On Exploring Proactive Cloud Elasticity for Internet of Things Demands

Vinicius Facco Rodrigues, Everton Correa, Cristiano André da Costa, Rodrigo da Rosa Righi
Programa de Pós Graduação em Computação Aplicada
Universidade do Vale do Rio dos Sinos
São Leopoldo, Rio Grande do Sul, Brazil – 93022–000
Email: {vfrodrigues,ecorrea,cac,rrrighi}@unisinos.br

***Abstract*—Today, Internet of Things (IoT) is an emergent concept in which billions of devices are connected to Internet capable of producing and exchanging data. One of the most used technologies in this area regards to the Radio Frequency Identification (RFID). It can produce large amount of data from many things like objects, persons and assets. Thus, it is needed middlewares which must support processing in large scales. However, the state-of-the-art does not present satisfactory solutions in which this kind of middlewares are capable of adapt themselves according to processing demands. In this context, this article presents a proactive cloud elasticity model called Proliot aiming at providing scalability to IoT middlewares. Proliot is capable of predicting load behavior combining time series techniques. In addition, it adapts cloud resources beforehand an overload or underload situation occurs. We evaluated our model comparing results with a reactive elasticity model. In our experiments, Proliot achieved best performance up to 76% when compared to Eliot.**

***Index Terms*—Internet of Things; cloud elasticity; middleware; RFID; EPCGlobal.**

I. Introdução

O termo Internet das Coisas – do inglês *Internet of Things* (IoT) – foi introduzido ao mundo por Kevin Ashton, em 1999, durante uma apresentação na Procter & Gamble (P&G) [1]. O RFID Group define Internet das Coisas como uma rede mundial, baseada em protocolos de comunicação padrão, de objetos interconectados e unicamente identificáveis [3]. Segundo Gubbi et al. [2], embora o termo seja utilizado de forma bem mais abrangente atualmente, incluindo aplicações de saúde, logística, segurança, agricultura, entre outros, o objetivo principal de fazer com que os computadores capturem as informações do mundo real sem ajuda de intervenção humana continua o mesmo. Tais sistemas podem gerar grandes quantidades de dados que devem ser posteriormente processados [5]. Dentre as diversas tecnologias IoT, como *Near Field Communications* (NFC) e *Wireless Sensor and Actuator Networks* (WSAN), RFID continua sendo a mais utilizada, principalmente por sua maturidade, baixo custo e, consequentemente, forte suporte do mercado [4].

Com o crescimento da adoção de tecnologias RFID, em 2003 a Global Standards One (GS1) desenvolveu o padrão EPCglobal [6] buscando normalizar sistemas que utilizam tais tecnologias. Atualmente, diversos sistemas IoT adotam o padrão EPCGlobal juntamente com tecnologias RFID. Em virtude disso, essa alta demanda de informação e a complexidade do sistema RFID exigem uma estrutura computacional robusta capaz de manter os níveis do serviço em uma qualidade aceitável. A robustez do sistema não se deve apenas à capacidade de processar grandes quantidades de dados, mas também a tolerância à falhas, alta disponibilidade e escalabilidade. Neste contexto, uma possível solução é o uso da Computação em Nuvem [7] dado sua capacidade de ajustar recursos conforme a demanda. Esta funcionalidade, chamada de elasticidade, permite que recursos computacionais sejam reorganizados através da adição ou remoção de recursos a qualquer momento. Isso permite que o sistema se ajuste à carga e evite subutilização ou sobrecarga.

Embora existam benefícios para sistemas IoT, a elasticidade em nuvem é mais largamente explorada em arquiteturas cliente-servidor, como vídeo sob-demanda, lojas online, aplicações BOINC (*Berkeley Open Infrastructure for Network Computing*) [8], governança eletrônica e *Web Services* [9]. Apesar de diversos estudos na direção de criar *middlewares* RFID compatíveis com o padrão EPCglobal, ainda há uma lacuna no que diz respeito à elasticidade de recursos para tais sistemas. Neste contexto, esse artigo propõe um modelo de elasticidade proativa para *middlewares* RFID padrão EPCglobal em Nuvem. Foi desenvolvido um gerenciador chamado Proliot que utiliza modelos de séries temporais para prever a carga de utilização dos recursos em que o *middleware* está sendo executado. É utilizada a elasticidade proativa como forma de mitigar o tempo de preparação e inicialização de um novo recurso computacional. Ao prever que o sistema estará sobrecarregado daqui alguns minutos, é possível já começar a alocação de uma nova máquina virtual. Dessa forma quando o sistema estiver de fato sobrecarregado o novo recurso já estará pronto para utilização. Os experimentos realizados demostraram que Proliot pode obter resultados muito semelhantes e em alguns casos até melhor em comparação com um sistema que utiliza a elasticidade reativa.

Este artigo está organizado em 8 seções. A seção II apresenta uma análise dos trabalhos relacionados com o tema proposto nesse artigo. Em seguida, o modelo Proliot é apresentado em detalhes na seção III. A seção IV é destinada para descrição da implementação do protótipo desenvolvido como prova de conceito da arquitetura proposta. Visando avaliar o modelo, a seção V detalha os resultados obtidos pelos experimentos

978-1-5386-3057-0/17/$31.00

realizados. Por fim, as considerações finais e oportunidades de trabalhos futuros são apresentadas na seção VI.

II. Trabalhos Relacionados

Nesta seção são apresentadas abordagens na área de *middleware* RFID compatível com os padrões e interfaces definidas pela EPCglobal com o intuito de verificar o estado da arte desse tema. Este estudo tem como principal objetivo comparar os diversos *middlewares* que utilizam as definições EPCglobal e com isso verificar características comuns e possíveis melhorias na área de balanceamento de carga e elasticidade.

Wang, Sung e Kim [10] propõe a arquitetura ESN, que utiliza as interfaces definidas pela EPCglobal com algumas extensões para suportar os dados advindos dos sensores de temperatura, umidade, pressão, entre outros possíveis em uma WSN. A arquitetura do *middleware* ESN é orientada a eventos e utiliza processamento de eventos complexos (CEP) para combinar os dados dos diferentes sensores. Assim como o ESN, o *RFID middleware System* (RMS) [11] também visa prover um *middleware* com capacidade de processamento de eventos complexos (CEP). No caso do RMS as informações de contexto não derivam de redes WSN, mas sim arquivos de fornecedor, rotinas de transporte, parceiros comerciais, ou até mesmo sistemas de gestão empresarial.

ETRI RFID Middleware Software Platform [12] é um *middleware* multicamada desenvolvido utilizando o *Avalon Framework*. O *middleware* é dividido em três camadas: monitoramento e gerenciamento de dispositivos; monitoramento e gerenciamento de dados; e integração com aplicações. Por outro lado, AspireRFID [13] é um *middleware* cujo objetivo é estender e modularizar os componentes definidos na arquitetura EPCglobal. Uma funcionalidade chave desse *middleware* é a capacidade de modularizar as camadas do sistema em *building blocks* que podem ser ligados ou desligados de acordo com a complexidade e tamanho da aplicação que vai utilizá-lo.

