Modelos Estocásticos para Análise de Desempenho e Custo de uma Arquitetura Distribuída Nuvem e Névoa

Francisco Airton Silva[∓], Glauber Dias Gonçalves[∓], Iure Fé[◇] Jederilson Luz[∓] e Erick MacGregor[∓]

[1] [∓]Universidade Federal do Piauí (UFPI)
[◇] Terceiro BEC - Exército Brasileiro
Picos - PI - Brasil
E-mail: faps@ufpi.edu.br

Abstract. *The fog and cloud computing may perform tasks together to attend different types of applications. However, taking into account variables such as latency, workload and computational capacity, it becomes complex to define under what circumstances it is more advantageous to use the cloud layer or the fog. This paper proposes a Stochastic Petri Net (SPN) to model such a scenario by considering cloud and fog, varying amounts of nodes and workloads. We also present a case study that is a practical guide to infrastructure administrators to adjust their architectures by finding the trade-off between cost and performance.*

Resumo. *A computação em nuvem e a computação em névoa podem trabalhar em conjunto para atender diferentes tipos de aplicações. Porém, levando em consideração variáveis como latência, carga de trabalho e capacidade computacional, se torna complexo definir em que circunstâncias é mais vantajoso usar a camada de nuvem ou a névoa. Este artigo propõe um modelo de Rede de Petri Estocástica (SPN) para modelar tal cenário considerando a nuvem e a névoa com variadas quantidades de nós e variadas cargas de trabalho. Apresentamos também um estudo de utilização do modelo que serve como um guia prático para auxiliar administradores de infraestruturas computacionais a adequar suas arquiteturas encontrando um compromisso satisfatório entre custo e desempenho.*

1. Introdução

A organização das Nações Unidas (ONU) estima que a proporção da população urbana planetária aumente em 68% até 2050, o que corresponde a um acréscimo de aproximadamente 2,5 bilhões de pessoas nas regiões urbanas[1]. Esse aumento da população inevitavelmente contribuirá para exacerbar os problemas já existentes das cidades como o congestionamento do tráfego, poluição e baixa qualidade de serviços públicos. O planejamento de cidades inteligentes é uma visão ambiciosa para lidar com tais problemas. Essa visão se baseia principalmente na integração entre tecnologias emergentes como IoT (do inglês *Internet of Things*), computação em nuvem e em névoa [Perera et al. 2017].

IoT é um termo para descrever objetos que podem se comunicar através da Internet. Tais objetos variam de sensores a atuadores que controlam objetos físicos com novas formas de interação, requerendo avanços nas interfaces humano-computador. Porém,

[1] `https://www.un.org/development/desa/en/news/population/2018-revision-of-world-urbanization-prospects.html`

apesar de promissores, os dispositivos IoT apresentam limitações de capacidade de processamento e armazenamento. Poucos dispositivos, mesmo em conjunto, não conseguem atender os requisitos de aplicações de alto desempenho ou com grande volume de dados.

Para mitigar esse problema, a computação em nuvem se apresenta como uma alternativa com recursos computacionais remotos virtualmente "ilimitados". O serviço de nuvens públicas, por exemplo, permite usuários diversos adquirirem recursos de computação de grandes provedores. Os usuários podem alugar instâncias de máquinas com capacidades diferentes conforme necessário e pagar pelo seu uso. Apesar de bem estabelecido no mercado, a computação em nuvem também possui algumas limitações. Além das preocupações com segurança e privacidade, existe o problema com relação a latência em caso de uma conexão de má qualidade [Wang and Ng 2010].

Os desafios relacionados à latência na comunicação com a nuvem fizeram surgir a chamada computação em névoa. A névoa se apresenta como uma camada mediadora de comunicação entre os dispositivos IoT e a nuvem, transferindo parte da capacidade da nuvem para a borda da rede. Por estar na borda da rede é possível usar recursos computacionais na névoa com capacidades limitadas para conter custos. Atualmente existe a possibilidade de usar micro-controladores com recursos restritos de forma colaborativa.

Estudos recentes buscam integrar computação em névoa e nuvem no processamento de dados de sensores IoT para aplicações das cidades inteligentes. Alguns estudos focam em arquiteturas híbridas (névoa e nuvem) para esse propósito e avaliam seus desempenhos via protótipos com dispositivos reais [Borthakur et al. 2017, Xu et al. 2017, Elkhatib et al. 2017]. Tal avaliação se tornaria inviável em termos de custos em análises variando quantidade e variedade de dispositivos. Modelos analíticos se tornam então uma alternativa para avaliação de desempenho desses sistemas.

Com esse propósito, alguns estudos modelam processamento de dados em arquiteturas híbridas névoa e nuvem como problemas de otimização [Li et al. 2018, Li et al. 2017] ou problemas de posicionamento de nós em topologias utilizando grafos [Amarasinghe et al. 2018, Mehta et al. 2016]. Tais modelos ainda apresentam limitações para análises de desempenho. Uma limitação importante é não permitir avaliações de uma arquitetura híbrida névoa e/ou nuvem, onde o número de dispositivos (nós) disponíveis para processamento de dados na névoa e/ou na nuvem sejam definidos livremente. Essa opção ofereceria maior flexibilidade ao analista para planejar e avaliar arquiteturas híbridas sob demanda para diferentes cenários.

Visando estender as opções para análises de desempenho dos modelos mais recentes da literatura (Seção 2), neste artigo, focamos na arquitetura híbrida névoa e nuvem (Seção 3) e propomos um modelo baseado em Redes de Petri Estocástica (SPN) para análise de desempenho de aplicações para cidades inteligentes que rodam sobre tal arquitetura (Seção 4 e 5). Demonstramos nosso modelo através de um exemplo de utilização baseado em dados coletados de experimentos reais, onde variamos configurações da arquitetura em termos de nós na névoa ou nuvem, e avaliamos seu desempenho para uma determinada carga de trabalho (Seção 6).

