Pós-Graduação em Ciência da Computação

Carlos Mágno

AVALIAÇÃO DA DISPONIBILIDADE DE *VIDEO SURVEILLANCE AS A SERVICE* (VSAAS)

Dissertação de Mestrado

Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
www.cin.ufpe.br/~posgraduacao

RECIFE
2015

Universidade Federal de Pernambuco
Centro de Informática
Pós-graduação em Ciência da Computação

Carlos Mágno

AVALIAÇÃO DA DISPONIBILIDADE DE *VIDEO SURVEILLANCE AS A SERVICE* (VSAAS)

Trabalho apresentado ao Programa de Pós-graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Orientador: *Prof. Dr. Paulo Romero Martins Maciel*

RECIFE
2015

Carlos Mágno
AVALIAÇÃO DA DISPONIBILIDADE DE *VIDEO SURVEILLANCE AS A SERVICE* (VSaaS)/ Carlos Mágno. – RECIFE, 2015-
113 p. : il. (algumas color.) ; 30 cm.

Orientador Prof. Dr. Paulo Romero Martins Maciel

Dissertação de Mestrado – Universidade Federal de Pernambuco, 2015.

1. Disponibilidade. 2. VSaaS. I. Paulo Romero Martins Maciel. II. Universidade Federal de Pernambuco. III. Avaliação da Disponibilidade de Video Surveillance as a Service (VSaaS)

CDU 02:141:005.7

Dissertação de mestrado apresentada por **Carlos Mágno** ao programa de Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, sob o título **AVALIAÇÃO DA DISPONIBILIDADE DE *VIDEO SURVEILLANCE AS A SERVICE* (VSaaS)**, orientado pelo **Prof. Prof. Dr. Paulo Romero Martins Maciel** e aprovado pela banca examinadora formada pelos professores:

Prof. Eduardo Antônio Guimarães Tavares
Centro de Informática - UFPE

Prof. Francisco Vieira de Souza
Departamento de Estatística e Informática - UFPI

Prof. Paulo Romero Martins Maciel
Centro de Informática - UFPE

RECIFE
2015

Dedico esta dissertação ao Senhor Deus, por mostrar a luz
nos momentos mais difíceis.

Agradecimentos

Meus sinceros agradecimentos a todos que contribuíram para o desenvolvimento deste trabalho, desde as que continuam presentes, assim como, as que por algum motivo não estão próximas.

Meu incomensurável agradecimento ao meu orientador, professor Dr. Paulo Maciel, que me acompanhou desde o início desta jornada, com sua cobrança e conselhos, confundidos a de um pai querendo o bem de um filho, sem os quais a conclusão estaria ameaçada. Ele será um exemplo a ser seguido na minha vida. Muito obrigado por acreditar em mim.

Agradeço a todos do grupo MoDCS pela amizade e parceria, em especial, minha turma: Igor, Eliomar, Verônica, Maria Clara, Júlio, Rosângela, Jonathan e Leandro Marques. Agradeço a Matheus pelo primeiro contato com a pesquisa dentro do Mestrado e a Jean que nessa reta final foi um dos grandes responsáveis pela conclusão deste trabalho.

Meu muito obrigado a minha família, especialmente a minha mãe, Sueli pelo amor incondicional e o acolhimento nos momentos de desespero e de lágrimas. Também ao meu pai, José Carlos, pelo apoio e experiências passadas. Todas as minhas decisões e conquistas obtidas em minha vida são dedicadas a eles, inclusive este meu sonho que sempre foi uma promessa feita a minha mãe, quando eu ainda tinha 14 anos: ser um cientista da Computação. Mesmo estando ausente por inúmeras vezes, nunca esqueci de vocês e os amo muito: Painho e Mainha. Agradeço a minha irmã Magda Priscila, por confabularmos juntos vários sonhos acadêmicos.

Agradeço a todos os parentes, sobretudo a uma pessoa que não posso esquecer, minha adorável Vó, dona Zefinha, que sempre perguntou por onde eu estava nos momentos ausentes e fazia a maior alegria quando passava 5 minutos conversando comigo; este trabalho também é dedicado a ela.

A minha namorada, Evelyne, obrigado por me mostrar alguns sentidos na vida, entre eles a felicidade e companheirismo, abrindo novas possibilidades e resgatando a pessoa desafiadora e alegre que há em mim. Quero que nosso amor seja eterno enquanto dure. Obrigado pela paciência e cumplicidade.

Ao Centro de Informática (CIn) da UFPE e ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), por todo o suporte recebido.

Por fim, quero agradecer ao Criador do Universo, DEUS. Só ele sabe o que passei nos momentos mais complicados da minha vida, por várias indefinições e problemas pessoais. Obrigado por me dar forças a cada vez que eu perguntei o PORQUÊ.

"Sem crise não há desafios, sem desafios, a vida é uma rotina."

—ALBERT EINSTEIN

Resumo

Nos últimos anos, sistemas de *Video Surveillance as a Service* (*VSaaS*) apresentam um aumento significativo na demanda por técnicas de segurança que elevem os níveis de confiabilidade do serviço. Em paralelo, o paradigma de Computação em Nuvem tornou-se uma importante ferramenta para serviços remotos da computação. O *VSaaS* entrega armazenamento de grande quantidade de dados. Em 2012, 50% do armazenamento em *big data* que necessitou serem analisados foram de vídeo de vigilância. Em geral, os vídeos têm um alto significado para seus proprietários, não permitindo longos períodos de interrupção. Com o objetivo de evitar baixos desempenhos e ampliar a qualidade dos serviços de vídeo são necessários mecanismos para garantir alta disponibilidade em *VSaaS*. Entretanto, esta tarefa é difícil sem gerar impacto no custo. O presente trabalho propõe dois sistemas de *VSaaS* que foram submetidos a análise de disponibilidade, por meio de modelos analíticos (RBD, CTMC e SPN). O primeiro sistema, denominado doméstico, foi caracterizado pelos elementos essenciais para uma estrutura básica do *VSaaS* para ser utilizado em casas e pequenos comércios. Estes sistemas geraram três arquiteturas que foram modeladas para a obtenção de fórmulas fechadas, elas são importantes para realização de análises. O modelo da arquitetura 1 foi validado e as outras arquiteturas variaram dessas. A arquitetura 3 teve a maior disponibilidade entre as outras arquiteturas, por possuir a quantidade maior de componentes replicados. O *downtime* (em horas) desta arquitetura comparada com a sem replicações foi em 36,89%. Por ela ter a maior disponibilidade, foi realizada uma análise de sensibilidade que mostrou o componente "Node" como o de maior impacto. No segundo sistema, foi apresentado um *VSaaS* de uma empresa, chamado empresarial, gerando 18 (dezoito) arquiteturas, uma delas comparada a arquitetura A1 (sem redundância), obteve uma redução significativa do *downtime* de 30% com um pequeno aumento no custo na ordem de 7%. Caso um determinado serviço exija um *downtime* menor, outra análise apontou uma arquitetura com redução de 80% ao aumentar 30% do custo. Diante desse panorama foram propostas e analisadas arquiteturas que podem auxiliar administradores a tomar importantes decisões na implementação de *VSaaS*.

Palavras-chave: Computação em Nuvem, Disponibilidade, VSaaS, Modelos Analíticos, RBD, SPN, CTMC.

Abstract

In the last few years, Video Surveillance as a Service VSaaS has shown the significant increase in demand for security mechanisms to ensure reliability higher levels. In parallel, the Cloud Computing paradigm has become an important tool for remote computing services. VSaaS, for example, allows for storage large amounts of data. In 2012, 50% of big data storage were surveillance video and in general, videos have a high significance for their owners, not allowing long periods interruption. To avoid video services with low performance and increase the quality, mechanisms to ensure high availability in VSaaS are required. However, this task is difficult without generating a major impact on cost, so this paper proposes two VSaaS systems who underwent an availability analysis, using analytical models (RBD, CTMC, and SPN). The first system, entitled domestic, was characterized by essential elements of a basic structure VSaaS, for use in homes and small businesses. This system generated three architectures that were modeled to obtain closed formulas; they are important to performing analyzes. The model architecture one was validated, and other architectures vary these. The architecture three had the highest availability of the other architectures, by owning the largest number of replicated components. The downtime (in hours) this architecture compared to a without replication was 36.89%. For having the highest availability, a sensitivity analysis showed the "Node"component as the most relevant. In the second system, was showed a VSaaS in a company and has generated eighteen architectures. One of them compared to a baseline, we obtained a significant reduction in downtime (30%) and a small increase in cost (on the order of 7%). In case, of the service requires less downtime, another analysis pointed an architecture with a reduction 80% of downtime and increased 30% in the cost. We propose and analyze architectures that can help administrators make important decisions in the VSaaS implementation.

Keywords: Cloud Computing, Availability Disponibilidade, VSaaS, Analytical Models, RBD, SPN, CTMC.

Lista de Figuras

Lista de Tabelas

Lista de Acrônimos

Sumário

1 Introdução

Se soubéssemos o que era que estávamos fazendo, não seria chamado pesquisa, seria?

—ALBERT EINSTEIN

Este Capítulo apresenta uma explanação acerca dos sistemas de vigilância em nuvem no contexto financeiro, além da importância da avaliação de disponibilidade desse serviço através de modelos matemáticos. Em seguida, descreve-se o foco da pesquisa na apresentação da motivação, justificativa, objetivos da pesquisa, trabalhos relacionados e a estrutura desta Dissertação.

1.1 Contexto

Os vídeos de vigilância, estão contribuindo para crescimento de *big datas*, apresentando uma fatia importante no universo digital. Esse fenômeno pode ser percebido pela presença de câmeras em elevadores, paredes de edifício e vias de trânsito, tanto para fiscalização quanto para proteção familiar. Outras áreas como laboratórios de pesquisa, escolas, empresas privadas, área médica, pesquisa e monitoramento de fauna e flora, monitoramento de relevo, condições climáticas e até mesmo controle de linhas de produção de fábricas.

A área de pesquisa em sistemas de vigilância na nuvem ainda é emergente (CHEN et al., 2014). Em sistemas de vigilância tradicionais, uma variedade de recursos relacionados à infraestrutura é necessária para realizar as operações de vigilância. Nos sistemas de vigilância baseados em nuvem a infraestrutura é provida pelo fornecedor da nuvem e geralmente cobrado pela quantidade de câmeras que estão monitorando.

O processo de modelagem consiste na arte de transformar situações da realidade em problemas matemáticos e resolvê-los (BASSANEZI, 2002). A modelagem usada para analisar

um serviço possibilita os benefícios aumentando a flexibilidade na alternância de parâmetros ou substituição de componentes, redução do tempo de implementação e custos quando comparados ao desenvolvimento e construção de sistemas ou protótipos reais. Além do risco de uma implementação de um serviço ou protótipo real podem não apresentar os resultados esperados, desperdiçando tempo e recursos financeiros da empresa (CHEN et al., 2014; CARVALHO, 2003; PRADO, 2000).

A computação em nuvem surgiu como uma opção viável para melhorar o desempenho e a confiabilidade das aplicações, tais como sistemas de vigilância em nuvem, *video on-demand* e armazenamento de dados. A fim de alcançar bons indicadores de qualidade de serviço (Qualidade de Serviço (QoS)) e cumprimento de acordos de níveis de serviços, do inglês *Service Level Agreement* (SLA), o comportamento desses sistemas precisam ser cuidadosamente caracterizados e compreendidos (CALHEIROS et al., 2011; CHEN et al., 2014).

Os sistemas de computação em nuvem geralmente são projetados para estarem disponíveis a todo o momento. Um exemplo é o que acontece com os sistemas de vigilância, onde a confiança de que o sistema estará disponível é importante. Há muitos desafios que devem ser superados para se chegar aos níveis de confiabilidade desejados (SUN et al., 2010). Os sistemas de vídeos de vigilância ainda são frequentemente encontrados no ambiente tradicional de circuito fechado de televisão (*Closed-circuit television* (CCTV)), vinculado aos servidores dedicados e mídias de armazenamento. Este tipo de serviço pode aproveitar os benefícios da nuvem e melhorar o seu fornecimento e aumentar sua confiabilidade.

Considere no decurso de um assalto onde o servidor seja roubado. Consequentemente, a prova do crime estará perdida. Esse tipo de ocorrência poderia ser evitada caso os vídeos estivessem armazenados em uma nuvem. A facilidade de transmissão de vídeo de câmeras de vigilância através da web é uma vantagem do sistema em nuvem. Neste caso, temos a situação que uma pessoa em viagem poderá monitorar e verificar a segurança de sua residência por meio de um computador ou dispositivo móvel com acesso à internet.

Diante do que foi exposto, este trabalho tem por finalidade analisar o comportamento da disponibilidade do *VSaaS*, um sistema para uso doméstico e outro para o setor empresarial. Esta pesquisa restringe sua análise na captura do vídeo no ambiente e na disponibilização do serviço na nuvem. Em sua metodologia o ponto de partida foi a definição das arquiteturas *VSaaS* a serem utilizadas no processo de modelagem hierárquica heterogênea, através de componentes sequenciais e paralelos representados em Diagrama de Blocos de Confiabilidade (*Reliability Block Diagrams* (RBD)) para comportamentos simples (KUO; ZUO, 2003; MACIEL et al., 2012), enquanto comportamentos mais detalhados e com dependências foram modelados com Cadeia de Markov de Tempo Contínuo (*Continuous-Time Markov Chain* (CTMC)) (TRIVEDI,

2002) e Redes de Petri Estocásticas (*Stochastic Petri Net* (SPN)).

Um dos modelos do sistema doméstico foi submetido a um processo de validação por meio de um experimento de injeção de falhas, e para obter o intervalo de confiança da disponibilidade foi utilizado o método proposto por (KEESEE, 1965). Nesta sistema, realizou-se uma análise de sensibilidade paramétrica na arquitetura três, que possui redundância *warm-standby* em alguns componentes, para identificar os pontos críticos da disponibilidade que podem ser melhorados.

O sistema empresarial, baseado em empresas de pequeno porte, iniciou-se com a definição de uma estratégia de disponibilidade para as câmeras. Na sequência, foi realizada uma análise de importância (*Availability Importance*) para apontar os componentes que têm maior contribuição para a disponibilidade do serviço e apoiou a proposição de novas arquiteturas. Por fim, uma análise entre custo e *downtime* foi aplicado nas 18 arquiteturas, sem e com redundâncias. Nesta análise, a alternância de prioridade foi considerada. Sendo utilizada para contribuir na decisão dos gestores de tecnologia da informação no momento de aumentar a disponibilidade do *VSaaS*.

1.2 Motivação e Justificativa

Vigilância por vídeo baseado em nuvem é um novo serviço de computação em nuvem que evoluiu como tema de investigação emergente (XIONG et al., 2014). Este novo modelo de serviço chamado *VSaaS* foi recentemente introduzido como uma alternativa para o desenvolvimento e gestão de sistemas de vigilância. O modelo permite uma maior escalabilidade de recursos a um baixo custo e pela concentração dos recursos computacionais presentes na nuvem poderão atender a mais de um cliente. Desta forma, evita que um cliente invista em uma arquitetura com 100 câmeras e a necessidade real seja de 40.

Para entender a grandeza desse tipo de serviço, uma pesquisa da *International Data Corporation* (IDC) (IDC, 2013), em 2012, revela que 50% de todos os materiais importantes a serem analisados no mundo digital foram de vídeos de vigilância e para 2015 espera-se alcançar o percentual de 65%. No Brasil, segundo a *Security Industry Association – Brasil* (SAI) (SIA, 2014), no ano de 2011, este mercado de vídeos de vigilância teve um volume de negócios de R$ 1,2 bilhões, e até 2017 espera-se um crescimento de 20,6%, atingindo R$ 3,7 bilhões. É um mercado que tende a crescer não só no Brasil, mas em todo o mundo, pela importância e necessidade de dados armazenados.

Uma pesquisa mundial realizada pela *MarketsandMarkets* (MARKETSANDMARKETS, 2012), sobre a expectativa do mercado de *VSaaS*, mostra que é esperado um crescimento de $474,00 milhões em 2011 para $2.390,90 milhões em 2017, com uma Taxa Composta de

Crescimento Anual (*Compound Annual Gowth Rate* (CAGR)) de 31,5% de 2012 até 2017. Um mercado com boas oportunidades financeiras, considerando essa taxa comparada com PIB Mundial esperado pelo FMI[1] em 2015 de 3,5%.

Como são apresentadas na literatura, as pesquisas existentes preveem um potencial significativo de *Video Surveillance as a Service* (VSaaS). No entanto, algumas questões como o custo (NEAL; RAHMAN, 2012), privacidade (PEARSON, 2009) e segurança (SABAHI, 2011) fazem algumas organizações ficarem em dúvida por optar ou não pelas soluções baseadas em nuvem (SABAHI, 2011).

Saranya e Vijayalakshmi (SARANYA; VIJAYALAKSHMI, 2011) afirmam a existência de benefícios que a infraestrutura de computação em nuvem pode oferecer para *VSaaS*: melhorar a disponibilidade, a centralização do armazenamento, memória, processamento e requisitos de banda larga; a redução de *hardware* e *software*, bem como os custos com licenças, a prestação de serviços de armazenamento de dados e de rede rápida e segura. Isto mostra que além de ser uma condição viável financeiramente, há ganhos de segurança e processamento.

1.3 Objetivos

Esta Dissertação tem por objetivo geral a concepção de modelos para avaliação da disponibilidade de Video Surveillance as a Service, por meio da combinação de modelos hierárquicos, utilização de redundância, análise de sensibilidade e validação de modelo para as arquiteturas propostas. Para atingirmos o objetivo geral, é necessário cumprir os seguintes objetivos secundários:

- Propor uma metodologia de avaliação de disponibilidade para o *VSaaS*;
- Desenvolver scripts para a injeção de falhas e monitoração dos componentes do sistema;
- Analisar a sensibilidade dos parâmetros do sistema para encontrar componentes críticos.

1.4 Trabalhos Relacionados

Avaliaçãao de disponibilidade em ambientes *VSaaS* tem sido tema de pesquisas que aborda o ambiente em diversas perspectivas, no entanto, foram encontrados poucos trabalhos que abordem a utilização de modelagem analítica para métricas de dependabilidade. Foram realizadas pesquisas nas *engines IEEEXplore*, *ACM*, *Scopus*, *ScienceDirect*, *SpringerLink* e, em trabalhos publicados no período de 2008 a 2015.

[1] Fundo Monetário Nacional

Os trabalhos que norteiam esta pesquisa foram divididos nas teorias usadas e nos trabalhos de *VSaaS*. Das teorias, encontram-se a modelagem analítica usada para analisar métricas de dependabilidade, análise de sensibilidade e análise de custo. Dos trabalhos de *VSaaS* encontraram-se implementações, arquiteturas e conceitos de dependabilidade em *VSaaS*. Estes trabalhos são explanados a seguir com suas respectivas contribuições a esta Dissertação.

Em XIONG et al. (2014) propuseram um projeto e método de implementação para sistemas de vídeo de vigilância baseado em nuvem. No protótipo, permite ao usuário posicionar uma *webcam* em determinado local que requer vigilância e realizar o monitoramento por um navegador. Dentre as vantagens apresentadas pelo trabalho, cita-se: utilização da computação paralela para tratar o grande ganho de espaço de armazenamento e fácil expansibilidade, o pouco investimento e baixo custo com manutenção, a alta portabilidade e segurança de dados superiores e, o bom compartilhamento. Isso mostra que a computação em nuvem, neste contexto, é capaz de resolver muitos problemas associados aos sistemas convencionais, tais como: alto custo de manutenção de dispositivos, insegurança na camada de dados, mau desempenho e baixa confiabilidade.

A pesquisa de KARIMAA (2011) estudou os conceitos de dependabilidade para a expansão da tecnologia de vigilância de vídeo através da infraestrutura de nuvem. Este conceitos foram a disponibilidade, segurança, confiabilidade e manutenção das soluções, além de apresentar as potencialidades identificadas nesta tecnologia. No entanto, não houve a proposição ou análise de modelos matemáticas neste contexto.

No trabalho de SUVONVORN (2008) foi implementado sob o *framework .NET* um sistema flexível para análise de vídeo em tempo real, com aspectos aplicáveis no domínio de vigilância, tais como, aquisição, análise, armazenamento, alerta e componentes de reprodução. Uma análise de performance nas métricas de detecção de movimento e de face foi realizada.

O artigo de SONG; TIAN; ZHOU (2014) projetou uma solução de um *display* remoto que permite aos usuários de vigilância assistam aos vídeos em tempo real e compartilhem as atualizações de tela entre os usuários de um *desktop* remoto. Foram adotados vários codificadores e método de codificação paralelo em *display* remoto para atender a qualidade de requisito do serviço em situações variadas. O sistema trata com carga de trabalho dinâmica melhor do que os métodos tradicionais de displays remotos. As tarefas de vigilância e de codificação foram gerenciados separadamente. Uma arquitetura de *VSaaS* é apresentada e dois modelos de filas foram projetados para lidar com o problema de provisionamento de recursos para diferentes codificadores. As métricas de desempenho analisadas foram consumos de largura de banda e da Unidade Central de Processamento do cliente.

A Dissertação de MELO (2014) propôs um conjunto de modelos matemáticos para

Tabela 1.1: Relação entre a proposta desta dissertação e outros trabalhos relacionados

	Principais contribuições da dissertação							
	Contexto	Arquiteturas VSaaS	Validação	Uso de Modelos	Custos	Análise de Sensibilidade	Métrica de Dependabilidade	VSaaS
Esta dissertação	Computação em Nuvem	✓	✓	✓	✓	✓	✓	✓
(XIONG et al., 2014)	Computação em Nuvem	✓						✓
(KARIMAA, 2011)	Computação em Nuvem						✓	✓
(MELO, 2014)	Nuvem Privada		✓	✓		✓	✓	
(DANTAS, 2012)	Nuvem Privada			✓	✓		✓	
(SUVONVORN, 2008)	Sistema de vigilância	✓						✓
(SONG; TIAN; ZHOU, 2014)	Nuvem Privada	✓			✓			✓

avaliação de disponibilidade na plataforma de computação em nuvem privada OpenNebula. Enfatizou-se o rejuvenescimento de software através de *Live Migration* de máquinas virtuais. Para aumentar a confiabilidade do modelo proposto, um dos modelos, foi submetido ao processo de validação por meio de experimentos de injeção de falhas. Essa pesquisa adotou a abordagem de validação igual.

Na Dissertação, DANTAS (2012) concebeu um conjunto de modelos analíticos para avaliação de dependabilidade das infraestruturas de computação em nuvem e utilizou a plataforma Eucalyptus. No segundo estudo de caso, as disponibilidades e os custos foram comparados entre as arquiteturas possíveis na construção redundante da nuvem Eucalyptus. Das arquiteturas foram calculadas as distâncias euclidianas até o ponto de origem (*0*), com a finalidade de apontar a arquitetura considerada boa, de acordo com estas métricas. No presente trabalho foi discutido o *trade-off* entre custo e *downtime* entre as arquiteturas propostas no sistema empresarial. Havendo um diferencial em relação a esta análise, que são resultados baseados em prioridades.

Na Tabela 1.1 é apresentado um estudo comparativo entre pontos de trabalhos relacionados e a pesquisa. Procurou-se com esses pontos saber qual o contexto do trabalho e se houve proposição de arquiteturas, utilização de técnicas de modelagem, validação de modelos, análises de sensibilidade, métricas de dependabilidade e por fim, se aborda *VSaaS*.

1.5 Estrutura da Dissertação

Este Capítulo contextualizou o tema da pesquisa ao tratar da relevância em avaliar a disponibilidade através de modelos matemáticos e o aspecto financeiro favorável. Em seguida, apresentou-se a motivação, justificativa e objetivos. Trabalhos relacionados a esse estudo foram selecionados e comentados as suas contribuições.

O Capítulo 2 apresenta o referencial teórico, abordando conceitos acerca da Computação em Nuvem, Dependabilidade, Intervalo de Confiança da Disponibilidade, Análise de Sensibilidade e *Video Surveillance as a Service*. Primeiro são introduzidos os conceitos básicos de computação em nuvem, seus modelos de serviços, benefícios e desvantagens deste paradigma. Em seguida é iniciada uma explicação sobre dependabilidade, em especial sobre disponibilidade, tema que é aplicado neste trabalho. Além de elucidar os tipos de modelos que podem ser utilizados para encontrar métricas de dependabilidade, esta Dissertação faz uso de RBD, CTMC e SPN. Para o entendimento sobre o processo validação é detalhada a obtenção dos intervalos de confiança de disponibilidade pelo método de Keesee. Por fim, são explicadas técnicas de análise de sensibilidade e conceitos de *Video Surveillance as a Service*.

O Capítulo 3, é composto por duas importantes seções: a metodologia e as arquiteturas propostas. Na Seção 3.1, é apresentada uma metodologia para auxiliar a avaliação de disponibilidade de um sistema *VSaaS*. As arquiteturas de *VSaaS* usadas nos estudos de casos são encontradas nesta Seção 3.2, com as respectivas restrições e funcionalidades dos componentes integrantes de uma plataforma *VSaaS*.

No Capítulo 4, são justificados os modelos concebidos em RBDs, CTMCs e SPNs, gerados com base nas arquiteturas utilizadas. Além disso, a validação do modelo da arquitetura básica é apresentada nesse Capítulo.

Os resultados estão contidos no Capítulo 5, através dos estudos de casos que avaliam a métrica de disponibilidade nas arquiteturas propostas. Também é apresentada uma análise de sensibilidade e *availability importance*. Na conclusão dos resultados é realizado um comparativo entre custo e disponibilidade nas arquiteturas do sistema empresarial.

Por fim, no Capítulo 6 são apresentadas as conclusões desta pesquisa e as principais contribuições e sugestões para possíveis trabalhos futuros.

2

Referencial Teórico

As oportunidades multiplicam-se à medida que são agarradas.

—SUN TZU (A Arte da Guerra)

Há conhecimentos básicos para esta Dissertação ser compreendida. Ressaltam-se neste Capítulo, os subsídios necessários para o entendimento das técnicas aplicadas ao longo deste trabalho e os conceitos podem ser melhor entendidos a partir das referências apontadas. Este Capítulo aborda os temas da seguinte forma: incialmente os conceitos básicos de computação em nuvem, bem como, suas vantagens e desvantagens. Em seguida, os conceitos sobre dependabilidade são explicados conforme a métrica de disponibilidade. Posteriormente, são apresentadas as técnicas de modelagem para a disponibilidade. Para a validação do modelo foi necessário entender o obtenção do intervalo de confiança da disponibilidade. Por fim, a análise de sensibilidade foi fundamentada e conceitos como trajetória, mercado e vantagens do *VSaaS* são abordados.