RFID middleware V1.0 [14] foi projetado com o objetivo de facilitar o gerenciamento de leitores RFID através de um sistema dinâmico de leitores virtuais chamado de *Virtual Reader*. O *middleware* busca principalmente diminuir o tempo e a complexidade da configuração de novos leitores ao sistema RFID oferecendo uma camada extra aos componentes padrões do EPCglobal que gerencie dinamicamente os leitores dos diferentes fornecedores. Outra abordagem focando EPCglobal, chamada LIT Middleware, é apresentada por Ashad et al. [15]. LIT Middleware é um *middleware* robusto construído de acordo com as definições EPCglobal e implementa tanto as interfaces *ALE* quanto *EPCIS*. O *middleware* oferece diversas técnicas de alta performance e gerenciamento dos dados, como o sistema de *Continuous Query Process* que serve para receber as requisições de aplicações agrupar os dados EPC capturados, filtrados e agrupados.

Fosstrak [16], também conhecido como Assada, é um dos *middlewares* RFID que implementam as definições EPCglobal mais utilizados no mundo acadêmico. Entre os requerimentos cobertos pelo Fosstrak podem ser destacados sua capacidade de disseminação dos dados RFID, mecanismos de filtragem e agregação dos dados tanto em nível de leitor quanto em nível de *middleware*, suporte a diversos tipos e modelos de leitores e recursos que interpretam os dados capturados e os transformam em eventos de negócio. O Fosstrak foi desenvolvido em três módulos básicos: leitor, filtro e agregação, e o *EPC Information Service* (EPCIS).

Devido a popularidade de Fosstrak, diversos outros trabalhos se basearam em sua arquitetura [17], [18], [19]. Oliot [17], [18] é um projeto da Auto-ID Lab, KAIST e busca integrar os padrões RFID da EPCglobal com outros protocolos IoT como código de barra, ZigBee e 6LoWPAN. Diferentemente do Oliot, o foco de Eliot [19] não é na integração de sensores e outras tecnologias IoT mas sim no balanceamento de carga e elasticidade do modulo EPCIS. Na arquitetura do Eliot, o Fosstrak foi utilizado como uma caixa preta, ou seja, não houve modificações no código fonte do Fosstrak, mas apenas na sua forma de implantação – distribuída ao invés de centralizada.

Visando uma analise comparativa entre os trabalhos analisados, a Tabela I apresenta as principais características de cada um. Apenas um dos projetos apresenta soluções de balanceamento de carga e elasticidade. Por outro lado, todos apresentam algum tipo de solução para o gerenciamento de leitores RFID, extração dos dados e processamento eficiente das informações. Alguns ainda implementam o modulo EPCIS para contextualização e distribuição dos eventos para aplicações e parceiros. Nota-se também uma grande tendência no desenvolvimento de soluções que buscam integrar outras tecnologias IoT ao *middleware* padrão EPCglobal e dessa forma reaproveitar a infraestrutura. Um ponto que chama atenção é que diversos autores salientaram que sistemas RFID geram grandes volumes de dados. Cada *middleware* buscou aplicar algum tipo de método de gerencia de eventos e dados afim de propor uma solução robusta em casos de alta demanda.

Embora a utilização de Computação na Nuvem seja uma solução viável e extremamente recomendável para processar o grande volume de dados gerados por sistemas RFID, Righi et al. [19] fazem uma ressalva quanto a utilização desse modelo. O resultado dos testes executados no modelo do Eliot distribuído e elástico foram bem superiores em comparação a versão não elástica e não distribuída, entretanto esses testes foram realizados com todas VMs já alocadas e ativas, apenas alterando o número de recursos sendo utilizados em uma tabela de recursos. O teste foi realizado dessa forma para desprezar o tempo de alocação e ativação de uma VM, que ficou entre três e cinco minutos. Apesar de válido, o teste demonstra que em casos de uso real essa demora na alocação gera uma baixa reatividade do sistema, bem como pode ocasionar negação de serviço. Uma possível solução para contornar essa demora seria a utilização de um método de elasticidade proativo, ao invés do atual reativo, que pudesse prever uma sobrecarga do sistema antes da mesma ocorrer e com isso começar os processos de alocação e ativação das VMs antecipadamente. Dessa forma no momento que a demanda realmente estiver alta os recursos já estarão prontos para o uso.

TABLE I
COMPARAÇÃO ENTRE *middlewares* RFID PADRÃO EPCGLOBAL

Middleware EPCglobal	*EPCglobal Reader Interface*	*EPCglobal ALE Interface*	*EPCglobal EPCIS Interface*	Balanceamento de carga	Elasticidade Proativa	Implementação *Open Source*
[10]	✓	✓	✓	×	×	✓
[11]	✓	✓	×	×	×	×
[12]	×	✓	×	×	×	×
[13]	✓	✓	✓	×	×	✓
[14]	✓	✓	×	×	×	×
[15]	✓	✓	✓	×	×	×
[16]	✓	✓	✓	×	×	✓
[18]	✓	✓	✓	×	×	✓
[19]	✓	✓	✓	✓	×	✓

III. MODELO PROLIOT

Nesta seção será apresentado o modelo Proliot, gerenciador de elasticidade proativa para *middlewares* IoT padrão EPCglobal. Primeiramente, são detalhadas as decisões que direcionam o modelo. Em seguida, a arquitetura do modelo é descrita. E por fim são apresentados os mecanismos de predição e elasticidade do gerenciador.

A. Decisões de projeto

Primeiramente, o gerenciador deve ser transparente tanto para o usuário quanto para o *middleware* ao qual ele está aplicando a elasticidade. Ou seja, o gerenciador não deve exigir nenhum tipo de configuração prévia do usuário e nem alterações no código do *middleware*. O gerenciador é direcionado em específico a *middlewares* RFID que seguem as especificações definidas pela EPCglobal. *Middleware* e gerenciador devem ser implantados na Nuvem. E o gerenciador deve ser capaz, através de uma API disponibilizada pelo provedor da Nuvem, de alocar e desalocar VMs de acordo com a utilização dos recursos computacionais já alocados. A elasticidade aplicada pelo gerenciador é apenas horizontal. Ou seja, a elasticidade é aplicada apenas na alocação e desalocação de VMs, e não na configuração de hardware das VMs.

O gerenciador deve usar elasticidade proativa e inicializar o processo de alocação de um novo recurso antecipadamente ao prever uma sobrecarga do sistema no futuro. Nesse contexto, sobrecarga do sistema se entende como um percentual de uso de CPU elevado e que cause negação do serviço e/ou perda de pacotes. Deve ser utilizado o modelo estatístico de séries temporais ARIMA para a previsão das métricas futuras que serão utilizadas nas tomadas de decisão do gerenciador. A métrica usada no Proliot, e a qual será feita a predição, é a carga média de CPU das VMs alocadas. O modelo Eliot, introduzido por [19], apresenta um gerenciador de elasticidade **reativa** para *middlewares* IoT, e é utilizado como base para o modelo Proliot. O Eliot também foi ser utilizado para comparação de resultados entre o modelo reativo versus o modelo proativo. O Eliot aplica a elasticidade reativa no *middleware* Fosstrak, mais especificamente no componente EPCIS Query Interface do Fosstrak. Sendo assim, o Proliot também irá aplicar a elasticidade sobre o mesmo componente do Fosstrak.