O modelo permitiu avaliar o compromisso entre tempo médio de execução (MRT) e custo arquitetural. Observamos, por exemplo, que o tempo de envio de dados para a nuvem é um dos fatores com maior impacto no MRT, ao passo que, configurações que

mesclam nós na névoa e na nuvem poucas vezes obtiveram vantagens sobre configurações apenas névoa ou nuvem em termos de MRT e também custo-benefício. Em suma, as principais contribuições desse artigo são: (i) um modelo analítico, que é uma ferramenta útil para administradores/desenvolvedores de sistemas para cidades inteligentes checar o desempenho de mudanças no sistema, antes que elas sejam implementadas, e (ii) um estudo de utilização do modelo que provê um guia prático para análise de desempenho em arquiteturas híbridas (névoa e nuvem).

2. Trabalhos Relacionados

Existe uma variedade de estudos que exploram computação na névoa e/ou nuvem. A integração névoa e nuvem foca, primordialmente, no processamento de dados e aquisição de conhecimento da IoT [Perera et al. 2017], ou seja, sensores adaptados para monitorar uma variedade de cenários, em especial, serviços nas metrópoles como trânsito, temperatura, segurança e etc. Em [Borthakur et al. 2017, Xu et al. 2017, Vilalta et al. 2018] são propostas arquiteturas para esse objetivo, que utilizam dispositivos na névoa para o processamento e análise primária dos dados, e máquinas virtuais (*VMs*) na nuvem para extração de padrões e predições. Esses trabalhos focam em avaliações baseadas em protótipos, e não visam, portanto, analisar configurações que otimizam desempenho das arquiteturas propostas com cargas de trabalho realistas. Tal análise se tornaria inviável em termos de custos para aquisição dispositivos em quantidade e variedade.

O desenvolvimento de modelos analíticos vem sendo uma abordagem utilizada para analisar o desempenho de várias configurações de arquiteturas que integram névoa e nuvem. Uma linha de estudo consiste em modelar o escalonamento de processos em nós na névoa ou nuvem como um problema de otimização [Li et al. 2018, Li et al. 2017, Souza et al. 2016]. O objetivo desses modelos é decidir onde processar os dados obtidos dos sensores, considerando uma série de características e minimização de recursos, especialmente consumo de energia dos dispositivos na névoa e atrasos na comunicação com a nuvem. Outra linha de modelos foca no problema de posicionar nós na névoa ou nuvem para diferentes topologias de redes, incorporando grafos aos problemas de otimização [Amarasinghe et al. 2018, Mehta et al. 2016]. Tais modelos ainda não são flexíveis o suficiente para avaliar o planejamento de arquiteturas híbridas, pois não permitem avaliações com o número de nós disponíveis para processamento de dados na névoa e/ou na nuvem definidos livremente. Essa opção permitiria ao analista avaliar o custo e o benefício de várias configurações da arquitetura para o cenário de seu interesse.

Para esse propósito, nesse artigo exploramos Redes de Petri Estocásticas (SPN). Uma rede de Petri é um grafo bipartido direcionado que pode ser usado na modelagem e descrição de sistemas; contém componentes em sua estrutura, como conjuntos de locais e transições; e é estocástica quando cada transição está associada a um atraso de disparo aleatório que segue um processo estocástico. SPNs são conhecidos pelo alto grau de representatividade, sendo mais intuitivos que opções convencionais, como cadeias de Markov, para representar concorrência, paralelismo, e sincronização, em sistemas variados, desde transporte urbano [Labadi et al. 2015] à segurança de aplicações na nuvem [Almutairi and Shetty 2017]. Mais próximo ao nosso estudo é o modelo SPN proposto em [Santos et al. 2018] para analisar tempo de resposta de uma aplicação de monitoramento cardíaco sobre a arquitetura névoa e nuvem. O uso de SPNs permite estimativas rápidas e acuradas dessa métrica, mas os autores não modelam escalonamento

de tarefas em múltiplos nós na névoa ou nuvem, assim como não avaliam o custo da arquitetura. Nosso artigo contempla essas questões, associado às vantagens de SPN.

3. Arquitetura

A Figura 1 ilustra a arquitetura que propomos modelar e analisar desempenho. Se trata de uma arquitetura híbrida que integra módulos névoa e nuvem direcionada a processar fluxo de dados, coletados de redes de sensores IoT metropolitanas, para aplicações com propósitos variados [Perera et al. 2017]. Por exemplo, alertas instantâneos sobre o trânsito ou previsão de tempo (Atuadores IoT), monitoramento de serviços para a população em geral como água, energia, segurança e etc.

No centro dessa arquitetura temos o componente intermediário *Front-End*, responsável por gerenciar a chegada de tarefas (fluxos de dados) continuamente. Em suma, o *Front-End* é composto de quatro etapas: (1) receber dados de um conjunto de sensores (tarefas); (2) distribuir tarefas para processamento nos módulos névoa e/ou nuvem; (3) gerenciar a finalização das tarefas nesses módulos; e por fim (4) retornar respostas aos serviços que requisitam as tarefas. Assumimos que o módulo *Front-End* está conectado aos sensores/atuadores IoT e dispositivos de usuários que obtém serviços da arquitetura via uma rede de baixa latência, por exemplo, uma rede Wi-Fi local. Esse deve ser um cenário típico previsto para as aplicações e serviços baseados em IoT.

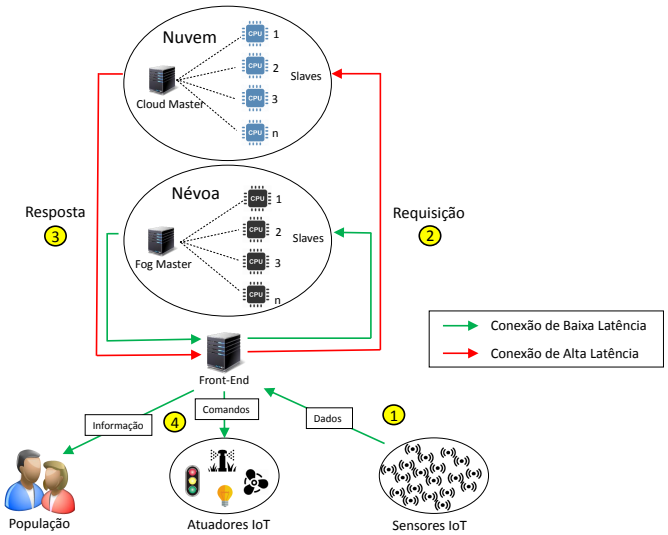


Figura 1. Arquitetura que integra Névoa e Nuvem para processar dados IoT.

