Estudo de viabilidade do uso de Raspberry PI na névoa*

Guilherme de Souza[1], Nelson Lago[2], Gerson Geraldo H. Cavalheiro[1], Alfredo Goldman[2]

[1]Programa de Pós-Graduação em Computação
Universidade Federal de Pelotas (UFPel)
Caixa Postal 354 – 96010.610 – Pelotas – RS – Brazil

[2]Instituto de Matemática e Estatística (IME) – Universidade de São Paulo (USP)
Rua do Matão, 1010 – São Paulo – SP – Brazil

{gdsdsilva,gerson.cavalheiro}@inf.ufpel.edu.br, {lago,gold}@ime.usp.br

Abstract. *The need for reduced latencies in many applications, often to improve end-user usability, brought about the* mist *paradigm, which moves processing or preprocessing out of the cloud and closer to the data source. To address the requirements of low cost, low energy consumption, small size, and others that are common with this paradigm, an option is the use of low capacity, general purpose and widely available equipment. This work aims at investigating the Raspberry Pi 3 as a dispositive for mist applications, assessing its characteristics with the NDBench benchmark. Results of experiments with read/write operations in a NoSQL database indicate the Raspberry's viability in scenarios with significant load (around up to 1.200 operations per second) while maintaining an average latency of 500ms. Such characteristics contemplate a vast amount of applications and suggest that the Raspberry Pi can be successfully used in mist and cloud environments.*

Resumo. *Em função da necessidade de se ter baixa latência em muitas aplicações e visando uma melhor usabilidade para o usuário final, nasce o paradigma de névoa, que traz o processamento ou pré processamento para um local mais próximo ao usuário. Buscando reduzir o consumo energético, optou-se pelo uso de dispositivos de baixa capacidade, dado seu propósito geral, baixo consumo e custo, além da disponibilidade no mercado. Este trabalho tem como objetivo investigar a Raspberry Pi 3 como dispositivo para névoa, avaliando seu uso através do benchmark NDBench realizando operações de escrita e leitura em um banco de dados NoSQL. Os resultados indicam a viabilidade da Raspberry em cenários onde são esperadas em torno de até 1.200 operações por segundo com latência média de 500ms, o que contempla uma grande quantidade de aplicações e demonstra que a Raspberry pode ser usada em ambientes de nuvem e névoa.*

1. Introdução

Em virtude da explosão do chamado *big data*, decorrente do crescimento da oferta e emprego de dispositivos que produzem e/ou consomem dados, como dispositivos inteligentes, vestíveis e sistemas em veículos, as aplicações próprias ao processamento de

*O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001

dados alcançaram novos patamares de importância e são cada vez mais requisitadas pelos indivíduos. A primeira alternativa para viabilizar computacionalmente toda a carga de processamento gerada foi empregar os recursos disponíveis na nuvem. Como vantagem adicional, a nuvem também reúne características que permitem a agregação de dados de diversas fontes. No entanto, como consequência direta do uso da nuvem, observou-se tanto um grande aumento no tráfego de rede como dificuldades quando o acesso à rede, pelo usuário final, é limitado ou mesmo inexistente.

Considerando que o processamento pode ocorrer em *data centers* geograficamente distantes, aplicações que dependem de uma baixa latência tendem a sofrer com o caminho percorrido até a nuvem. Inúmeras dessas aplicações, oriundas da Internet das Coisas (*Internet of Things* – IoT) [Gubbi et al. 2013], encontram-se nessa situação. Nesses casos, a qualidade da rede entre o ponto onde o dado é gerado ou necessário até o ponto onde existe o processamento impacta diretamente a qualidade do resultado obtido. Como mecanismo de compensação, *data centers* são forçados a se adequar, seja aumentando recursos ou fazendo um maior aproveitamento dos existentes. Muitas vezes, esse esforço de adequação resulta em um aumento no consumo energético de tais centros de computação, o qual, em 2012, aproximava-se de 270 TWh, crescendo anualmente [Van Heddeghem et al. 2014].

De forma a atender melhor essa demanda, um novo paradigma foi proposto, a *Computação em Névoa* [Bonomi et al. 2012], que estende a Computação em Nuvem [Mell and Grance 2011], trazendo a possibilidade de se ter dispositivos habilitados ao processamento no meio do caminho. A computação em névoa traz, dessa forma, o processamento ou pré-processamento de dados para perto do usuário, mantendo seus quatro pontos essenciais: baixa latência, maior confiabilidade, peso reduzido de *hardware* (tamanho, local e número de máquinas) e baixo consumo energético [Byers 2017].