B. Arquitetura

A Figura 1 apresenta a arquitura simplificada de um sistema RFID, componentes básicos de um *middleware* padrão EPCglobal e - destacado em vermelho - o gerenciador de elasticidade proativa e balanceador de carga Proliot. Resumidamente, os dados das etiquetas RFID lidos pelos leitores RFID são enviados para o componente ALE do *middleware*, que é responsável por filtrar e agrupar os dados brutos transformando-os em eventos. Uma vez processados, os eventos são enviados parar a Capturing Application, que aplica regras de negocio definidas pelo usuário, como adicionar informações de endereço ou disparar um alarme. A Capturing Application então utiliza a Capture Interface para gravar os dados dos eventos no banco de dados. Uma vez que os dados estão gravados, eles podem ser consumidos a qualquer momento através de consultas enviadas a Query Interface.

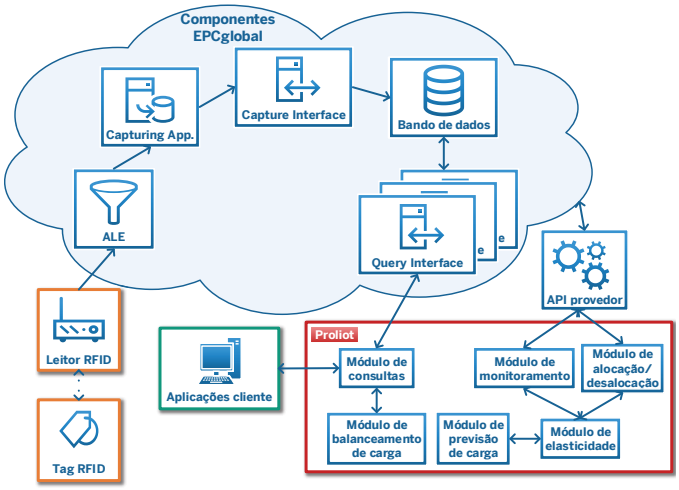


Fig. 1. Arquitetura do modelo Proliot.

Em uma implantação regular de um *middleware* padrão EPCglogal, todos seus componentes são instalados em um único equipamento de forma centralizada e sem oferecer elasticidade ou balanceamento de carga. Na arquitetura proposta no Proliot se sugere a implantação desses componentes de forma distribuída, onde cada componente é instalado em um VM separada. Ademais, é adicionado o gerenciador de elasticidade Proliot que é responsável por monitorar as VMs do componente Query Interface, adicionar ou remover VMs desse componente de acordo com a carga do sistema, receber requisições de consulta das aplicações cliente e encaminhar as requisições para as VMs Query Interface ativas. Para realizar essas tarefas o Proliot conta com os seguintes módulos: (i) monitoramento; (ii) previsão de carga; (iii) elasticidade; (iv)

alocação/desalocação de recursos; (v) recebimento/envio de consultas; e (vi) balanceamento de carga.

O módulo de monitoramento é responsável por fazer chamadas periódicas à API do provedor de nuvem consultando o estado atual das VMs. Esses dados são utilizados nos cálculos de previsão de carga. Já o módulo de previsão de carga é o componente que agrupa os dados das VMs e executa a previsão das métricas através do modelo estatístico de séries temporais ARIMA. Por sua vez, o módulo de elasticidade é responsável por analisar as métricas calculadas pelo módulo de previsão de carga, juntamente com outras métricas, e decidir qual ação deve ser tomada: alocação de um novo recurso, desalocação do recurso menos utilizado ou nenhuma ação. Uma vez que o módulo de elasticidade toma uma decisão ele dispara o módulo de alocação/desalocação que contata a API do provedor da nuvem para executar a ação desejada e depois atualiza os atributos internos do gerenciador para que aquele recurso seja removido do pool de VMs utilizáveis. Por outro lado, o módulo de recebimento/envio de consultas é o responsável por coordenar a comunicação do modelo. Proliot conta com um servidor HTTP que deve receber as requisições das aplicações clientes e então, utilizando um cliente HTTP, encaminha-las para uma das VMs ativas. Uma vez que a VM processe a requisição, a resposta é então enviada ao Proliot, que por sua vez encaminha à aplicação cliente. A requisição enviada ao Proliot deve ser exatamente igual a uma requisição que se enviaria diretamente a Query Interface. Nenhum parâmetro ou modificação é necessária. Por fim, para decidir a qual VM encaminhar a requisição recebida, o módulo de recebimento/envio consulta o módulo de balanceamento de carga que utiliza o algoritmo de *round-robin* para escolher a VM destino. Nas subseções seguintes serão abordados em mais detalhes os módulos de previsão de carga e elasticidade.

C. Módulo de previsão de carga

O módulo de previsão de carga é o principal componente de Proliot. É esse módulo que atribui a característica proativa ao gerenciador elástico. A Figura 2 ilustra a diferença entre um modelo elástico reativo e proativo. No modelo reativo, ilustrado na Figura 2 (a), o sistema atua com *thresholds* inferior e superior para ativar a elasticidade. Quando o índice da métrica analisada - no caso do Proliot a métrica aplicada é utilização de CPU - ultrapassada o *threshold* inferior, uma VM é removida. Quando a métrica ultrapassa o *threshold* superior, uma nova VM é adicionada. O problema desse modelo, e que é destacado na Figura 2 (a), é que o tempo de alocação de uma nova VM é um tempo considerável. Nos testes realizados com a VM do componente Query Interface, o tempo de alocação da VM foi de aproximadamente 150 segundos. Como o gerenciador elástico reativo só inica a alocação de uma nova VM quando o sistema já está sobrecarregado, isso significa que o sistema vai continuar sobrecarregado por pelo menos mais 150 segundos. Uma forma de contornar esse problema é iniciando a alocação da nova VM antes do sistema estar sobrecarregado. E para isso é necessário aplicar o modelo proativo.

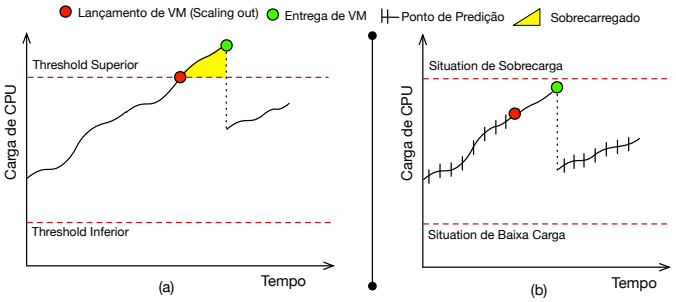


Fig. 2. Modelos de elasticidade: (a) reativo; (b) proativo.