O módulo névoa é composto de um servidor central (*Fog Master*) responsável por receber tarefas do módulo *Front-End* e distribuí-las entre os nós processadores da névoa. Esses nós devem ser preferencialmente computadores compactos ou SBCs (do inglês Single Board Computers) que não requerem espaço físico dedicado e condições especiais de refrigeração. Isso permite flexibilidade para instalação ou modificação do local desses computadores, bastando apenas alimentação por energia elétrica. Assumimos que os módulos *Front-End* e névoa estão conectados por uma rede de baixa latência, por exemplo, uma rede Wi-Fi local. Note que *Front-End* e *Fog Master* podem estar fisica-

mente em um mesmo computador, embora haja a separação lógica entre eles mostrada na Figura 1.

Por sua vez, o módulo Nuvem também é composto por um nó central (*Cloud Master*) responsável por receber tarefas do módulo *Front-End* e distribuí-las entre os nós processadores da nuvem, contudo os nós escravos da nuvem possuem maior poder computacional. Esses nós devem ser preferencialmente computadores virtuais (VMs) alugados em provedor(es) de nuvem pública. Isso permite flexibilidade para escalar o poder computacional da arquitetura de acordo com o volume de tarefas a ser processado e também restrições de tempo de resposta de aplicações IoT. Assumimos que os módulos *Front-End* e nuvem estão conectados por uma rede de alta latência, por exemplo, conexões providas por provedores de Internet (ISPs). Por padrão assumimos que o *Cloud Master* está em uma VM separada de nós escravos.

4. Proposta de Modelo SPN

Nessa seção, descrevemos a nossa proposta de modelo baseado em redes de Petri estocásticas (SPN) para representar a arquitetura que integra névoa e nuvem apresentada na Seção 3. Enfatizamos que o objetivo do nosso modelo é auxiliar administradores de sistemas baseados nessa arquitetura na tarefa complexa de ajustar vários parâmetros adequadamente para alcançar níveis de desempenho desejáveis. Logo, nosso modelo deve ser útil para checar o efeito de mudanças no sistema, antes que elas sejam implementadas.

A Figura 2 apresenta o nosso modelo SPN composto dos seguintes componentes: (i) *Admission* que trata da chegada de trabalhos; (ii) *Front-End*, que representa a máquina responsável por receber as requisições e escalonar para nuvem ou para névoa; (iii) *Transmission*, que representa o envio de dados para a nuvem; e por fim, os componentes (iv) *Fog* e (v) *Cloud* que recebem dados e os processam, distribuindo entre os nós que os compõem. Os componentes são representados por elementos gráficos do tipo lugares (círculos), transições temporizadas (barras vazias) e marcações de locais (barras preenchidas). A Tabela 1 apresenta todos os elementos do modelo extensivamente. As transições temporizadas são parametrizadas com distribuições de probabilidade. O administrador do sistema deve informar essas distribuições de acordo com a literatura ou realizando medições e caracterizações do sistema.

Dado a visão geral do modelo, descrevemos agora o fluxo de processamento de dados entre seus componentes. A sub-rede *Admission* é composta por dois lugares **P_Arrival** e **P_ArrivalQueue**, que representam a espera entre chegada de trabalhos e a aceitação desses trabalhos na fila, respectivamente. Os tokens em **P_Arrival** e **P_ArrivalQueue** representam quaisquer tipos de requisições que envolvam entrada de dados e que possam ser distribuídos. Os tempos entre chegadas de trabalhos são atribuídos à transição **T_Arrival**. Consideramos que os tempos entre disparos são exponencialmente distribuídos, essa suposição pode ser modificada, alterando essa distribuição. A sub-rede de admissão também não considera o atraso proveniente da transmissão do cliente para o sistema. A transição **T_Arrival** leva em conta apenas o tempo que as requisições entraram no sistema, ou seja, não são levadas em conta as perdas provenientes da rede. A transição **T6** representa o recebimento da requisição, note que é uma transição imediata, não possuindo atraso associado. **T6** dispara assim que estiver um token em **P_ArrivalQueue** e