2.1 Computação em Nuvem

Computação em nuvem é um paradigma em contínuo desenvolvimento que foi originado da combinação de diferentes tecnologias (SOUSA et al., 2012). Os recursos como: poder de processamento, rede, armazenamento e softwares são ofertados e acessados remotamente pela Internet. Algumas definições de computação em nuvem são encontradas na literatura, por exemplo: um sistema paralelo e distribuído consistido de uma coleção de computadores interligados e virtualizados que são provisionados dinamicamente e apresentados como um ou mais recursos de computação unificados. São baseados em acordo de nível de serviço estabelecido através da negociação entre o fornecedor e o consumidor (BUYYA; YEO; VENUGOPAL, 2008).

Em BAKSHI (2009), a computação em nuvem é definida como recursos de TI e serviços que são abstraídos da infraestrutura subjacente e são fornecidos *"on-demand"* e "em escala" num ambiente de vários utilizadores. A computação em nuvem é um modelo que permite o acesso à rede sob demanda para um conjunto de recursos computacionais configuráveis partilhados (por exemplo, redes, servidores, armazenamento, aplicativos e serviços) que podem ser rapidamente provisionados e liberados, com o mínimo esforço de gestão ou de interação com quem fornece os serviços (MELL; GRANCE, 2014). A computação em nuvem é composta por cinco principais características (MELL; GRANCE, 2014):

- *On-demand self-service* (autoatendimento sob demanda) - um consumidor pode, unilateralmente, abastecer-se dos recursos computacionais, tais como tempo de servidor e armazenamento de rede, conforme necessidade, de maneira automática e sem requerer interação humana com o provedor de serviço;

- *Broad network access* (amplo acesso à rede) - os recursos estão disponíveis sobre a rede, sendo acessado por mecanismos padrões que promovem o uso por plataformas de clientes finos e grossos, heterogêneos (exemplo: *smartphones*, *tablets*, *laptops* e estações de trabalho);

- *Resource pooling* (geração de pools de recursos) - os recursos de computação (armazenamento, processamento, memória e largura de banda) são reunidos para servir a múltiplos consumidores usando um modelo multi-inquilino (*multi-tenant*), com recursos físicos e virtuais diferentes atribuídos dinamicamente e realocados de acordo com a demanda do cliente. O modelo multi-inquilino é uma arquitetura essencial para um ambiente em nuvem, pois permite que múltiplos inquilinos (empresas/clientes) compartilhem os mesmos recursos físicos, o que acontece com um sistema integrado de gestão ao permanecer os módulos relevantes, logicamente isolados;

- *Rapid elasticity* (elasticidade rápida) - recursos podem ser provisionados e liberados elasticamente, em alguns casos automaticamente, escalando rapidamente para dentro e para fora, de acordo com a demanda. Para o usuário, os recursos disponíveis para fornecimento parecem ilimitados e podem ser utilizados em qualquer quantidade e a qualquer momento;

- *Measured service* (serviço medido) - sistemas de nuvem podem monitorar, controlar e informar o uso de recursos pela capacidade de medição em níveis de abstração adequada para o tipo de serviço. Oferecendo transparência tanto para o fornecedor quanto para o consumidor dos serviços utilizados.

O princípio básico de computação em nuvem é atribuir computação a grande número de computadores distribuídos, em vez de computadores locais ou serviços remotos. Caracterizada

pela utilização eficiente dos recursos, empregando virtualização, monitoramento de recursos e mecanismos de balanceamento de carga (SARANYA; VIJAYALAKSHMI, 2011).

Há três principais modelos de negócio de Computação em Nuvem, pela *National Institute of Standards and Technology* (NIST) (MELL; GRANCE, 2014). São eles: Software como Serviço (SaaS), Plataforma como Serviço (PaaS) e Infraestrutura como Serviço (IaaS), conforme a Figura 2.1.

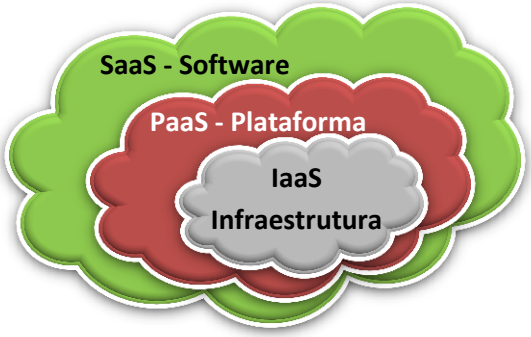


Figura 2.1: Modelos de negócio de Computação em Nuvem

- **IaaS -** através do uso da virtualização, este modelo de negócio particiona os recursos de um *datacenter* entre os usuários sobre a forma de recursos virtualizados. Neste modelo de negócio, geralmente o usuário paga pelo serviço de acordo com a capacidade da máquina virtual (*Virtual Machine - VM*) alocada e pela quantidade de tempo que a VM permanece em execução. O usuário não gerencia ou controla a infraestrutura de nuvem subjacente, mas é responsável por controlar sua própria pilha de software na qual os seus serviços irão executar, o sistema operacional e o armazenamento.

- **PaaS -** neste modelo o provedor de serviço abstrai a VM e o sistema operacional, oferecendo ao usuário uma plataforma de software de alto nível. Este modelo tem como vantagem o dimensionamento transparente dos recursos de hardware durante a execução dos serviços (VAQUERO et al., 2008). Com PaaS os usuários podem desenvolver, compilar, *debugar* e ainda realizar testes na nuvem.

- **SaaS -** este modelo é voltado para usuários comuns de Internet, em vez de desenvolvedores e administradores de sistema. Os softwares e os dados gerados por eles ficam armazenados na nuvem e podem ser acessados a partir de qualquer computador conectado à Internet. Atrelado a esta comodidade de acesso aos dados e ao software está à vantagem financeira que o cliente pagará um valor relativamente menor do que o valor de compra do software.

Nas sociedades modernas, a maior parte dos serviços essenciais são disponibilizados de uma forma transparente. O abastecimento de água, eletricidade, gás e telefone, bens imprescindíveis no nosso dia a dia, têm esta caraterística. A sua exploração é feita com o modelo de pagamento baseado no uso, PAYG (GOMES, 2012). Esses modelos de negócios seguem o conceito de "pague pelo que usar". O valor pago pelo serviço é flexível de acordo com a necessidade de utilização da organização em qualquer tempo.

Os recursos computacionais são mantidos e gerenciados na nuvem por provedores. Entre suas particularidades no fornecimento dos serviços aos seus clientes, alguns focam na redução de custo da operação e outros podem ter interesse em entregar um serviço com maior confiabilidade e segurança de dados (ZHANG; CHENG; BOUTABA, 2010). Dessa forma, três tipos de modelos de implantação de Nuvem são considerados e apresentados, a seguir (MELL; GRANCE, 2014):

Nuvem Privada: a infraestrutura da nuvem é fornecida para uso exclusivo de uma organização que possui múltiplos clientes. Por exemplo, um centro de dados de uma empresa sediada numa nuvem;

Nuvem Pública: a infraestrutura é fornecida para uso aberto ao público geral que permanecem entidades únicas, mas serão unidas por uma padronização ou uma tecnologia proprietária que possibilitará a portabilidade dos dados e das aplicações. Os recursos são disponibilizados dinamicamente e numa perspectiva *self-service* a partir da Internet;

Nuvem Híbrida: é caracterizada quando a nuvem é composta de duas ou mais infraestruturas distintas (privada, comunitária ou pública), que continuarão como entidades únicas. A vantagem deste modelo é a economia de custos e o acesso a uma infraestrutura escalável com mecanismos de segurança. Pela agregação das vantagens dos dois outros modelos, privado e público. Esta metodologia é uma opção interessante para as empresas. Porém, as nuvens híbridas introduzem uma complexidade adicional que se refere à distribuição de aplicações pelos dois modelos (SUBRAMANIAN, 2011).

A computação em nuvem é importante para a distribuição e acesso de recursos de computação, oferecendo algumas vantagens de recursos computacionais (ARAUJO et al., 2011). Na Tabela 2.1, é possível verificar através de uma análise de SWOT do modelo computacional de computação em nuvem, realizada por (HERBERT; ERICKSON, 2011) e (MARSTON et al., 2011), em que elencou os pontos fortes e fracos, além das oportunidades e ameaças, considerando sua evolução, integração e as perspectivas de mercado.

Sucintamente, dos benefícios em utilizar os serviços da nuvem destacam-se: gestão centralizada, redução de consumos energéticos e diminuição dos custos de manutenção das infraestruturas tradicionais. A nuvem disponibiliza uma diversidade de serviços, onde a rapidez de acesso a recursos virtuais favorece a agilidade do negócio (MILLER, 2008; TERRY, 2011).

Tabela 2.1: Análise de SWOT do paradigma computação em nuvem, (HERBERT; ERICKSON, 2011) e (MARSTON et al., 2011)

Pontos fortes	**Pontos fracos**
- Serviço escalável em curto prazo; - Redução dos custos iniciais de gestão de energia e de servidores subutilizados; - Configurações predefinidas de servidores e de máquinas virtuais com controle dos recursos adquiridos; - O acesso e a gestão do sistema organizacional, a partir de API, são disponibilizados pelos fornecedores, através de dispositivos com inferiores capacidades computacionais.	- Os consumidores podem perder o controle dos dados alojados na nuvem; - Os fornecedores não garantem total qualidade e disponibilidade de serviço, durante todo o ano, o que pode ser insuficiente para as empresas; - Possibilidade de falha nos serviços disponibilizado pelos fornecedores da nuvem.
Oportunidades	**Ameaças**
- Sem necessidade dos tradicionais gastos iniciais, as empresas têm facilitada à criação e a expansão de negócios; - Criação de *mashups*, páginas Web que combinam diversos serviços externos, originando novos serviços e novas oportunidades de negócio; - Uso mais racional da energia e dos recursos naturais.	- Possibilidade de falência dos fornecedores da nuvem; - Preocupação com a segurança, a confiabilidade e o desempenho dos serviços da nuvem; - Regulamentação variável e sujeita a eventuais alterações, conforme as decisões nacionais, no que respeita à privacidade, aos requisitos de auditoria e de localização de dados.

2.2 Dependabilidade

O desempenho e a dependabilidade são duas importantes características para a análise de sistemas. Usualmente, devem ser avaliadas separadamente, considerando que a primeira assume que o sistema e seus componentes irão falhar, e que a segunda baseia-se nas análises da falha e do reparo e na estrutura do sistema (DAS, 1998). O termo dependabilidade é uma tradução literal do termo inglês *dependability*, que indica a qualidade do serviço fornecido por um dado sistema e a confiança depositada no serviço fornecido.

Muitas características esperadas em sistemas distribuídos, incluídas nos ambientes em nuvem, são relatadas com a concepção de dependabilidade. Não há uma simples definição

de dependabilidade, mas pode ser entendida como a capacidade de entregar uma determinada funcionalidade específica que pode ser justificadamente confiável (AVIZIENIS et al., 2004; SALEH; MAHMOUD; ABDEL-SALAM, 2013).

No desenvolvimento de um sistema com os atributos de dependabilidade desejados, um conjunto de métodos e técnicas devem ser empregadas. Esses métodos e técnicas são classificados e citados a seguir (WEBER, 2003):

- **Prevenção de falhas** - impede a ocorrência ou introdução de falhas. Envolve a seleção de metodologias de projeto e de tecnologias adequadas para os seus componentes;
- **Tolerância a falhas** - fornece o serviço esperado mesmo na presença de falhas. Técnicas comuns: mascaramento de falhas, detecção de falhas, localização, confinamento, recuperação, reconfiguração, tratamento;
- **Validação** - remoção de falhas e verificação da presença de falhas;
- **Previsão de falhas** - estimativas sobre presença de falhas e estimativas sobre consequências de falhas.

Com o passar das três últimas décadas, dependabilidade é um conceito que integra os seguintes atributos (AVIŽIENIS et al., 2004): **Disponibilidade** - prontidão de um sistema para entregar o serviço correto para um usuário. **Confiabilidade** - continuidade com o serviço correto. **Segurança** - ausência de consequências catastróficas sobre o usuário e o ambiente. **Integridade** - ausência de alterações impróprias no sistemas. **Manutenabilidade** - capacidade de sofrer modificações e reparos.

Este estudo é focado no conceito de disponibilidade, que está relacionado com confiabilidade. Confiabilidade é a probabilidade que o sistema S não irá falhar em um tempo t. Um dos atributos mais importantes para sistemas computacionais é a disponibilidade. Disponibilidade pode ser expressa como a relação da expectativa do sistema ativo em um total de tempo observado (AVIZIENIS et al., 2001) (AVIZIENIS et al., 2004), como apresentado na Equação 2.1.

$$Disponibilidade = \frac{TempoAtivo}{TempoAtivo + TempoInativo} \tag{2.1}$$

A disponibilidade em sistemas *online* (ex. sistemas bancários, sites da Internet) é um dos pontos cruciais que garantem a qualidade do serviço, uma vez que as solicitações geradas a esses serviços são frequentes e necessitam de respostas imediatas. Além disso, é importante estimar a disponibilidade do sistema porque paralisações e interrupções inesperadas na entrega dos serviços acarretam prejuízos financeiros e podem comprometer a reputação de seus provedores (HAGEN; SEIBOLD; KEMPER, 2012).

No processo de avaliação da disponibilidade de sistema por intermédio de medições significa uma operação custosa. As medidas mais importantes na avaliação são Tempo Médio para Falha (*Mean Time To Failure* (MTTF)) e para Reparo (*Mean Time To Repair* (MTTR)) (MACIEL et al., 2012).

O MTTF representa o tempo médio para falhar. É uma métrica que corresponde ao tempo estimado para acontecer uma falha. O MTTF pode ser calculado através da Equação 2.2.

$$\int_0^\infty R(t) \times dt \tag{2.2}$$

O MTTR, por sua vez, é o tempo médio para reparo. O MTTR é baseado num atributo conhecido como manutenabilidade. Entende-se como manutenabilidade, a probabilidade de um Sistema S ser reparado com um tempo t. Geralmente a manutenabilidade é denotada pela função $M(t)$ (MACIEL et al., 2011). A obtenção do MTTR pode ser realizada pela Equação 2.3.

$$\int_0^\infty M(t) \times dt \tag{2.3}$$

O MTTF e MTTR sendo exponencialmente distribuídos podem ser representados pelas variáveis λ e μ, respectivamente e são denotadas de taxas. Observe que λ pode ser calculado como o inverso do tempo médio de falha do sistema, enquanto que μ é o inverso do tempo médio de reparo. As taxas mostram a quantidade de ocorrências de falhas/reparos em um determinado intervalo de tempo. Dessa forma, pode-se encontrar a disponibilidade através da Equação 2.4:

$$Disponibilidade = \frac{\mu}{\mu + \lambda}, \tag{2.4}$$

Com a obtenção de disponibilidade, duas novas fórmulas podem ser expressadas, formando métricas de dependabilidade. A primeira delas, a *indisponibilidade*, representa a probabilidade do sistema não estar disponível e é calculada pela Equação 2.2. A outra, o *downtime*, significa estimar o intervalo de tempo que o sistema ficou em interrupção. Por exemplo, para encontrar o *downtime* anual em horas, usaríamos o resultado da *Indisponibilidade* $\times$ quantidade anual de horas, conforme Equação 2.2.

$$Indisponibilidade = 1 - Disponibilidade$$

$$DowntimeAnual = Indisponibilidade \times 8760$$

A curva da banheira representa o comportamento da taxa de falhas de um equipamento ou sistema produtivo ao longo do tempo. Suas fases estão associadas ao fator de forma γ de uma

eventual distribuição de *Weibull* que descreva a confiabilidade do equipamento, conforme se observa na Figura 2.2 e descrito seus comportamentos a seguir:

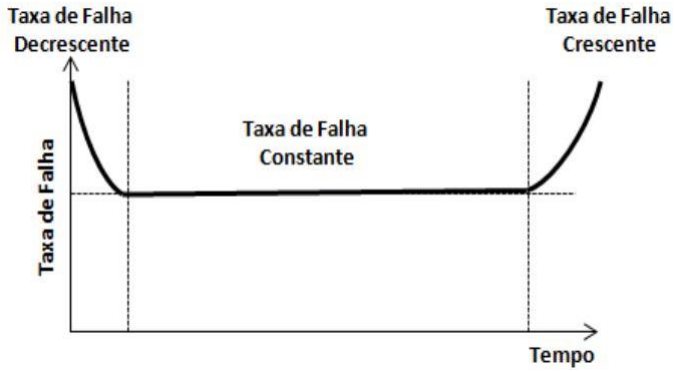


Figura 2.2: Curva da Banheira

a) Taxa de Falha Decrescente - acontece no início da vida de um componente, onde suas falhas começam elevadas, porém é decrescente. Na fabricação de hardwares ou máquinas essas falhas são justificadas pelos erros de fabricação, erros de projeto, peças defeituosas, processos de fabricação inadequados, mão de obra destreinada, entre outros. Essa taxa de falha diminui conforme são detectados e corrigidos os erros;

b) Taxa de Falha Constante - o sistema entra numa fase chamada de vida útil, haja visto que o valor médio da taxa de falha é constante. As falhas ocorrem por causas aleatórias, geralmente externas ao equipamento, na maioria das vezes são de difícil controle;

c) Taxa de Falha Crescente - nesta fase inicia-se o período final de vida e é caracterizada pelo desgaste do componente. Sua taxa torna-se crescente.

2.3 Técnicas de Modelagem para Disponibilidade

Vários tipos de modelos analíticos podem ser usados para avaliação de disponibilidade. Diagrama de Blocos de Confiabilidade (*Reliability Block Diagram* - RBD), (MACIEL et al., 2012), Árvore de Falhas (KUO; ZUO, 2003), Cadeias de Markov de Tempo Contínuo (*Continuous-time Markvov Chain* - CTMCs) (TRIVEDI, 2002) e Redes de Petri Estocásticas (*Stochastic Petri Net* - SPNs) (KARTSON et al., 1994; AJMONE MARSAN; CONTE; BALBO, 1984) têm sido técnicas de modelagem utilizadas para modelar sistemas tolerantes a falha e avaliar métricas de dependabilidade. Esses tipos de modelos divergem em dois aspectos: sua facilidade em ser usado em aplicações específicas e o respectivo poder de modelagem. Sendo assim, o sucesso pode ser alcançado através da composição de modelos distintos.

Os modelos RBDs e árvores de falhas representam as condições sob as quais um sistema pode se encontrar em falha, ou estar operacional, em termos de relações estruturais entre seus componentes, são classificados como modelos combinatórios. Modelos CTMC e SPN representam o comportamento do sistema (atividades de falha e reparo) por seus estados e ocorrência de eventos expressados em transições de estados marcados. Estes permitem restrições e são classificados como modelos baseados em estados (MACIEL et al., 2012).

A combinação de ambos os tipos de modelo também é possível, permitindo obter o melhor dos dois mundos, através da modelagem hierárquica (MALHOTRA; TRIVEDI, 1993; KIM; GHOSH; TRIVEDI, 2010). Diferentes modelos, do mesmo tipo ou de tipos distintos, podem ser combinados em diferentes níveis de compreensão levando a modelos hierárquicos maiores (MALHOTRA; TRIVEDI, 1993).

Modelos hierárquicos heterogêneos têm sido usados para lidar com a complexidade de vários tipos de sistemas, como por exemplo: redes de sensores (KIM; GHOSH; TRIVEDI, 2010), redes de telecomunicação (TRIVEDI et al., 2006) e nuvens privadas (DANTAS et al., 2012).

2.3.1 Diagrama de Bloco de Confiabilidade

Diagrama de Blocos de Confiabilidade é um tipo de modelo combinatorial proposto para analisar a confiabilidade de sistemas baseados nas relações de seus componentes. Entendido como um método utilizado para mostrar como a confiabilidade de um componente pode contribuir para o sucesso ou o fracasso de um sistema complexo. Por conseguinte, esse formalismo foi ampliado para a análise de disponibilidade e manutenibilidade. Nesta seção, atentaremos para o uso de RBDs para o cálculo de disponibilidade, uma vez que este é o foco deste trabalho.

Um RBD é formado pelos vértices de origem e de destino, componentes representados por blocos, e arcos conectando os blocos e os vértices. Sua estrutura é organizada com base essencial para o sistema funcionar, ou seja, o diagrama representa o modo operacional do ambiente que está sendo modelado (MELO et al., 2013). O modo operacional apresenta quais componentes do sistema devem estar em funcionamento para que o sistema deva responder corretamente.

Para o RBD está disponível é necessário que exista algum caminho do vértice de origem até o vértice de destino, no qual componentes indisponíveis representam uma interrupção em uma parte do caminho. Na Figura 2.3 está mostrado que o sistema continua disponível mesmo acontecendo a falha de um dos componentes, uma vez que há um caminho contínuo dos vértices de início e fim do RBD. Na Figura 2.4 a falha do componente é crítica para o sistema, porque ele impede que se repita o comportamento do caminho no RBD.

Em um modelo de diagrama de blocos de confiabilidade os componentes são represen-

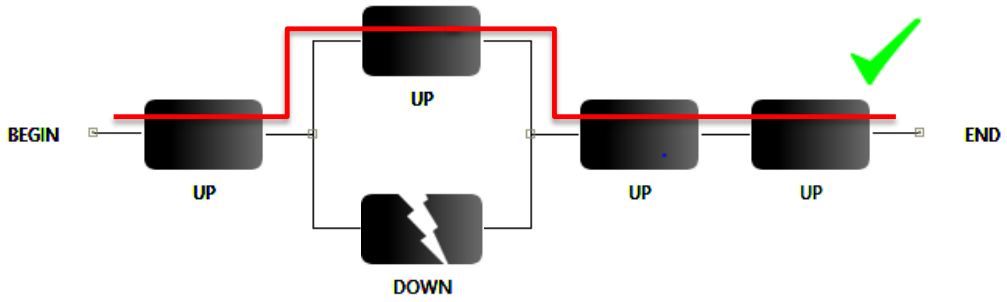


Figura 2.3: Sistema disponível.

tados com blocos combinados com outros blocos. Se os componentes forem ligados em série, Figura 2.5, todos os componentes deverão estar operacionais para que conjunto esteja disponível. Nos paralelos, Figura 2.6, é necessário que pelo menos um esteja funcionando. Por fim, outra estrutura típica é a *k-out-of-n*, na Figura 2.7, determina que k componentes devem estar em funcionamento de uma determinada quantidade n de componentes, ou ainda em combinações entre essas organizações (TRIVEDI et al., 1996; MACIEL et al., 2011).

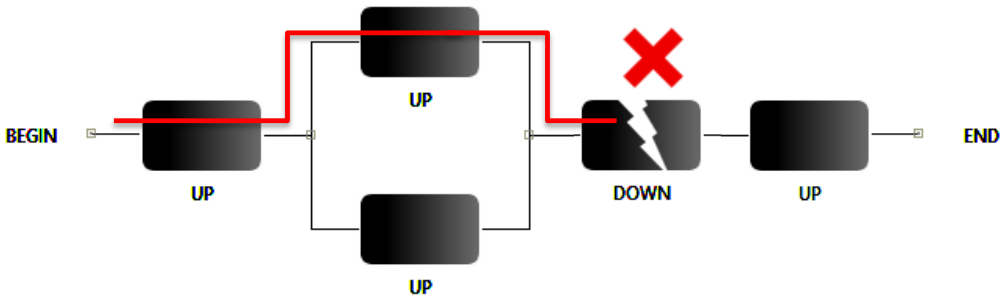


Figura 2.4: Sistema indisponível.

A confiabilidade de dois blocos conectados em série é obtida através da Equação 2.5:

$$R_S = R_1 \times R_2 \tag{2.5}$$

onde:
R_1 descreve a confiabilidade do bloco 1.
R_2 descreve a confiabilidade do bloco 2.

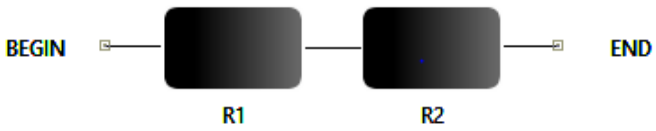


Figura 2.5: Componentes em série.

A confiabilidade de dois blocos conectados em paralelo é obtida através da Equação 2.6:

$$R_P = 1 - \prod_{i=1}^{2}(1 - R_i) \tag{2.6}$$

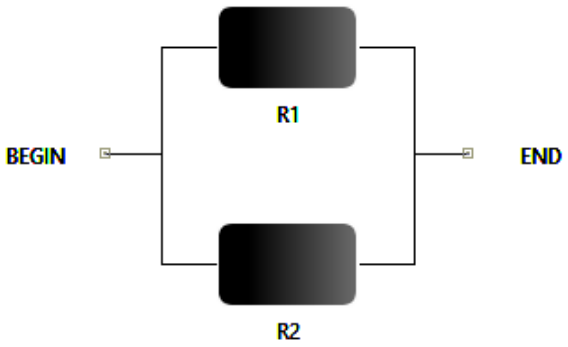


Figura 2.6: Componentes em paralelo.

A confiabilidade de blocos conectados em *k-out-of-n* é obtida pela Equação 2.7:

$$R_{k-out-of-n}(t) = \sum_{i-k}^{n} P(X = i) \tag{2.7}$$

Figura 2.7: Componente em *k-out-of-n*.

Modelos RBDs são utilizados, geralmente, em sistemas modulares que consistam de muitos módulos independentes, em que cada um pode ser facilmente representado por um bloco de confiabilidade. Assim, havendo a necessidade de modelar sistemas complexos, onde se exige a adição de redundância em módulos do sistema. O usuário deve recorrer as técnicas de modelagem hierárquica, que utiliza em conjunto modelos como CTMC e RBD ou RBD e SPN, na tentativa de obter resultados mais expressivos.

2.3.2 Cadeias de Markov

Cadeia de Markov (STEWART, 1994), (TRIVEDI et al., 2006) é um formalismo matemático para a modelagem de sistemas, modelando o funcionamento de um sistema através de um conjunto de estados e transições entre os estados, tanto para fins descritivos quanto preditivos

(MENASCE et al., 2004). Entendido ainda, como uma máquina de estados, em que os nós são estados e os arcos representam as transições entre os estados do modelo.