A Figura 2 (b) demonstra o funcionamento do modelo proativo. Nesse modelo o gerenciador coleta dados históricos da métrica analisada e aplica esses dados à um algoritmo de predição. A predição deve ser feita levando em consideração o tempo necessário para alocação da VM. No caso do Proliot, as observações das métricas são feitas a cada 10 segundos. Ou seja, o ciclo de análise do gerenciador é de 10 segundo. Considerando que uma nova VM do componente Query Interface leva 150 segundos para ser alocada e o ciclo de análise é de 10 segundos, então o gerenciador precisa prever qual será o valor da métrica daqui a 15 ciclos. Caso o gerenciador identifique uma situação de sobrecarga 15 ciclos no futuro, então ele inicia imediatamente a preparação do novo recurso. Dessa forma, quando o sistema estiver quase a ponto de sobrecarga, o novo recurso vai estar disponível e vai fazer com que o sistema estabilize novamente. Em um exemplo prático: no segundo 220 o gerenciador coleta o valor de utilização da CPU atual, agrupa o valor atual com os valores das observações anteriores e envia os dados para o algoritmo de predição. O algoritmo de predição por sua vez analisa os valores recebidos e gera a predição de carga do sistema para o segundo 370. Se essa carga prevista for acima de 80%, então o gerenciador inicia a alocação de uma nova VM do componente Query Interface.

O algoritmo de predição utilizado no Proliot foi o modelo *autoregressive integrated moving average* (ARIMA). O ARIMA foi escolhido como modelo preditivo pois, segundo Dinda e O'Hallaron [20], é um modelo comprovadamente efetivo para predição de utilização de CPU. Além disso, ARIMA também é um modelo bem conhecido no meio acadêmico e amplamente utilizado para predição em várias áreas de aplicação [21]. Por fim, a execução do ARIMA também se mostrou mais rápida, precisa e com necessidade de menos dados históricos que outros modelos preditivos testados. O modelo ARIMA é formado pela combinação de dois modelos distintos: autoregressive (AR) e moving average (MA). Esses modelos são combinados juntos ao grau de integração. Para a utilização do ARIMA é necessário informar a ordem desejada para cada um desses componentes. A ordem deve ser escolhida de acordo com perfil da série temporal informada. No Proliot, foi utilizada a função *auto.arima* do pacote de *forecast* do R[1]

[1]Ambiente de software voltado para computação estatistia. https://www.r-project.org/

que aplica um pré-processamento na série temporal informada e define qual a melhor ordem do ARIMA para aqueles dados.

A cada ciclo de análise do gerenciador, o resultado da média aritmética das CPUs ativas é armazenado e serve como base para a predição do ARIMA. Segundo [22], são necessários no mínimo 5 valores em uma série temporal para que o ARIMA seja capaz de fazer a predição. No Proliot, o mínimo histórico utilizado foi de 8 ciclos, pois nos testes foi o valor que demonstrou o melhor custo-benefício em relação a precisão. Durante esses 8 ciclos nenhuma previsão é gerada pelo ARIMA, esse período foi denominado de período de *warm-up*. Após passado o período de *warm-up*, o ARIMA começa então a gerar predições para 15 ciclos à frente. Uma vez que haja alguma mudança na quantidade de recursos alocados, os dados históricos coletados são apagados e se começa um novo *warm-up period*. Isso é feito pois o perfil da aplicação muda com mais ou menos recursos, então os dados históricos se tornam inúteis quando uma mudança ocorre.

D. Módulo de elasticidade

O módulo de elasticidade é o responsável por orquestrar e coordenador o funcionamento do gerenciador de elasticidade. É no módulo de elasticidade que são tomadas as decisões de alocação e desalocação de recursos através da combinação de uma série de algoritmos distintos. A Figura 3 demonstra o fluxo de decisão do módulo de elasticidade e os diferentes estados possíveis em que o gerenciador pode se encontrar.

O processo inicia no momento que o gerenciador é iniciado. Como o processo de avaliação é feito de forma cíclica com intervalos de 10 segundos, a primeira validação feita é se o gerenciador continua ativo. Uma vez validado que o gerenciador ainda se encontra ativo, o módulo de elasticidade dispara o módulo de monitoramento, que - através de uma API fornecida pelo provedor da Nuvem - recupera o estado atual de cada uma das VMs do componente Query Interface que estão alocadas ou em processo de alocação/desalocação. O módulo de elasticidade analisa as informações recuperadas pelo módulo de monitoramento e valida se existe algum processo de alocação/desalocação de VM em andamento. Caso exista, o módulo de elasticidade encerra seu processamento para esse ciclo e se coloca em espera para execução do próximo ciclo. Esse comportamento é definido, pois nenhuma medida de elasticidade deve ser tomada enquanto alguma elasticidade prévia ainda estiver em execução.

Caso nenhuma alocação/desalocação esteja em processo, o módulo de elasticidade agrupa as informações coletadas das VMs do componente Query Interface ativas e calcula uma média aritmética da porcentagem de uso de CPU das VMs. Essa média calculada é adicionada à série temporal que será utilizada na predição de carga. Logo após, o módulo valida a quantidade de registros já adicionados à série. Essa validação é feita pois para utilizar tanto o algoritmo de predição ARIMA quanto o algoritmo de suavização Aging - que serão abordados em mais detalhes logo em seguida - é necessário uma quantidade minima de dados históricos. Com base na literatura e em testes realizados, o valor minimo de dados históricos para o Proliot foi definido em 8 registros.

Se a série tiver menos que 8 registros históricos os algoritmos de predição e suavização ainda não podem ser utilizados, então o módulo de elasticidade entra no modo de tratamento de casos extremos. Ou seja, durante os primeiros 8 ciclos o gerenciador não é capaz de gerar uma predição, mas para que não fique completamente bloqueado para casos de súbita queda ou súbito crescimento, o gerenciador conta com um modo de elasticidade emergencial. Isso é realizado testando se o valor da média de CPUs no atual ciclo é maior que 90% ou menor que 10%. Caso algum desses valores seja ultrapassado, a elasticidade é ativada. Se a média for maior que 90%, uma nova VM para o componente Query Interface começa a ser alocada. Se a média for menor que 10%, a VM com a menor porcentagem de uso de CPU é desalocada. Após 8 ciclos de *warm-up*, o módulo de elasticidade já é capaz de contar com os dados de predição de carga superior e suavização de carga inferior. Esses dados são utilizados de forma complementar e independente. E funcionam da seguinte forma:

- **Predição de carga superior**: A predição da carga de CPU é feita através do modelo estatístico ARIMA - explicado na subseção anterior - e serve para simular a carga do sistema 150 segundos no futuro. Esse valor foi determinado em função do tempo de alocação de uma nova VM. Caso seja verificado que a carga estimada é maior que 80%, então a alocação de uma nova VM é disparada. Ou seja, a predição é utilizada apenas para determinar se uma nova alocação deve ser iniciada ou não;
- **Suavização de carga inferior**: Para definir se o sistema está subutilizado e um recurso pode ser removido, é utilizado o algoritmo Aging. Esse algoritmo serve para fazer uma suavização dos dados e evitar falso-positivos. Em outras palavras, evitar que uma desalocação seja iniciada apenas por um único pico baixo no sistema. Para isso, o algoritmo atribui um peso mais elevado para a observação mais recente, dividindo por uma potência de 2 em cada elemento subsequente da série histórica. Por exemplo, considerando que o limiar para desalocação seja um valor resultante abaixo de 20% e a seguinte série temporal: 19, 22, 21, 23, 21, 20, 23, 18. O algoritmo aplicaria o seguinte cálculo: $smoothed\_load = \frac{19}{2} + \frac{22}{4} + \frac{21}{8} + \frac{23}{16} + \frac{21}{32} + \frac{20}{64} + \frac{23}{128} + \frac{18}{256} = 20{,}28$. Nesse exemplo, podemos observar que mesmo que a média de utilização de CPU observada no ciclo atual seja de 19% - valor abaixo do limiar de 20% -, ainda sim o gerenciador não removeria um recurso pois o valor suavizado é de 20,28. Com isso, o sistema evita remover um recurso necessário apenas por uma observação isolada. Vale ressaltar que o algoritmo de Aging é utilizado apenas para o teste de desalocação de VMs.