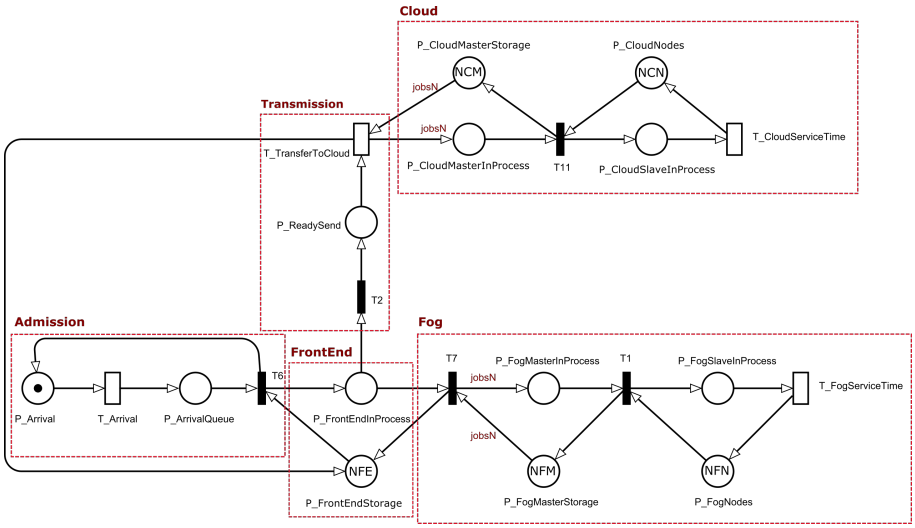


Figura 2. Modelo SPN para uma Arquitetura Névoa/Nuvem

Tabela 1. Descrição dos Elementos do Modelo

Tipo	Nome do Elemento	Descrição
Lugares	P_Arrival	Espera por novas requisições (conjuntos de trabalhos)
	P_ArrivalQueue	Espera pela disponibilidade da fila
	P_FrontEndInProcess	Trabalhos na fila do Front-End
	P_FrontEndStorage	Capacidade do Front-End
	P_FogMasterInProcess	Trabalhos na fila do master da névoa
	P_FogMasterStorage	Capacidade do master da névoa
	P_FogSlaveInProcess	Trabalhos na fila dos nós escravos da névoa
	P_FogNodes	Capacidade da névoa
	P_ReadySend	Trabalho pronto para envio à nuvem
	P_CloudMasterInProcess	Trabalhos na fila do master da nuvem
	P_CloudMasterStorage	Capacidade do master da nuvem
	P_CloudSlaveInProcess	Trabalhos na fila dos nós escravos da nuvem
	P_CloudNodes	Capacidade da nuvem
Transições Temporizadas	T_Arrival	Tempo de chegada entre conjuntos de trabalhos
	T_FogServiceTime	Tempo para execução de um único trabalho em um único nó da névoa
	T_TransferToCloud	Tempo para transferência de uma requisição (conjunto de trabalhos)
	T_CloudServiceTime	Tempo para execução de um único trabalho em um único nó da nuvem
Marcações dos Lugares	NFE	Número de espaços disponíveis de armazenamento no FrontEnd
	NFM	Número de espaços disponíveis de armazenamento no master da névoa
	NFN	Número de nós disponíveis para execução na névoa
	NCM	Número de espaços disponíveis de armazenamento no master da nuvem
	NCN	Número de nós disponíveis para execução na nuvem
	jobsN	Número de trabalhos que constituem a requisição atual

pelo menos um token em **P_FrontEndStorage**.

Quando **T6** dispara, a sub-rede do Front-End é alcançada, um token é retirado de **P_ArrivalQueue** e **P_FrontEndStorage** e tem como pós-condição a marcação do lugar **P_Arrival** (permitindo um novo disparo) e a adição de um token em

P_FrontEndInProcess. A quantidade de tokens em **P_FrontEndInProcess** representa o enfileiramento de requisições no FrontEnd. O enfileiramento ocorre quando não há capacidade disponível para servir a requisição recém-chegada. Se houver capacidade disponível na nuvem ou na névoa, as transições **T2** ou **T7** são disparadas e posteriormente a requisição segue em frente para ser processada. As transições imediatas possuem um atributo de peso indicando a probabilidade que ela pode ser disparada. Portanto, o avaliador pode definir se prefere dar prioridade de envio para a nuvem ou para a névoa simplesmente configurando os pesos de **T2** e **T7**, respectivamente.

Quando existe um token na sub-rede *Transmission*, ou seja, no lugar **P_ReadySend**, isto significa que a nuvem foi escolhida para executar a requisição. **T_TransferToCloud** refere-se ao tempo que uma requisição (conjunto de arquivos por exemplo) levará para chegar até a nuvem em questão. **T_TransferToCloud** só é disparada se o servidor master da nuvem possuir recurso suficiente de armazenamento, ou seja, a variável NCM (do lugar **P_CloudMasterStorage**) for maior do que 0. Quando o FrontEnd envia todo o conjunto de dados da requisição para a nuvem, tais dados são apagados do FrontEnd, por isso existe um arco entre **T_TransferToCloud** e **P_FrontEndStorage**.

Quando a requisição chega no servidor master da névoa ou da nuvem, ambas possuem o mesmo funcionamento, por isso iremos explicar as duas sub-redes em conjunto. As variáveis NFM (da névoa) e NCM (da nuvem) indicam a quantidade de espaço de armazenamento das máquinas master respectivas. Caso haja espaço de armazenamento suficiente, os lugares **P_FogMasterInProcess** ou **P_CloudMasterInProcess** consomem um número de trabalhos representados pela variável **jobsN**. A variável **jobsN** indica a multiplicidade do arco, ou seja, a quantidade possível de trabalhos que uma requisição pode se subdividir. Nosso modelo considera que cada trabalho (*job*) terá o mesmo tamanho. Por exemplo, em uma aplicação de processamento de logs, **jobsN** representaria uma quantidade de arquivos de texto de tamanhos iguais.

O disparo de **T11** e **T1** representa o início do processamento de mais um trabalho. Estes disparos são condicionados à quantidade de nós disponíveis para processamento. As marcações em NCN de **P_CloudNodes** e NFN de **P_FogNodes** indicam a quantidade de nós disponíveis na nuvem e na névoa, respectivamente. À medida que os trabalhos vão sendo consumidos pelos lugares **P_CloudSlaveInProcess** e **P_FogSlaveInProcess**, tokens são retirados de **P_CloudNodes** e **P_FogNodes**. Este fluxo significa que cada trabalho será alocado em um recurso à medida que for chegando. O tempo que os trabalhos permanecem em processamento em um nó depende das transições **T_CloudServiceTime** e **T_FogServiceTime**. Tais transições possuem a semântica *infinite server*, então, cada trabalho é processado independentemente. É importante notar que o tempo de processamento depende muito da capacidade computacional de cada nó. Neste trabalho consideramos que todos os nós de cada camada possuem a mesma capacidade computacional. Assim, as transições **T_CloudServiceTime** e **T_FogServiceTime** devem ser configuradas com um tempo de processamento para um único trabalho em um tipo de recurso específico. Como falado anteriormente, esse tempo pode ser medido em experimentos reais ou serem encontrados na literatura. Em caso de execução de experimentos, deve-se associar uma distribuição aos valores amostrais.

O modelo proposto permite avaliar um número muito grande de cenários pois o avaliador deve configurar 12 parâmetros (vide Tabela 1). Os parâmetros incluem: as 4

transições temporizadas; as 6 marcações de lugares relacionados a recursos ou carga de trabalho; e ainda os pesos das 2 transições imediatas (**T2** e **T7**). Qualquer alteração em um destes parâmetros pode impactar significativamente no tempo médio de resposta do sistema e consequentemente no custo de infraestrutura. A variação das possibilidades de cenários considerando um grande número de fatores da arquitetura híbrida é o que torna este modelo a principal contribuição deste trabalho.

5. Métricas de Desempenho

Nesta seção, definimos métricas para avaliar a arquitetura névoa e/ou nuvem com base no modelo proposto.