Para o processamento desses dados, nem sempre há como utilizar uma máquina com grande poder computacional e, muitas vezes, não se apresenta essa necessidade. Com isso, a névoa traz a possibilidade do uso de dispositivos de baixa capacidade que possam suprir as necessidades geradas pelas aplicações. Em [G. Souza and Pilla 2018], foi proposto o uso de dispositivos com arquitetura ARM, em específico a Raspberry Pi 3, como parte de uma névoa, mantendo o foco em avaliar o consumo energético gerado por ela.

Dando continuidade, o trabalho aqui apresentado tem por objetivo avaliar e quantificar seu desempenho com relação à latência por meio de estudos práticos a partir da simulação de escrita e leitura em banco de dados NoSQL com o *benchmark Netflix Data Store Benchmark (NDBench)* [Papapanagiotou and Chella 2018]. Este experimento nos permite criar cenários próximos à realidade de diversas aplicações dado o contexto de serviços *web*. Por exemplo: *blogposts*, *sites* de informações, pequenos *marketplaces*, além do processamento de dados oriundos da coleta de sensores, seja dentro da agroindútria ou em empresas autônomas, ou seja, é possivel pensar em inúmeras aplicações dada a quantidade de atendimento necessário.

Fazendo uso de diferentes tipos de carga, tornou-se possível encontrar resultados adequados próximos ao limite do dispositivo, obtendo-se assim resultados de latência nos percentis 95 e 99, latênca média e operações por segundo. Os resultados demonstraram que a Raspberry Pi 3 possui capacidade de atendimento de aplicações que se assemelham

aos experimentos aqui realizados.

Existe uma grande gama de *small boards* no mercado, porém, entre elas a Raspberry Pi 3 foi selecionada por ser de fácil reprodução e facilmente encontrada. Ainda, ela possibilita o uso tanto para aplicações específicas quanto para mais genéricas, já que conta com um sistema próprio que nos permite tal maleabilidade. Sua arquitetura ARM é pensada para realização de diferentes tarefas, porém visando um baixo consumo energético, além de abrir possibilidades de realizar sua amplificação com o uso de *shields* específicos que aumentam a capacidade da placa [Maksimović et al. 2014].

O presente artigo está organizado da seguinte forma: na Seção 2 serão apresentados trabalhos relacionados, na Seção 3 são apresentadas as ferramentas utilizadas para realização dos experimentos. Na Seção 4, são descritos os resultados obtidos considerando os cenários representativos. As considerações finais e trabalhos futuros se encontram na Seção 5.

2. Trabalhos Relacionados

Com o Paradigma da IoT, houve um grande aumento de dispositivos conectados à Internet e a previsão é de se ter mais de 24 bilhões até 2020 [Gubbi et al. 2013]. Essa previsão gerou uma nova gama de desafios e oportunidades a serem exploradas e, desde então, estudos vêm sendo realizados para uso desse *hardware* de baixo custo.

Em [R. Nakhkash et al. 2019], os autores realizaram testes em dispositivos de baixa capacidade computacional e baixo consumo energético, fazendo uso do *Benchmark LOCUS (Low-power customizable many-core architecture for wearables)* para simular alguns serviços realizáveis pelos vestíveis, comparando nove serviços como: *ECG, AES encrypt/descrypt, AStar, DTW, SVM, Histogram, 2dConv e Haar Transform*, podendo ser executados de forma serial ou em *threads*, obtendo como resultados o tempo de execução e consumo gerado pelos dispositivos. Dentre os equipamentos comparados estão Raspberry Pi Zero e Odroid-Xu4, arquiteturas com processadores distintos, utilizados em vestíveis e em celulares respectivamente. A questão levantada é que alguns serviços poderiam ser executados pelo *gateway* ao invés do sensor, levando em conta a arquitetura e os tipos de serviços. Após os testes pode-se observar que o processador é um forte contribuinte tanto no consumo quanto no tempo e que portas de entrada e saída podem fazer os resultados variarem. As placas mais próximas à Raspberry Pi demonstram um desempenho e consumo que se adequam aos serviços, sugerindo que algumas aplicações poderiam ser repassadas ao *gateway*, aproveitando melhor sua capacidade para não sobrecarregar os vestíveis, assim fazendo um melhor aproveitamento de ambos equipamentos conectados.

Sendo possível que o processamento possa ser transferido para uma camada posterior na rede, ou seja, não ficando somente no final da borda, assim, uma menor latência é exigida, porém para isso deve-se levar em conta a localidade do usuário e o posicionamento do nodo da névoa. Assim, [Huang et al. 2019] apresenta o *MultiCopyStorage*, uma técnica que permite a seleção de testes necessários para a definição da topologia adequada na replicação de dados, obtendo uma menor latência. Essa importância é demonstrada em [Bittencourt et al. 2017], onde os autores fazem uso do jogo *electroencephalogra-phy* (EEG) *Tractor Beam* e um aplicativo de vigilância por vídeo / rastreamento de objetos.