Na ciência da computação, em particular, este formalismo é bastante conveniente para descrever e analisar propriedades dinâmicas de sistemas computacionais (BOLCH et al., 2006). As transições são modeladas por um processo estocástico de tempo contínuo ou discreto, definido por distribuições exponenciais ou geométricas, respectivamente (SOUSA, 2009). Por isso, há duas classificações para modelos que seguem o formalismo de cadeia de Markov (STEWART, 1994; TRIVEDI et al., 2006): Cadeias de Markov à escala de Tempo Contínuo CTMC ou Cadeias de Markov à escala de Tempo Discreto (*Discrete-Time Markov Chain* (DTMC)). A diferença entre essas classificações é que, em modelos CTMC, suas transições entre estados podem ocorrer em qualquer instante de tempo e não em pontos discretos de tempo.

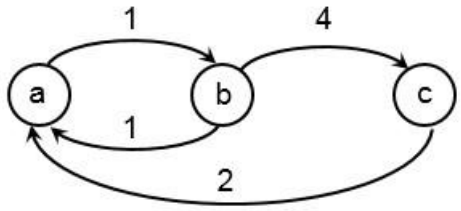


Figura 2.8: Exemplo de cadeia de Markov.

Sua representação gráfica é similar a dos autômatos, como um diagrama de estado, onde os vértices representam os estados e os arcos do diagrama representam as possíveis transições entre os estados. As transições entre estados representam a ocorrência de eventos (MACIEL et al., 2011). A Figura 2.8 apresenta um exemplo de CTMC com dois estados e duas transições. Neste modelo o estado 1 representa que o sistema estar funcionando. A partir deste a falha pode ocorrer (λ), levando ao estado 2. Do estado 2 apenas o reparo (μ) é possível.

Cadeias de Markov também são representadas em forma de matriz, a chamada **matriz de taxa de transição Q**. Na matriz Q estão contidas as informações sobre as transições dos estados na cadeia de Markov e utilizada para a resolução de cadeias de Markov. Cada elemento localizado fora da diagonal principal representa a taxa de ocorrência dos eventos que efetivam a transição dos estados do sistema. Os elementos contidos na diagonal principal são os valores necessários para que a soma dos elementos de cada linha seja igual a zero. Por exemplo, utilizando a CTMC da Figura 2.8, teremos a matriz da Equação 2.8.

$$Q = \begin{pmatrix} -\lambda & \lambda \\ \mu & -\mu \end{pmatrix} \tag{2.8}$$

As probabilidades de transição dos estados podem ser calculadas através da Equação 2.9.

$$p_{i,j}(s,t) = PX(t) = j|X(s) = i \tag{2.9}$$

A análise estacionária de uma cadeia de Markov consiste em encontrar a probabilidade de o sistema alcançar determinado estado durante um longo tempo de execução. Estas probabilidades são independentes do estado inicial do sistema e são representadas pelo vetor $\pi = [\pi_1, \pi_2, ..., \pi_n]$, onde π é a probabilidade estacionária para o estado i. Uma vez que a cadeia de Markov considerada, é uma cadeia ergódica, podemos encontrar as probabilidades estacionárias através do sistema linear formado pelas Equações 2.10 e 2.11 (BOLCH et al., 2006). Se em uma cadeia de Markov é possível alcançar qualquer estado a partir de qualquer estado, em um número n finito de passos, esta cadeia é dita ergódica.

$$\pi Q = 0, \tag{2.10}$$

$$\sum_{i=1}^{n} \pi_i = 1 \tag{2.11}$$

No qual (Equação 2.10), Q é a matriz de estados e π (vetor de probabilidade) é o autovetor correspondente ao autovalor unitário da matriz de transição. Ressalta-se que a soma dos elementos do vetor de probabilidade π deve ser igual 1, ou seja $||\pi|| = 1$ (ARAúJO, 2009). A Equação 2.11 é a condição de normalização, adicionada para assegurar que a solução obtida é um único vetor de probabilidade. A Equação 2.10 tem um conjunto de soluções infinitas. Normalizando as soluções, chega-se a um único vetor de probabilidades.

Para os modelos em CTMC, a matriz de transição de estados Q denominada de gerador infinitesimal, onde cada elemento não diagonal da linha i e coluna j da matriz representa a taxa de transição do estado i para o estado j do modelo. Os elementos diagonais de Q representam o ajuste necessário para que a soma dos elementos de cada linha seja zero. Para os modelos em DTMC, a matriz de transição de estados Q é denominada de matriz estocástica, em que cada elemento representa a probabilidade de transição entre os estados do modelo.

Desta forma, as cadeias de Markov têm papel fundamental no processo de modelagem de sistemas redundantes, comportamentos especiais em pilhas de *softwares* e avaliação de dependabilidade em diversos sistemas. A utilização de CTMCs permite avaliações tanto de desempenho, quanto de disponibilidade. Suas aplicações vão desde a quantificação da vazão de uma linha de produção até a determinação de tempos de falha e reparo em sistemas críticos (BAIER et al., 2003).

Como será apresentado na próxima seção, Redes de Petri estocásticas podem ser analisadas através da obtenção da cadeia de Markov associada. De forma similar, a teoria de redes

de filas, uma fila pode ser reduzida a uma cadeia de Markov e analisada e, além disso, alguns teoremas importantes desta teoria são provados através de modelos markovianos (KLEINROCK, 1975).

2.3.3 Rede de Petri Estocásticas

Propostas por Carl Adam Petri em 1962 (PETRI, 1962), em sua tese de doutorado intitulada "Comunicação com autômatos", as Redes de Petri (PN) constituem uma ferramenta de modelagem gráfica e matemática aplicável a várias categorias de sistemas. As Redes de Petri são adequadas para descrever e estudar sistemas caracterizados por serem concorrentes, assíncronos, distribuídos e paralelos (MURATA, 1989).

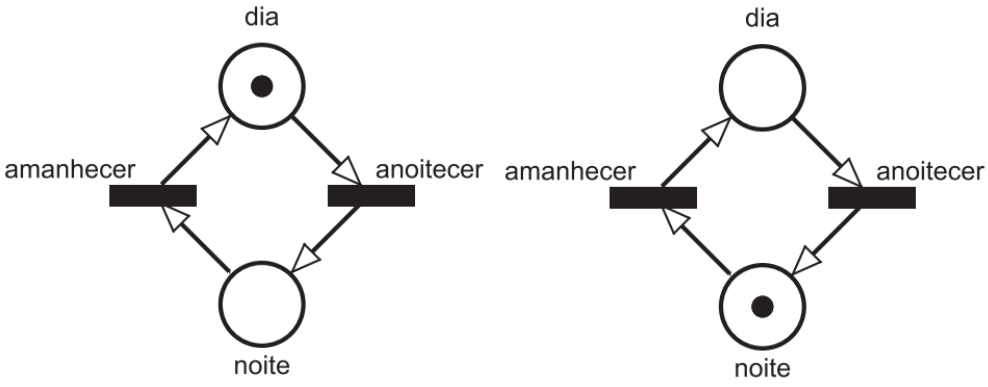
A representação gráfica das Redes de Petri é formada por lugares (Figura 2.9(a)), transições (Figura 2.9(b)), arcos (Figura 2.9(c)) e *tokens* (Figura 2.9(d)). Os lugares equivalem às variáveis de estado e as transições correspondem às ações realizadas pelo sistema (MACIEL; LINS; CUNHA, 1996). Esses dois componentes são ligados entre si através de arcos dirigidos. Os arcos podem ser únicos ou múltiplos. A distribuição de tokens nos lugares da Rede de Petri determinam o estado do sistema ou a quantidade de recursos.

(a) Lugar (b) Transição (c) Arco (d) *Token*

Figura 2.9: Elementos de uma rede de Petri

Na Figura 2.10, apresenta-se uma Rede de Petri a qual os lugares representam os períodos do dia (*dia* e *noite*), já as transições alteram o período do dia (*amanhecer* ou *anoitecer*). Nesse exemplo, o arco dirigido do lugar *dia* para a transição *anoitecer* indica que, para que anoiteça, é necessário que haja um *token* no lugar *dia*. Tal qual, o arco dirigido do lugar *noite* para a transição *amanhecer* indica que, para que amanheça, é necessário que haja um *token* no lugar *noite*. A localização do token na rede indicará, portanto, se é dia (Figura 2.10(a)) ou noite (Figura 2.10(b)).

As Redes de Petri possui extensões, onde uma delas são as redes temporizadas que buscam acrescentar às redes de Petri a possibilidade de análise no domínio de tempo (MARRANGHELLO, 2005). As SPNs oferecem possibilidade de unir a habilidade do formalismo de Redes de Petri para descrever sistemas com atividades assíncronas, concorrentes e não-determinísticas, além de conflitos ou mecanismos de redundância *cold-standby* ou *warm-standby*,



(a) Dia (b) Noite

Figura 2.10: Rede de Petri representando um dia

com um modelo estocástico, permitindo a descrição de um comportamento dinâmico na modelagem de desempenho e dependabilidade de sistemas (MARSAN; BALBO; CONTE, 1986), (GERMAN, 2000), (MURATA, 1989).

Segundo Sales (SALES, 2002), uma característica importante do formalismo de Redes de Petri estocásticas é que ele pode ser facilmente compreendido por pessoas que não tenham familiaridade com métodos de modelagem probabilística.

Graficamente, as transições estocásticas e os arcos inibidores costumam ser representados conforme exibido na Figura 2.11. São atribuídas as transições estocásticas (Figura 2.11(a)) uma taxa obedecendo a uma distribuição exponencial. A taxa da transição é inversamente proporcional ao tempo x, de modo, taxa será de $1/x$.

Os arcos inibidores ((Figura 2.11(b))), originalmente não presentes em Redes de Petri, são utilizados para inibir o disparo das transições da rede. De maneira semelhante aos arcos tradicionais, eles também podem ter um peso associado. A introdução da concepção de arco inibidor aumenta o poder de modelagem de Rede de Petri, adicionando a capacidade de testar se um lugar não tem *tokens*. Arcos inibidores e transições estocásticas podem ser substituídos por guardas associadas às transições.

(a) Transição estocástica (b) Arco Inibidor

Figura 2.11: Elementos das SPNs que estendem o comportamento das PNs.

As SPNs (CIARDO, 1994), (BALBO, 2001), (GERMAN, 2000) são obtidas através da

associação de um tempo exponencialmente distribuído com o disparo de cada transição da rede. Há muitas maneiras diferentes de representar SPNs. Este trabalho adota uma definição genérica adotada em (GERMAN, 2000), na qual uma SPN é uma tupla definida da seguinte forma:

Definição 1: Uma SPN é definida pela 9-tupla $SPN = (P, T, I, O, H, \Pi, G, M_0, Atts)$, onde:

- $P = \{p1, p2, ..., pn\}$ é o conjunto de lugares;
- $T = \{t1, t2, ..., tm\}$ é o conjunto de transições imediatas e temporizadas, $P \cap T = \emptyset$;
- $I \in (\mathbb{N}^n \rightarrow \mathbb{N})^{n \times m}$ é a matriz que representa os arcos de entrada (que podem ser dependentes de marcações);
- $O \in (\mathbb{N}^n \rightarrow \mathbb{N})^{n \times m}$ é a matriz que representa os arcos de saída (que podem ser dependentes de marcações);
- $H \in (\mathbb{N}^n \rightarrow \mathbb{N})^{n \times m}$ é a matriz que representa os arcos inibidores (que podem ser dependentes de marcações);
- $\Pi \in \mathbb{N}^m$ é um vetor que associa o nível de prioridade a cada transição;
- $G \in (\mathbb{N}^n \rightarrow \{true, false\})^m$ é o vetor que associa uma condição de guarda relacionada a marcação do lugar à cada transição;
- $M_0 \in (\mathbb{N}^n)$ é o vetor que associa uma marcação inicial de cada lugar (estado inicial).
- $Atts = (Dist, Markdep, Policy, Concurrency, W)^n$ compreende o conjunto de atributos associados às transições:

$Dist \in \mathbb{N} \rightarrow F$ é uma função de distribuição de probabilidade associada ao tempo de cada transição, sendo que $0 \leq F \geq \infty$. Esta distribuição pode ser dependente de marcação;

$Markdep \in \{constante, enabdepg\}$, define se a distribuição de probabilidade associada ao tempo de uma transição é constante ou dependente de marcação (*enabdepg* - a distribuição depende da condição de habilitação atual);

$Policy \in \{fprd, prsg\}$ define a política de memória adotada pela transição (*fprd - preemptive repeat different*, valor padrão, de significado idêntico à *race enabling policy*; *prsg -preemptive resume*, corresponde à *age memory policy*);

$Concurrency \in \{ss, is\}$ é o grau de concorrência das transições, onde *ss* representa a semântica *single server* e *is* representa a semântica *infinity server*.

$W \in \mathbb{N} \rightarrow R^+$ é a função peso, que associa um peso (w_t) às transições imediatas e uma taxa λ_t às transições temporizadas.

O formalismo de Redes de Petri estocásticas permite duas classes diferentes de transições no modelo: transições imediatas e transições temporizadas (HAVERKORT, 1998). As transições imediatas possuem um tempo de disparo igual a zero com prioridade superior às transições temporizadas e são representadas graficamente no modelo por barras finas. Estas transições disparam assim que são habilitadas. Níveis de prioridade podem ser atribuídos às transições. Para solucionar conflitos de disparos das transições são aplicados pesos às transições imediatas (HAVERKORT, 1998).

Uma transição temporizada dispara após um tempo aleatório, distribuído exponencialmente, associado à mesma quando habilitada. Estas transições são representadas graficamente no modelo por barras espessas. As taxas de disparos estão associadas somente às transições temporizadas, e estas podem ser dependentes da marcação da SPN para serem habilitadas e disparadas (SALES, 2002), (HAVERKORT, 1998).

As transições temporizadas podem ser caracterizadas por diferentes semânticas e indicam quantos disparos podem ser feitos por unidade de tempo numa transição. Conhecidas como *single server*, *multiple server* e *infinite server*, mostradas a seguir (MARSAN; BALBO; CONTE, 1986):

- ***Single server*** - Apenas um *token* é disparado por vez, ou seja, a capacidade de um lugar/transição é 1. Utilizada em modelos de disponibilidade, considerando-se a existência de uma única equipe de manutenção, quando vários componentes do sistema passam para o estado de falha;

- ***Multiple server*** - É possível fazer k disparos por vez, ou seja, a capacidade de um lugar/transição é um k inteiro. Em modelos de disponibilidade é usada quando existe um número de equipes de manutenção menor do que o número de componentes em falha, deixando-os em excesso na fila;

- ***Infinite server*** - É possível fazer infinitos disparos de uma única vez. É utilizada em modelos de disponibilidade, quando há uma equipe de manutenção exclusiva e independente para cada componente que falhar. Nesse tipo de semântica, todas as marcações são processadas em paralelo.

As SPNs marcadas com um número finito de lugares e transições são isomórficas às cadeias de Markov (HAVERKORT, 1998), (MARSAN et al., 1994). O isomorfismo de um modelo SPN com uma cadeia de Markov é obtido a partir do gráfico de alcançabilidade reduzido, que é dado através da eliminação dos estados voláteis e do rótulo dos arcos com as taxas das transições temporizadas e dos pesos das transições imediatas.

Os modelos SPN descrevem as atividades de sistemas através de gráficos de alcançabilidade. Esses gráficos podem ser convertidos em modelos Markovianos, que são utilizados para avaliação quantitativa do sistema analisado. As medições de desempenho e dependabilidade são obtidas através de simulações e de análises em estado estacionário e transiente baseadas na cadeia de Markov embutida no modelo SPN (BOLCH et al., 2006).

2.4 Intervalo de Confiança da Disponibilidade

As técnicas de modelagem apresentadas anteriormente resultaram em modelos matemáticos, que por sua vez, precisam de dados de entrada. Esses dados são derivados de medição ou pesquisas anteriores. Há casos de modelos criados a partir da união de um ou mais componentes que foram estudados anteriormente. Portanto, esses novos modelos precisam passar por um processo de validação que compare o modelo proposto com o ambiente real.

A validação pode ser realizada usando o método de Keesee (KEESEE, 1965). Este método determina um intervalo de confiança da disponibilidade. Considerando que o resultado esteja em um intervalo, é possível indicar uma correlação entre o modelo e o ambiente real. Considera-se satisfatório quando o resultado do modelo está contido no intervalo. Neste processo são usadas duas métricas de falha e reparo: tempo de execução e número de ocorrências. Estes dados são extraídos de um experimento real e servem como entrada para as fórmulas abaixo apresentadas.

Primeiro é adaptada a equação da disponibilidade, simplificando-a conforme é apresentado na Equação 2.12.

$$A = \frac{\mu}{\lambda + \mu} = \frac{1}{1 + \frac{\lambda}{\mu}} = \frac{1}{1 + \rho} \tag{2.12}$$

Assume-se que ρ é a relação entre $\frac{\lambda}{\mu}$. Considera-se para o experimento o n sendo os eventos de falha e reparo. Para o tempo total de falha é S_n e o tempo total de reparo Y_n. No trabalho, propõe-se um método para estimar o valor de λ, o qual é definido na equação.

$$\hat{\Lambda} = \frac{n}{S_n} \tag{2.13}$$

Um intervalo de confiança 100*(1 - α) para λ é dado por uma distribuição Qui-quadrada com os parâmetros definidos na Equação 2.14.

$$\left(\frac{X^2_{2n;1-\frac{\alpha}{2}}}{2S_n}, \frac{X^2_{2n;\frac{\alpha}{2}}}{2S_n}\right) \tag{2.14}$$

Para estimar μ, segue o mesmo processo usado para calcular λ, de acordo com a Equação 2.15.

$$\hat{M} = \frac{n}{Y_n} \tag{2.15}$$

Para o intervalo de confiança 100*(1 - α) para μ é similar a Equação 2.14, conforme Equação reajustada 2.16.

$$\left(\frac{X^2_{2n;1-\frac{\alpha}{2}}}{2Y_n}, \frac{X^2_{2n;\frac{\alpha}{2}}}{2Y_n}\right) \tag{2.16}$$

Consequentemente, o estimador de probabilidade máxima para a relação $\frac{\lambda}{\mu}$ é $\hat{\rho}$, definida pela Equação 2.17

$$\hat{\rho} = \frac{\hat{\Lambda}}{\hat{M}} = \frac{\frac{n}{S_n}}{\frac{n}{Y_n}} = \frac{Y_n}{S_n} \tag{2.17}$$

Um intervalo de confiança para ρ é dado entre ρl e ρu, seguindo uma função de probabilidade da distribuição F, vista na Equação 2.18.

$$\rho l = \frac{\hat{\rho}}{f2n;2n;\frac{\alpha}{2}} \;\; e \;\; \rho u = \frac{\hat{\rho}}{f2n;2n;1-\frac{\alpha}{2}} \tag{2.18}$$

Finalmente, o estimador de probabilidade máxima para a disponibilidade é $\hat{A} = \frac{1}{1+\hat{\rho}}$. Isto acontece porque A é uma função monotonicamente decrescente de ρ e o intervalo de confiança 100*(1-α) para A é a apresentada na Equação 2.19.

$$\left(\frac{1}{1+\rho u}, \frac{1}{1+\rho l}\right) \tag{2.19}$$

2.5 Análise de Sensibilidade

Análise de sensibilidade é um conjunto de técnicas usadas para determinar os fatores mais críticos em relação às medições ou saídas de um modelo (HAMBY, 1994). A técnica mais simples adotada para avaliar a sensibilidade de uma métrica com seus respectivos parâmetros é escolher um parâmetro para variar enquanto mantém os outros fixos. Isso é uma forma de obter um *ranking* de sensibilidade que registra as mudanças da saída do modelo correspondente à variação nos parâmetros de entrada (HAMBY, 1994).

Quando relacionada com modelos analíticos como cadeias de Markov, RBD e SPN, a análise de sensibilidade paramétrica torna-se uma técnica importante para que os efeitos

causados através de mudanças nas taxas de transição nas métricas de interesse sejam computados. Essa abordagem é utilizada para detectar gargalos de desempenho ou confiabilidade no sistema, guiando a otimização dos processos (BLAKE; REIBMAN; TRIVEDI, 1988; ABDALLAH; HAMZA, 2002). Análise de sensibilidade beneficia a identificação dos parâmetros que podem ser removidos sem efeitos significantes aos resultados. Modelos grandes, com uma dúzia de taxas, podem ser drasticamente reduzidos através do uso dessa abordagem.

Entre outras técnicas aplicáveis para análise de sensibilidade incluem projetos experimentais fatoriais, análise de correlação, análise de regressão, análise de perturbação (PA) bem como análise diferencial, conhecida como análise de sensibilidade paramétrica ou método direto (HAMBY, 1994), (Matos Júnior, 2011), (MATOS et al., 2012). A análise diferencial foi escolhida para esse trabalho porque pode ser eficientemente realizada nos tipos de modelos analíticos, normalmente empregados nos estudos de disponibilidade e desempenho. Na análise diferencial as derivadas parciais das métricas de interesse são calculadas dos respectivos parâmetros de interesse. Se métrica Y depende de um parâmetro (λ), a sensibilidade de Y com respeito a λ é calculada pela Equação 2.20 ou 2.21, quando adotado a sensibilidade escalada (FRANK, 1978).

$$S_{\lambda}(Y) = \frac{\partial Y}{\partial \lambda} \tag{2.20}$$

$$S^{*}_{\lambda}(Y) = \frac{\partial Y}{\partial \lambda}\left(\frac{\lambda}{Y}\right) \tag{2.21}$$

Outros métodos de escala podem ser empregados, dependendo da natureza do parâmetro, entre eles a métrica de interesse, a necessidade para compensar grandes diferenças em unidades e magnitude de medida (Matos Júnior, 2011). $S_{\lambda}(Y)$ e $S^{*}_{\lambda}(Y)$ também são referidos como coeficientes de sensibilidade (HAMBY, 1994), cujos valores são ordenados em um ranking, que compara o grau de influência entre todos os parâmetros.

2.6 *Availability Importance*

A importância de componentes pode ser analisada em termos de confiabilidade, de disponibilidade e destas relacionadas com custos. Este trabalho restringirá a importância do componentes em relação a disponibilidade, denominada *Availability Importante*. Essa análise destaca quais componentes tem maior impacto na disponibilidade do sistema, ou quais componente que, aumentando sua disponibilidade, aumentarão mais significativamente a disponibilidade (FIGUEIRÊDO; ROMERO MARTINS MACIEL, 2011).

O índice de *Availability Importante* $\left(I_i^A\right)$ é baseado no índice de *Reliability Importance*. Este considera o tempo médio para falha dos componentes, justificado por ser uma métrica de

confiabilidade. Para o avaliador ao executa os cálculos, ao invés de utilizar a confiabilidade, a métrica utilizada é a disponibilidade, incluindo, além do tempo médio para falha, o tempo médio de recuperação dos componentes. O cálculo utilizado é mostrado na Equação 2.22, e os resultados normalizados pelo maior valor é calculado pela Equação 2.23 (FIGUEIRÊDO; ROMERO MARTINS MACIEL, 2011).

$$I_i^A = A_s(1_i, p^i) - A_s(0_i, p^i) \tag{2.22}$$

Onde $\mathbf{p}_i$ representa o vetor de disponibilidades dos componentes com o *i*ésimo componente removido; 0_i representa a condição quando o componente i é falho e 1_i a condição quando o componente i está no modo operacional.

$$In_i = \frac{I_i}{I_x} \tag{2.23}$$

Onde In_i é o índice normalizado para o componente i; I_i é o valor do índice não normalizado para o componente i e, I_x é o valor do maior índice não normalizado entre os componentes.

Essa técnica de análise será usada na subseção 5.2.1, do Capítulo 5. Com o objetivo de apontar quais componentes serão replicados no Estudo de Caso II, referente ao contexto do sistema empresarial.

2.7 *Video Surveillance as a Service*

Serviços de *live video streaming* são proferidos na Internet (PICCONI; MASSOULIÉ, 2008). Ao mesmo tempo, os desafios específicos desta tecnologia, tais como taxa bits constantemente variável, transmissões especiais de agendamento, bem como a necessidade de métodos de gerenciamento de armazenamento para lidar com tráfego intenso (CHANG; VETRO, 2005), estão incentivando consideravelmente as pesquisas e desenvolvimentos nesta área. Um dos principais representantes dos serviços de *live video streaming* é o *VSaaS*. Em pesquisa da IMS Research, sobre as tendências em matéria de tecnologias vigilância visuais prevê crescente demanda de nuvem baseadas soluções de vigilância LAWTON (2009). O conceito de sistema de vigilância de vídeo baseada em nuvem está em sua fase de infância SHARMA; KUMAR (2014).

O conceito *VSaaS* é bastante novo. Os serviços são restritos, principalmente, ao armazenamento de dados de vídeo na nuvem, redirecioná-los o cliente, ou para enviar relatórios analíticos após a triagem manual dos dados SHARMA; KUMAR (2014).

Dificilmente, há prestadores de serviços que conseguem transmitir e analisar os fluxos de vídeos em tempo real, porque existem vários desafios para alcançar este nível de operação

simultânea. A Smartvues lista 10 desafios para *VSaaS* M (2012). Entre estes a largura de banda, velocidade, gestão de dados, segurança e transferência são desafios chaves. Com a transmissão realizada pelas tecnologias 3G e 4G a solução das questões como a largura de banda e a velocidade estão quase lá SHARMA; KUMAR (2014).

Além disso, os serviços *VSaaS* começaram a compreender as preocupações dos seus clientes relacionadas com a segurança e privacidade, então eles apresentaram políticas mais transparentes e justas relativas ao armazenamento e uso de dados de vídeo dos clientes SHARMA; KUMAR (2014).

Com a maturidade do mercado de *VSaaS* nos últimos anos o serviço é oferecido em três modelos de negócios: o serviço hospedado, o sistema gerido pelo fornecedor ou um modelo híbrido no qual os potenciais clientes precisam perceber as principais diferenças. O primeiro modelo é o serviço hospedado de vigilância por vídeo que grava as imagens fora das instalações do cliente com as imagens a serem transferidas sobre a rede do cliente para o fornecedor *data centers*, onde são geridas e armazenadas, implicará em capital de investimentos para aquisição dos equipamentos de armazenamento.