5.1. Tempo Médio de Resposta (*MRT*)

O tempo médio de resposta (*MRT*) pode ser obtido a partir da Lei de Little [Little 1961]. A Lei de Little relaciona o número médio de requisições em progresso em um sistema ($RequestsInProgress$), a taxa de chegada ($Arrival_{Rate}$) e o tempo médio de resposta (MRT). Como mostrado anteriormente, uma requisição se subdivide em um conjunto de trabalhos, e é diretamente impactada pela taxa de chegada. A taxa de chegada é o inverso do tempo de chegada. Considerando a transição para tempo entre chegadas do modelo, temos que $Arrival_{Rate} = \frac{1}{T\_Arrival}$. Vale ressaltar que a Lei de Little requer um sistema estável, ou seja, que possua uma taxa de requisições menor que a taxa de processamento dos servidores. No modelo proposto pressupomos que a taxa de chegada real não necessariamente é a taxa de chegada efetiva, pois alguns trabalhos podem se perder, ou por termos filas finitas, serem descartados. Então, como recomendado pelo autor Jain (1990), nós subtraímos a probabilidade de descartes representado pela variável $N\_Discard$. Portanto, a equação correspondente à Lei de Little para MRT utilizada no nosso modelo é expressa na Equação 1.

$$\textbf{MRT} = \frac{RequestsInProgress}{Arrival_{Rate} \times (1 - N\_Discard)} \tag{1}$$

A equação 2 define $RequestsInProgress$. Para calcular o número de requisições em progresso no sistema, precisa-se somar a quantidade de tokens em cada um dos lugares que representam uma requisição em andamento. Como uma requisição é subdividida em um conjunto de trabalhos, nesta equação deve-se compensar esta multiplicidade nos lugares onde relaciona-se com trabalhos e não requisições. Na equação 2, $Esp(Lugar)$ representa a esperança estatística de existir tokens em "Lugar", onde $Esp(Lugar) = (\sum_{i=1}^{n} P(m(Lugar) = i) \times i)$. Em outras palavras, $Esp(Lugar)$ indica quantos tokens ocupam aquele Lugar.

$$\begin{aligned}\textbf{RequestsInProgress} = {} & Esp(P\_FrontEndInProcess) + Esp(P\_ReadySend) + \\ & \frac{Esp(P\_CloudMasterInProcess)}{jobsN} + \frac{Esp(P\_CloudSlaveInProcess)}{jobsN} + \\ & \frac{Esp(P\_FogMasterInProcess)}{jobsN} + \frac{Esp(P\_FogSlaveInProcess)}{jobsN}\end{aligned} \tag{2}$$

A equação 3 define $N\_Discard$. Para calcular o descarte é necessário existir token na fila de entrada (P_ArrivalQueue) e não restar mais nenhum recurso disponível dentro

das sub-redes. $P(Lugar = n)$ calcula a probabilidade de existirem n tokens em "Lugar". Por fim, além do *MRT* nós calculamos também o nível de utilização dos recursos, a utilização da nuvem é dada por $UC = \frac{Esp(P\_CloudSlaveInProcess)}{NCN}$ e a utilização da névoa é dada por $UF = \frac{Esp(P\_FogSlaveInProcess)}{NFN}$.

$$\begin{aligned}\textbf{N\_Discard} = P((P\_ArrivalQueue = 1) \wedge (P\_FrontEndStorage = 0) \wedge \\ (P\_FogMasterStorage = 0) \wedge (P\_FogNodes = 0) \wedge \\ (P\_CloudMasterStorage = 0) \wedge (P\_CloudNodes = 0))\end{aligned} \tag{3}$$

5.2. Custo da Arquitetura

Propomos avaliar o custo da arquitetura do ponto de vista do provedor/administrador da aplicação (IoT) com opções de adquirir uma infraestrutura própria na névoa, alugar VMs na nuvem ou ambos, considerando requisitos de MRT. Logo, esse custo, corresponde ao consumo de energia da névoa, custo de aquisição de nós da névoa e aluguel de nós na nuvem em um período de tempo t. O custo é dado por

$$C_t = (N_n \times E_{n,t}) + (N_n \times P_{n,t}) + (N_c \times M_{c,t}) \tag{4}$$

onde N_n e N_c são número de nós da névoa e da nuvem respectivamente, $E_{n,t}$ é o custo com energia elétrica por nó da névoa, $P_{n,t}$ é o custo de propriedade, ou seja, compra do dispositivo nó da névoa, e $M_{c,t}$ é o custo de locação de uma máquina virtual na nuvem. Esses custos correspondem a valores no intervalo de tempo t, e o cálculo da Equação 4 está integrado à resolução do nosso modelo.

Custos de energia e propriedade de computadores ao longo do tempo podem ser estimados detalhadamente, considerando utilização de CPU para consumo energia e depreciação do valor de propriedade à medida que computadores se tornam obsoletos [Walker 2009]. Nesse trabalho assumimos simplificações para esses custos. Consideramos o custo de energia ($E_{n,t}$) dado pela potência nominal do dispositivo em relação ao custo em kWh, e o custo de propriedade ($P_{n,t}$) dado pelo valor de compra do dispositivo sem depreciações. Note que tais simplificações são estimativas conservadoras de custo quando utiliza-se menos que 100% da capacidade de processamento do dispositivo e o renova/descarta em períodos menores ou igual a um ano [Walker 2009].

6. Considerações Práticas

Nesta seção apresentamos exemplos práticos a partir da resolução do modelo SPN para ilustrar como um administrador de sistema, baseado na arquitetura que integra névoa e nuvem, pode utilizar o modelo para avaliações de desempenho.

Com esse propósito, implementamos um algoritmo de processamento de texto distribuído. Consideramos que esse algoritmo deve ser executado no componente *Front-End* do modelo para processar logs coletados de vários sensores, um cenário comum em IoT. Assim, simulamos a coleta de dados em 20 sensores, que a cada 500 segundos gera 15MB de dados, totalizando uma requisição de 300MB a ser processada. A parametrização do modelo foi baseada em execuções do algoritmo em SBCs para representar nós da névoa e VMs para representar nós da nuvem. Utilizamos um SBC Raspberry Pi Modelo B CPU de 700 MHz 1 Core e 512 MB de RAM e uma VM modelo *t2.medium* da Amazon com processador Intel 3.3 GHz 2 Cores e 4 GB de RAM. Assumimos que os sensores enviam

logs para os nós da névoa via rede Wi-Fi local e enviam logs para os nós da nuvem via conexão de Internet dedicada com velocidades de 40 ou 160 Mbps.