No primeiro exemplo, o jogo atua a partir da concentração do usuário em um determinado objeto, o que o traz até o jogador, logo uma baixa latência no processamento e

análise dos dados de concentração gerados pelos sensores permite uma maior realidade e por consequência melhor desenvoltura. O mesmo pode-se mostrar para vigilância por vídeo, identificando o objeto e sua posição: uma latência menor garante o acerto da resposta do dado transmitido naquele momento. Aplicando algumas políticas de propriedades e se preocupando com a locomoção do usuário para determinadas posições, realizaram testes com 3 nodos entre o usuário e a nuvem, dessa forma possibilitou analisarem o comportamento da latência conforme as aplicações mudavam de nodo ou eram migrados para a nuvem.

Com os estudos aqui apresentados, observa-se a importância de se ter um dispositivo entre o caminho da nuvem e usuário final, porém não é discutido um dispositivo em específico como ocupante deste papel de névoa. Dessa forma, queremos evidenciar a Raspberry Pi 3 como dispositivo que pode vir à desempenhar esta função e ao mesmo tempo atender o paradigma além de suprir as necessidades apresentadas.

3. Metodologia

Visto que a demanda de serviços *web* ocupa a maior parte das aplicações na nuvem, para a realização deste trabalho optou-se pelo uso de um banco de dados NoSQL, dada sua adequação ao uso em sistemas distribuídos e em aplicações com alta demanda e dependentes de elasticidade [Cattell 2011]. Tendo tudo isso em vista, optamos por fazer uso do banco de dados NoSQL Apache Cassandra.

Da mesma forma, para escolha do simulador entre os *benchmarks* disponíveis para testes em sistemas de nuvem, optou-se pelo que seria capaz de atender nossas exigências de latência e que nos permitisse uma livre escolha das cargas de trabalho para criarmos cenários restritos para execução dos experimentos com cargas ideais e próximas à capacidade de processamento da Raspberry. Dadas tais necessidades, entre outras funções, o simulador *NDBench* foi o selecionado para o presente trabalho.

3.1. Apache Cassandra

Com a necessidade de os sistemas trabalharem com uma alta demanda de transações por segundo e os bancos de dados estarem preparados para atender tal situação, o Cassandra nasce oriundo desta necessidade observada na época e das apresentadas na seção anterior. Sendo uma solução desenvolvida pela empresa *Facebook*, de forma a suprir o armazenamento do seu motor de busca de mensagens, para a empresa isso significa lidar com uma taxa de bilhões de gravações por dia e também escalar pela quantidade de usuários [Lakshman and Malik 2010].

Como os usuários são atendidos por *data centers* geograficamente distribuídos, é necessário ser capaz de manter réplicas de dados entre os *data centers*. Esta replicação permite manter baixos valores para latência de pesquisa, além de ser expansível. Ele adicionalmente oferece outras facilidades, como componentes de persistência de dados, detecção e recuperação em cenários de falhas, permitindo que, na ocorrência de uma falha em uma réplica, o usuário seja encaminhado para o sítio mais próximo onde o dado desejado também esteja disponível [Featherston 2010].

Na empresa *Netflix*, o Cassandra é o principal repositório de dados, suportando milhares de microsserviços e mais de 125 milhões de assinantes. Sua aplicação serve

como uma base sólida de conjuntos de dados mundialmente replicados pela empresa, trazendo transmissão *online* de séries e filmes. Contando com centenas de *clusters*, "dezenas de milhares de nós e peta bytes de dados, o Cassandra atende a vários milhões de operações por segundo com vários nodos de disponibilidade na camada de armazenamento" [Papapanagiotou and Chella 2018].

3.2. NDBench

Entre os inúmeros *benchmarks* existentes e conhecidos para testes deste tipo, o *YAHOO! Cloud Serving Benchmark* (YCSB), é o mais utilizado em ambientes de nuvem, até então, e também é o que mais se assemelha ao *NDBench*. Porém, ele não nos permite configurar parâmetros de um *cluster* a partir de um ponto central. Além disso, a ferramenta executa testes em horizonte finito e não permite a alteração dinâmica das cargas de trabalho ou modelo de dados durante a execução. Assim, consideramos que ela não atendia os requisitos de testes aos quais nos propusemos a realizar neste trabalho [Papapanagiotou and Chella 2018].

O *NDBench* foi desenvolvido para testar o crescente número de serviços na empresa *Netflix*, atendendo a real necessidade da mesma, simulando o tráfego em seus *data centers*, para que assim possa ser disponibilizado todo suporte necessário. Logo, é possível verificar a capacidade da infraestrutura e suas aplicações demonstrando possíveis necessidades de melhoras nos serviços fornecidos. Rotinas de testes são mantidas nas quais os resultados são utilizados em certas decisões dentro da empresa, como dados de usuários, pagamentos, listas de *stream* já vistas ou que desejam ser vistas.

