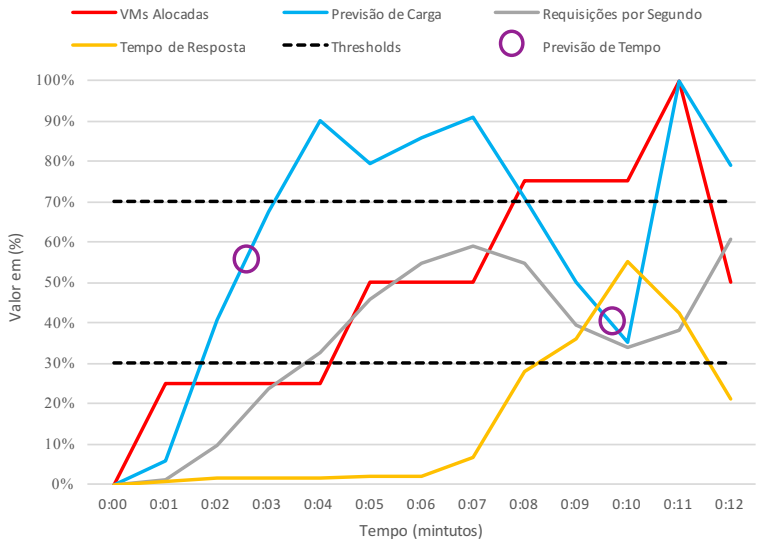
A Figura 6 apresenta o comportamento dos testes do perfil descendente para a arquitetura escalável com no máximo 210 requisições simultâneas. É possível verificar em ambas partes da figura que as linhas dos indicadores de previsão de carga e requisições por segundo são próximas, o que caracteriza uma tendência. As duas partes da Figura 6 apresentam diferença em relação ao indicador de máquinas virtuais ativas, pois a Figura 6 (a) inicia o processo de instanciação de uma nova máquina virtual no minuto 2, enquanto a Figura 6 (b) começa o mesmo processo no minuto 3. Por outro lado, é possível verificar que no minuto 6 da Figura 6 (b) inicia-se o processo de liberação de recursos computacionais, pois o tempo de resposta diminuiu consideravelmente entre os minutos 5 e 6 indicando que as máquinas virtuais ativas estavam com recursos disponíveis. Logo, não foi necessário chegar ao *threshold* inferior para iniciar o processo de liberação de recursos na nuvem conforme demonstrado na Figura 6 (b).

5.3. Discussão dos resultados

Com base nos testes realizados no modelo MMEliot, pode-se identificar momentos em que a técnica de elasticidade proposta agiu de acordo com o esperado, conforme demonstrado na Tabela 2. O tempo de saturação das máquinas virtuais após atingir o *threhold* superior foi parecido entre os testes realizados. Entretanto, o modelo proposto no perfil de teste ascendente teve melhor desempenho se comparado ao *middleware Eliot*, pois ele ficou meio minuto a menos saturado. O modelo proposto não teve bom desempenho em relação ao tempo de resposta, possuindo um valor maior do que o *middleware Eliot* nos dois perfis de teste avaliados. O modelo proposto teve melhor desempenho em relação ao *Eliot* em relação ao percentual de pacotes perdidos. Isto ocorreu pois as máquinas virtuais do MMEliot ficaram menos tempo saturadas se comparado ao *Eliot* no perfil de teste ascendente. O modelo proposto teve melhor desempenho sobre a energia consumida, logo,



(a) Eliot



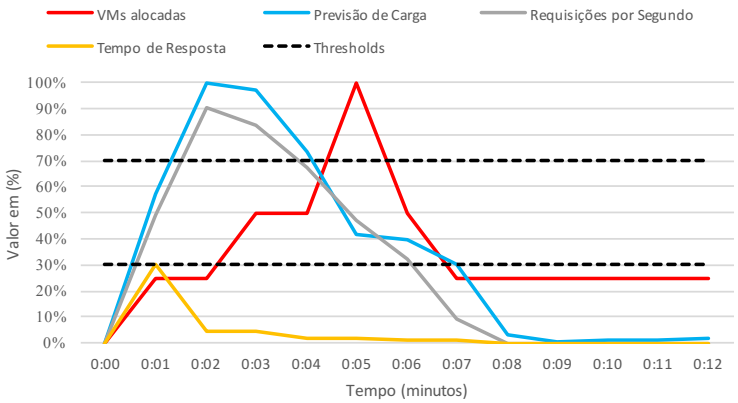
(b) MMEliot

Figura 5. Avaliação dos *middlewares* (a) Eliot e (b) MMEliot com carga ascendente com no máximo 210 requisições simultâneas.