A Figura 3 mostra o fluxo de execução do algoritmo de processamento de texto. Trata-se do algoritmo clássico Conta Palavras[2], bem conhecido na área de *Big Data* e procedimento *MapReduce*. Esse algoritmo conta o número de ocorrências únicas de determinadas palavras de um arquivo de texto (log). Para fazer a divisão e distribuição das tarefas, o sistema distribuído realiza um processo de divisão lógica dos blocos de dados (*split de dados*). A quantidade de tarefas *map* é determinada pela quantidade de *splits*. A partir dessa divisão, a etapa *Mapping* distribui as tarefas entre os nós (escravos) do(s) componente(s) névoa e/ou nuvem. Ao executar em cada nó, essa etapa gera um par chave-valor, sendo a chave uma palavra e o valor a quantidade 1. Para chegar ao resultado final, a etapa de ordenação gera uma lista de todos os valores por chave e a etapa *reduce* processa essa lista, gerando um arquivo com a frequência de cada palavra. Por fim, coletamos o tempo de processamento para executar as etapas *mapping* e ordenação de um arquivo de 15MB em um único nó, e os utilizamos para parametrizar o modelo (*T_FogServiceTime* e *T_CloudServiceTime*). Note que as tarefas na nuvem e névoa são balanceadas (*round-robin*) com um peso de 50% para cada componente e o armazenamento temporário foi delimitado para 20 tokens nos componentes *FrontEnd* e nós mestres dos componentes nuvem e névoa.

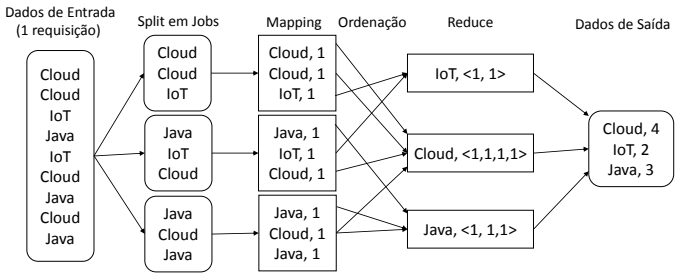


Figura 3. Fluxo de execução do algoritmo de processamento de texto.

Nesse exemplo de utilização do modelo, analisamos *qual configuração da arquitetura possui tempo médio de resposta (MRT) e custo desejáveis, variando número de nós da nuvem e/ou névoa*. Para simplificar a análise quanto a possibilidades de configurações, definimos quatro configurações representativas para a arquitetura: (i) apenas a nuvem com Internet de 160 Mbps (Nuvem A), (ii) apenas a nuvem com Internet de 40 Mbps (Nuvem B), (iii) apenas a Névoa, e (iv) configuração Híbrida que mescla nós da névoa e nuvem. Assim, primeiramente definimos alguns parâmetros fixos, que são eles: *T_Arrival* = 500s, *T_FogServiceTime* = 57,2s, *T_CloudServiceTime* = 6,47s, T2 = 50%, T7 = 50%, NFE = 20, NFM = 20, NCM = 20, e jobsN = 20. Vale ressaltar que coletamos os valores para *T_FogServiceTime* e *T_CloudServiceTime* através de experimentos reais. Posteriormente, avaliamos MRT e custo de cada configuração variando quantidades de nós da névoa e/ou nuvem. Para comparar as arquiteturas variamos cinco quantidades de nós em cada configuração: Névoa (#Rasp.) = $\{5,10,15,20,25\}$; Nuvem (#VM) = $\{1,2,3,4,5\}$; Híbrida (#VM+#Rasp.) = $\{2+5,2+10,2+15,2+20,2+25\}$. Identificamos as variações como C1, C2, C3, C4, e C5 para representar o crescimento em número de nós da arquitetura,

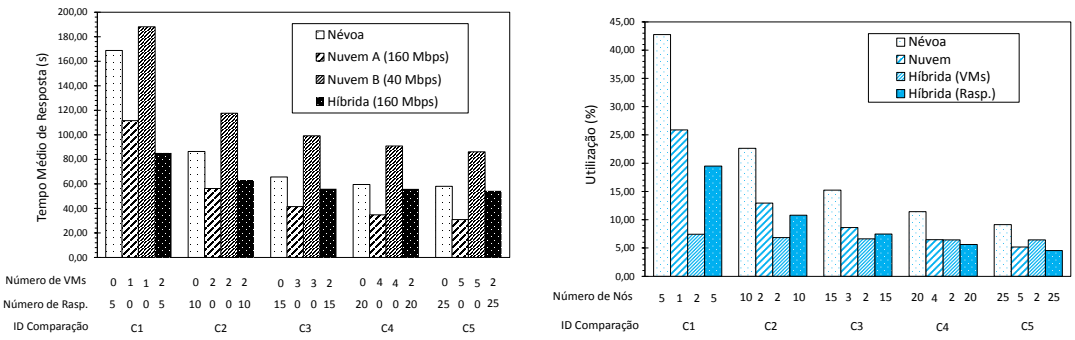
[2]WordCount https://gist.github.com/kzk/712029/9d0833aac03b23ec226e034d98f5871d9580724e

e cada identificador possui 4 configurações da arquitetura que consideramos da mesma classe, ou seja, configurações com desempenhos parecidos.

Finalmente, para analisar o custo da arquitetura, consideramos um período de um ano, que é um tempo mínimo para justificar o investimento em compra de SBCs (Raspberries) e obter VMs com menor custo via contrato antecipado de instâncias reservadas (RI) com um provedor de nuvem. Dado esse período, os custos unitários utilizados para a análise tem como referência os preços adotados por provedores de infraestrutura e comunicação mais populares no momento da escrita desse artigo: serviço de Internet com custo anual de US$ 959,88 e US$ 1319,88 para velocidades de 40 Mbps e 160 Mbps (Verizon USA); um kit Raspberry PI com custo de US$50,00 (Loja virtual da Amazon USA); o preço do kWh em $ 0,13, seguindo a média nacional no EUA[3]; e enfim, o preço anual da VM *t2.medium* por US$ 235.00 (Nuvem da Amazon).