Ele foi inteiramente planejado para execução em plataformas de nuvem, possibilitando sua aplicação em diferentes *hardware*, o que permite uma plataforma híbrida, podendo ser executado em qualquer sistema de *software*. Ele ainda conta com um conjunto de bancos de dados e testes para eles, tais como:

- Apache Cassandra.
- *Dynomite*.
- *Elasticsearch and RDS*.

Os padrões de teste e cargas podem ser executados por linha de comando ou através da interface de usuário (UI) fornecida pelo simulador. Propriedades de algumas principais cargas podem ser vistas na Tabela 1. As cargas de testes podem ser alteradas conforme a necessidade, além do *NDBench* permitir que o usuário crie e integre suas próprias aplicações para que sejam testadas através do simulador. Os resultados gerados pelo *NDBench* são mostrados em tempo real, gerando como resultados latência acumulada em percentil, assim como resultados de memória entre outros, disponibilizando um gráfico na sua UI.

A partir dos testes fornecidos pelo simulador, *benchmarks* são capazes de ser alcançados e analisados dada a aplicação desejada. Sendo possível que com um teste simples possamos avaliar a infraestrutura, economizando dezenas de testes antes realizados de forma manual. A carga de trabalho gerada pelo simulador pode ser alterada em tempo real, como ressaltado anteriormente, garantindo um cenário mais realista, como por exemplo: o dia da *black friday*, em que em um único dia a circulação de usuários extrapola o normal, porém normalizando no outro dia, dessa forma tais cenários podem ser simulados constantemente, fazendo com que serviços de melhor qualidade sejam fornecidos.

Tabela 1. Algumas das principais propriedades das cargas aplicadas pelo NDBench.

Config. Nome	Descrição
numkeys	Espaço amostral para as chaves geradas aleatoriamente
numValues	Espaço amostral para os valores gerados
dataSize	Tamanho de cada valor
numWriters/numReaders	Número de *threads* por nodo NDBench para gravações/leituras
writeEnabled/readEnabled	Booleano para ativar ou desativar escrita ou leitura
writeRateLimit/readRateLimit	Número de gravações por segundo e leituras por segundo
userVariableDataSize	Booleano para ativar ou desativar a capacidade da carga a ser gerada aleatoriamente

3.3. Metodologia dos Testes

A Figura 1 ilustra a Raspberry como nodo da névoa, em um cenário representando uma aplicação no agronegócio. Implementos agrícolas visitam a lavoura e trazem informações coletadas sobre o estado da cultura, como condição de umidade, qualidade nutritiva do solo e infestação de pragas. Essas informações são adicionadas à base de dados no decorrer do dia. À noite, o sistema lê índices pluviométricos de um *data center*, ou repositório como o Ana[1] e a previsão meteorológica de forma a poder programar os implementos sobre as ações a serem realizadas sobre a cultura (hidratação, adubagem e pulverização de pesticidas) no dia seguinte. Os gestores da fazenda, por sua vez, tem a possibilidade de visualizar o estado da lavoura e, eventualmente, interferir na programação de atividades.

Com ideias de aplicações que possuam características similares ao cenário apresentado, iniciamos a preparação dos casos de estudo. Inicialmente, foi instalado no Raspberry Pi 3 o ambiente do Cassandra na sua versão 2.2.14, compatível com o *NDBench*, de acordo com a documentação. Similarmente, configurou-se a máquina-cliente que manteve o *NDBench* ativo, de mesmo modo seguindo-se a documentação disponível no GitHub[2].

Junto ao *NDBench*, são disponibilizadas as configurações das tabelas dos bancos de dados necessárias para que estejam aptos a receber a simulação. Logo, o Cassandra foi configurado de acordo, sendo criado somente um único nodo, o qual julgamos já ser suficiente para testar o desempenho do dispositivo.

A comunicação entre as máquinas foi realizada por canal criado pelas portas *ethernet*, via cabo de rede, de forma a diminuir o ruído nos resultados obtidos. A configuração da máquina cliente manteve as configurações *default*, ou seja, alguns valores pré selecionados pelo simulador foram mantidos. Somente opções como *numWrites / numReads* e *writeRateLimit / readRateLimit* foram alteradas, como demonstrado na Tabela 2, de forma a alcançar um *benchmark* no qual fosse possível chegar próximo a um limite do dispositivo e analisar um real cenário de aplicação.