O segundo modelo é o serviço gerido de vigilância por vídeo, onde a gravação das imagens é realizada nas instalações do cliente e a vigilância é gerida remotamente pelo fornecedor. O último modelo é o híbrido. Neste caso, as imagens são transmitidas para o site do fornecedor, mas também são armazenadas nas instalações do cliente de várias maneiras – nas câmaras, em um aparelho suplementar ou em um suporte de armazenamento anexado à rede ou NAS (*Network Attached Storage*).

Por muitas razões, incluindo segurança e a popularidade das câmeras IP para vídeos de vigilância estão crescendo rapidamente. Na Terceira geração dos sistemas de vídeo de vigilância, são necessárias informações dos vídeos, como o fluxo ou sequência das imagens digitais, fornecendo uma abordagem inteligente e eficiente para análise automática de sistemas.

No mercado de *VSaaS*, existem algumas empresas ofertando esse serviço, entre elas: a Axis, Ivideon, MagonCam, K Group Companies, Neovsp, Security Station. A seguir estão listadas algumas vantagens apresentadas pela empresa IVIDEON na utilização do serviço de *VSaaS*:

- **Serviço Global** - Acessar todas as suas câmeras em diferentes continentes;
- **Incorporar o vídeo em seu website** - adicionar um vídeo ao vivo da sua câmera em um website ou blog, ou compartilhar um link direto para o Facebook;
- **Compatibilidade** - Utilizar uma Câmera IP cara, uma CCTV ou até uma simples webcam;

- **Dar acesso a outros usuários** - definir a permissão de acesso de vários usuários as câmeras de forma simultânea;
- **Aplicativos Móveis** - vê o que está acontecendo em qualquer lugar usando um navegador ou um aplicativo móvel para iPhone, iPad e Android;
- **Marcar as câmeras em um mapa** - Criar marcação de todas suas câmeras no Google Maps;
- **Segurança** - Todos os dados - incluindo transmissões de vídeo - são criptografados;
- **Arquivos armazenados em nuvem** - opção de armazenar os vídeos localmente ou mantê-los seguros na nuvem.

A empresa *Security Station*, esboça na Figura 2.12 como é a forma de funcionamento do *VSaaS* oferecido por ela. A empresa utiliza Câmeras IP que são gerenciadas por uma nuvem, que também pode realizar a gravação dos vídeos. Esta nuvem disponibiliza para o usuário final a visualização das imagens em tempo real por meio dos navegados dos dispositivos (*tablet*, *smarthphone* e computador).

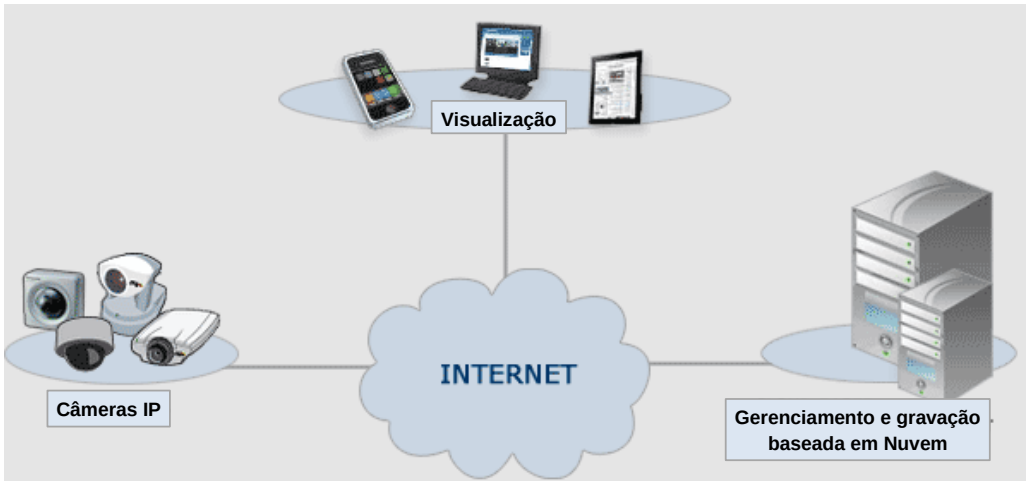


Figura 2.12: Estrutura do *VSaaS*, da empresa *Security Station*

3

Metodologia de Avaliação e Arquiteturas *VSaaS*

Não desista nas primeiras tentativas, a persistência é amiga da conquista.
Se você quer chegar onde a maioria não chega, faça o que a maioria não faz.
—BILL GATES

Este Capítulo apresenta a metodologia utilizada nessa pesquisa e os sistemas utilizados nos estudos de casos com suas respectivas arquiteturas modeladas analiticamente.

3.1 Metodologia de Avaliação de Disponibilidade

Nesta Seção externa-se uma metodologia para auxiliar a avaliação e os modelos de disponibilidade para *VSaaS*. A metodologia propõe diversas etapas, entre as quais, o entendimento do sistema, definição de métricas de interesse, obtenção dos modelos, validação do modelo, avaliação do sistema e, por fim, a interpretação dos resultados atraves de análises. Na Figura 3.1 as etapas da metodologia adotada são exibidas em forma de fluxograma.

A primeira etapa da metodologia ora proposta diz respeito à compreensão dos sistemas *VSaaS*, dos seus componentes e das suas interações. É nesta fase que são adquiridos os insumos necessários para a modelagem do sistema. Caso o entendimento do sistema não seja claro, o avaliador provavelmente cometerá erros de interpretação e, consequentemente, comprometerá as etapas seguintes. Esta etapa também divide o sistema em subsistemas menores para mitigar a complexidade na avaliação do modelo final do sistema. Tal subdivisão facilitará na etapa de obtenção dos modelos.

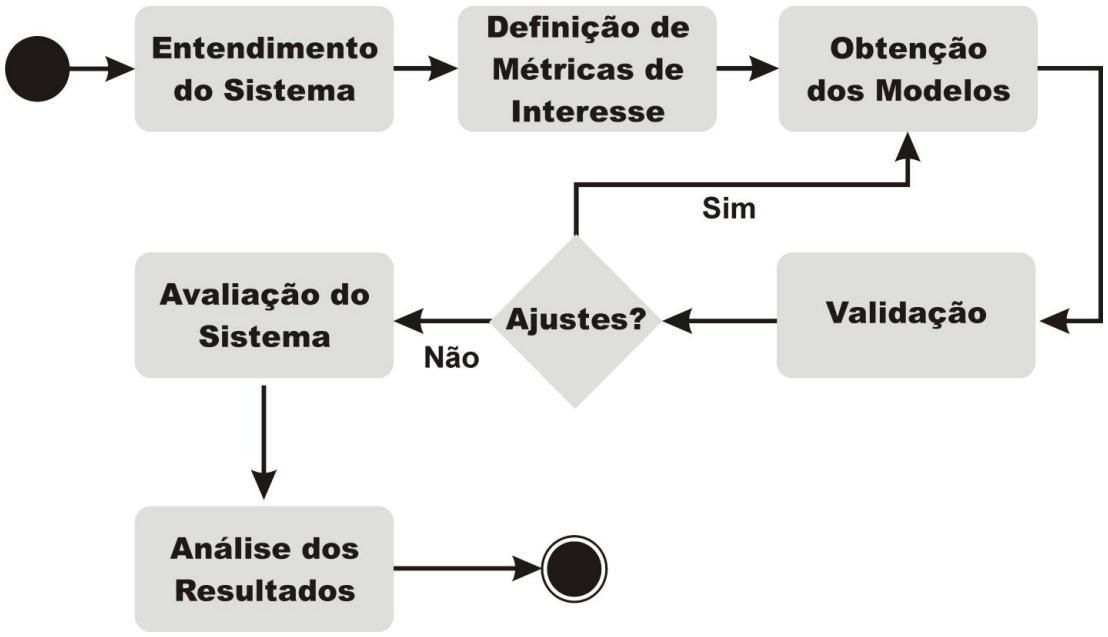


Figura 3.1: Metodologia

Com base nos entendimentos do sistema e atrelado a pesquisas encontradas e/ou visitas em organizações, os sistemas e suas arquiteturas a serem utilizadas durante o estudo são definidas nessa etapa. Nesta fase, deve-se obter os parâmetros de entrada para avaliação dos modelos, procurando o MTTF, MTTR ou *Mean Time Between Failures* (MTBF) de todos componentes que compõem a arquitetura. Esses parâmetros foram obtidas através de pesquisas anteriormente publicadas, porém eles poderiam ser obtidos por meio de medição.

Na segunda etapa são definidas as métricas relevantes para a avaliação de um sistema *VSaaS*, bem como a definição de métricas de dependabilidade. As métricas usadas foram disponibilidade, *downtime* anual e custo. Essas métricas foram especificadas no Capítulo 2 e serão formuladas no Capítulo 4 seguindo a sintaxe suportada pelas ferramentas de avaliação utilizadas.

Em seguida, os modelos são criados para cada subsistema utilizando a técnica de modelagem mais adequada, como RBD, CTMC ou SPN (para citar apenas os modelos adotados neste trabalho). Tais modelos são então adotados para obter as métricas de interesse. Nesta etapa, deve-se realizar a escolha das ferramentas apropriadas para modelar (entre as quais *Symbolic Hierarchical Automated Reliability and Performance Evaluator* (SHARPE), Mercury e TimeNet), em função da técnica usada e ferramentas matemáticas (o Mathematica) para análise e obtenção de fórmulas fechadas.

A etapa de validação é responsável por verificar se o modelo representa o sistema de

forma satisfatória ou se não cumpre os requisitos do sistema. Um método viável de realização destes estudos é através de injeção automatizada de falhas, permitindo observar o comportamento do sistema sob muitas condições. Para esse experimento de injeção de falha, foi necessário a montagem do ambiente de teste, onde é instalado e configurado o sistema conforme a arquitetura básica. Por conseguinte, desenvolveu-se algoritmos em Shell Script para inserir falhas nos componentes do sistema com o intuito de simular o comportamento do ambiente real, e outro para monitorar o serviço com a finalidade de verificar se o sistema está operacional ou não.

As taxas de falha e reparo dos componentes usadas no experimento são tempos exponencialmente distribuídos, ressalva-se que estas taxas foram aceleradas, a fim de reduzir o tempo total do experimento, tendo em vista que falhas são imprevisíveis e podem levar um longo tempo para se manifestar. Este processo é mais bem detalhado na Seção 4.1.3. Com a execução do experimento é obtido resultados que serão utilizadas no processo de validação do mesmo. Para que o modelo seja considerado válido, é necessário que este proporcione o cálculo de métricas com erros de exatidão dentro de níveis considerados aceitáveis, denominado intervalo de confiança da disponibilidade. Uma vez que satisfaça as exigências, o processo segue o passo seguinte, de outro modo, os ajustes nos modelos devem ser realizados de modo a representar corretamente o comportamento do sistema.

O próximo passo é muito importante porque é onde a disponibilidade será avaliada. Os modelos de subsistema são avaliados considerando várias arquiteturas que foram propostas mediante a avaliação da arquitetura inicial, considerada *baseline*. As arquiteturas criadas devem representar as situações de interesse. Essas arquiteturas serão comparadas com finalidade de determinar se certa estratégia de redundância é satisfatória ou não.

Por fim, os resultados obtidos são analisados de forma adequada com apoio de ferramentas matemáticas (Mathematica, Mercury e o Minitab). Pode-se identificar o impacto dos diferentes parâmetros na disponibilidade do sistema. É responsabilidade do avaliador interpretar os dados e apontar as possíveis melhorias na infraestrutura. Os resultados podem apontar pontos de falhas da infraestrutura mais críticos que outros. As técnicas que auxiliam o avaliador nessa intepretação foram *Availability Importance*, Análise de Sensibilidade Paramétrica e Análise de Custo, por meio da distância euclidiana até seu ponto de origem. Além disso, a conclusão das arquiteturas com gráficos e tabelas auxiliam a apresentação desses resultados.

3.2 Arquiteturas *VSaaS*

Nesta Seção serão detalhados os dois sistemas adotados nesta Dissertação. O primeiro deles é denominado Doméstico para ser utilizada em casos simples, por exemplo, casas, escolas

ou comércios. No segundo, tratou-se de uma concepção dirigida à empresa, chamado Empresarial. Por inúmeras particularidades possíveis de outras arquiteturas, esse trabalho propõe escolher restrições genéricas que possibilitem aplicá-las em outras situações. As restrições, os recursos, operações, componentes e as explicações sobre o funcionamento desses sistemas são apresentadas nas subseções a seguir.

3.2.1 Arquiteturas do Sistema Doméstico

A primeira arquitetura do sistema doméstico teve um papel fundamental, porque serviu de entendimento do *VSaaS* para implantação do ambiente de experimento e sua validação. Esta arquitetura foi adaptada da plataforma *VSaaS* do trabalho de Xiong (XIONG et al., 2014) e ilustrada pela Figura 3.2.

Vale ressaltar a importância de compreender os motivos pelo qual este estudo optou analisar um sistema de vigilância em nuvem, ao invés do sistema convencional. Os sistemas convencionais são entendidos com uma central de monitoramento na própria empresa. Desta forma, pode haver problemas do custo inicial elevado na implantação e uma segurança que pode se transformar em insegurança, tendo em vista que a central pode ser alvo dos criminosos. Então, com o serviço nas nuvens, existe o benefício de baixo custo na instalação (pela escalabilidade dos recursos) e o aumento nos níveis de segurança.

Uma plataforma *VSaaS*, normalmente, possui dois momentos com a interação do cliente. O primeiro em relação a geração das *streaming* de vídeo e o segundo no acesso remoto as essas imagens. Os vídeos gerados são enviados pela Internet até uma nuvem privada, que no nosso estudo foi o OpenNebula 3.6 [1]. Esta plataforma privada de nuvem é composta por Frontend, Node, dispositivo de armazenamento e uma máquina virtual, esta por sua vez possui um servidor web (Apache[2] com uma aplicação que recebe e disponibiliza a *streaming* de vídeo (Ffmpeg[3]).

Contudo, esta pesquisa limita-se a verificar a disponibilidade da arquitetura do conjunto de câmeras no lado do cliente até um ambiente de nuvem que oferece a transmissão pelo serviço *web*, não considerando a visualização e monitoramento do usuário através de qualquer que seja o dispositivo com acesso à Internet e um navegador. Da mesma forma, também não foram considerados o componente e o software de armazenamento.

Salienta-se que nos experimentos e na modelagem não se detém aos problemas do *link* de conexão com a Internet, tais como: *jitter*, latência, perda de pacotes e taxa de transferência. Esses aspectos não foram levados em consideração pelas dificuldades dos provedores de Internet

[1] http://opennebula.org/
[2] http://www.apache.org/
[3] https://www.ffmpeg.org/

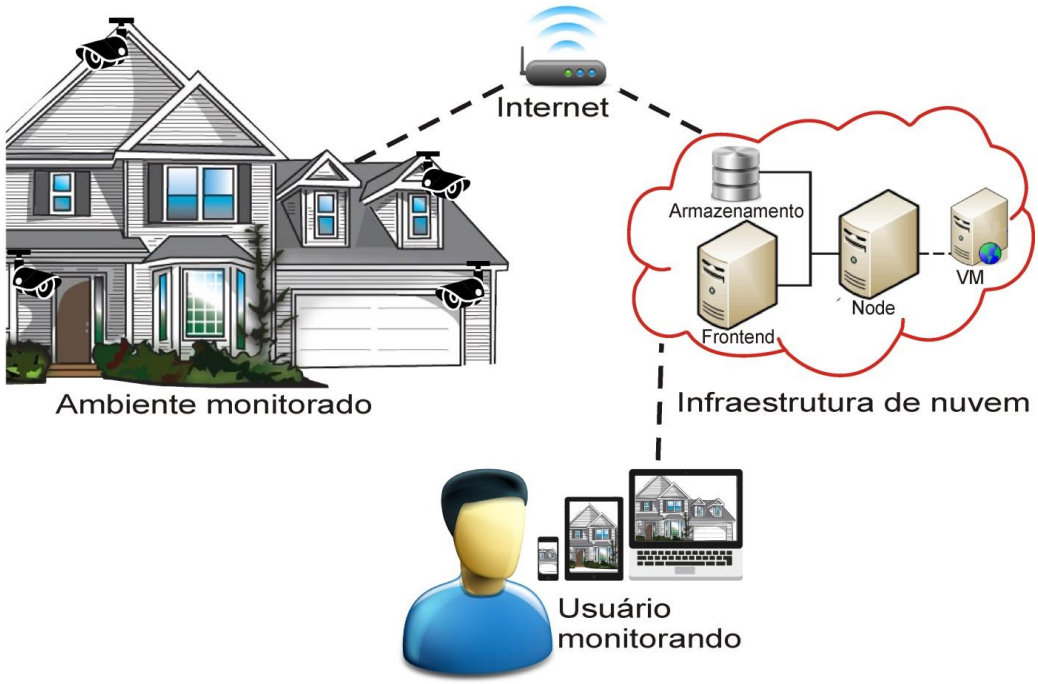


Figura 3.2: Pataforma *VSaaS*.

em garantir taxas constantes de transmissão. Utilizou-se ainda, tecnologias de conexão sem fio (a 3G e WiFi) que podem receber interferência do ambiente. Desta forma, deixa-se explícito que esses aspectos podem afetar os resultados desta pesquisa.

Na Figura 3.3 é visualizada a arquitetura 1, denominada básica, por possuir apenas os componentes necessários para o seu funcionamento. Percebe-se que é uma arquitetura com ausência de componentes replicados. Esses componentes são apresentados com suas respectivas interações. Sendo composto por Câmera (C), um computador de apoio chamado de *Streaming Transmitter* (ST), conexões de *Wireless Fidelity* (WiFi), *Frontend* (F), *Node* (N) e subconjunto denominado *Live Video* (LV), todos esses componentes serão explicados adiante.

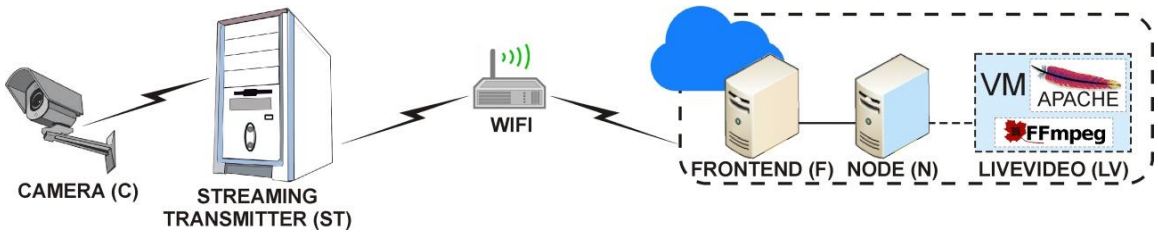


Figura 3.3: Arquitetura 1 - Básica.

Quanto ao modo operacional dessa arquitetura, entende-se que a plataforma estará indisponível quando ocorrer alguma falha na câmera, no *streaming transmitter*, na WiFi, no

Frontend ou no serviço LiveVideo, dessa forma, o modo operacional para arquitetura da Figura 3.3 pode ser sintetizado na Expressão Lógica 3.1, (onde os componentes são representados por *C*, *ST*, *WiFi*, *F*, *N* e *LV*, respectivamente). Uma falha na câmera impossibilita o acesso as imagens e compromete o funcionamento do sistema. Essas imagens podem sofrer interrupções no lado do cliente caso o *ST* ou a conexão de rede *WiFi* falhem. No âmbito do ambiente de nuvem a falha do *F* afetará todos os componentes do sistema, tornando o serviço da plataforma indisponível. Da mesma forma, um defeito no *N* ocasionará na indisponibilidade do serviço, bem como uma falha no conjunto do *LV* que pode ocorrer em qualquer um dos componentes de software que formam a pilha de serviço ilustrada na Figura 3.3.

$$ArqVSaaS1_{Ativa} = C \wedge ST \wedge WiFi \wedge F \wedge N \wedge LV \tag{3.1}$$

Explicando o funcionamento desses componentes, a **Câmera** é o componente que caracteriza esta arquitetura em um sistema de vídeo de vigilância, tendo a finalidade de gravar e transmitir as imagens dos locais definidos estrategicamente. Porém, saber qual o melhor posicionamento, com a menor extensão de pontos cegos não é uma decisão fácil. Para esta decisão, as arquiteturas 2 e 3 com redundância (das Figuras 3.4 e 3.5) considera-se o trabalho de (SUTOR et al., 2008). Neste trabalho é apresentada uma Cobertura da Qualidade Visual de Vigilância (Cobertura da Qualidade Visual de Vigilância (VSCQ)), que é um parâmetro de desempenho usado para determinar a localização, distribuição e número de câmeras que asseguram a cobertura mínima, se uma ou mais câmeras falharem. Assume-se para estas arquiteturas o parâmetro chamado de *VSCQ1* (cobertura total do espaço), o que requer que pelo menos duas das três câmeras funcionem. É considerado um tipo alternativo de redundância que aumenta os níveis de segurança mesmo como possível fracasso de uma câmera.

O ***Streaming Transmitter*** é responsável pela gestão das câmeras no ambiente que será monitorado, recebimento da *streaming* de vídeo e a transmissão para a plataforma. Este sistema é composto por um computador e um software de transmissão Ffmpeg. Este foi incluso na arquitetura básica em função do baixo custo do computador (considerando a aquisição e o consumo de energia) e principalmente pela facilidade na configuração de diferentes tipos de câmeras, desde uma simples *webcam* até uma câmera IP (escolhida para o modelo estudado). No entanto, este componente pode ser substituído por câmeras que se ligam diretamente à Internet ou a um dispositivo de rede que receba e envie *streaming* de vídeo.

As **Conexões de Rede** permitem o acesso à Internet via Wi-Fi ou 3G para transmissão. Estas conexões podem ser usadas separadas ou de forma redundantes de acordo com a disponibilidade desejada, na arquitetura 1 considera-se apenas Wi-Fi. Já nas arquiteturas 2 e 3 foram integrada conexão 3G, conforme visto nas Figuras 3.4 e 3.5, respectivamente. O ambiente de

computação em nuvem é estruturado com um **Frontend** e vários *Nodes* que formam um *cluster*. A gestão dos recursos da infraestrutura e controle de *Nodes* é atribuição do *Frontend*, além de decidir pelas instanciação das VMs nos *Nodes*. Por sua vez, os *Nodes* executarão o serviço de *LiveVideo* e poderá programar políticas de alta disponibilidade, tais como (*heartbeat*[4]), *live migration* ou balanceamento de carga.

O subsistema **LiveVideo** é composto de uma VM e um servidor web (Apache[5]). Este componente permite que o usuário monitore (via computadores ou dispositivos móveis) as imagens que estão sendo capturadas. Para as transmissões será utilizado um mesmo aplicativo de *streaming* de vídeo do (FFmpeg). Outra função desempenhada neste subsistema é a compactação do vídeo para um formatado desejado, já que o administrador pode optar por maior qualidade de imagem possuindo uma rápida conexão de Internet ou uma menor qualidade para não haver travamento nas imagens ao serem assistidas.

A segunda arquitetura do sistemas domésticos é exibida na Figura 3.4, nesta há redundância na câmera e no serviço *LiveVideo*, justificada pelo *VSCQ1* do trabalho (SUTOR et al., 2008). Além desta, a 3G foi incorporada como outro componentes redundante na conexão de rede. O modo operacional desta arquitetura é sintetizada na Expressão Lógica 3.2 e difere do modo operacional da arquitetura básica, pelo fato da redundância. No caso das câmeras, para estarem disponível é necessário as seguintes combinações, *C1* e *C2*, ou *C1* e *C3* ou *C2* e *C3* estarem funcionando. Esta mesma lógica é aplicada para o *LiveVideo*. Para as conexões de rede basta a WiFi ou o 3G funcionarem para o sistema está disponível.

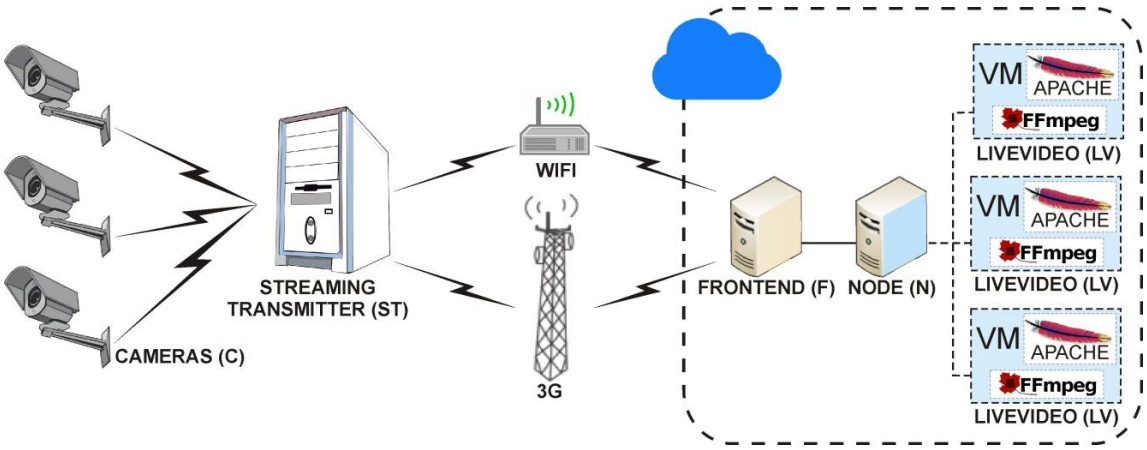


Figura 3.4: Arquitetura 2 - Com redundância.

[4]http://linux-ha.org/wiki/Heartbeat
[5]http://www.apache.org/

$$ArqVSaaS2_{Ativa} = ((C_1 \wedge C_2) \vee (C_1 \wedge C_3) \vee (C_2 \wedge C_3)) \wedge ST \wedge (WiFi \vee 3G) \wedge F \wedge N \wedge ((LV_1 \wedge LV_2) \vee (LV_1 \wedge LV_3) \vee (LV_2 \wedge LV_3)) \quad (3.2)$$

A última arquitetura do sistema doméstico (Figura 3.5) é semelhante à segunda, apenas com redundância no componente *Node*. O modo operacional desta arquitetura encontra na Expressão Lógica 3.3. O princípio para o componente *Node* esteja em funcionamento é o mesmo utilizado para o conjunto das câmeras.