6.1. Tempo Médio de Resposta (MRT)

Nesta parte, os resultados das análises estacionárias são apresentados e discutidos observando o tempo médio de resposta (MRT). As Figuras 4(a) e 4(b) mostram o resultado da comparação das 4 configurações de arquiteturas: apenas a Nuvem A (160 Mbps), apenas a Nuvem B (40 Mbps), apenas a Névoa, e a configuração Híbrida. A Figura 4(a) mostra especificamente os valores de MRT. A Figura 4(b) apresenta os níveis de utilização dos recursos em cada comparação. Observe que na Figura 4(b) é disposta apenas uma coluna para a arquitetura de Nuvem, pois Nuvem A e Nuvem B obtiveram níveis de utilização iguais, e são dispostas duas colunas para a utilização na forma Híbrida (VM e Rasp.).



(a) (b)

Figura 4. Comparação entre as Arquiteturas enviando 20 Arquivos de 15MB Variando o Número de Nós: (a) Tempo Médio de Resposta (s) e (b) Nível de Utilização de Recursos (%).

Configuração com Pior MRT: Considerando 300MB de dados a serem transportados para a nuvem não foi vantajoso adotar a Nuvem B em nenhuma das comparações. Uma Internet de 40 Mbps eleva notavelmente o tempo para a transferência de dados. Na comparação C1, a Nuvem B obteve um MRT de 180s. Na comparação C2, este valor baixou para 120s, ainda bem distante do melhor caso. Na comparação C5, o valor chegou a 82s. Portanto, *caso a conexão de Internet não possua uma alta velocidade deve-se cogitar fortemente a adoção de uma configuração apenas com a Névoa.*

[3]https://www.bls.gov/regions/west/news-release/averageenergyprices_losangeles.htm

Configuração com Melhor MRT: A configuração da Nuvem A obteve o melhor MRT em todas as comparações, exceto C1. Na comparação C2, por exemplo, a Nuvem A foi melhor do que a Névoa com diferença de 30 segundos e melhor do que a forma Híbrida com diferença de 7 segundos. A Nuvem A tem melhor desempenho na maioria das comparações devido a redução do tempo de transferência de dados para a nuvem. Portanto, *se há conexão de Internet de alta velocidade, sugere-se adotar apenas a nuvem*.

Quando a configuração Híbrida é Preferível: Na comparação C1, observamos que a configuração Híbrida tem um MRT menor do que as demais: 188, 168, 111 e 85 segundos respectivamente para a Nuvem B, Névoa, Nuvem A e a configuração Híbrida. Portanto, *quando o número de nós tanto da nuvem quanto da névoa é baixo (exemplo: até 5 nós na névoa e 2 nós na nuvem), a união névoa e nuvem resulta em melhor MRT*. Nas quatro comparações seguintes, aumentando-se o número de nós, nem sempre a configuração híbrida provê o melhor MRT.

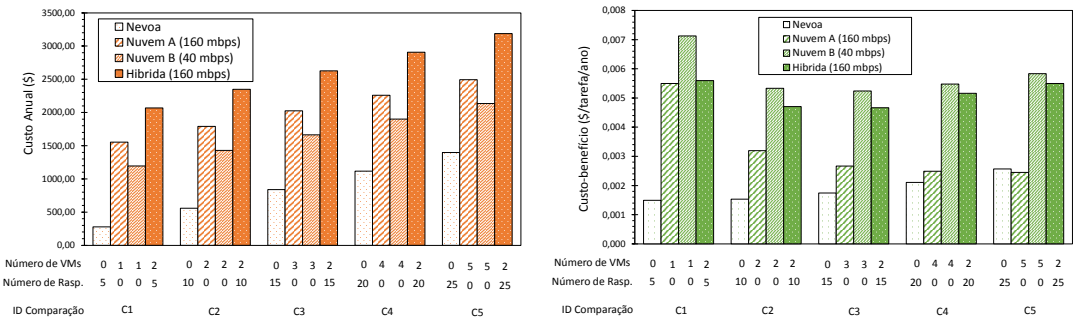
Baixa Variação do MRT a Partir da Comparação C3: Note que o MRT de todas as configurações praticamente não muda a partir da comparação C3, ou seja, há uma tendência de baixa variação o MRT à medida que se aumenta número de nós. Ainda nesta comparação C3, observamos também que o incremento de uma VM na Nuvem A e B tem baixo impacto no MRT. De fato, isso ocorre porque a utilização da nuvem a partir da comparação C3 se torna baixo (menor que 10%), como mostra a Figura 4(b). Assim, sob o ponto de vista de MRT, é recomendável a adoção da Nuvem A com 3 VMs ou da configuração Híbrida com 2 VMs e 15 Raspberries. Portanto, *deve-se atentar ao nível de utilização dos recursos pois a partir de um certo ponto não faz diferença a adição de mais nós*.

6.2. Custo da Arquitetura

Nessa seção analisamos o custo da arquitetura para as mesmas variações de números de nós apresentadas na análise de MRT. Esse custo corresponde ao consumo de energia da névoa, custo para aquisição de nós da névoa e custo de locação de nós na nuvem (VMs), conforme definição na Equação 4, para um período de um ano e custos unitários discutidos na Seção 6.

A Figura 5(a) mostra o custo anual da arquitetura para as cinco comparações em que se analisou o MRT. Observa-se que as configurações apresentam o mesmo padrão em termos qualitativos de custo para todas as comparações. Enumerados do menor para o maior custo anual, tem-se as configurações Névoa, Nuvem B, Nuvem A e Híbrida. Sendo que essa última alcança até US$ 3086,40 por ano, segundo estimativas de nosso modelo. Os custos apresentados estão alinhados com os valores de MRT em cada comparação, visto que espera-se aumentar custo à medida que se investe em mais nós na névoa ou nuvem para reduzir o MRT. A Figura 5(b), por sua vez, mostra a relação custo-benefício dado pelo custo por tarefa executada anualmente pela arquitetura, ou seja, a taxa de serviço anual. Note que a configuração Híbrida, que tem o menor MRT para a comparação com poucos nós (C1), tem custo-benefício intermediário, mas deve ser adotada em acordos de nível de serviço com restrição de MRT. Por outro lado, a configuração apenas Névoa tem o menor custo-benefício na comparação C1, mas à medida que se aumenta número de nós (para reduzir MRT em casos de restrições de nível de serviço), a configuração Nuvem A é a única que apresenta tendência de queda do custo-benefício.