Todos os experimentos contaram com a duração de 25 minutos, sem sofrer alteração na configuração do simulador durante a execução, embora o simulador nos permitisse isso. Experimentos foram executados anteriormente para analisar as possíveis variações, cargas e falhas dada pelo simulador, não sofrendo variações, empregou-se cargas aproximadas do limite de resposta da Raspberry Pi 3 para a obtenção dos resultados analisados na Seção 4. Com as cargas selecionadas, criaram-se três cenários, de forma a simular

[1]Agência Nacional de Águas: `https://www.ana.gov.br/`
[2]GitHub: `github.com/Netflix/ndbench`

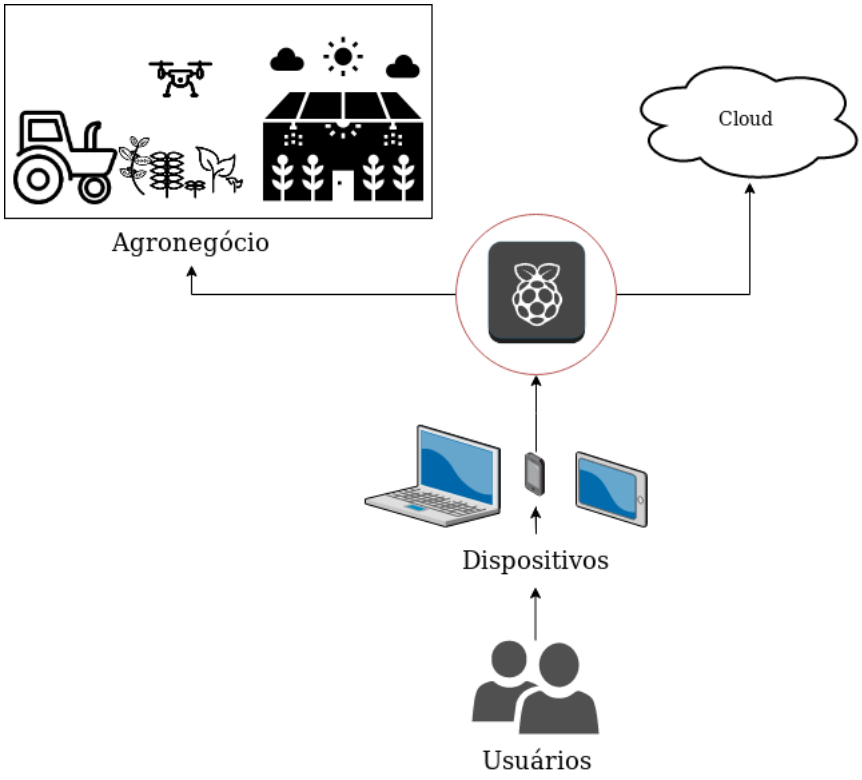


Figura 1. Exemplo visual do comportamento da Raspberry PI 3 como nodo da névoa em aplicações de Agronegócio

Tabela 2. Valores atribuídos às variáveis de execução do *NDBench* durante os diferentes experimentos executados.

Parâmetro	Valores atribuídos		
	Experimento 1	**Experimento 2**	**Experimento 3**
numWrite	256 Threads	256 Threads	256 Threads
numRead	256 Threads	256 Threads	256 Threads
writeRateLimit	2.000 e/s	0 e/s	500 e/s
readRateLimit	0 l/s	2.000 l/s	1.500 l/s

determinadas situações. Inicialmente realizamos um teste que simula somente escritas no banco, depois outro que simula somente leituras e, por fim, uma simulação onde ambos ocorrem, com 75% de leituras e 25% de escritas.

Para execuções mais precisas, fizemos uso de *scripts python* para realizar as operações necessárias para execução do simulador pois, como o simulador fornece a interface de usuário, o inicio poderia ser dado por meio dela, porém a inicialização e finalização não seriam precisas. Dado que o simulador permite a sua utilização via linha de comando com chamadas *curl*, optou-se por automatizar essa parte. Todo o roteiro de testes e os arquivos necessários aqui apresentados estão disponíveis no GitHub[3].

[3]GitHub: `github.com/SouzaGuilherme/research\_PI3\_evaluate\_experiment`

4. Resultados

Após a análise dos resultados obtidos a partir da execução do simulador nos cenários criados, focamos nos resultados de latência nos percentis 95 e 99, latência média e requisições por segundo. Para uma melhor visualização dos resultados, desconsideramos o tempo de lançamento do simulador. Todos os resultados, incluindo os tempos de lançamento, estão disponíveis para consulta no GitHub[4] junto aos fontes.

A escolha em demonstrar os resultados não somente em latência média, mas também em percentil 95 e 99 vem das informações que a média nos esconde, sendo atualmente a métrica mais utilizada em servidores. Logo, com isso temos resultados como 95% dos usuários são atendidos abaixo de uma certa latência, ou seja, o servidor pode estar atendendo alguns usuários com um latência ruim e isso variará com a quantidade de usuários que o servidor esta atendendo (o mesmo ocorre ao utilizar o percentil 99).