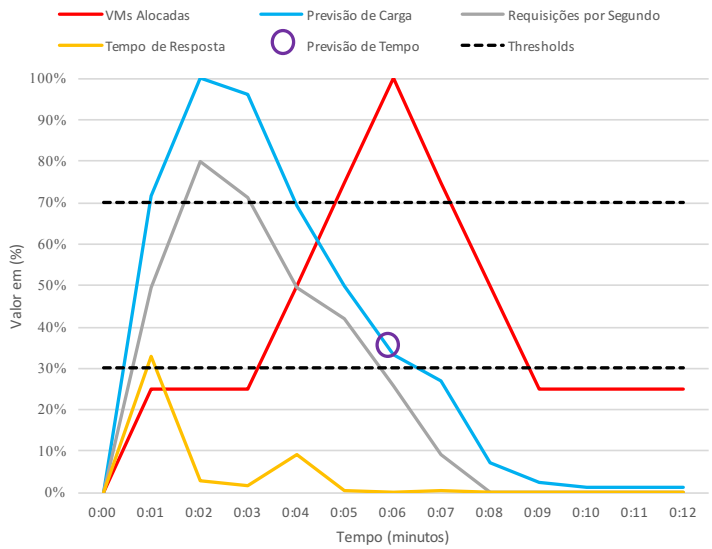
o MMEliot proporcionou melhor aproveitamento dos recursos computacionais. Por fim, o MMEliot obteve melhorias satisfatórias em relação ao *middleware Eliot*, pois foi possível adiantar a alocação e liberação de recursos da nuvem, evitando futuros gargalos e economizando recursos, respectivamente.

6. Conclusão

Atualmente, percebe-se que trabalhos presentes na literatura de Internet das Coisas focam em estratégias para lidar com escalabilidade e balanceamento de carga. Entretanto, nenhum deles combina múltiplas métricas para a tomada de decisão quanto a elasticidade. Observando essa lacuna, o presente artigo apresentou MMEliot como uma extensão do modelo de elasticidade em nuvem Eliot. Diferentemente de *Eliot*, MMELiot permite o monitoramento das métricas CPU e tempo de resposta para a tomada de decisão quanto à



(a) Eliot



(b) MMEliot

Figura 6. AAvaliação dos *middlewares* (a) Eliot e (b) MMEliot com carga descendente com no máximo 210 requisições simultâneas.

Tabela 2. Comparação do desempenho entre MMEliot e Eliot.

Carga	Gerenciador	Tempo de saturação (min)	Tempo de resposta (ms)	Perda de pacotes (%)	Energia
Ascendente	Eliot	5	465	10,9%	24
	MMEliot	4,5	337	9,32%	15
Descendente	Eliot	3	147	25,3%	18
	MMEliot	3	99	24,4%	19

elasticidade de recursos. Uma vez que as métricas sejam coletadas, a técnica *Aging* é aplicada em ambas com o objetivo de evitar falsos-positivos e falsos-negativos nas ações de alocação e liberação de máquinas virtuais. Sendo assim, algoritmos que usam as duas métricas citadas previamente juntamente com a técnica de *Aging* representam a contribuição científica do presente artigo.

A metodologia de avaliação considerou o desenvolvido de um protótipo baseado no modelo MMEliot, o qual permitiu avaliar a técnica de elasticidade reativa multi métrica no sentido de antecipar a alocação e liberação de recursos da nuvem. Os testes mostraram que MMEliot consegue de fato antecipar as ações de elasticidade, acarretando num melhor tempo de execução para a aplicação em relação ao *Eliot*. Em adição, MMEliot ofereceu melhor gerenciamento da energia, além de perder menos pacotes do que o *Eliot* em situações de saturação das máquinas virtuais ativas. Apesar do foco do artigo ser Internet das Coisas, acredita-se que a arquitetura e os algoritmos apresentados sejam também pertinentes em outros contextos, como comércio eletrônico, computação de alto desempenho e suporte para computação ubíqua.