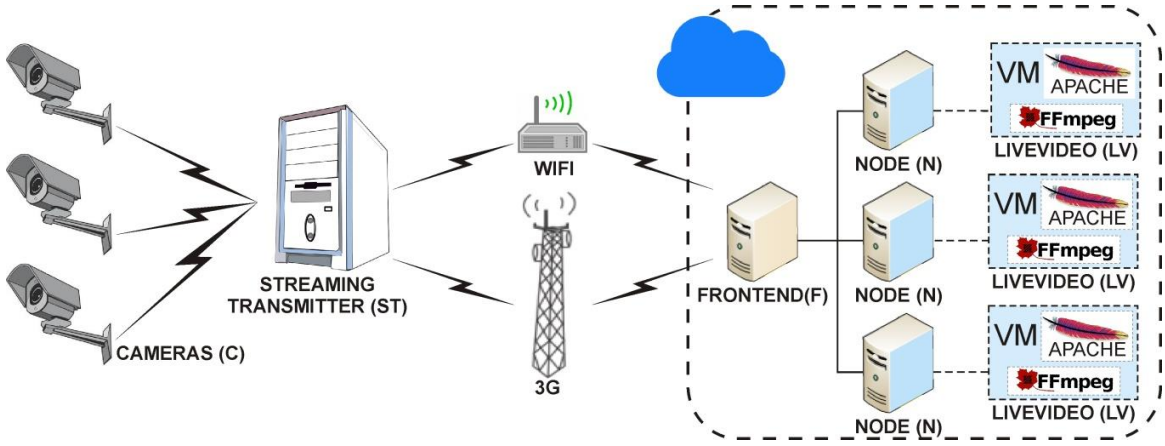


Figura 3.5: Arquitetura 3 - Com redundância.

$$ArqVSaaS3_{Ativa} = ((C_1 \wedge C_2) \vee (C_1 \wedge C_3) \vee (C_2 \wedge C_3)) \wedge ST \wedge (WiFi \vee 3G) \wedge F \wedge ((N_1 \wedge N_2) \vee (N_1 \wedge N_3) \vee (N_2 \wedge N_3)) \wedge ((LV_1 \wedge LV_2) \vee (LV_1 \wedge LV_3) \vee (LV_2 \wedge LV_3)) \quad (3.3)$$

3.2.2 Arquiteturas do Sistema Empresarial

Mediante visita a uma empresa de contabilidade do município de Caruaru, que por motivo de privacidade será chamada de Empresa X. Os componentes são os mesmo utilizados na arquitetura básica da Seção anterior, com suas respectivas funcionalidades. Porém, as restrições que compreendem a disponibilidade, ausência da conexão de rede 3G e a organização na distribuição das câmeras são diferentes. Este sistema foi rotulada de Empresarial e é apresentado na Figura 3.6.

A Empresa X é formada por cinco andares, onde cada um deles é constituído por um corredor que dá acesso a quatro salas. Em um contexto, sem adotar redundância, em cada sala e

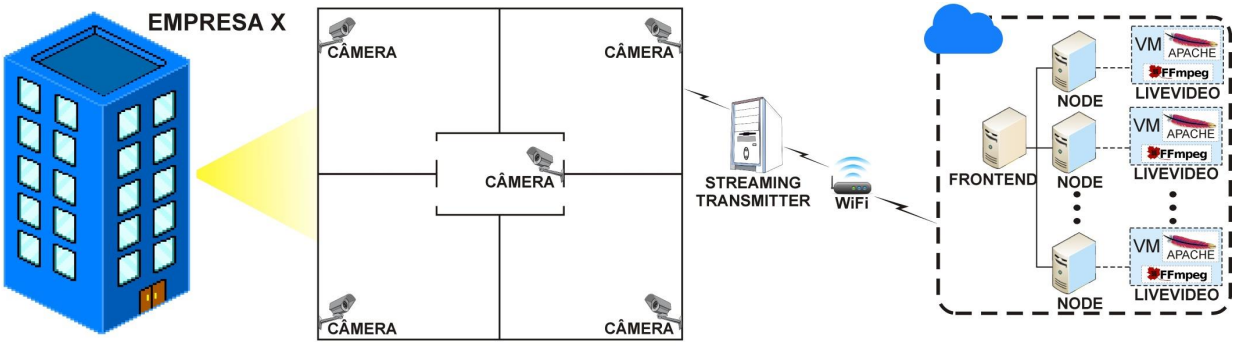


Figura 3.6: Arquitetura de uma empresa com *VSaaS*

nos corredores foram implantadas as câmeras direcionadas as portas de acesso de cada sala, com o intuito de registrar a entrada e a saída das pessoas.

Em uma das salas possui o *Streaming Transmitter* que tem a função de receber o sinal de todas as câmeras da empresa e, por meio de uma conexão (WiFi) com a Internet, comunica e distribui os vídeos com o ambiente de Computação em Nuvem. Este componente, possibilita compreender uma diferença entre o sistema de vídeo de vigilância convencional com o *VSaaS*. O sistema convencional é monitorado em tempo real por uma guarda ou terá de armazenar tudo que é gravado pelas câmeras. Com isso, irá gerar um risco de segurança, onde o computador poderá ser roubado ou quebrado, consequentemente perdendo todos os dados. Custos elevados são necessários para montar um servidor de armazenamento localmente, sendo importante saber que em um dos seus modelos de negócios, o *VSaaS* oferece o serviço de armazenamento.

No funcionamento de *VSaaS*, o ambiente em nuvem proporcionará um menor custo, pelo fato de seu uso ser sob demanda e os dados não serem armazenados localmente, resolvendo o problema de segurança em caso de roubo ou o computador ter um dano irreparável. Todavia, no contexto de segurança em *VSaaS* encontra-se a geração de novos problemas, como é o caso da invasão de privacidade. No caminho da transmissão dos dados, *hackers* podem ter acesso às imagens. Então, é algo que também se deve ter cuidado. Mas, com os avanços da computação em Nuvem esse problema vem sendo trabalhado e os riscos minimizados.

Na atuação do *Frontend*, *Node* e *LiveVideo* que compõem a infraestrutura na nuvem, somente o *Node* teve restrição. Foi limitada a instanciação de 10 VMs para cada *Node* que continuará com as funcionalidades de gerenciar os recursos de processamento e disponibilizando pela Internet os vídeos de vigilância.

Porém, como afirmar que o *VSaaS* desta empresa esteja disponível? Se o modo operacional fosse que todas as câmeras devessem estar funcionando para ser considerado ativo teria uma baixa disponibilidade por causa da restrição rígida que poderia ser utilizada, porém nem sempre o radicalismo é a estratégia mais sensata. Então, a pesquisa considerou a restrição do modo

operacional definido por andar, onde cada um deles possui cinco câmeras. Se considerarmos que todas as informações e materiais importantes estão nas salas, pode-se afirmar que, de modo operacional, pelo menos quatro das cinco câmeras estejam funcionando para que o sistema continue ativo. Por exemplo, considera-se que, caso uma das câmeras instaladas nas salas falhe por meio da câmera implantada no corredor, será descoberto quem teve acesso ao local. Da mesma forma, se a câmera do corredor falhar não haverá influência na disponibilidade, já que todas as outras salas tiveram os conteúdos gravados.

Nesta arquitetura, tem-se uma nova abordagem em relação ao serviço *LiveVideo* e a câmera. Considera-se, pois que a câmera um esteja associada ao serviço *LiveVideo* um. Com isso, uma Expressão Lógica 3.4 foi gerada para encontrar modo operacional. Ressalva que está Expressão Lógica é utilizada para entender a disponibilidade de um andar.

$$ArqVSaaSEmpresarial_{Ativo} = CameraLiveVideo_{Ativo} \wedge WiFi \wedge F \wedge (N \vee N \wedge N) \quad (3.4)$$

A notação única da câmera um com o *LiveVideo* ficou estabelecido em CLV. A notação dos demais componentes já apresentados nas arquiteturas da Seção Anterior. Para não comprometer a estética e compreensão na apresentação das informações, optou por apresentar o modo operacional da câmera com serviços *LiveVideo* a parte, na Expressão Lógica 3.5. Esta estratégia é melhor explicada no Capítulo 5.

$$\begin{aligned} CameraLiveVideo_{Ativo} = {} & (CLV_1 \wedge CLV_2 \wedge CLV_3 \wedge CLV_4) \vee (CLV_5 \wedge CLV_2 \wedge CLV_3 \wedge CLV_4) \vee \\ & (CLV_1 \wedge CLV_5 \wedge CLV_3 \wedge CLV_4) \vee (CLV_1 \wedge CLV_2 \wedge CLV_5 \wedge CLV_4) \vee \\ & (CLV_1 \wedge CLV_2 \wedge CLV_3 \wedge CLV_5) \vee (CLV_1 \wedge CLV_2 \wedge CLV_3 \wedge CLV_4 \wedge CLV_5) \end{aligned} \quad (3.5)$$

Portanto, o uso da estratégia das câmeras e dos serviços *LiveVideo* é um bom exemplo de segurança que se combina com alta disponibilidade. Porém, outras interpretações podem ser geradas, por exemplo, se duas ou três das cinco câmeras estiverem funcionando em cada andar, será que aumentará a disponibilidade? Para se ter essa percepção estas três interpretações são verificadas em parte do estudo de caso 2. Com base nesta arquitetura inicial do Sistema Empresarial outras arquiteturas foram propostas com o uso da técnica de *availability importance*. Por fim, serão aplicadas uma análise comparativa de custo e o *downtime* das arquiteturas, com o objetivo de encontrar uma arquitetura eficaz. Para isso, os dados das arquiteturas serão normatizados para calcular a distância euclidiana e encontrar a menor distância em relação a origem, entre os resultados obtidos.

4

Modelos de Disponibilidade

A única maneira de enfrentar o obstáculo maior é ultrapassar os obstáculos menores."

—(JULIO VERNE)

Este capítulo apresenta a fase de modelagem analíticas das arquiteturas dos sistemas doméstico e empresarial, com suas respectivas variações. O modelo da primeira arquitetura proposta foi submetido ao processo de validação. As técnicas de modelagem mediante complexidade que surgiu durante o concebimento dos modelos, foram utilizadas RBD, CTMC e SPN. Os mecanismos de redundância incorporados nas arquiteturas são elucidados. Por meio dos modelos gerados foi possível obter as fórmulas fechadas, importantes para calcular métricas de disponibilidade e realização da análise de sensibilidade e do *availibity importance*. As modelagens partem da técnica RBD para os comportamentos simples, por indicar como que o funcionamento dos componentes que forma um sistema pode afetar no funcionamento de determinado sistema. Por RBD possibilita o cálculo de métricas de dependabilidade, como a disponibilidade (MACIEL et al., 2011), métrica principal dessa Dissertação. Alguns componentes dos modelos RBD são formados por outros componentes, onde alguns podem ser representados por outros RBDs, enquanto que outros apresentam interações mais complexas entre seus componentes, justificando o uso de modelos baseados em estados em tais situações, como é o caso do CTMC ou SPN (MACIEL et al., 2011).

4.1 Modelagem das Arquiteturas do Sistema Doméstico

Para a modelagem e a validação desta arquitetura, esta seção será divida em três subseções: a primeira contém os componentes que foram modelados com RBD; a segunda contém os

componentes modelados pela CTMC, caso específico do serviço *LiveVideo*; e por conseguinte, são explicados os resultados do processo de validação.

A Figura 4.1 representa o resultado da fase de modelagem da arquitetura básica ilustrada na Figura 3.3, na Seção 3.2. Este modelo de alto nível está representado por um conjunto de RBDs, no total de seis submodelos. Suas notações foram simplificadas para possibilitar que as fórmulas fechadas sejam expressas: *Câmera (C)*, *Streaming Transmitter (ST)*, *WiFi (WF)*, *Frontend F)*, *Node (N)* e *LiveVideo (LV)*. Alguns blocos estão com sombreamento e sinalizam blocos que são representados por submodelos de forma hierárquica, por outros blocos RBDs ou aplicada outra técnica de modelagem, CTMC.

4.1.1 Modelos RBDs

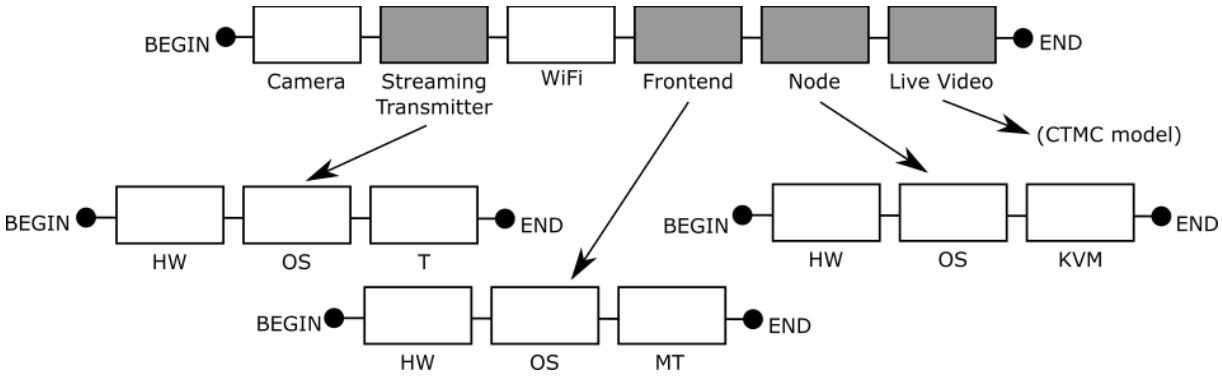


Figura 4.1: Modelo RBD da arquitetura 1.

Para facilitar a didática de apresentação às três variações desta arquitetura inicial, as fórmulas fechadas são exibidas mediante ilustração das figuras nos respectivos modelos. Elas foram usadas para calcular a disponibilidade destas arquiteturas. O resultado da fórmula fechada do modelo da Figura 4.1 é a Equação 4.1. Os componentes ST, F e N estão sombreados para indicar o fato que foram avaliados por outros modelos RBDs e o *LV* foi por CTMC. A $A_{Arquitetura1}$ é a disponibilidade da arquitetura básica, sendo formada pela multiplicação das disponibilidade dos componentes A_C, A_{ST}, A_{WF}, A_F, A_N e A_{LV}.

Sua particularidade da modelagem é ter somente blocos em série, justamente pela inexistência de componentes redundantes. Não esquecendo o serviço *LiveVideo*, explicado adiante, que utiliza uma CTMC para sua modelagem e não possui redundância.

$$A_{Arquitetura1} = A_C \times A_{ST} \times A_{WF} \times A_F \times A_N \times A_{LV} \tag{4.1}$$

A partir da arquitetura básica, alguns mecanismos de redundância são acrescentados

nos componentes. Para evitar a repetição dos submodelos nas figuras das arquiteturas com redundância, optou-se por exibir o primeiro nível do modelo. Na Figura 4.2, que representa a arquitetura 2, três componentes são replicados, presentes na Equação 4.2: a A_C e o A_{LV} com blocos de operação *2-out-of-3*, significa que das três câmeras ou serviços *LiveVideo*, pelo menos duas devem funcionar para o subsistema esteja operacional. Sua fórmula se dá por α_1, mesmo sendo o exemplo da câmera, é o mesmo usado para outros componentes do tipo. A câmera em *2-out-of-3* é A_C mais três vezes ${A_C}^2$ vezes 1 menos A_C.

A A_{LV} é a disponibilidade resultada de uma CTMC com mecanismo de redundância, são duas redundâncias em um mesmo serviço. Por fim, uma redundância na conexão de rede com a Internet é obtida com a adição da tecnologia 3G com o *WiFi* por meio de blocos paralelos. Este é representado por β_1 e significa que a disponibilidade é (1 - (1 A_{WF}) $\times$ (1 - A_{3G})).

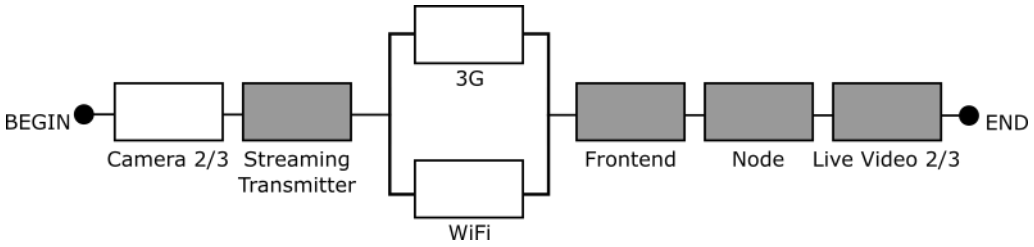


Figura 4.2: Modelo RBD do arquitetura 2 da arquitetura básica com a estratégia de redundância 1.

$$A_{Arquitetura2} = \alpha_1 \times (A_{ST}) \times (1 - (1 - A_{WF}) \times (1 - A_{3G})) \times (A_F) \times (A_N) \times \beta_1 \quad (4.2)$$

onde,
$\alpha_1 = ({A_C}^3 + (3 \times {A_C}^2 \times (1 - A_C))$
$\beta_1 = ({A_{LV}}^3 + (3 \times {A_{LV}}^2 \times (1 - A_{LV}))$

Na Figura 4.3, da arquitetura 2, existe uma diferenciação com a arquitetura 2 que é a redundância no componente A_N que também usará o bloco de operação *2-out-of-3*. Conforme a Equação 4.3 equivalente a Figura 4.3. Esta arquitetura é considerada a de maior número de redundâncias, totalizando cincos.

$$A_{Arquitetura3} = \alpha_1 \times (A_{ST}) \times \beta_1 \times (A_F) \times \gamma_1 \times \alpha_2 \quad (4.3)$$

onde,
$\alpha_1 = ({A_C}^3 + (3 \times {A_C}^2 \times (1 - A_C))$
$\beta_1 = (1 - (1 - A_{WF}) \times (1 - A_{3G}))$

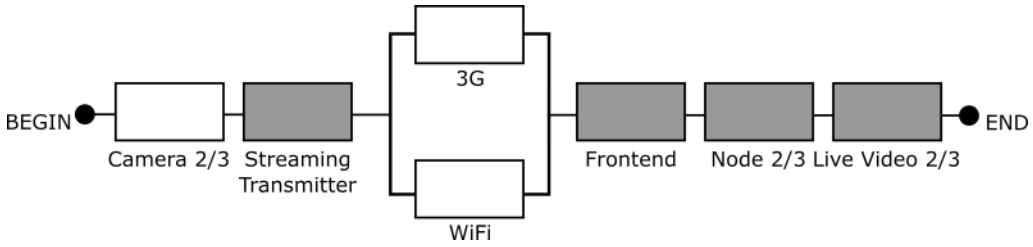


Figura 4.3: Modelo RBD da arquitetura 3 com a estratégia de redundância 2.

$$\gamma_1 = ({A_N}^3 + (3 \times {A_N}^2 \times (1 - A_N))$$
$$\alpha_2 = ({A_{LV}}^3 + (3 \times {A_{LV}}^2 \times (1 - A_{LV}))$$

Os três subsistemas RBDs representados por blocos de cinza na Figura 4.1 (*Transmissão Transmitter*, *Node* e *Frontend*) contêm três componentes de série, representando o hardware (HW), o sistema operacional (SO) e uma aplicação que varia de acordo com o subsistema específico. A *Streaming Transmitter* inclui um aplicativo de transmissão de vídeo (T); Frontend possui um serviço de gerenciamento da nuvem (MT), e por último, o *Node* inclui um virtualizador de máquinas virtual (KVM). As Equações 4.4, 4.5 e 4.6 permitem o cálculo da disponibilidade dos subsistemas *ST*, *F* e *N*, respectivamente. E, como todos os blocos são em série a disponibilidade é obtida com a multiplicação entre si das disponibilidades destes componentes.

$$A_{ST} = A_{HW} \times A_{SO} \times A_T \tag{4.4}$$

$$A_F = A_{HW} \times A_{SO} \times A_{MT} \tag{4.5}$$

$$A_N = A_{HW} \times A_{SO} \times A_{KVM} \tag{4.6}$$

4.1.2 Modelos CTMCs

Como descrito nas especificações dos componentes no Capítulo 3, o subsistema *LV* é composto por uma *VM* e os softwares *Apache* e *Ffmpeg*. Optou-se por utilizar uma CTMC em sua modelagem para verificar duas relações comportamentais entre os componentes. Primeiro verificar se o *Apache* e o *Ffmepg* estão sob o funcionamento de uma *VM*. Caso essa mude seu estado para inativo, resulta em ambos os sistemas fiquem inativos e devem ser recuperados os três componentes. A segunda é a independência do *Apache* em relação ao *Ffmpeg* e vice versa, porque uma falha no *Apache* deixará o componente indisponível, porém não causa danos ao funcionamento do *Ffmpeg*. O mesmo acontece com a falha do *Ffmpeg* que não interfere no

Apache.

Tabela 4.1: Estados do subsistema LiveVideo sem redundância.

Estado	Descrição
UUU	O Sistema está Ativo
UUD	O Sistema está Inativo, o FFmpeg está Inativo
UDU	O Sistema está Inativo, o Apache está Inativo
UDD	O Sistema está Inativo, o FFmpeg e Apache estão Inativo
DDD	O Sistema está Inativo, todos os componentes estão Inativo

O modelo CTMC para o subsistema *LV* sem redundância é ilustrado na Figura 4.4. Seus estados estão listados e descritos na Tabela 4.1 e suas taxas descritas na Tabela 4.2. Foi utilizada a notação *XXX*, onde $X \in \{U, D\}$ representa a condição de cada componente: U para ativo, e D para inativo. A primeira caractere representa a VM, a segunda o Apache e a última o Ffmpeg. O estado *UUU* (sombreado) indica que o sistema está disponível, uma vez que todos os três componentes estão operacionais. Todos os outros estados, *UUD*, *UDU*, *UDD* e *DDD* representam que o subsistema *LV* encontra-se com algum problema de funcionamento e está interrompido.

Tabela 4.2: Taxas do subsistema LiveVideo sem redundância.

Taxa	Descrição
λ_A	Falha do Apache
λ_F	Falha do Ffmpeg
λ_V	Falha do VM
μ_A	Reparo do Apache
μ_F	Reparo do Ffmpeg
μ_V	Reparo do VM

Iniciando com o estado *UUU*, ao ocorrer falha em FFmpeg, com taxa de λ_F, causa a indisponibilidade do serviço, indicado pelo estado *UUD*. A reparação da aplicação FFmpeg ocorre com a taxa de μ_F, retornando para o estado *UUU*. Enquanto esteja no estado *UUU* falhas podem ocorrer no Apache, ou na VM, representada pelas transições com taxas λ_A e λ_V, respectivamente. Quando ocorre no Apache o estado *UDU* é alcançado. Podendo ser reparado com a taxa de μ_A e o modelo atinge o estado *UUU*.

Quando há falhas simultânea de ambos (Apache e FFmpeg) o estado é *UDD* e uma reparação de qualquer deles se torna possível, disparando uma transição para *UDU* ou *UUD*. Falhas podem ocorrer antes que o sistema esteja completamente reparado, isso acontece quando a VM falha (com a taxa de λ_V) e o estado de *DDD* é alcançado, não importando o estado de

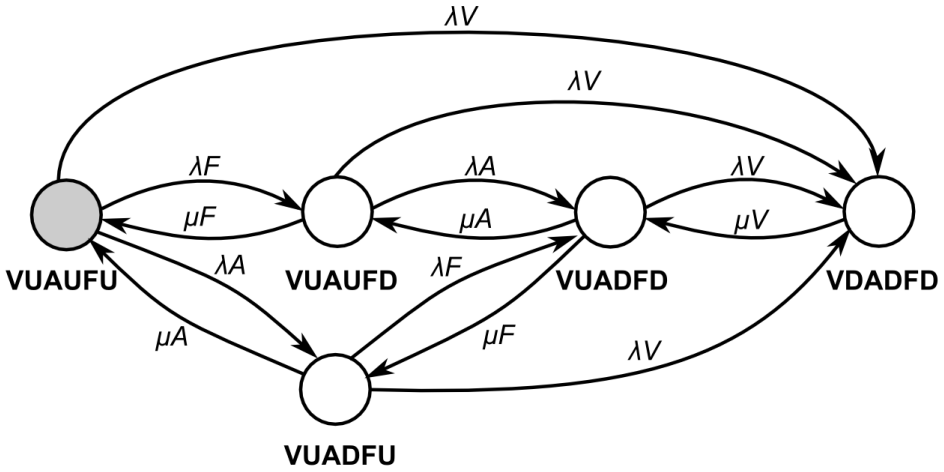


Figura 4.4: CTMC do subsistema LiveVideo sem redundância.

origem, porque *Apache* e *FFmpeg* são dependentes da VM. Por essa dependência os estados *DUU*, *DDU* e *DUD* não são alcançados nesta CTMC.

Para obtenção das fórmulas fechadas das CTMCs foram utilizadas duas ferramentas, a Mercury (SILVA et al., 2012; CALLOU et al., 2013) usadas para modelagem de todos os modelos apresentados neste Capítulo e a ferramenta Mathematica[1]. Com a ferramenta Mercury, após o término da modelagem foi possível exportar o modelo formatado para o próprio Mathematica e então gerada a fórmula fechada obtida para disponibilidade do subsistema LV, Equação (4.7). Pelo tamanho da expressão foi necessário realizar redução em conjuntos menores.

$$A_{LV} = \frac{\mu_A \times \mu_F \times \alpha_1 + 2\lambda_V + \mu_F \times \mu_V}{\beta_1 \times \lambda_F + \lambda_V + \mu_F \times \alpha_1 + \lambda_V + \mu_F \times \lambda_V + \mu_V} \tag{4.7}$$

onde,

$\alpha_1 = \lambda_A + \lambda_F + \mu_A$

$\beta_1 = \lambda_A + \lambda_V + \mu_A$

A Figura 4.5 descreve o serviço *LiveVideo* em que usa redundância *"Warm-Standby"* utilizando um modelo CTMC. Esta redundância do serviço é através de uma VM de reposição configurada de forma idêntica a VM primária (inclusive com Apache e Ffmpeg). Foi utilizada a notação *XXX_XXX*, onde $X \in \{U, D, W\}$ representa a condição de cada componente: U para ativo, D para inativo e W para espera. O primeiro caractere representa a VM, a segunda o Apache e a terceira o Ffmpeg. Do quarto caractere até a final, segue a mesma sequência dos caracteres iniciais, porém, elas compõem a VM reserva. Os estados em cinza (*UUU_WWW* e *UDD_UUU*) representam o sistema disponível. Todavia, os estados restantes simbolizam o sistema com mau

[1] www.wolfram.com/mathematica

funcionamento, são eles: *UUD_WWW*, *UDU_WWW*, *UDD_WWW*, *DDD_UUD*, *DDD_UDU*, *DDD_UDD* e *DDD_DDD*. Todos os estados são descritos na Tabela 4.3 e as taxas foram as mesma da CTMC sem redundância.