Tabela 3. Total de escritas e leituras alcançadas com sucesso e falhas ao final de 25 minutos.

Experimentos	Escrita		Leitura	
	Sucesso	Falha	Sucesso	Falha
Experimento 1	1.990.963	2.585	-	-
Experimento 2	-	-	2.155.112	0
Experimento 3	834.606	26	808.983	0

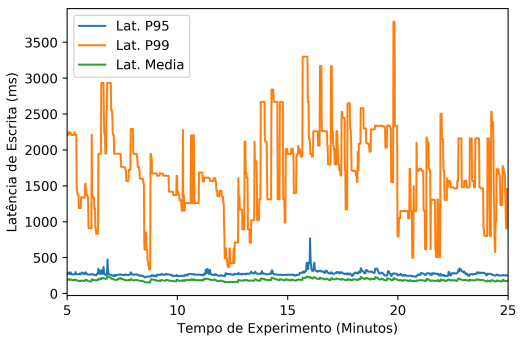
Podemos observar, na Tabela 3, o montante de operações realizadas durante os 25 minutos de execução do simulador, podendo ser visto que são alcançadas em torno de dois milhões de operações em ambos cenários. O Experimento 1 apresentou uma taxa de falhas em torno de $\pm$ 0.1298%, enquanto os Experimentos 2 e 3 apresentaram zero e 26 falhas respectivamente, caindo em *time out*, possivelmente pela demora da Raspberry Pi 3 em atender a demanda.

No primeiro cenário, no qual realizamos somente escritas, observou-se que a Raspberry atendia a demanda conforme sua capacidade, alcançando um atendimento de mais de 1.200 operações por segundo. Mantendo a latência média e a percentil 95 abaixo dos 500ms, essas informações estão contidas na Figura 2 (a) e (b).

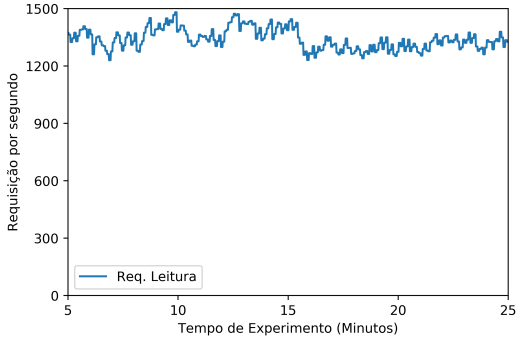
Onde havia somente leituras, a Raspberry Pi 3 demonstrou uma ótima desenvoltura, dado que conseguiu atender um número equivalente ao primeiro cenário em requisições por segundo entre 1.200 e 1.500 como pode ser visto na Figura 3 (b). Igualmente, manteve uma latência de qualidade como pode ser vista no Figura 3 (a), mantendo a média abaixo de 500ms e percentil 95 abaixo de 1000ms.

Dados os resultados anteriores, alguns valores já eram esperados neste último cenário proposto, conforme pode-se observar na Figura 4 (a) e (b). Neste cenário mais realista de escritas e leituras simultâneas, a *small board* manteve a qualidade no serviço, mesmo realizando ambas operações. Demonstrou uma baixa latência, tanto em percentil 95 estando um pouco a mais de 500ms, quanto a média, que como nos outros testes se manteve abaixo dos 500ms. Ao olharmos para as operações que realizou durante o período de execução do simulador, podemos ver que chegava em torno de mais de 450 requisições por segundo de escritas e leituras, como demonstrado na Figura 4 (c). Visto que

[4]GitHub: `github.com/SouzaGuilherme/research_PI3_evaluate_experiment`



(a) Latência de escritas



(b) Requisições por segundo

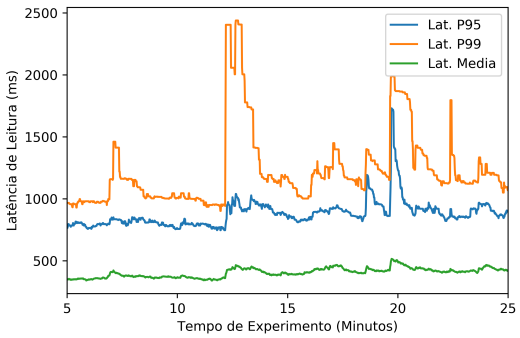
Figura 2. Latências e requisições por segundo durante escrita no período de execução de 25 minutos desconsiderando os 5 primeiros do simulador.

nos resultados anteriores os alcances em requisições foram maiores, pudemos observar que a placa tenta distribuir igualmente o trabalho de forma a atender ambas necessidades.