A cargo de trabalhos futuros, sugere-se a utilização de outras métricas como uso de memória e dados da rede de comunicação. Em adição, os autores pretendem executar o protótipo MMeliot em outras aplicações de Internet das Coisas, capturando dados da empresa de logística FEDEX, por exemplo. Por fim, destaca-se a possibilidade de exploração da elasticidade proativa no contexto de Internet das Coisas, não precisando mais do uso de *thresholds* na configuração do motor de elasticidade.

Agradecimentos

Este trabalho foi parcialmente suportado pelos seguintes órgãos brasileiros de fomento: CAPES, CNPq e FAPERGS.

Referências

Atzori, L., Iera, A., and Morabito, G. (2010). The internet of things: A survey. *Comput. Netw.*, 54(15):2787–2805.

Bartholomew, D. (2010). Sql vs nosql. Disponível em: <http://www.linuxjournal.com/article/10770>. Acesso em: novembro. 2016.

Byun, J. and Kim, D. (2015). Oliot epcis: New epc information service and challenges towards the internet of things. In *2015 IEEE International Conference on RFID (RFID)*, pages 70–77. IEEE.

Galante, G. and d. Bona, L. C. E. (2012). A survey on cloud computing elasticity. In *2012 IEEE Fifth International Conference on Utility and Cloud Computing*, pages 263–270.

Gomes, M., da Rosa Righi, R., and da Costa, C. A. (2014). Internet of things scalability: Analyzing the bottlenecks and proposing alternatives. In *2014 6th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT)*, pages 269–276. IEEE.

Gubbi, J., Buyya, R., Marusic, S., and Palaniswami, M. (2013). Internet of things (iot): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, 29(7):1645 – 1660. Including Special sections: Cyber-enabled Distributed

Computing for Ubiquitous Cloud and Network Services & Cloud Computing and Scientific Applications Ů Big Data, Scalable Analytics, and Beyond.

Liu, F., Jie, Y., and Hu, W. (2008). Distributed ale in rfid middleware. In *2008 4th International Conference on Wireless Communications, Networking and Mobile Computing*, pages 1–5. IEEE.

Lorido-Botran, T., Miguel-Alonso, J., and Lozano, J. A. (2014). A review of auto-scaling techniques for elastic applications in cloud environments. *Journal of Grid Computing*, 12(4):559–592.

Mell, P. M. and Grance, T. (2011). Sp 800-145. the nist definition of cloud computing. Technical report, ., Gaithersburg, MD, United States.

Nguyen, H. M., Kim, S. H., Le, D. T., Heo, S., Im, J., and Kim, D. (2015a). Epcloud flow: Load prediction and migration optimizations for epc network on cloud. In *2015 IEEE 8th International Conference on Cloud Computing*, pages 981–984. IEEE.

Nguyen, H. M., Kim, S. H., Le, D. T., Heo, S., Im, J., and Kim, D. (2015b). Optimizations for rfid-based iot applications on the cloud. In *Internet of Things (IOT), 2015 5th International Conference on the*, pages 80–87.

Nikravesh, A. Y., Ajila, S. A., and Lung, C.-H. (2015). Towards an autonomic auto-scaling prediction system for cloud resource provisioning. In *Proceedings of the 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS '15, pages 35–45, Piscataway, NJ, USA. IEEE Press.

OpenNebula (2016). Why opennebula? Disponível em: <http://opennebula.org/about/why/>. Acesso em: novembro. 2016.

Righi, R. R., Reis, E. S., Rostirolla, G., Costa, C. A., and Alberti, A. M. (2016). Exploring cloud elasticity on developing an epcglobal-compliant middleware. In *2016 IEEE International Conference on RFID (RFID)*, pages 1–4.

Schmidt, L., Mitton, N., Simplot-Ryl, D., Dagher, R., and Quilez, R. (2011). Dht-based distributed ale engine in rfid middleware. In *RFID-Technologies and Applications (RFID-TA), 2011 IEEE International Conference on*, pages 319–326. IEEE.

Tanenbaum, A. S. (2010). *Computer Networks*. Prentice Hall, fifth edition.

Welbourne, E., Battle, L., Cole, G., Gould, K., Rector, K., Raymer, S., Balazinska, M., and Borriello, G. (2009). Building the internet of things using rfid: The rfid ecosystem experience. *IEEE Internet Computing*, 13(3):48–55.

Ziekow, H., Fabian, B., Müller, C., and Günther, O. (2010). Rfid in the cloud: A service for high-speed data access in distributed value chains. In *AMCIS*, page 256.