São utilizados dois algoritmos distintos pois a alocação de um novo recurso pode levar até 150 segundos, já a desalocação é praticamente instantânea. Sendo assim, para alocação se uti-

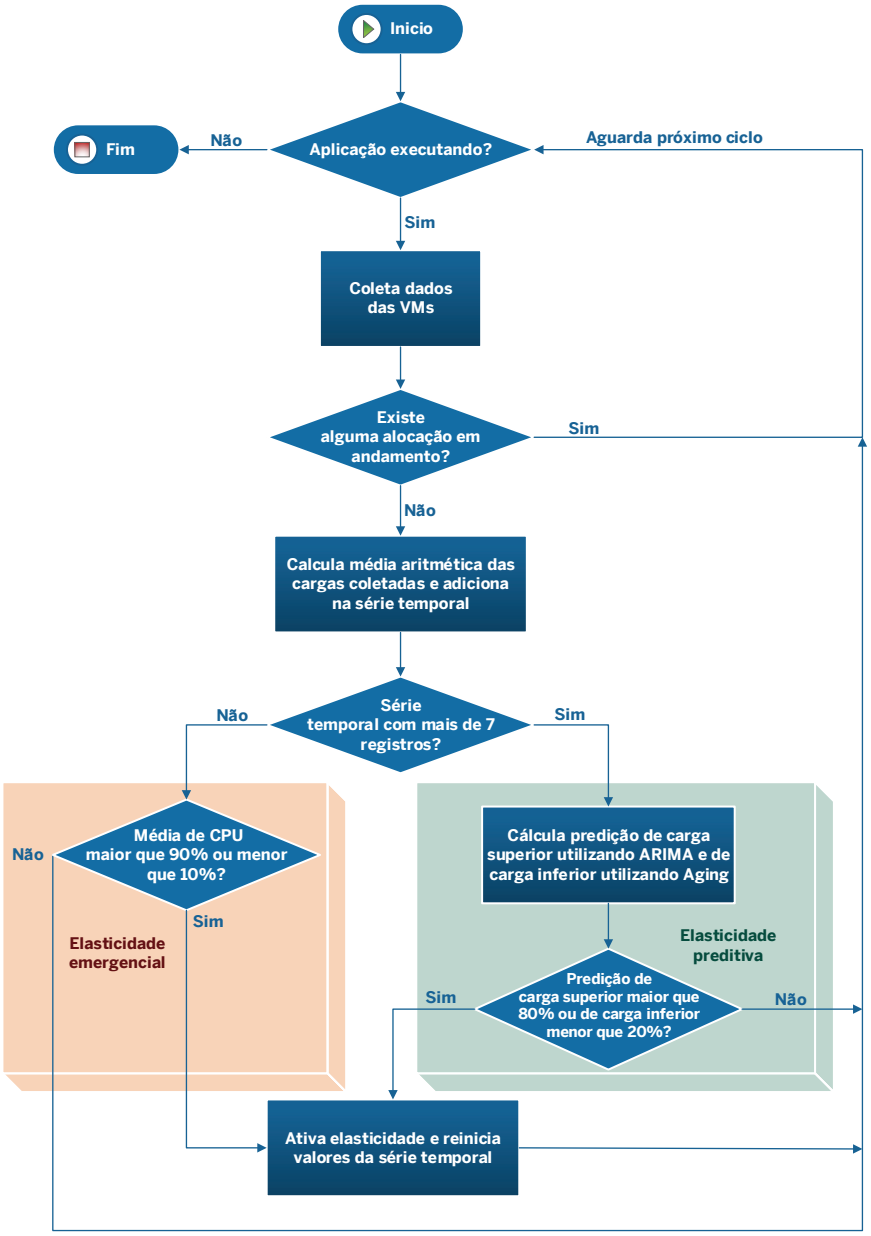


Fig. 3. Fluxo de processamento do módulo de elasticidade.

liza predição através do ARIMA, e para desalocação se utiliza suavização através de Aging. Após os cálculos das cargas superior e inferior o módulo de elasticidade verifica se a carga prevista superior está acima de 80% ou se a carga suavizada inferior está abaixo de 20%. Caso alguma das situações seja verdade, então o módulo de alocação/desalocação é ativado. Após a execução da elasticidade, o módulo termina seu processamento e aguarda para a execução do próximo ciclo. Um ponto importante ressaltar é que sempre que uma elasticidade é ativada os dados da série temporal são reiniciados. Já que o cenário muda então os dados históricos também precisam ser reiniciados.

IV. Metodologia de Avaliação

Para a avaliação do modelo proposto, Proliot e Eliot [19] foram submetidos a uma bateria de testes com a mesma aplicação. Ambos gerenciadores aplicam a elasticidade sobre a VM que executa o componente Query Interface do *middleware* Fosstrak. Sendo assim, os testes foram realizados utilizando a aplicação MIB para simular variadas demandas de consulta aos dados EPC através de requisições a este componente. Utilizando diferentes perfis de requisição é esperado observar o comportamento elástico do sistema, resultando em alocações e desalocações de VMs.

Nos testes realizados foram utilizados 3 perfis de carga: (i) ascendente; (ii) descendente; e (iii) constante. Para cada um dos perfis, o tempo de execução da aplicação foi definido para 420 segundos. No perfil ascendente, MIB foi configurado para iniciar a execução com uma *thread* e adicionar uma nova *thread* a cada 3 segundos. Por outro lado, no perfil descendente, MIB foi configurado para iniciar a execução com 140 *threads* fazendo requisições simultâneas e a cada 3 segundos uma *thread* é removida. Por fim, no perfil constante, o MIB foi configurado para rodar 140 *threads* simultâneas em todo o tempo de execução.

A avaliação de desempenho foi feita utilizando as seguintes métricas: tempo de saturação do sistema, tempo de resposta médio, quantidade de pacotes perdidos e energia. A métrica do tempo de saturação do sistema mede quanto tempo o sistema ficou executando com uma carga de CPU superior à 80%. Esse valor é analisado pois há uma relação diretamente proporcional de altas cargas de CPU à diminuição na qualidade de serviço. Em outras palavras, quanto mais tempo o sistema rodar com uma carga de CPU elevada, maior a chance de existir perdas de pacote ou aumento no tempo de resposta. Os valores de tempo de resposta e perdas de pacotes também são avaliados pois eles refletem à qualidade de serviço ao usuário. Por fim, é analisado o consumo de energia de cada modelo. O índice energético é obtido observando períodos de tempo e multiplicando essa quantidade de tempo pela quantidade de VMs ativas naquele período. Esse cálculo de energia é demonstrado pela equação: $Energia = \sum_{i=1}^{I}(i \times T(i)))$. Onde *i* é a quantidade de VMs ativas no tempo $T(i)$.