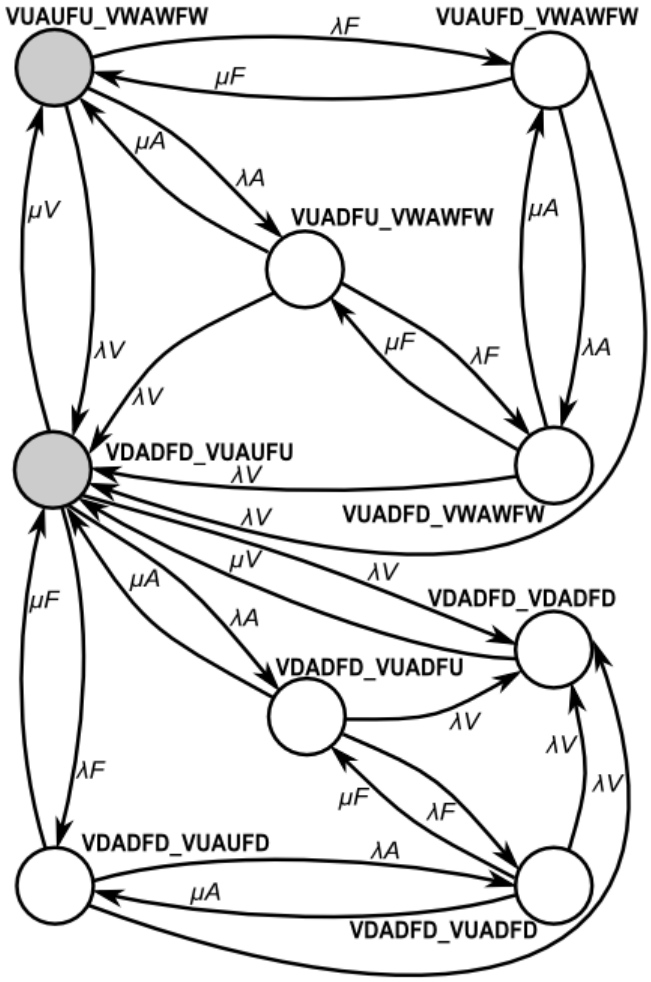


Figura 4.5: Cadeia de Markov do LiveVideo com Redundância

O modelo atinge o estado *UDU_WWW* quando Apache falha, ou *UUD_ VWAWFW* quando FFmpeg falha e outras falhas podem acontecer durante a reparação de cada um desses componentes. Quando ocorrem falhas no Apache ou FFmpeg, o reparo acontece com as taxas de μ_A e μ_F, respectivamente. A VM secundária é ativada somente com a falha da VM principal (por meio da transição λ_V), atingindo o estado *DDD_UUU*. Um reparo pode ser realizado na VM que falhou e assim o sistema retorna ao estado inicial, colocando a VM secundária na condição de "Warm-Standby" novamente. No entanto, Apache e FFmpeg também pode falhar na VM secundária após sua ativação, atingindo os estados *DDD_UDU* e *DDD_UUD*, respectivamente. Quando a VM secundária falhar enquanto ativada, o sistema atinge o estado de *DDD_DDD*, com

Tabela 4.3: Estados do subsistemas do *LiveVideo* com redundância.

Estado	Descrição
UUU_WWW	O Sistema está Ativo
UUD_WWW	O Sistema está Inativo, o FFmpeg está Inativo
UDU_WWW	O Sistema está Inativo, o Apache está Inativo
UDD_WWW	O Sistema está Inativo, o FFmpeg e Apache está Inativo
DDD_UUU	O Sistema está Ativo, a VM secundária Ativa e VM principal Inativa
DDD_UUD	O Sistema está Inativo, o FFmpeg está Inativo
DDD_UDU	O Sistema está Inativo, o Apache está Inativo
DDD_UDD	O Sistema está Inativo, o FFmpeg e Apache está Inativo
DDD_DDD	O Sistema está Inativo

todos os componentes para inativo.

Por esta CTMC possui as mesmas relações de dependência entre os componentes, os estados *DUU_WWW*, *DUD_WWW*, *DDU_WWW*, *DDD_DUU*, *DDD_DUD*, *DDD_DDU* são inatingíveis neste modelo. Não é representada a falha da segunda VM no modelo durante a VM principal esteja em funcionamento, porque se entende e considera a redundância "Warm-Standby" em nível do *Node* que instancia essa VM secundária.

A Equação 4.8 representa a expressão da fórmula fechada para esse modelo.

$$A_{LV} = \frac{(\alpha_1\alpha_2 + \lambda_A\alpha_3\mu_F + \gamma_4\alpha_4\mu_V(\gamma_1 + \alpha_1\beta_1\gamma_2 + \lambda_A\gamma_3\mu_V + \gamma_4(\alpha_6 + \alpha_4\alpha_5)))}{((\lambda_A + \alpha_3)\beta_1(\lambda_A + \beta_4)(\gamma_1\alpha_5 + \lambda_A({\lambda_V}^2\beta_1\delta_1 + \gamma_4(\beta_3\delta_2)))} \quad (4.8)$$

onde,

$\alpha_1 = \lambda_A\lambda_V$

$\alpha_2 = \lambda_F + \lambda_V + \mu_A$

$\alpha_3 = \lambda_V + \mu_A$

$\alpha_4 = \lambda_V + \mu_F$

$\alpha_5 = \lambda_V + \mu_{VAF}$

$\alpha_6 = \lambda_F\lambda_V$

$\beta_1 = \lambda_F + \alpha_4$

$\beta_2 = \lambda_A\alpha_1$

$\beta_3 = \alpha_6\alpha_5 + \alpha_4$

$\beta_4 = \alpha_2 + \mu_F$

$\gamma_1 = \beta_2\beta_1$

$\gamma_2 = \lambda_F + 2\alpha_3 + \mu_F$

$\gamma_3 = \alpha_6 + \alpha_3\alpha_4$

$\gamma_4 = \alpha_3\beta_4$

$$\delta_1 = \lambda_F + 2\alpha_3 + \alpha_4\beta_1\gamma_2\mu_V + \gamma_3{\mu_V}^2$$
$$\delta_2 = {\lambda_V}^2 + \lambda_V\mu_V + {\mu_V}^2$$

4.1.3 Validação

Ao lidar com modelos analíticos, geralmente é importante verificar se o comportamento do modelo da Figura 3.3 da Seção 3.2.1 do capítulo anterior, se o modelo não tem evidências para ser rejeitado como um modelo que representa o ambiente de execução (GOEL, 1985). Então, um experimento de injeção de falhas foi realizado, onde modelo e ambiente de execução foram submetidos às mesmas condições de entrada. Por conseguinte, realizou-se a validação do modelo através dos cálculos estatísticos propostos por (KEESEE, 1965), para calcular o intervalo de confiança da disponibilidade.

A validação de modelos de desempenho é realizada por meio da comparação dos resultados do modelo com as medidas extraídas a partir de um ambiente real (LAVENBERG, 1983). Validação de modelos de dependabilidade pode ser mais difícil, pois as falhas reais são imprevisíveis e podem levar um longo tempo para se manifestar (frequentemente mais de um ano (SCHROEDER; GIBSON, 2007)).

Este obstáculo é superado pelo emprego de técnicas de injeção de falhas (ARLAT et al., 1993). Tais técnicas empregam um software ou hardware para gerar elemento de falhas nos componentes do sistema de uma forma controlada. A taxa de falha de um componente pode ser acelerado, a fim de reduzir o tempo total do experimento (CHAN, 2004; VACARO; WEBER, 2006). Um sistema de monitoramento é geralmente incorporado ao lado de um *testbed* de injeção de falhas para observar como o sistema reage na presença de falhas.

Essa arquitetura foi baseada na plataforma de nuvem privada OpenNebula 3.6, Os *nodes* da nuvem utilizam o KVM[2] (versão 1.0) como *hypervisor*, implantada em um ambiente real composto por quatro máquinas físicas (apresentadas na Tabela 4.4), um roteador e uma câmera do tipo AXIS M10[3] (Figura 4.7).

Para aumentar o grau de confiança dos resultados obtidos a partir do estudo de caso, um processo de validação foi realizado no modelo. O processo de injeção de falhas foi baseado em (SOUZA et al., 2013), que emprega uma ferramenta de software (Eucabomber) para avaliar a disponibilidade de um ambiente de nuvem. Este trabalho utilizou para a validação o fluxo de trabalho proposto por (MELO, 2014) e é apresentado na Figura 4.6:

- **Conceber o modelo** – seleção do modelo a ser validado de acordo com o ambiente estudado;

[2]http://www.linux-kvm.org/
[3]http://www.axis.com/sk/en/products/axis-m1031-w

Tabela 4.4: Componentes do ambiente de testes do experimento de Injeção de Falhas.

Nome	Descrição	Processador	RAM	SO
Frontend	Gerente da nuvem	Intel Core i3 4 núcleos de 3.10 GHz	4 GB	12.04 Ubuntu Server 12.04 (kernel 3.2.0-23)
Node	Nó executor da nuvem	Intel Core i3 4 núcleos de 3.10 GHz	4 GB	12.04 Ubuntu Server 12.04 (kernel 3.2.0-23)
Stresser	Monitor da nuvem	Intel Core 2 Quad 4 núcleos de 2.66Gh	4 GB	12.04 Ubuntu Desktop 12.10 (kernel 3.5.0-36))
Streaming Transmitter	Gerenciar câmeras	Intel Core i5 4 núcleos de 2.50Gh	4 GB	12.04 Ubuntu Desktop 12.10 (kernel 3.5.0-36))

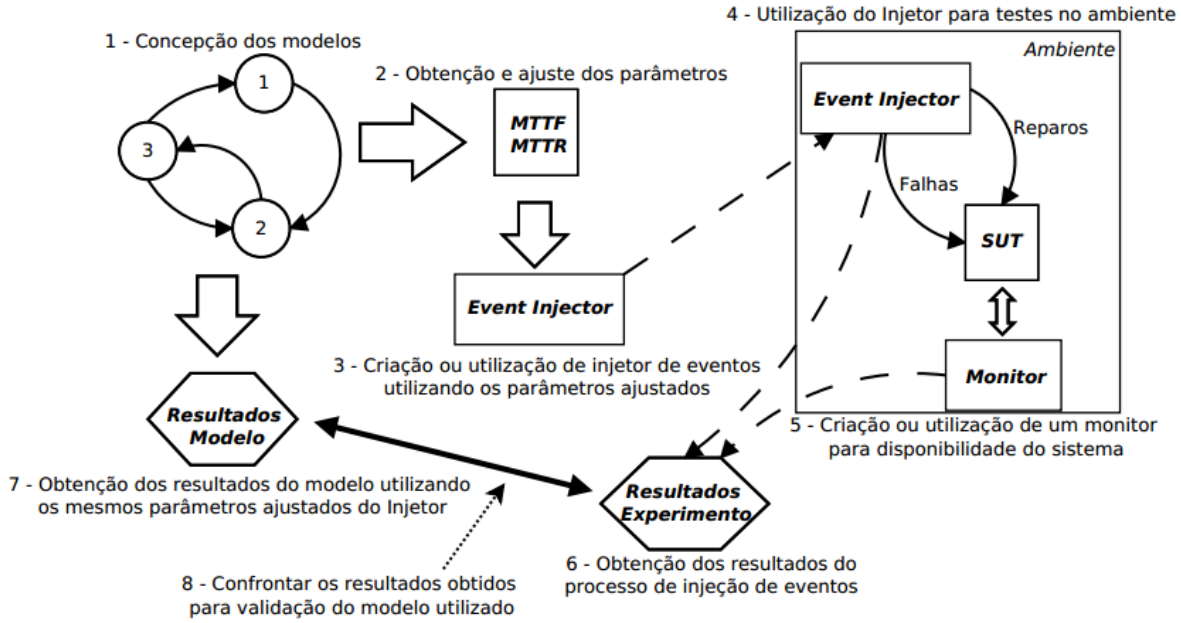


Figura 4.6: Fluxo de trabalho para validação dos modelos.

- **Obter os parâmetros de entrada** – o *MTTF* e *MTTR* são determinados através de medições ou encontrados em relevante literatura. Nesta pesquisa, os tempos de falhas foram ajustados de acordo com um fator de aceleração de 400;

- **Desenvolver scripts para injeção de falhas** – Os *scripts* foram baseado no Algoritmo 1, durante o tempo que o componente estiver ativo é gerado um tempo exponencial da taxa de falha dele (a ferramenta R[4] foi utilizada nesta geração), então é injetada a falha e os dados de ordem da falha, tempo de falha e horário da falha são coletados. Sabendo que ele não está ativo, então entra o processo reparo que também precisa da geração de um tempo

[4]www.r-project.org

Figura 4.7: Câmera AXIS utilizada no estudo

exponencial, agora para reparo. Por fim, os mesmo dados serão coletados para os reparos. Em paralelo outro script é responsável por monitorar a disponibilidade do sistema.

Estes *scripts* foram elaborados a partir do *Shell Script*, que estão ao final deste trabalho nos Apêndices A, B,C e D, que representam os scripts de injeção de falhas e reparos dos módulos cliente, *frontend*, *node* e *livevideo*. Em mesma ordem, os Apêndices E, F,G e H são usados para monitorar estes módulos. Os *scripts* são explicados a seguir.

O ambiente foi supervisionado durante um período de 3 semanas. Com interrupções porque a validação foi fragmentada em quatro módulos. O módulo denominado **cliente** engloba as câmeras, *streaming transmitter* e WiFi. Para injetar a falha e o reparo neste módulo utilizou-se o protocolo de Protocolo de Fluxo Contínuo em Tempo Real (Real-Time Streaming Protocol - RTSP). Este protocolo é utilizado no transporte de streaming de vídeo das câmeras do modelo Axis, usada no experimento. Para monitorá-lo foi utilizada a ferramenta *nmap* para escutar a porta 554.

O módulo Frontend foi composto por um sistema operacional, hardware e o sistema *oneadmin*, responsável por gerenciar a nuvem. Para provocar sua falha aplicou o comando *shutdown* que desliga o computador. Sendo o reparo realizado por meio do comando *wakeonlan* que consegui ligar um computador que esteja conectado na mesma rede do computador que solicitou o comando. O monitoramento foi realizado pelo comando *ping* que retorna “up” se o computador conectou a rede ou “down” caso o computador não esteja conectado, nesse caso, é entendido que o computador está desligado.

Este processo de injeção de falha, reparo e monitoramento, explicado no módulo anterior foi o mesmo utilizado no módulo *Node*. Por fim, o módulo *livevideo* formado pela Máquina Virtual, Apache e Ffmpeg, foi escolhido o serviço do Apache para ora utilizar o comando *stop* e simular uma falha e na recuperação usou-se o comando *start* do serviço. Para monitorá-lo se fez o uso da ferramenta *nmap* para escutar a porta 80.

- **Executar o experimento de injeção de falhas** – os *scripts* de injeção de falhas são executados

de acordo com o comportamento do sistema real com os valores dos parâmetros obtidos a partir da segunda etapa. Os scripts de injeção e monitoramento foram executadas em um computador paralelo a estrutura da nuvem OpenNebula e estava conectado na mesma rede, este foi denominado *Stresser*;

- **Monitorar o ambiente** – um monitor observa o estado de funcionamento do sistema e seus componentes, enquanto os experimentos estão sendo executados. Um relatório chamado *log* é produzido, que inclui os dados captados pelo monitor, ou seja, os eventos de falha e reparo ao longo do tempo. Este relatório é essencial para a comparação dos resultados dos experimentos com os resultados do modelo;

- **Sintetizar os resultados do experimento** – os resultados servirão de base para o cálculo do intervalo de confiança de disponibilidade;

Algorithm 1 Injetor de Eventos

while true **do**
  if ComponenteEstaAtivo **then**
    $TempoFalha = TempoExponencial(TaxaDeFalhaComponente)$;
    $Esperar(TempoFalha)$;
    $InjetarFalhaComponente()$;
    $ColetarDados(OrdFalha, TempoFalha, timestamp)$
  else
    $TempoReparo = TempoExponencial(TaxaDeReparoComponente)$;
    $Esperar(TempoReparo)$;
    $InjetarReparoComponente()$;
    $ColetarDados(OrdReparo, TempoReparo, timestamp)$
end if
  end while

- **Avaliar o modelo** – a disponibilidade em estado estacionário do modelo é calculado usando os parâmetros de entrada descritos na segunda etapa. Estes resultados são comparados com o intervalo de confiança para a disponibilidade (KEESEE, 1965) obtido anteriormente;

- **Confrontar os resultados para validação** – os resultados do modelo são analisados para verificar se eles estão dentro dos limites do intervalo de confiança. Se assim for, então o modelo é considerado coerente com o sistema real.

Na Tabela 4.5, encontram-se os valores dos parâmetros de entrada para fórmula que gera o intervalo de confiança da disponibilidade. Nesse trabalho, considera-se o intervalo de confiança de 95%.

Tabela 4.5: Parâmetros de Entrada para o Método de Keese.

Parâmetros	**Valores**
Tempo total de falha do *Frontend*	$12885,71s$
Tempo total de reparo do *Frontend*	$9603,35s$
Quantidade de Falhas e Reparos da *Frontend*	50
Tempo total de falha do *Node*	$16208,51s$
Tempo total de reparo do *Node*	$8113,50s$
Quantidade de Falhas e Reparos da *Node*	50
Tempo total de falha do Cliente	$3270000,28s$
Tempo total de reparo do Cliente	$1072241,22s$
Quantidade de Falhas e Reparos da Cliente	50
Tempo total de falha do *LiveVideo*	$6940,14s$
Tempo total de reparo do *LiveVideo*	$16973,71s$
Quantidade de Falhas e Reparos da *LiveVideo*	50

A fim de simplificar o processo, o modelo foi dividido em quatro partes: i) da câmera, *Streaming Transmitter* e WiFi; ii) *Frontend*; iii) Node; iv) *LiveVideo*. A Tabela 4.6 mostra a disponibilidade para as partes distintas avaliadas pelo modelo, seguida por um intervalo de confiança obtida por meio dos experimentos. Os resultados de cada parte, bem como a disponibilidade geral do sistema 0,03552, revela que tivemos seus resultados dentro dos seus respectivos intervalos de confiança, (0,01422, 0,07987).

Tabela 4.6: Validação das disponibilidades

Componente	**Disponibilidade do Modelo**	**95% C.I.**
Cliente	$0,35622$	$(0,24150;0,42525)$
FrontEnd	$0,53889$	$(0,47496;0,66553)$
Node	$0,63857$	$(0,57389;0,74764)$
LiveVideo	$0,28982$	$(0,21609;0,37747)$
Disponibilidade geral	$0,03552$	$(0,01422;0,07987)$

Os baixos valores das disponibilidades nos resultados são creditados ao fator de redução de 400 que aplicado para aceleração das falhas. Portanto, o modelo proposto não tem evidências para ser refutado como modelo que representa o comportamento do ambiente real, desta forma, demonstra ser um modelo confiável para ser utilizado na avaliação de disponibilidade do ambiente, bem como a proposição de novas arquiteturas.

4.2 Modelagem das Arquiteturas do Sistema Empresarial

Nesta Seção, as diferenças na modelagem entre a arquitetura básica do Sistema Doméstico e Empresarial são detalhadas, alterando ou acrescentando novas restrições do seu funcionamento. Como ilustrado na Figura 4.8, este modelo é o suficiente para encontrar a disponibilidade de um dos andares e foi orientada por uma visita a um empresa de pequeno porte da cidade de Caruaru que por motivos de privacidade será chamada de Empresa X. Nele foi realizada a junção dos componentes Câmera e *LiveVideo*, formando um componente *k-out-of-n* denominado CLV. O subsistema *LiveVideo*, nesta segunda arquitetura é modelado por RBD, não mais por uma CTMC. Com ênfase à explicação da SPN, proposta para a política de manutenção da câmera reserva com a redundância *cold-standby*.

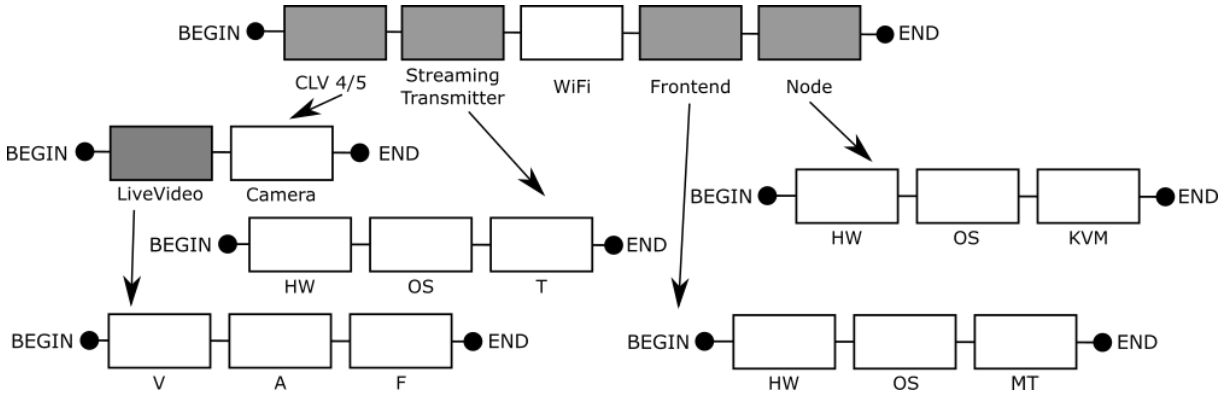


Figura 4.8: Modelo RBD da Arquitetura de *VSaaS* da Empresa X

Para a disponibilidade completa da arquitetura empresarial. Considera-se a multiplicação da disponibilidade de todos os andares considerando as câmeras e o *LiveVideo*. Pela visita na empresa, constatou-se a necessidade suficiente de um *Streaming Transmitter* e uma conexão de rede (optada pela WiFi) para toda a estrutura da empresa. A sequência continua com um *Frontend* e um *Node* no ambiente da nuvem, o *Node* por sua condição de instanciar no máximo 10 câmeras terá sua disponibilidade potencializada ao cubo para comportar as 25 câmeras iniciais. Conforme resumida na Equação 4.9, utilizando notações e outras equações da arquitetura básica.

$$A_{ArquiteturaEmpresa} = (A_{CLV})^5 \times A_{ST} \times A_{WF} \times A_F \times (A_N)^3 \tag{4.9}$$

Na Figura 4.9 é exibida a apresentação hierárquica heterogênea do bloco CLV do modelo RBD, da Figura 4.8, por meio da técnica de modelagem SPN.

A modelagem é composta por dois componentes, os serviços *LiveVideos* e as Câmeras, onde uma delas poderá ser substituída. O modelo SPN é constituído de 31 lugares, sendo 10

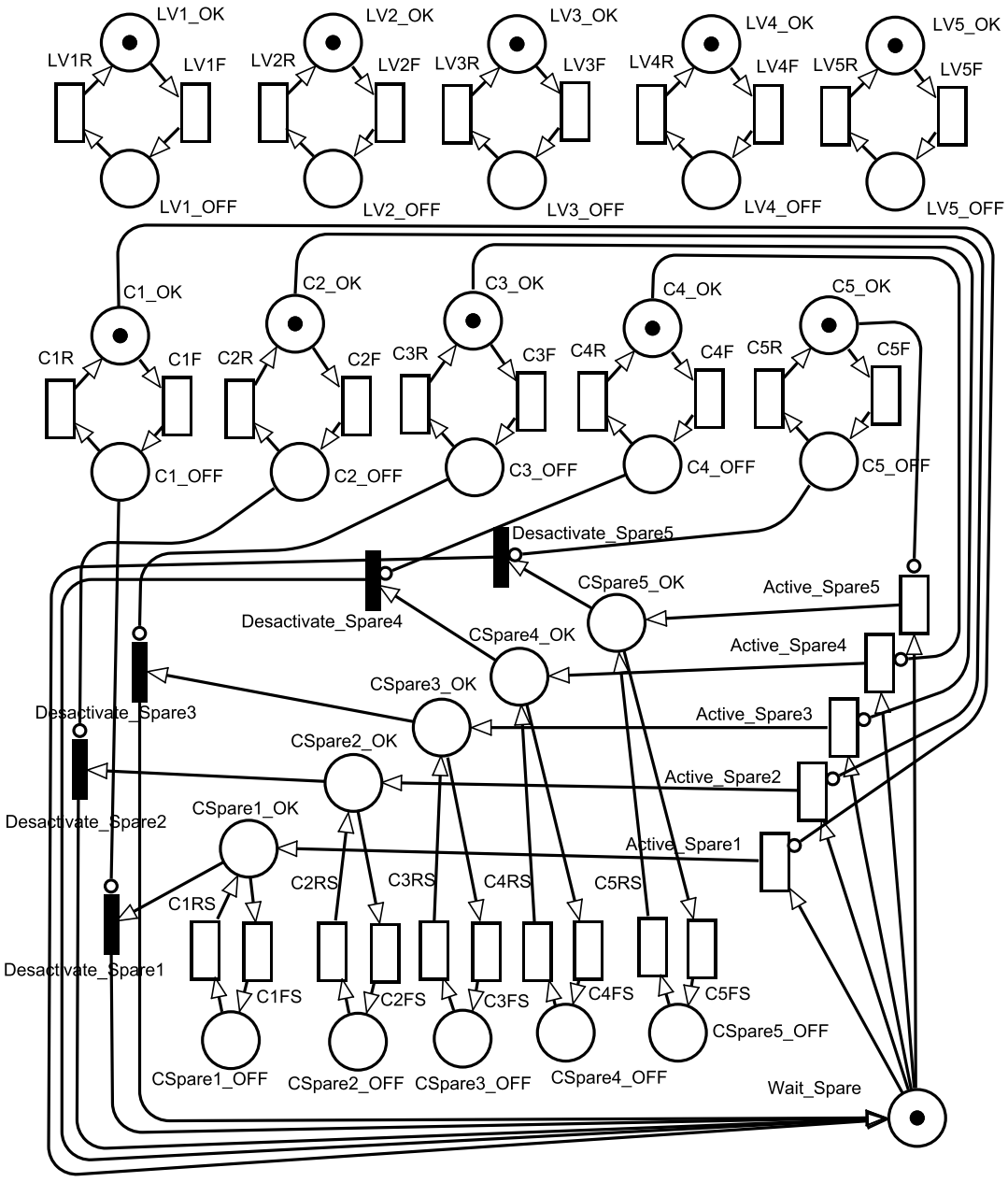


Figura 4.9: Rede de Petri dos componentes Câmera e *LiveVideo* relacionados com redundância.

para o serviço *LiveVideo*: LV1_OK, LV2_OK, LV3_OK, LV4_OK e LV5_OK que representam os estados operacionais e LV1_OFF, LV2_OFF, LV3_OFF, LV4_OFF, LV5_OFF são os seus estados falhos. Os atributos relacionados a cada transição do *LiveVideo* descrito neste trabalho são apresentados na Tabela 4.7. Nas transições temporizadas LV1F, LV2F, LV3F, LV4F e LV5F que ocasionam as falhas nos respectivos serviços, deixando-os em lugares OFF, seguem uma distribuição exponencial (exp), bem como as transições temporizadas de reparo LV1R,

LV2R, LV3R, LV4R e LV5R que retorno o *token* para os lugares OK. Sobre todas as transições temporizadas modeladas nesta SPN utilizou-se a semântica *single-server*.