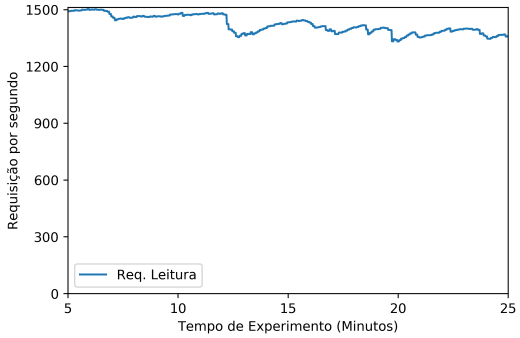
5. Conclusão

De acordo a seção anterior, a Raspberry Pi 3 mostrou-se eficiente nos cenários propostos, alcançando boas respostas em latência, dado que ao observar os resultados em percentil 95 se mantém entre 500ms e 1.000ms. Observando a média, em todas operações se mantiveram abaixo dos 500ms que, para inúmeras aplicações, torna-se mais que necessário e viabiliza o ajuste de muitas aplicações que podem vir a se enquadrar. Podemos ver que ela atende uma boa demanda de requisições por segundo, mantendo-se em torno de 1.100 operações, que representa uma quantidade de atendimento adequada a uma grande gama de aplicações.

Com tais dados, pudemos demonstrar a capacidade da Raspberry Pi 3 em atender a necessidades de muitas aplicações, suprindo os requisitos exigidos pelo paradigma de névoa, tendo em vista os quatro pontos essenciais da névoa. O trabalho aqui apresentado

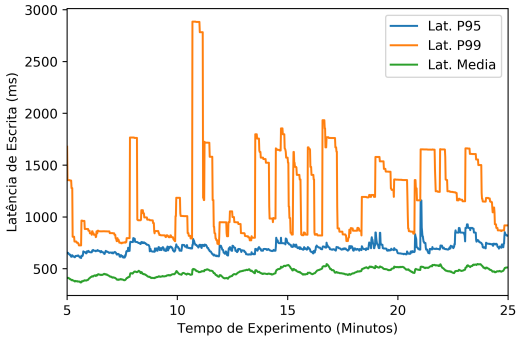


(a) Latências de leituras

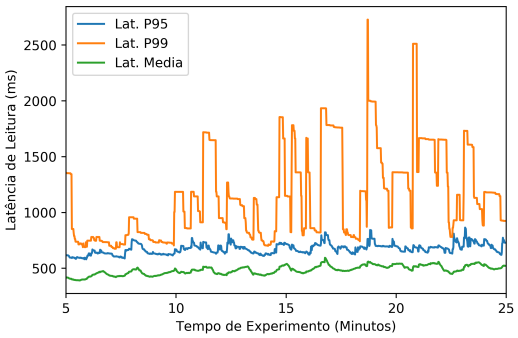


(b) Operações por segundo

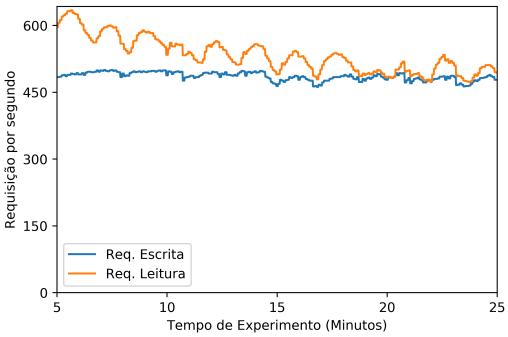
Figura 3. Latências e requisições por segundo durante leitura no período de execução de 25 minutos desconsiderando os 5 iniciais do simulador.



(a) Latências de escritas



(b) Latências de leituras



(c) Operações por segundo

Figura 4. Latências e requisições por segundo de escrita e leitura no período de execução de 25 minutos desconsiderando os 5 iniciais do simulador.

quantificou um pouco do poder do dispositivo em resposta de latência para aplicações semelhantes aos cenários aqui propostos. Como já demonstrado em outros trabalhos, os quesitos energia e peso também são supridos pelo dispositivo.

Em suma, uma possível aplicação seria como integrante em um sistema de carros autônomos [Gerla et al. 2014], para operações não críticas, porém que dependem de baixa latência e uma grande confiabilidade nos dados, dado que, ao percorrer o caminho da rede, pode haver perca de pacotes. Dessa forma é possível diminuir essa perda e o tráfego na rede, enviando somente dados necessários ajudando no treinamento e controle das plataformas. Semelhantemente, aplicações industriais podem ser viáveis, com a chegada do que chamam de indústria 4.0, que integra o uso de IoT em suas aplicações, de forma a possibilitar o uso da *small board*, mantendo o processamento mais próximo do equipamento assim ganhando uma baixa latência e um pré-processamento dos dados [Radanliev et al. 2019].