O gerenciador Proliot foi desenvolvido em Java utilizando o Eclipse Neon como IDE. Para o uso do algoritmo de predição ARIMA, foi utilizada a Java/R Interface (JRI), biblioteca de comunicação do Java com a linguagem R. No R foi utilizado o pacote *forecast* que contêm a função auto.arima, como explicado na subseção do módulo de predição. Para o gerenciamento das requisições recebidas foi criado um HTTP server utilizando a biblioteca httpserver[2] da Sun. E para encaminhar as requisições para o componente Query Interface foi utilizado a biblioteca http.client[3] do Apache. Foram utilizadas duas ferramentas auxiliares para a simulação de um ambiente real de um sistema RFID. A primeira ferramenta utilizada foi o simulador Rifidi Emulator[4]. A segunda ferramenta auxiliar utilizada foi o "MIB: Micro Benchmark para Avaliação de Middlewares de Internet das Coisas". Na implementação do Proliot, o MIB foi utilizado para simular requisições de uma aplicação cliente ao componente Query Interface. Mais detalhes sobre os perfis de teste executados no Proliot serão descritos na seção a seguir.

V. Avaliação de Resultados

Nesta seção serão apresentados os resultados dos testes comparativos executados conforme descrito na seção anterior. A subseções V-A, V-B e V-C apresentam, respectivamente, os resultados obtidos com as execuções da carga crescente, descendente e constante. Por fim, uma análise comparativa é realizada na subseção V-D.

A. Ascendente

Inicialmente é possível observar que a Figura 4 apresenta os resultados obtidos durante o teste com carga ascendente. Esse foi o perfil de carga que demonstrou mais diferenças entre os modelos de elasticidade. Como pode ser observado na Figura 4 (a), o gerenciador Proliot começa a execução do teste com um período de warm up onde armazena resultados históricos para então começar a gerar previsão de carga. Utilizando a previsão gerada através do ARIMA, o Proliot – no segundo 140 – é capaz de prever que haveria uma sobrecarga do sistema no futuro. Sendo assim, o Proliot inicia imediatamente a alocação de uma nova máquina virtual, mesmo o sistema não estando saturado ainda. Isso resultou em um período pequeno de saturação. Já na Figura 4 (b), é possível observar que o gerenciador Eliot só inicia a alocação de uma nova VM no momento em que o sistema já está saturado – no segundo 230. Isso ocasionou um grande período de saturação do sistema, causando em negação de serviço e aumento do tempo de resposta.

B. Descendente

Na Figura 5 é possível observar o comportamento dos gerenciadores trabalhando com carga descendente. Nesse cenário ambos gerenciadores apresentam um comportamento muito parecido. Na Figura 5 (a), pode-se notar que O Proliot logo no inicio da execução observa uma carga de CPU acima de 90% e aciona a elasticidade emergencial, adicionando uma nova máquina virtual. No segundo 300, o Proliot – utilizando o algoritmo de AGING – nota que a carga de CPU é menor que 20% e desaloca uma de suas VMs. O Eliot – ilustrado na Figura 5 (b) – tem um comportamento muito parecido ao do Proliot nesse perfil de carga. A diferença fica apenas no momento da desalocação da VM que, no caso do Eliot, ocorre no segundo 290.

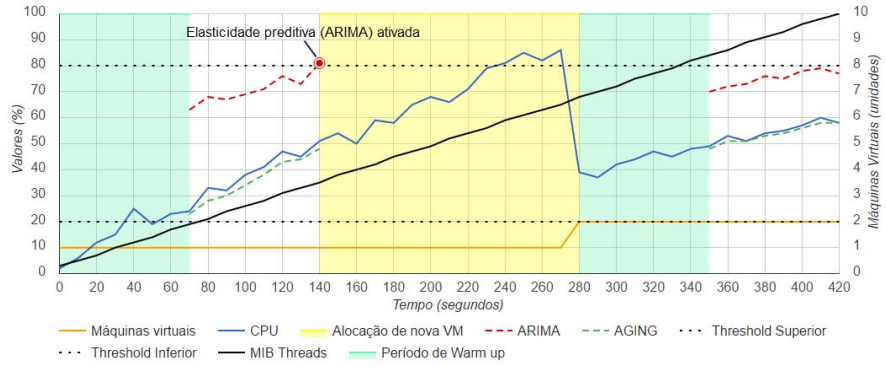
C. Constante

O resultado dos testes de carga constante são apresentados na Figura 6. Nesse cenário, novamente ambos modelos apresentaram um comportamento muito similar. A Figura 6 (a) demonstra o funcionamento do Proliot com uma carga constante. Mais uma fez o Proliot ativou a elasticidade emergencial no inicio da execução ao se deparar com uma carga de CPU superior à 90%. O sistema continua saturado por alguns segundos enquanto a VM adicional é preparada para alocação. A nova VM é entregue no segundo 140, o que resulta em uma estabilização do sistema. Por ter um número constante de requisições por segundo, as duas máquinas continuam em funcionamento até o final do teste. Como pode ser observado na Figura 6 (b), o Eliot apresenta um comportamento idêntico ao do Proliot nesse cenário de teste.