Tabela 4.7: Atributos das transições dos *LiveVideos* do modelo SPN *cold-standby*.

Transição	Tipo	Tempo ou peso
$LV1F$	exp	$MTTF$
$LV1R$	exp	$MTTR$
$LV2F$	exp	$MTTF$
$LV2R$	exp	$MTTR$
$LV3R$	exp	$MTTR$
$LV3F$	exp	$MTTF$
$LV4R$	exp	$MTTR$
$LV4F$	exp	$MTTF$
$LV5R$	exp	$MTTR$
$LV5R$	exp	$MTTR$

No componente câmera são 21 lugares na SPN, sendo cinco lugares que sinalizam as câmeras em estado operacional, C1_OK, C2_OK, C3_OK, C4_OK e C5_OK. Há outros cinco lugares que assumem que o estado operacional de uma câmera, ao substituí-la em uma das salas ou do corredor, alcançando os lugares CSpare1_OK, CSpare2_OK, CSpare3_OK, CSpare4_OK e CSpare5_OK, a partir do disparo de uma das transições de falha da câmera, entre elas: C1F, C2F, C3F, C4F e C5F. Todas as transições inerentes ao processo de falha ou reparo das câmeras reservas e principais são exibidas na Tabela 4.8. Estes lugares continuam em processo de substituição até que um *token* continue nos lugares falhos das câmeras principais, C1_OFF, C2_OFF, C3_OFF, C4_OFF e C5_OFF, porque com os disparos das transições de reparo das câmeras principais, C1R, C2R, C3R, C4R e C5R passarão para os lugares operacionais.

Porém, considerando os lugares das câmeras reservas estando ativos, isso não garante que eles sejam livres de falhas, ao disparo de qualquer transição de falha, C1FS, C2FS, C3FS, C4FS e C5FS. Isso resultará na indisponibilidade destas câmeras e os lugares CSpare1_OFF, CSpare2_OFF, CSpare3_OFF, CSpare4_OFF e CSpare5_OFF, poderão ser alcançados, que por sua vez, conseguirão retornar aos lugares em OK, por meio do disparo de alguma das transições de reparo, C1RS, C2RS, C3RS, C4RS e C5RS. Todas essas transições temporizadas também são exponencialmente distribuídas, alternando entre MTTF e MTTR das câmeras.

O lugar Wait_Spare possui a quantidade de recursos disponíveis, câmeras reservas. Este lugar é o centro do mecanismo de redundância *cold-standby*. Ele é responsável pela substituição das câmeras principais que, por ventura, falhar. O Wait_Spare pode ser acionado com o disparo de cinco transições temporizadas (com tempo de um hora), Active_C1, Active_Spare2, Active_Spare3, Active_Spare4 e Active_Spare5, apresentadas na Tabela 4.9. São essas transições

Tabela 4.8: Atributos das transições das câmeras do modelo SPN *cold-standby*.

Transição	Tipo	Tempo ou peso
$C1F$ e $C1FS$	exp	$MTTF$
$C1R$ e $C1RS$	exp	$MTTR$
$C2F$ e $C2FS$	exp	$MTTF$
$C2R$ e $C2RS$	exp	$MTTR$
$C3F$ e $C3FS$	exp	$MTTF$
$C3R$ e $C3RS$	exp	$MTTR$
$C4F$ e $C4FS$	exp	$MTTF$
$C4R$ e $C4RS$	exp	$MTTR$
$C5F$ e $C5FS$	exp	$MTTF$
$C5R$ e $C5RS$	exp	$MTTR$

que indicam qual câmera principal está falha e permitirá sua substituição, com a condição que haja recurso disponível (representado graficamente pelo número de *tokens* no lugar Wait_Spare). Essas transições estão interligadas com os lugares das câmeras em funcionamento por meio de um arco inibidor que não permite que essas transições sejam disparadas com *tokens* nesses lugares.

Supondo que a câmera 1 falhe e o disparo da transição Active_Spare1 seja concretizado por haver um *token* disponível no lugar Wait_Spare, desta forma a câmera reserva é acionada e representada com um *token* no lugar CSpare1_OK. Na hipótese da câmera dois também falhar neste momento ela não será substituída pela ausência de *token* disponível no lugar Wait_Spare.

Outra câmera em falha só será substituída novamente depois que seja realizado o disparo da transição de reparo da câmera principal. Quando isso ocorrer uma das transições, Desactive_Spare1, Desactive_Spare2, Desactive_Spare3, Desactive_Spare4 e Desactive_Spare5 terá de ser disparada instantaneamente, por ela ser uma transição imediata. Estas transições são interligadas com respectivos lugares de câmera principal em OFF por meio de um arco inibidor, sendo habilitada com a ausência de *token* nesses lugares e com a existência do *token* nos lugares CSpare_OK. Com isso ela estará devolvendo o *token* para o lugar Wait_Spare e permitindo a substituição de outras câmeras falhas. Para aumentar o número de câmeras reservas na SPN deverá ser aumentado o número de *tokens* no lugar Wait_Spare.

A primeira obrigatoriedade para a SPN estar disponível, é considerar a dependência entre *LiveVideo* e câmera, por exemplo, se a câmera 1 estiver ativa, representada pelo estado C1_OK, com o serviço *LiveVideo* 1 também ativo, representado por LV1_OK. Com a ressalva de quando a câmera principal falha, o serviço da mesma forma poderá ser relacionado com a câmera reserva CSpare1_OK. Esta obrigatoriedade é aplicada em todas as outras câmeras com seus respectivos serviços.

Tabela 4.9: Atributos das transições de disparo da Câmera Reserva do modelo SPN *cold-standby*.

Transição	Tipo	Tempo ou peso
*Active_Spare*1	*exp*	1 hora
*Desactive_Spare*1	*im*	1
*Active_Spare*2	*exp*	1 hora
*Desactive_Spare*2	*im*	1
*Active_Spare*3	*exp*	1 hora
*Desactive_Spare*3	*im*	1
*Active_Spare*4	*exp*	1 hora
*Desactive_Spare*4	*im*	1
*Active_Spare*5	*exp*	1 hora
*Desactive_Spare*5	*im*	1

A seguir é apresentada a Expressão Probabilística utilizada para obter a disponibilidade da SPN, considerando a obrigatoriedade explicada no parágrafo anterior. A estratégia de disponibilidade adotada que será explicada no Capítulo 5, na Seção 5.2. Esta determina que para o andar estar disponível, quatro das cinco câmeras deverão estar funcionando:

P{(((#C1_OK=1 OR #CSpare1_OK=1) AND #LV1_OK=1) AND ((#C2_OK=1 OR #CSpare2_OK=1) AND #LV2_OK=1) AND ((#C3_OK=1 OR #CSpare3_OK=1) AND #LV3_OK=1) AND ((#C4_OK=1 OR #CSpare4_OK=1) AND #LV4_OK=1)) OR (((#C5_OK=1 OR #CSpare5_OK=1) AND #LV5_OK=1) AND ((#C2_OK=1 OR #CSpare2_OK=1) AND #LV2_OK=1) AND ((#C3_OK=1 OR #CSpare3_OK=1) AND #LV3_OK=1) AND ((#C4_OK=1 OR #CSpare4_OK=1) AND #LV4_OK=1)) OR (((#C1_OK=1 OR #CSpare1_OK=1) AND #LV1_OK=1) AND ((#C5_OK=1 OR #CSpare5_OK=1) AND #LV5_OK=1)AND ((#C3_OK=1 OR #CSpare3_OK=1) AND #LV3_OK=1) AND ((#C4_OK=1 OR #CSpare4_OK=1) AND #LV4_OK=1)) OR (((#C1_OK=1 OR #CSpare1_OK=1) AND #LV1_OK=1) AND ((#C2_OK=1 OR #CSpare2_OK=1) AND #LV2_OK=1) AND ((#C5_OK=1 OR #CSpare5_OK=1) AND #LV5_OK=1) AND ((#C4_OK=1 OR #CSpare4_OK=1) AND #LV4_OK=1)) OR (((#C1_OK=1 OR #CSpare1_OK=1) AND #LV1_OK=1) AND ((#C2_OK=1 OR #CSpare2_OK=1) AND #LV2_OK=1) AND ((#C3_OK=1 OR #CSpare3_OK=1) AND #LV3_OK=1) AND ((#C5_OK=1 OR #CSpare5_OK=1) AND #LV5_OK=1)) OR (((#C1_OK=1 OR #CSpare1_OK=1) AND #LV1_OK=1) AND ((#C2_OK=1 OR #CSpare2_OK=1) AND #LV2_OK=1) AND ((#C3_OK=1 OR #CSpare3_OK=1) AND #LV3_OK=1) AND ((#C4_OK=1 OR #CSpare4_OK=1) AND #LV4_OK=1) AND((#C5_OK=1 OR #CSpare5_OK=1) AND #LV5_OK=1))}

Portanto, esse Capítulo apresentou os modelos e as fórmulas para obtenção da disponibilidade, estas sendo umas das principais contribuições desta Dissertação. Foi apresentado o

processo de validação que aumentou a credibilidade do modelo e possibilita a utilização dos *scripts* e da metodologia em outros modelos com ou sem o contexto de *VSaaS*.

5

Estudos de casos

"A nossa maior glória não reside no fato de nunca cairmos, mas sim em levantarmo-nos sempre depois de cada queda".

—CONFÚCIO

Neste capítulo, os resultados das métricas de disponibilidade e *downtime* são apresentados em dois estudos de casos oriundos das arquiteturas dos dois sistemas. Além desses resultados, os estudos de casos tem por objetivo utilizar técnicas que apoiem a decisão dos administradores de nuvens públicas ou privadas, com necessidade de alta disponibilidade na plataforma *VSaaS*.

5.1 Estudo de Caso 1 - Sistema Doméstico

Este primeiro estudo de caso compara a disponibilidade e o *downtime* anual entre as três arquiteturas *VSaaS*, propostos no Capítulo 3, na Seção 3.2.1. Cujos modelos analíticos RBDs são representados pelas Figuras 4.1, 4.2 e 4.3. A particularidade comportamental do subsistemas *LiveVideo* que foram modelados em CTMCs estão nas Figuras 4.4 e 4.5. Por fim, foi realizada uma análise de sensibilidade paramétrica no modelo da Figura 4.3 com objetivo de encontrar pontos críticos com maior impacto na disponibilidade.

Espera-se que os resultados sejam subsídios aos administradores na utilização de mecanismos de redundância para o aumento da disponibilidade dos serviços ofertado em suas empresas. Colaborando para o estabelecimento e cumprimento de SLAs, manutenção ou elevação da confiabilidade perante os clientes, evitar prejuízos financeiros enquanto o sistema estar indisponível ou pelas multas contratuais.

No decorrer da pesquisa, houve a necessidade de estabelecer algumas restrições, pois elas determinam de que forma as análises desse estudo de caso foram realizadas. As três políticas

restritivas foram:

- As câmeras foram reparadas ou substituídas no prazo de 24 horas;
- O mecanismo de redundância empregado aos componentes foi o *warm-standby*, isto é, eram ativados automaticamente após a detecção da falha, exceto as câmeras;
- A arquitetura estava disponível, se pelo menos duas das três câmeras estivessem funcionando.

A arquitetura 1 (Figura 4.1) foi extraída de um sistema com os seus requisitos mínimos para estar em funcionamento (ou seja, sem componentes paralelos e ausência de redundância). Na arquitetura 2 (Figura 4.2), semelhante à arquitetura 1, porém com a redundância dos componentes da Câmera, da conexão de rede e do serviço *LiveVideo*. A arquitetura 3 consistiu de um modelo RBD representado pela Figura 4.3 que, em relação à arquitetura 2, teve a inclusão da redundância no componente *Node*. Ressalte-se no componente *LiveVideo*, presente nas arquiteturas 2 e 3, houve a modelagem da redundância *"Warm-Standby"* por CTMC, exibido na Figura 4.5.

Todos os valores dos parâmetros foram baseados em artigos pesquisados e na documentação dos fabricantes. Não houve necessidade de obter esses parâmetros de entrada pelo método da medição. Os parâmetros do componente Câmera foram obtidos a partir das especificações da câmera AXIS. Contudo, nela não havia explicitamente o MTTF e MTTR, havendo somente MTBF (tempo médio entre falhas). Considerou-se então o MTBF como o MTTF da câmera. A restrição determina que a câmera fosse reparada em 24 horas e foi justamente criada para suprir a ausência deste informação, o MTTR. O reparo correspondeu a substituição da câmera e não conserta-la.

A Tabela 5.1, fornece tanto os parâmetros da Câmera, bem como todos os parâmetros utilizados neste estudo, os componentes *Streaming Transmitter*, *Node*, *Frontend* e *LiveVideo*, extraídos de (MATOS et al., 2012), (KIM; MACHIDA; TRIVEDI, 2009) e (DANTAS, 2012). Os parâmetros de entrada para as redes de comunicação, Wi-Fi e 3G foram provenientes de (D-LINK, 2012) (COOPER; FARRELL, 2007), respectivamente. A Equação 2.4 foi adotada a para a análise de disponibilidade e para a transformação dos MTTF e MTTR em taxa, notação usual de λ e μ de cada um dos componentes, na devida ordem.

Diante dos valores obtidos, foi possível realizar a análise de disponibilidade das três arquiteturas do sistema doméstico do serviço *VSaaS*. Os resultados dessa primeira análise do Estudo de Caso I, em termos de disponibilidade e downtime anual em horas, são apresentados na Tabela 5.2. Para ampliar e facilitar o entendimento, esses resultados são ilustrados nas Figuras 5.1 e 5.2, por meio de gráficos.

Como resultado, a disponibilidade encontrada para a arquitetura 1, sem redundância, foi de 99,503712%, correspondente a um *downtime* de 43,47482113 horas. Em seguida, a

Tabela 5.1: Parâmetros da Câmera, *WiFi*, conexão 3G, *Streaming Transmitter, Node, Frontend* e *LiveVideo*.

Parâmetros	**Valores** (h^{-1})
$\lambda_{HW}S,F,N$	$0,000114155$
$\mu_{HW}S,F,N$	$0,6$
$\lambda_{SO}S,F,N$	$0,000694444$
$\mu_{SO}S,F,N$	1
λ_T	$0,005714286$
μ_T	60
λ_{MT}	$0,001268392$
μ_{MT}	1
λ_{KVM}	$0,000347222$
μ_{KVM}	1
λ_A	$0,005714286$
μ_A	60
λ_F	$0,005714286$
μ_F	60
λ_V	$0,000345661$
μ_V	$1,875$
λ_{WF}	$0,00001$
μ_{WF}	$0,6$
λ_{3G}	$0,000012019$
μ_{3G}	$0,083333333$
λ_C	$0,00001$
μ_C	$0,04167$
$Active_Spare_{1,2,3,4,5}$	1

Tabela 5.2: Resultados da comparação entre os modelos.

Arquiteturas	**Disponibilidade**	***Downtime* h/ano**
1	$99,503712\%$	$43,474821$
2	$99,564588\%$	$38,142043$
3	$99,686835\%$	$27,433189$

arquitetura 2, a primeira com redundância, aumentou sua disponibilidade para 99,564588% e reduziu o *downtime* para 38,142043 horas. Na comparação entre essas arquiteturas (1 e 2) houve uma diferença de 5,3 horas. Ou seja, o período de aproximadamente dois dias que o serviço ficou sem funcionar na arquitetura 1 foi reduzido para um dia e meio na arquitetura 2. No caso, a arquitetura 2 reduziu o *downtime* em 12,26% em relação à primeira arquitetura.

Por fim, a arquitetura 3 obteve a maior disponibilidade e menor *downtime*, 99,686835% e 27,433189 horas, respectivamente. À medida que se compara a arquitetura 2 com a arquitetura 3 encontra-se a diminuição de 10,7 horas de interrupções, gerando um impacto na redução do

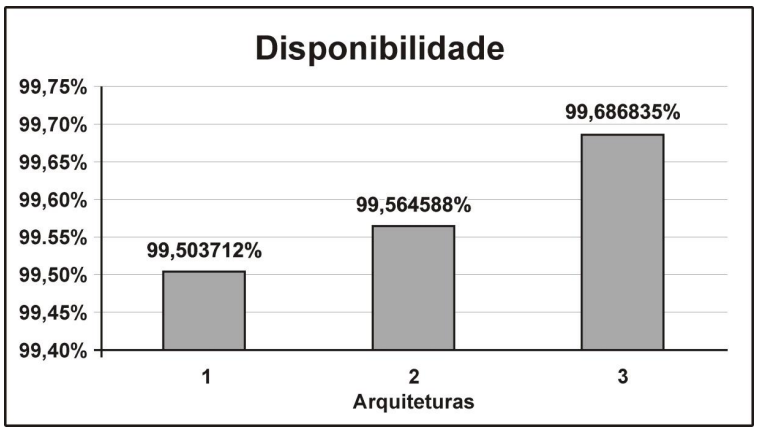


Figura 5.1: Disponibilidade das arquiteturas

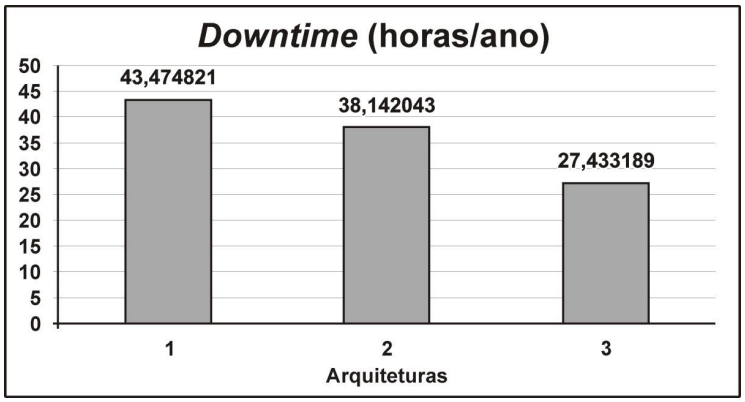


Figura 5.2: *downtime* das arquiteturas

downtime em percentagem de 28,07%. Estas arquiteturas diferenciam apenas com a redundância aplicada no componente *Node* da arquitetura 3. Por conseguinte, verificou-se a diferença de 16 horas de inatividade, mais que a metade de um dia, na comparação entre a arquitetura 3 e a arquitetura inicial. Esta diferença equivale a 36,89% de redução do *downtime* e significa o serviço indisponível em um dia e 3 horas.

Nos trabalhos pesquisados e nas empresas que ofertam o serviço de *VSaaS*, há poucas informações sobre o custo referente ao *downtime* desse serviço. Justificado por ser um valor complexo de ser estimado e genericamente utilizado para todas as áreas, pela particularidade do conteúdo ter um valor e/ou um significado diferente para cada cliente. Como forma de representar e compreender os prejuízos financeiros deste *downtime* nas 27,43 horas, considerou-se outro serviço de vídeo, o *pay-per-view*, segundo a (INTRANSA, 2010). A Equação 5.1 mostra como foi realizado o cálculo, as horas de interrupção $\times$ o custo por hora (*pay-per-view* $\bar{\$}$ 150.000,00 dólares), resultando um prejuízo de $ 4.114.500,00 dólares. Esse valor pode ser considerado exagerado, entretanto, imagine o serviço ofertado a um Museu com famosas obras de artes de

Pablo Picasso ou Leonardo da Vinci, certamente constará altíssimas cláusulas financeiras no contrato.

$$CustoDowntimeAnual_{pay-per-view} = 27,43 \times \$150.000,00 = \$4.114.500,00 \quad (5.1)$$

Embora os valores de *downtime* observados para os modelos que continham redundância demonstrarem melhorias significativas em relação a uma arquitetura inicial, os resultados também indicaram que há oportunidades de melhoria, já que o valor para a melhor arquitetura ainda corresponde a mais de um dia de interrupção durante o período de um ano.

5.1.1 Análise de Sensibilidade

A partir do resultado encontrado na arquitetura 3, de aproximadamente um dia de interrupção, optou-se por investigar possíveis melhorias sobre essa arquitetura. Considerando uma empresa de *VSaaS* com sua infraestrutura montada e com existência de redundância em algum componente (por exemplo, o *Frontend* na estrutura da nuvem e esta escolhida de maneiro intuitiva e/ou orientações de outros profissionais de TI). Todavia, a empresa pretende aumentar a disponibilidade da arquitetura atual fundamentada em alguma técnica matemática, por exemplo, uma análise de sensibilidade.

Como mencionado na Seção 2.5 do Capítulo 2, o método direto, baseado em derivadas parciais é a espinha dorsal de muitas técnicas de análise de sensibilidade. Usou-se tal técnica para começar a explorar os parâmetros do modelo, com o objetivo de identificar aqueles com impacto significativo sobre os resultados. O *ranking* de sensibilidade obtido através do cálculo de derivadas parciais também permite ignorar parâmetros, justificadamente, que têm menos impacto sobre as métricas de interesse.

Nesse caso, a análise de sensibilidade tornou-se uma das alternativas viáveis para a detecção de gargalos ou componentes chaves que aumentam a disponibilidade e, assim, incorporar novas melhorias. Todos os cálculos desta análise foram realizados com o suporte da ferramenta *Mathematica*. Por uma questão de concisão, as próprias expressões derivadas não foram apresentadas aqui pelo grande tamanho das expressões, sendo apresentado apenas os valores obtidos para os índices de sensibilidade.

Na Tabela 5.3, encontra-se o *ranking* de sensibilidade para os parâmetros considerados nesta análise. Foram computados pelas derivadas parciais das expressões da Equação 4.3, indicada na Seção 4.1, do Capítulo 4. O *ranking* de sensibilidade usou um índice escalado para remover as influências indesejáveis das unidades, porque parâmetros com ordens de magnitude

muito diferentes foram utilizados neste modelo. Os parâmetros tiveram seus índices modulados e organizados em ordem decrescente, conforme o parâmetro de maior impacto sobre a disponibilidade do *VSaaS*. Aqueles que possuem um sinal de menos (-) indica apenas uma relação inversa entre o parâmetro e a disponibilidade.

Tabela 5.3: *Ranking* de Sensibilidade.

Parâmetro	$\mathbf{S}^*(\mathbf{A})$
μ_{OSN}	$-5,596 \times 10^{-1}$
λ_{OSN}	$5,596 \times 10^{-1}$
μ_{KVM}	$-2,799 \times 10^{-1}$
λ_{KVM}	$2,799 \times 10^{-1}$
μ_{HWN}	$-1,534 \times 10^{-1}$
λ_{HWN}	$1,534 \times 10^{-1}$
μ_{MT}	$-1,266 \times 10^{-3}$
λ_{MT}	$1,266 \times 10^{-3}$
μ_{OSF}	$6,939 \times 10^{-4}$
λ_{OSS}	$-6,939 \times 10^{-4}$
λ_{OSF}	$-6,939 \times 10^{-4}$
μ_{OSS}	$6,939 \times 10^{-4}$
μ_{HWS}	$1,902 \times 10^{-4}$
λ_{HWF}	$-1,902 \times 10^{-4}$
λ_{HWS}	$-1,902 \times 10^{-4}$
μ_{HWF}	$1,902 \times 10^{-4}$
λ_{T}	$-9,522 \times 10^{-5}$
μ_{T}	$9,522 \times 10^{-5}$
λ_{C}	$-3,452 \times 10^{-7}$
μ_{C}	$3,452 \times 10^{-7}$
λ_{A}	$-1,087 \times 10^{-7}$
λ_{F}	$-1,087 \times 10^{-7}$
μ_{F}	$1,087 \times 10^{-7}$
μ_{A}	$1,087 \times 10^{-7}$
λ_{WF}	$-2,403 \times 10^{-9}$
μ_{WF}	$2,403 \times 10^{-9}$
μ_{3G}	$2,403 \times 10^{-9}$
λ_{3G}	$-2,403 \times 10^{-9}$
μ_{VAF}	$7,764 \times 10^{-11}$
λ_{V}	$-7,639 \times 10^{-11}$

Verificou-se que os seis parâmetros com maior impacto foram associados com o componente do *Node*: sistema operacional (λ_{OSN} e μ_{OSN}), *hypervisor* (λ_{KVM} e μ_{KVM}) e *hardware* (λ_{HWN} e μ_{HWN}). Um gerente de TI que precisa melhorar a sua disponibilidade do sistema, quando estiver operando um serviço semelhante a esta arquitetura, deve concentrar seus esforços

nesses parâmetros ou no componente inteiro.