Por exemplo, a configuração Nuvem A tem redução de 0,005 a 0,002 US$/tarefa/ano nas comparações de C1 a C5. Enfim, a análise de custo-benefício mostra indícios de saturação em número de nós da configuração, ou seja, aumento de custo sem redução de MRT.



(a) (b)

Figura 5. Análise de custo em relação ao número de nós: (a) estimativa de custo anual ($/ano) e (b) estimativa de custo-benefício ($/tarefa/ano).

7. Conclusão

Neste artigo propomos um modelo SPN para representar uma arquitetura distribuída nuvem e/ou névoa, e auxiliar a configuração dos parâmetros de tal arquitetura. Para demonstrar a utilização e praticidade do modelo em análises de desempenho de sistemas baseados nessa arquitetura, propomos uma aplicação de processamento de texto para prover estimativas de tempo de resposta médio (MRT) e custo da arquitetura. Os resultados com a métrica MRT permitiram observar que há casos onde apenas o uso da névoa é adequado (conexão de baixa qualidade), há casos onde é sugerível adotar apenas a nuvem (conexão de alta qualidade) e há casos onde o uso das duas camadas é mais proveitoso (baixa quantidade de nós). Quanto aos custos da arquitetura, mostramos que reduzir MRT requer aumento de custos com a adição de nós na névoa e/ou nuvem. Contudo, análises de custo-benefício, por exemplo $/tarefa/ano, podem justificar investimentos na expansão da arquitetura, visando um MRT menor. Mostramos também que em cenários com restrições de MRT por acordos de níveis de serviço, a configuração híbrida mínima (nós névoa e nuvem) ou apenas nuvem com adição de mais nós apresentam custos-benefícios satisfatórios. Como trabalhos futuros pretendemos utilizar o modelo a outros tipos de aplicações e maior variabilidade de comparações com nós na nuvem e/ou na névoa. Pretendemos também adicionar no modelo o fator disponibilidade da nuvem.

Agradecimentos

Este trabalho tem o apoio do Conselho Nacional de Desenvolvimento Científico e Tecnológico – CNPq e da empresa de telecomunicações Virtex Telecom.

Referências

Almutairi, L. M. and Shetty, S. (2017). Generalized stochastic petri net model based security risk assessment of software defined networks. In *Proc. of IEEE MILCOM*.

Amarasinghe, G., de Assunção, M. D., Harwood, A., and Karunasekera, S. (2018). A data stream processing optimisation framework for edge computing applications. In *Proc. of IEEE ISORC*.

Borthakur, D., Dubey, H., Constant, N., Mahler, L., and Mankodiya, K. (2017). Smart fog: Fog computing framework for unsupervised clustering analytics in wearable internet of things. In *Proc. of IEEE GlobalSIP*.

Elkhatib, Y., Porter, B., Ribeiro, H. B., Zhani, M. F., Qadir, J., and Rivière, E. (2017). On using micro-clouds to deliver the fog. *arXiv preprint arXiv:1703.00375*.

Labadi, K., Benarbia, T., Barbot, J., Hamaci, S., and Omari, A. (2015). Stochastic petri net modeling, simulation and analysis of public bicycle sharing systems. *IEEE Transactions on Automation Science and Engineering*, 12(4):1380–1395.

Li, H., Ota, K., and Dong, M. (2018). Learning iot in edge: Deep learning for the internet of things with edge computing. *IEEE Network*, 32(1):96–101.

Li, Y., Orgerie, A., Rodero, I., Parashar, M., and Menaud, J. (2017). Leveraging renewable energy in edge clouds for data stream analysis in iot. In *Proc. of IEEE/ACM CCGRID*.

Little, J. D. (1961). A proof for the queuing formula: L= λ w. *Operations research*, 9(3):383–387.

Mehta, A., Tärneberg, W., Klein, C., Tordsson, J., Kihl, M., and Elmroth, E. (2016). How beneficial are intermediate layer data centers in mobile edge networks? In *Proc. of IEEE FAS*W*.

Perera, C., Qin, Y., Estrella, J. C., Reiff-Marganiec, S., and Vasilakos, A. V. (2017). Fog computing for sustainable smart cities: A survey. *ACM Comp. Surv.*, 50(3):32:1–32:43.

Santos, G. L., Takako Endo, P., Ferreira da Silva Lisboa Tigre, M. F., Ferreira da Silva, L. G., Sadok, D., Kelner, J., and Lynn, T. (2018). Analyzing the availability and performance of an e-health system integrated with edge, fog and cloud infrastructures. *Journal of Cloud Computing*, 7(1):16.

Souza, V. B., Masip-Bruin, X., Marin-Tordera, E., Ramirez, W., and Sanchez, S. (2016). Towards distributed service allocation in fog-to-cloud (f2c) scenarios. In *Proc. of IEEE GLOBECOM)*.

Vilalta, R., Vía, S., Mira, F., Casellas, R., Muñoz, R., Alonso-Zarate, J., Kousaridas, A., and Dillinger, M. (2018). Control and management of a connected car using sdn/nfv, fog computing and yang data models. In *Proc. of IEEE NetSoft*.

Walker, E. (2009). The real cost of a cpu hour. *Computer*, 42(4):35–41.

Wang, G. and Ng, T. E. (2010). The impact of virtualization on network performance of amazon ec2 data center. In *Proc. of IEEE Infocom*.

Xu, Y., Mahendran, V., Guo, W., and Radhakrishnan, S. (2017). Fairness in fog networks: Achieving fair throughput performance in mqtt-based iots. In *Proc. of IEEE CCNC*.