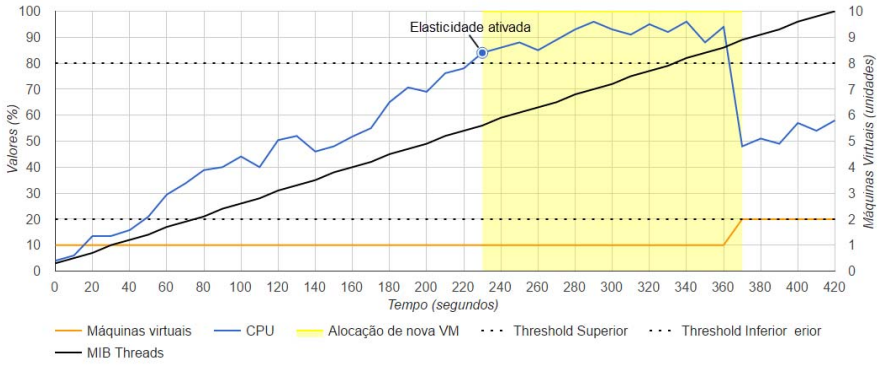
[2]com.sun.net.httpserver

[3]http://org.apache.http.client

[4]https://sourceforge.net/projects/rifidi/files/Rifidi%20Emulator/

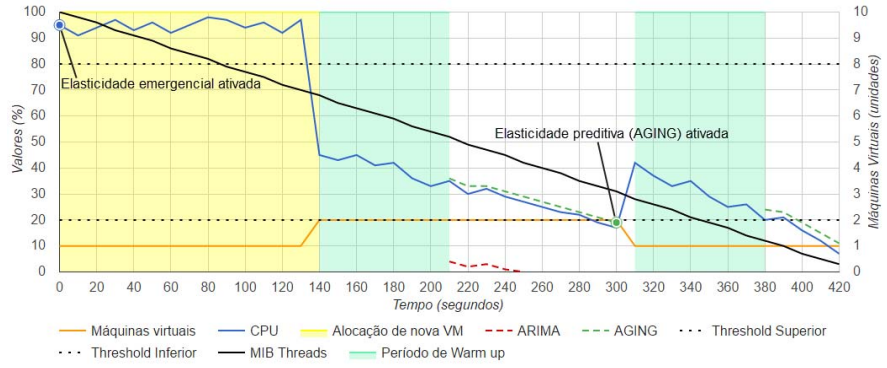


(a) Proliot

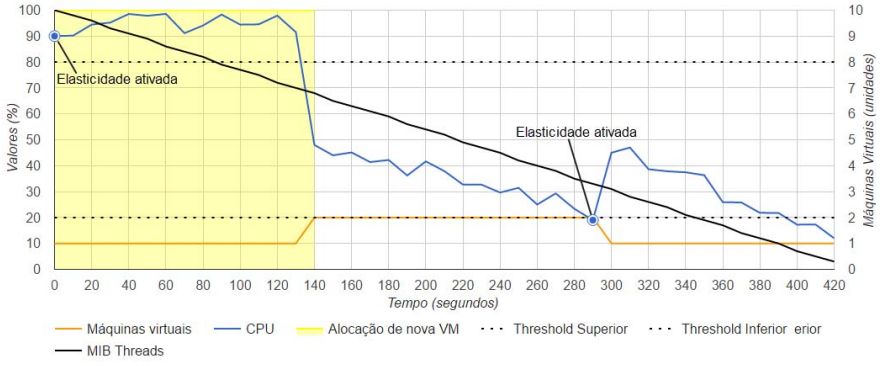


(b) Eliot

Fig. 4. Execução da aplicação processando a carga ascendente com os gerenciadores (a) Proliot e (b) Eliot.

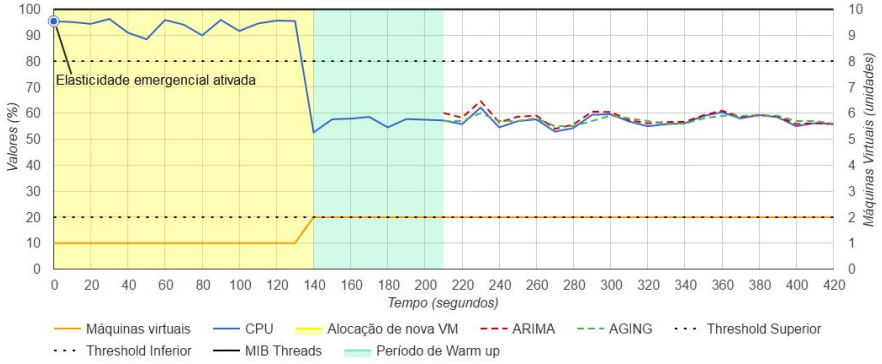


(a) Proliot

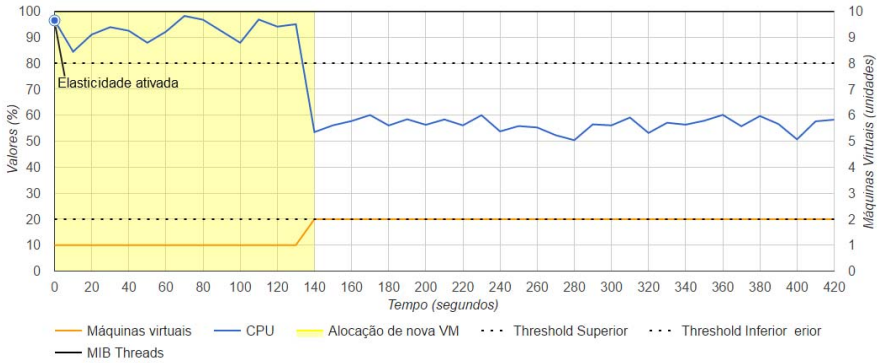


(b) Eliot

Fig. 5. Execução da aplicação processando a carga descendente com os gerenciadores (a) Proliot e (b) Eliot.



(a) Proliot



(b) Eliot

Fig. 6. Execução da aplicação processando a carga constante com os gerenciadores (a) Proliot e (b) Eliot.

D. Análise Comparativa

A Tabela II apresenta um comparativo das métricas coletadas de cada gerenciador elástico durante os testes executados. Conforme já foi destacado nas demais figuras dessa seção, a análise das métricas também confirma a similaridade no desempenho dos gerenciadores Proliot e Eliot para os perfis de carga descendente e constante. Como pode ser observado na Tabela II, nos perfis descendente e constante, ambos gerenciadores apresentaram o mesmo tempo de saturação do sistema e percentualmente quase os mesmos valores de tempo de resposta, pacotes perdidos e energia - na comparação individual para cada perfil. Isso demonstra que para esses perfis de cargas os dois gerenciadores demonstram uma performance praticamente equivalente.

A maior diferença entre os gerenciadores se mostra no perfil de carga ascendente. Nesse perfil de carga é possível observar grandes ganhos de performance do gerenciador proativo Proliot. Por exemplo, no tempo de saturação do sistema, o Proliot apresentou um valor 76% menor que o do Eliot. No tempo de resposta, o Proliot também demonstrou um valor 28% menor em relação ao Eliot. E em se tratando da quantidade de pacotes perdidos, o Proliot teve um valor 37% menor que o Eliot. Em contrapartida, o Proliot apresentou um valor 16% maior no índice de energia. Em resumo, o Proliot apresenta uma performance equivalente ao Eliot nos perfis descendente e constante. E uma performance superior ao Eliot no perfil ascendente - considerando tempo de saturação, tempo de resposta e pacotes perdidos. Entretanto, esse ganho de performance vem ao custo de um consumo maior de energia.

TABLE II
COMPARATIVO DAS MÉTRICAS DOS GERENCIADORES PROATIVO E REATIVO

Perfil de carga	**Gerenciador**	**Tempo de saturação (s)**	**Tempo de resposta (ms)**	**Pacotes perdidos**	**Energia**
Ascendente	Proliot	30	86	753	560
	Eliot	130	120	1198	470
Descendente	Proliot	140	126	1477	580
	Eliot	140	127	1494	570
Constante	Proliot	140	186	5620	700
	Eliot	140	188	5735	700

VI. CONCLUSÃO

Atualmente, mais de 20 bilhões de dispositivos estão conectados à internet e estima-se que até 2020 serão mais de 30 bilhões [23]. Muitos desses dispositivos contam com sensores que possibilitam capturar informações do mundo real e levá-las para o universo virtual – processo idealizado no conceito de Internet das Coisas (IoT). Abordagens combinando *middlewares* IoT padrão EPCglobal com as capacidades da

Computação em Nuvem geralmente abordam a elasticidade de recursos de maneira reativa. Focando um serviço transparente capaz de oferecer elasticidade proativa à *middleawres* IoT padrão EPCglobal, esse artigo propôs o modelo de arquitetura do gerenciador Proliot. No modelo proposto pelo Proliot, o *middleware* é implantado na Nuvem de forma distribuída e um gerenciador responsável por controlar a elasticidade, monitoramento e balanceamento de carga é adicionado ao sistema. O diferencial do gerenciador proativo proposto nesse trabalho frente a outros modelos de elasticidade é a combinação de diversos algoritmos distintos para as tomadas de decisão em relação a ativação da elasticidade. Além disso, outra vantagem é a praticidade ao usuário que não necessita fazer nenhum tipo de adaptação em sua aplicação para utilizar o gerenciador Proliot. E por fim, o gerenciador também oferece total compatibilidade com qualquer *middlware* IoT que implemente o componente Query Interface da EPCglobal sem necessidade de nenhuma alteração no *middleware*.