Porém, o dispositivo apresenta limitações, dado que algumas vezes ele consegue atender uma demanda maior de requisições por segundo, mas logo cai novamente. Ao tentar aumentar a demanda via simulador, ocorre um grande número de falhas o que acaba por atrapalhar a avaliação. Isso, no entanto, não é muito diferente de plataformas mais robustas. Outro fator restritivo é a memória, dado que maioria das *small boards* contam com 1GB. Tendo em vista que tanto o *NDBench* e o Cassandra são aplicações

robustas, que disponibilizam uma série de parâmetros para que atuem melhor em cenários específicos, através de técnicas como *tuning*, que não foram exploradas neste trabalho, talvez seja possível obter resultados melhores para certas aplicações.

Como trabalhos futuros, cenários diferentes podem vir a ser testados, além de novas aplicações, sendo possível criar um cenário próximo do gerado pelas empresas *Netflix* e *Facebook*. Teríamos, assim, exemplos tanto de aplicações quanto resultados concretos sobre seu uso atual. Da mesma forma, pode-se explorar aplicações mais restritas a cidades inteligentes e aplicações adequadas à indústria 4.0, como pospostos aqui. Possibilidades são abertas quanto ao próprio *hardware*, como o uso da Raspberry Pi 4[5], além do aumento horizontal do equipamento, que nos possibilita um *cluster* de Raspberry Pi, pois mesmo com o aumento em número de dispositivos, dado seu tamanho, ele ainda continuaria a atender o paradigma. Porém, estudos sobre seu comportamento devem ser realizados para uma quantificação adequada, até mesmo para verificar se o ganho em desempenho e o aumento em consumo de energia é justificado.

Referências

Bittencourt, L. F., Diaz-Montes, J., Buyya, R., Rana, O. F., and Parashar, M. (2017). Mobility-aware application scheduling in fog computing. *IEEE Cloud Computing*, 4(2):26–35.

Bonomi, F., Milito, R., Zhu, J., and Addepalli, S. (2012). Fog computing and its role in the internet of things. In *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, MCC '12, pages 13–16, New York, NY, USA. ACM.

Byers, C. C. (2017). Architectural imperatives for fog computing: Use cases, requirements, and architectural techniques for fog-enabled iot networks. *IEEE Communications Magazine*, 55(8):14–20.

Cattell, R. (2011). Scalable SQL and NoSQL data stores. *Acm Sigmod Record*, 39(4):12–27.

Featherston, D. (2010). Cassandra: Principles and application. *Department of Computer Science University of Illinois at Urbana-Champaign*.

G. Souza, J. O. and Pilla, M. (2018). Comparação de desempenho do workload ycsb em raspberry pi b+ e 3. In *XVIII Escola Regional de Alto Desempenho*, pages 117–120, Porto Alegre/RS, Brasil.

Gerla, M., Lee, E., Pau, G., and Lee, U. (2014). Internet of vehicles: From intelligent grid to autonomous cars and vehicular clouds. In *2014 IEEE World Forum on Internet of Things (WF-IoT)*, pages 241–246.

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.

Huang, T., Lin, W., Li, Y., He, L., and Peng, S. (2019). A latency-aware multiple data replicas placement strategy for fog computing. *Journal of Signal Processing Systems*.

[5] `www.raspberrypi.org/products/raspberry-pi-4-model-b`

Lakshman, A. and Malik, P. (2010). Cassandra: A decentralized structured storage system. *SIGOPS Oper. Syst. Rev.*, 44(2):35–40.

Maksimović, M., Vujović, V., Davidović, N., Milošević, V., and Perišić, B. (2014). Raspberry pi as internet of things hardware: performances and constraints. *Design Issues*, 3(8).

Mell, P. and Grance, T. (2011). The NIST definition of cloud computing. NIST Special Publication 800–145.

Papapanagiotou, I. and Chella, V. (2018). NDBench: Benchmarking microservices at scale. *arXiv preprint arXiv:1807.10792*.

R. Nakhkash, M., Nguyen gia, T., Azimi, I., Anzanpour, A., Rahmani, A. M., and Liljeberg, P. (2019). Analysis of performance and energy consumption of wearable devices and mobile gateways in iot applications. In *International Conference on Omni-Layer Intelligent Systems – COINS '19*, pages 68–73.

Radanliev, P., De Roure, D. C., Nurse, J. R., Montalvo, R. M., and Burnap, P. (2019). The Industrial Internet-of-Things in the Industry 4.0 supply chains of small and medium sized enterprises. *University of Oxford*.

Van Heddeghem, W., Lambert, S., Lannoo, B., Colle, D., Pickavet, M., and Demeester, P. (2014). Trends in worldwide ict electricity consumption from 2007 to 2012. *Comput. Commun.*, 50:64–76.