Foi desenvolvido um protótipo de Proliot e comparado ao gerenciador reativo Eliot. Foram utilizados 3 perfis de carga nos testes: ascendente, descendente e constante. Os resultados foram avaliados segundo as métricas de tempo de saturação do sistema, tempo de resposta médio, total de pacotes perdidos e energia. Nos testes dos perfis descendente e constante os gerenciadores demonstraram resultados praticamente iguais com uma pequena vantagem do Proliot. Já nos testes com o perfil de carga ascendente o gerenciador Proliot demonstrou resultados expressivamente melhores. O tempo de saturação foi 76% menor, o tempo de resposta foi 28% menor e a quantidade de total de pacotes perdidos foi 37% menor em relação ao Eliot. O contraponto foi um aumento de 16% no índice energético.

Para trabalhos futuros, destacam-se a análise de cargas irregulares e possíveis adaptações para esse tipo de perfil de carga. Ainda, pode-se citar a implementação do gerenciador em um ambiente produtivo com demandas reais de carga. Em adição, uma possível extensão desse modelo poderia ser realizada para suportar outros componentes de um sistema EPCglobal, como a Capture Application ou o ALE. E por fim, abordagens utilizando bancos de dados baseados em documentos e distribuídas ao invés do banco MySQL surge como uma oportunidade futura.

REFERENCES

[1] K. Ashton, "That Śinternet of thingsŠ thing," *RFiD Journal*, vol. 22, no. 7, pp. 97 – 114, 2009. [Online]. Available: http://www.itrco.jp/libraries/RFIDjournal-That\%20Internet\%20of\%20Things\%20Thing.pdf

[2] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of things (iot): A vision, architectural elements, and future directions," *Future Generation Computer Systems*, vol. 29, no. 7, pp. 1645 – 1660, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X13000241

[3] INFSO and EPOSS, "Internet of things in 2020: A roadmap for the future," 2008. [Online]. Available: http://www.smart-systems-integration.org/public/documents/publications/Internet-of-Things_in_2020_EC-EPoSS_Workshop_Report_2008_v3.pdf

[4] M. PRESSER and A. GLUHAK, "The internet of things: Connecting the real world with the digital world," *The Magazine for Telecom Insiders*, vol. 2, 2009. [Online]. Available: http://archive.eurescom.eu/message/messageSep2009/The-Internet-of-Thing\%20-Connecting-the-real-world-with-the-digital-world.asp

[5] A. H. Ngu, M. Gutierrez, V. Metsis, S. Nepal, and Q. Z. Sheng, "Iot middleware: A survey on issues and enabling technologies," *IEEE Internet of Things Journal*, vol. 4, no. 1, pp. 1–20, Feb 2017.

[6] J. I. Aguirre, "Epcglobal: a universal standard," Ph.D. dissertation, Massachusetts Institute of Technology, Massachusetts , Estados Unidos, 2007. [Online]. Available: http://dspace.mit.edu/handle/1721.1/42350

[7] M. Gomes, R. da Rosa Righi, and C. A. da Costa, "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)*, Oct 2014, pp. 269–276.

[8] D. Anderson, "Boinc: a system for public-resource computing and storage," in *Grid Computing, 2004. Proceedings. Fifth IEEE/ACM International Workshop on*. IEEE, Nov 2004, pp. 4–10.

[9] A. Raveendran, T. Bicer, and G. Agrawal, "A framework for elastic execution of existing mpi programs," in *Proceedings of the 2011 IEEE Int. Symposium on Parallel and Distributed Processing Workshops and PhD Forum*, ser. IPDPSW '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 940–947.

[10] W. Wang, J. Sung, and D. Kim, "Complex event processing in epc sensor network middleware for both rfid and wsn," in *2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC)*, May 2008, pp. 165–169.

[11] L. Dong, D. Wang, and H. Sheng, "Design of rfid middleware based on complex event processing," in *2006 IEEE Conference on Cybernetics and Intelligent Systems*, June 2006, pp. 1–6.

[12] T. Cheong, Y. Kim, and Y. Lee, "Rems and rbpts: Ale-compliant rfid middleware software platform," in *2006 8th International Conference Advanced Communication Technology*, vol. 1, Feb 2006, pp. 699–704.

[13] N. Kefalakis, N. Leontiadis, J. Soldatos, and D. Donsez, *Middleware Building Blocks for Architecting RFID Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 325–336. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-03819-8_31

[14] C.-H. Kuo, L.-C. Hwang, P.-H. Cheng, M.-L. Lee, and J.-R. Hu, "The robustness rfid middleware system for epcglobal network," in *2010 8th IEEE International Conference on Industrial Informatics*, July 2010, pp. 831–835.

[15] A. Kabir, B. Hong, W. Ryu, and S. Ahn, *LIT Middleware: Design and Implementation of RFID Middleware Based on the EPC Network Architecture*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 221–229. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-76862-3_21

[16] C. Floerkemeier, C. Roduner, and M. Lampe, "Rfid application development with the accada middleware platform," *IEEE Systems Journal*, vol. 1, no. 2, pp. 82–94, Dec 2007.

[17] J. Byun and D. Kim, "Oliot epcis: New epc information service and challenges towards the internet of things," in *2015 IEEE International Conference on RFID (RFID)*, April 2015, pp. 70–77.

[18] A.-I. L. KAIST, "Oliot project," 2016. [Online]. Available: http://gs1oliot.github.io/oliot/

[19] R. da Rosa Righi, E. S. dos Reis, G. Rostirolla, C. A. da Costa, and A. M. Alberti, "Exploring cloud elasticity on developing an epcglobal-compliant middleware," in *2016 IEEE International Conference on RFID (RFID)*, May 2016, pp. 1–4.

[20] P. A. Dinda and D. R. O'Hallaron, "An evaluation of linear models for host load prediction," in *High Performance Distributed Computing, 1999. Proceedings. The Eighth International Symposium on*, 1999, pp. 87–96.

[21] H. M. A. E. Hag and S. M. Sharif, "An adjusted arima model for internet traffic," in *AFRICON 2007*, Sept 2007, pp. 1–6.

[22] K. Kalpakis, D. Gada, and V. Puttagunta, "Distance measures for effective clustering of arima time-series," in *Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on*, 2001, pp. 273–280.

[23] IDC, "The digital universe of opportunities:rich data and increasing the value of the internet of things," 2014. [Online]. Available: http://www.emc.com/leadership/digital-universe/2014iview/index.htm