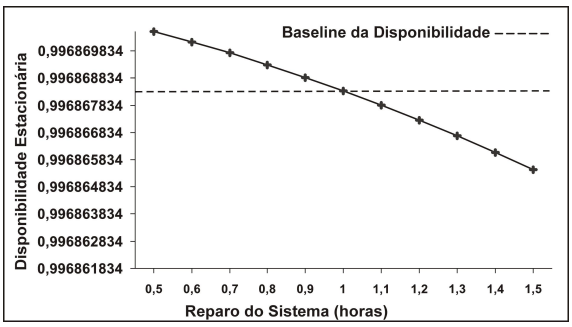
Na sequência, os parâmetros que também apresentam relevância estão relacionados aos componentes do *Frontend* e do *Streaming Transmitter*, com resultados numéricos similares, na seguinte ordem: μ_{MT}, λ_{MT}, μ_{OSF}, λ_{OSS}, λ_{OSF}, μ_{OSS}, μ_{HWS}, λ_{HWF}, λ_{HWS}, μ_{HWF}, λ_T e μ_T. Pela semelhança dos resultados, após o melhoramento do componente *Node*, qualquer um dos dois componentes (*Frontend* ou *Streaming Transmitter*) transformaram-se em bons componentes a serem melhorados.

A possibilidade de trabalhar e analisar parâmetro por parâmetro é uma das vantagens da abordagem de análise de sensibilidade utilizada neste estudo de caso. Que considerou nos cálculos os parâmetro até último nível dos componentes que compõe a arquitetura no processo de modelagem. Havendo possibilidade em uma visão mais detalhada, de não realizar a redundância em todo o componente, por exemplo, priorizando somente o sistema de gerenciamento da nuvem ou a mudança para um sistema operacional mais confiável, no caso com um MTTF maior ou MTTR menor. Existem outras abordagens menos custosas que verificam o impacto do componente inteiro, sem os seus respectivos detalhamentos.

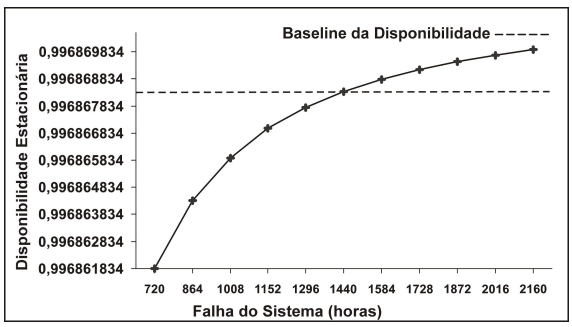
Em outros resultados da análise, observou-se baixo impacto da disponibilidade na taxa de falha e de reparo da câmera. Devido à restrição do tempo de reparo ser de 24 horas e pela aplicação da estratégia *2-out-of-3*, nesse componente. Os aplicativos que compreendem o subsistema *LiveVideo* (*VM*, *Apache*, *FFmpeg*) também possuem a estratégia redundância *2-out-of-3* e outra estratégia de redundância realizada por meio de uma CTMC, tornando-os com pouca relevância na disponibilidade, bem como os componentes das conexões de redes (*WiFi* e 3G). Por conta dessas redundâncias, os administradores não devem considerar esses componentes como prioritários na política aplicada para aumentar a disponibilidade.

As Figuras 5.3 e 5.4 enfatizam os resultados da análise de sensibilidade paramétrica, usando a variação de um parâmetro no tempo. É importante observar que os gráficos são organizados de acordo com a ordem do *ranking* de derivadas parciais. Os valores dos parâmetros foram mudados em passos médios de 10% e manteve-se com uma linha tracejada a disponibilidade estacionária da arquitetura 3.

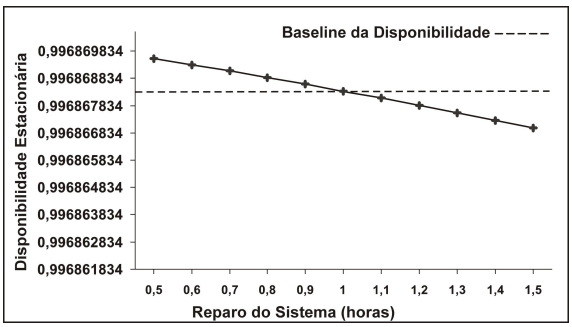
A Figura 5.3, apresenta os parâmetros com maior impacto. Por exemplo, MTTR do sistema operacional do *Node*, visualmente, é compreendido no seu distanciamento do valor original de 1 hora, o que causou um impacto maior da disponibilidade, tanto no aumento máximo, considerando o tempo de 30 minutos, como na redução mínima com valor de 1 hora e 30 minutos. Comportamento visual semelhante ao MTTF do sistema operacional, desse mesmo componente. No caso do MTTR e MTTF do KVM, percebe-se uma redução mediana em comparação ao impacto dos componentes anteriores, mesmo sabendo que eles são o terceiro e quarto no *ranking*



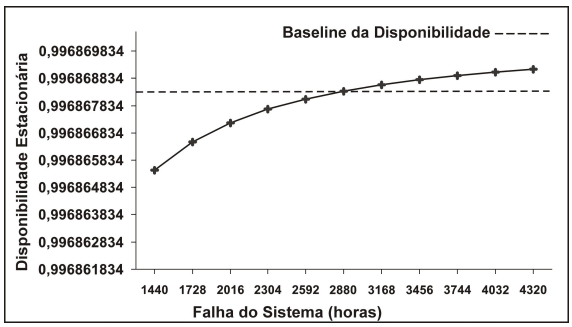
(a) MTTR do SO - *Node* (horas).



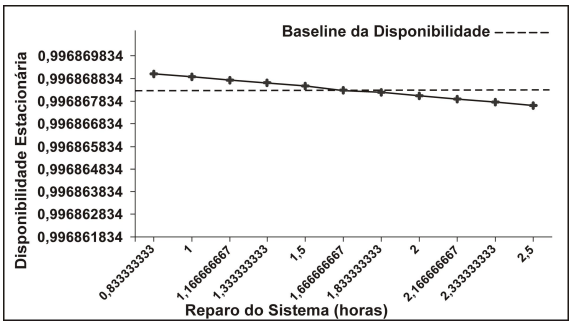
(b) MTTF do SO - *Node* (horas).



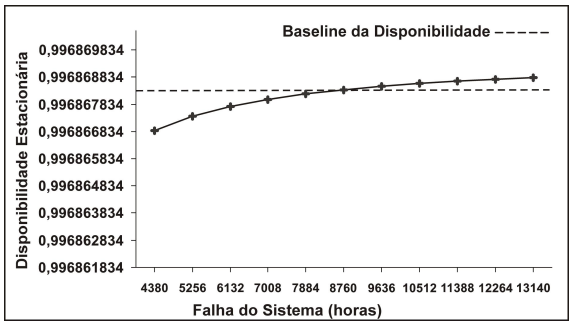
(c) MTTR do KVM (horas).



(d) MTTF do KVM (horas).



(e) MTTR do HW - *Node* (horas).



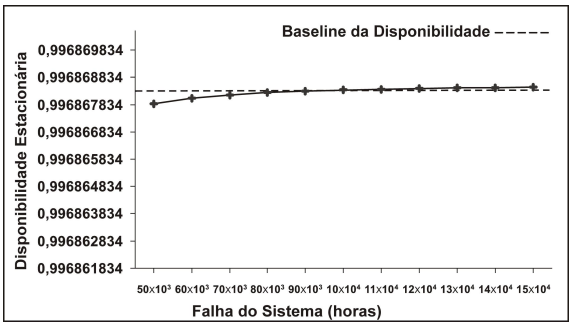
(f) MTTF do HW - *Node* (horas).

Figura 5.3: Análise de Sensibilidade - Gráfico com parâmetros de maior impacto

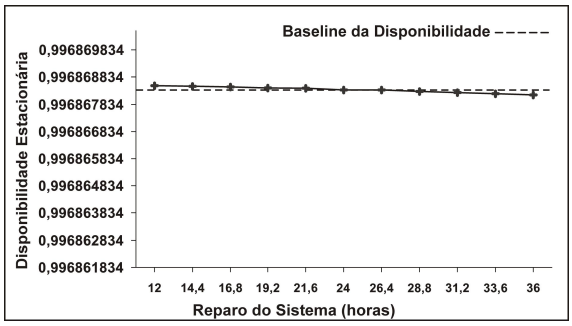
de sensibilidade. Observa-se o intrigante comportamento em relação aos MTTF e MTTR do hardware no *Node* que suas curvaturas no gráfico aproximam-se dos parâmetros de menor impacto.

Na Figura 5.4 apresenta os parâmetros com menor impacto. Fenômeno contrário do ocorrido no parágrafo anterior, entre os parâmetro com menor impacto, sobressai o MTTR do 3G, o MTTR e MTTF da câmera. Eles tiveram suas curvaturas praticamente iguais ou muito próximas da linha tracejada da disponibilidade.

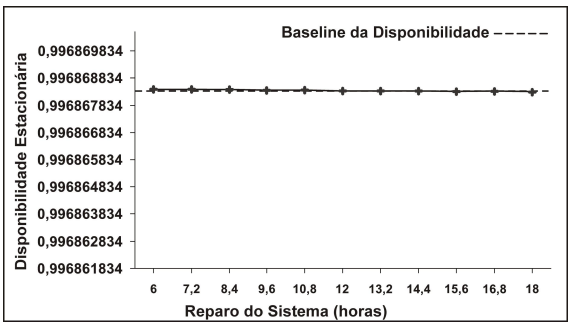
Nesta seção, foi proposta através da análise de sensibilidade, algumas melhorias na arquitetura 3. Dependendo das necessidades e dos recursos financeiros disponíveis é que o



(a) MTTF da câmera (horas)



(b) MTTR da câmera (horas)



(c) MTTR do 3G (horas)

Figura 5.4: Análise de Sensibilidade - Gráfico com parâmetros de menor impacto

administrador ou gestor de TI escolherá as mudanças. Com isso, finaliza-se o primeiro estudo de caso.

5.2 Estudo de Caso 2 - Sistema Empresarial

Os resultados desse segundo estudo de caso refere-se à arquitetura empresarial do serviço de *VSaaS*, exposta na Figura 3.6, cujos modelos analíticos RBDs são representados pela Figura 4.8. Para representar o subsistema da política de manutenção da câmera e o serviço LiveVideo foi modelado com SPN e exibido na Figura 4.9. Para a proposição das novas arquiteturas com redundância nesse estudo, utilizou-se a técnica de *Availability Importance*. Para em seguida realizar a análise de disponibilidade em todas as arquiteturas. Em paralelo uma análise de custo foi aplicada as arquiteturas com o objetivo de comparar o *downtime* e o custo de cada arquitetura. Esta comparação foi guiada por uma técnica de cluster, a distância euclidiana, para encontrar uma arquitetura eficaz que combinem justamente o custo com o tempo que este ficará indisponível no ano.

Um dos principais problemas para obter altas disponibilidades em sistemas em nuvem estão na escassez de recursos computacionais e na limitação dos recursos financeiros. Por

isso, a importância de utilizar técnicas que contribuirão para decisões coerentes em relação à redundância, que estas permitam o estreitamento do número de arquiteturas possíveis e sugestão de boas arquiteturas baseados na prioridade traçada. Este estudo de caso, pontualmente, destina-se a colaborar para essas decisões.

Os cálculos para avaliação destes modelos foram baseados na Equação 4.9. Com o acréscimo dos resultados gerados pela rede de Petri 4.9. Inicialmente houve a definição de estratégia abordada para a disponibilidade do serviço *LiveVideo* com a câmera. Explicadas a seguir:

- **Primeira estratégia** – um andar estava disponível se pelo menos quatro das cinco câmeras estivessem funcionando com seus respectivos serviços de *LiveVideo*;
- **Segunda estratégia** – um andar estava disponível se pelo menos três das cinco câmeras estivessem funcionando com seus respectivos serviços de *LiveVideo*;
- **Terceira estratégia** – um andar estava disponível se pelo menos duas das cinco câmeras estivessem funcionando com seus respectivos serviços de LiveVideo.

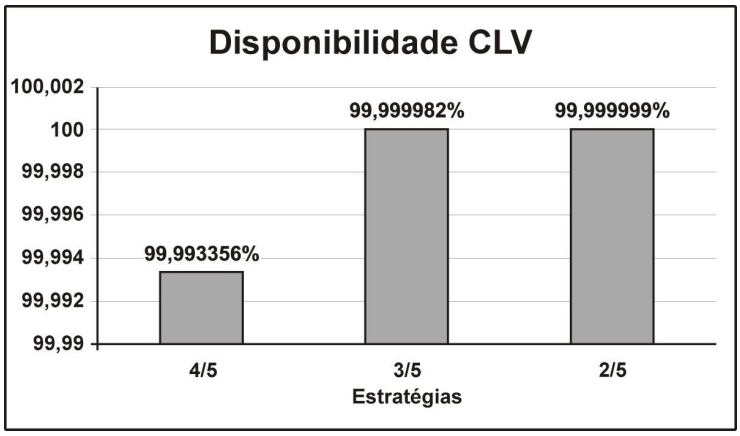


Figura 5.5: Disponibilidade do RBD da Câmera com o serviço LiveVideo

Para obter os resultados de disponibilidade dessas estratégias que envolve estes dois componentes (câmera e *LiveVideo*), modelou-se um bloco em RBD com a estrutura *k-out-of-n*, chamado de CLV, conforme a Figura 4.8, do Capítulo 4. Esses resultados são exibidos na Figura 5.5 e percebeu-se uma aproximação dos 100% de disponibilidade entre as estratégias. A primeira estratégia atingiu 99,993356%, a segunda 99,999982% e a terceira 99,999999%. Para ampliar a compreensão em um contexto geral, calculou-se as métricas de disponibilidades e *downtime* do modelo da Empresa X respeitando justamente essas estratégias e só então foi escolhida a mais adequada.

Tabela 5.4: Estratégias adotadas

Estratégia	Disponibilidade	Downtime h/ano
4/5 Câmeras	99,270138%	63,935857
3/5 Câmeras	99,303036%	61,053977
2/5 Câmeras	99,303122%	61,046509

Através da Tabela 5.4 e nas Figuras 5.6 e 5.7, para disponibilidade e *downtime*, respectivamente. Os resultados observados da disponibilidade na primeira estratégia foi de 99,270138%, enquanto na segunda 99,303036% e na última 99,303122%. O período de interrupção anual aproximou-se das 60 horas em todas as estratégias. Percebeu-se na primeira estratégia algo em torno de 64 horas que comparada com as outras duas estratégias houve um pequena diferença de 3 horas, porque os *downtime*s foram de 61 horas.

Porém, como explicado na Seção 3.2.2, na apresentação da arquitetura da empresa, a primeira estratégia possui maior segurança em relação as demais. Esse estudo adotou a primeira estratégia para continuar a análise de disponibilidade do modelo. Pela conveniência dessa estratégia ser mais rígida e segura que as demais. Essa rigidez é pelo fato que o andar é composto de quatro salas e um corredor, com câmeras em todos os compartimentos. Existe a concepção que mesmo com a falha da câmera do corredor, todas os documentos e materiais das salas estarão sendo monitorados. Caso a falha ocorra em uma das salas é possível identificar as pessoas que tiveram acesso a elas, por meio da câmera instalada no corredor.

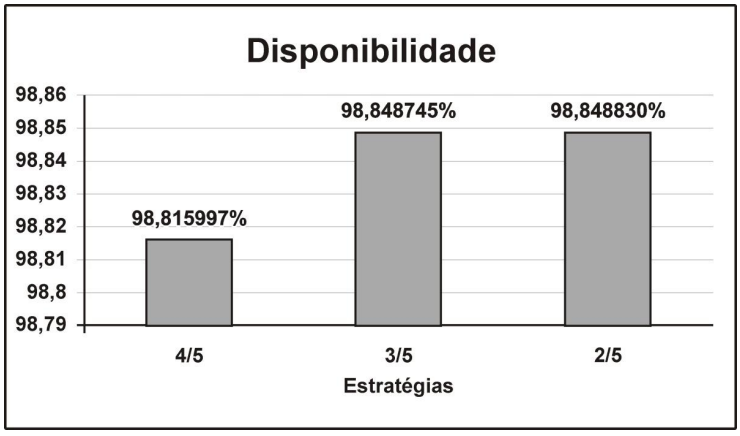


Figura 5.6: Disponibilidade das estratégias analisadas

5.2.1 *Availability Importance*

Nesta subseção, usou-se análise de sensibilidade *Availability Importance* (explicada na Seção 2.6, do Capítulo 2) para determinar quais componentes seriam replicados e auxiliar na

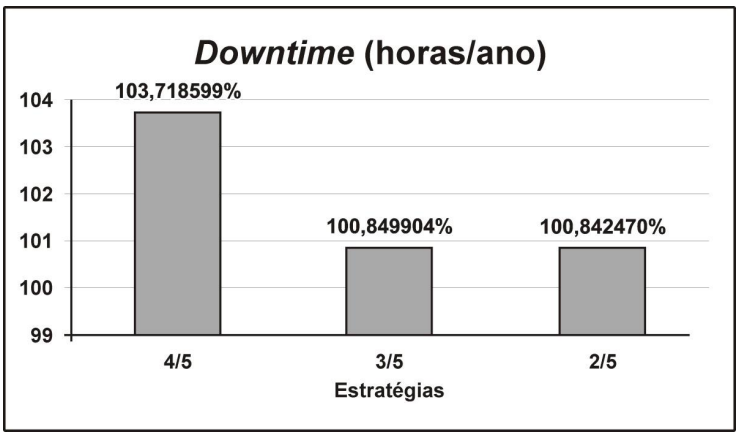


Figura 5.7: *downtime* das estratégias analisadas

geração das arquiteturas. Diferentemente da análise de sensibilidade paramétrica realizada no estudo de caso anterior, esta abordagem considera apenas os componentes do modelo RBD. Essa análise foi auxiliada pela ferramenta Mercury conforme o modelo da Figura 4.8. Esses resultados estão apresentados na Figura 5.8 e na Tabela 5.5. Com índice 1 na CameraLiveVideo, do *Frontend* 0,999564, do *Node* 0,998644, do *Streaming Transmitter* 0,998393 e da WiFi 0,997432.

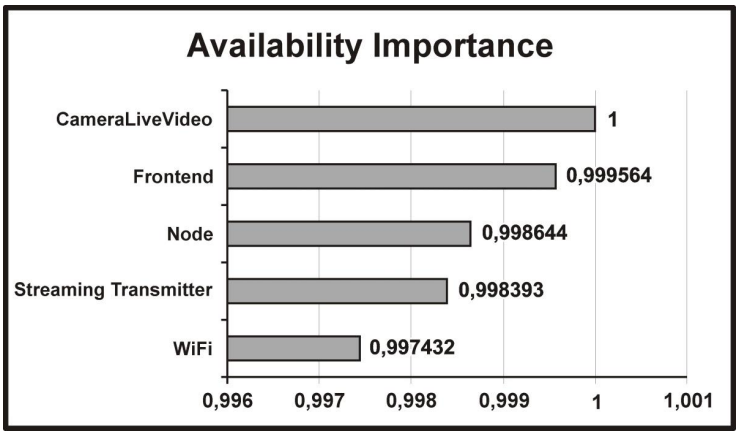


Figura 5.8: Resultado da *Availability Importance*

Tabela 5.5: Resultado da *Availability Importance*

Availability Importance	
CameraLiveVideo	1
Frontend	0,999564
Node	0,998644
Streaming Transmitter	0,998393
WiFi	0,997432

Notou-se que os três primeiros componentes com maior impacto na disponibilidade foram *CameraLiveVideo, Frontend* e *Node*. Desta forma, as arquitetura criadas a partir desta receberam redundância nestes componentes, comparados na próxima seção.

5.2.2 Custo versus *Downtime*

Uma das contribuições desse estudo de caso é a análise de custo relacionado com o *downtime*. Para isto, as arquiteturas e os valores dos componentes são apresentado. Para depois os resultados serem analisados, considerando a disponibilidade, o *downtime* e o custo. Com propósito de determinar arquiteturas eficientes técnicas de distância euclidiana foi utilizada sobre as arquiteturas geradas.

Na Tabela 5.6, cada arquitetura foi descrita e definida a quantidade de vezes que os componentes foram replicados, do total de 18 arquiteturas. Iniciando-se com a arquitetura A1 que é ausente de redundância. Na arquitetura A8 temos redundância única nos componentes *Node* e *Frontend*. No caso da arquitetura A10 com redundância na *Node* e a disponibilização de uma câmera reserva. Houve dez arquiteturas que foram adicionados redundância no *Frontend* (A2, A3, A8, A9, A11, A12, A13, A15, A17 e A18), dez no *Node* (A4, A5, A8, A10, A11, A13, A14, A15, A16 e A18) e por final, dez arquiteturas com redundância na câmera (A6, A7, A9, A10, A11, A12, A14, A16, A17 e A18). A arquitetura com o máximo de redundância é o A18 com três componentes de *Node* e de *Frontend* e duas câmeras reservas, logicamente, tendo o maior custo e disponibilidade. Porém, nem sempre a empresa está com recursos financeiros disponíveis e procura qual o melhor custo benefício.

A segunda etapa definiu os valores de cada componente. Sendo uma etapa crucial no processo de avaliação, porque a eficiência e confiabilidade dos resultados estão relacionados a esses dados de entrada. Então, quanto mais próximo dos valores do mercado mais esta comparação faz sentido. Os valores encontrados estão postados na Tabela 5.7. Os preços dos componentes foram pesquisados nos sites dos fabricantes e serviu para obtenção do custo final de cada arquitetura. O componente com maior custo é o computador que foi *Frontend* e *Node*, por R$ 2.529,00 e seguido pelo custo da câmera de R$ 879,51. O terceiro maior valor foi R$ 848,00, referente ao computador com configuração simples que foi usado para o *Streaming Transmitter*. Por fim, o equipamento de conexão da internet custou R$ 189,05. Não houve gastos com os *softwares* utilizados nesta pesquisa por serem *open source*.

Depois da obtenção dos custos e a escolha das arquiteturas começou a fase de calcular as métricas de disponibilidade e o *downtime* anual. Estes resultados estão expressos na Tabela 5.8 e para uma melhor compreensão, foram gerados os gráficos para estas duas métricas, Figuras 5.9, 5.10 e 5.11 para os custos. As arquiteturas encontradas com maior disponibilidade foram A8

Tabela 5.6: Apresentação das Arquiteturas

Arquiteturas	Descrição da redundância
A1	Sem redundância
A2	2 *Frontend*
A3	3 *Frontend*
A4	2 *Node*
A5	3 *Node*
A6	1 Câmera reserva
A7	2 Câmeras reservas
A8	2 *Frontend* e 2 *Node*
A9	2 *Frontend* e 1 Câmera reserva
A10	2 *Node* e 1 Câmera reserva
A11	2 *Frontend*, 2 *Node* e 1 Câmera reserva
A12	3 *Frontend* e 1 Câmera reserva
A13	3 *Frontend* e 2 *Node*
A14	3 *Node* e 1 Câmera reserva
A15	3 *Node* e 2 *Frontend*
A16	2 Câmeras reservas e 2 *Node*
A17	2 Câmeras reservas e 2 *Frontend*
A18	3 *Frontend*, 3 *Node* e 2 Câmeras reservas

Tabela 5.7: Descrição e valor dos componentes

Marca/Modelo	Componente	Descrição
DELL - PowerEdge T110II	HD	500 GB
	Memória	4 GB
	CPU	Intel Xeron - 3.10 GHz
Custo(R$)		2.529,00
PC Lenovo	HD	500 GB
	Memória	2 GB
	CPU	Intel Celeron - 2.6 GHz
Custo(R$)		848,00
Câmera AXIS	M1011	45s VGA (640 x 480 pixels)
Custo(R$)		878,51
Roteador Wireless Intelbras		300Mbps
Custo(R$)		189,05

(99,851341%), A15 (99,851794%), A13 (99,851802%), A11 (99,883969%) e A18 (99,884936%), essa última era esperada porque possui mais componentes replicados. Caso a empresa busque apenas maior disponibilidade, esses resultados seriam subsídios para a tomada de decisão dos gestores.

No contexto do *downtime*, as arquiteturas com períodos de interrupção maiores foram

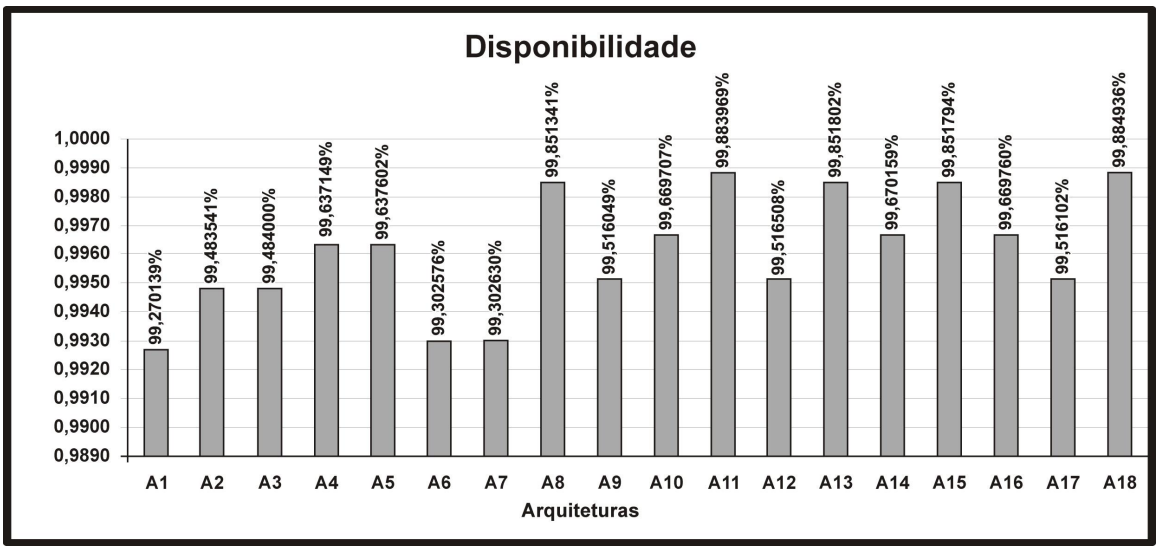


Figura 5.9: Disponibilidade das arquiteturas da empresa X

selecionados com valor acima de 40 horas, entre elas: A12 (42,353915), A17 (42,389451), A9 (42,394115), A3 (45,201584), A2 (45,241771), A7 (61,08), A6 (61,089656) e A1 (63,935857). Elas possuem uma interrupção entre dois dias até dois dias e meio. Porém, a diferença de horas entre essas arquiteturas com o menor *downtime* foi considerável. Por exemplo, comparando a arquitetura A18 (10,7 horas) com o A1, chega a 53,86 horas, o que significa dois dias e 5 horas de interrupções a mais. Foram arquiteturas extremas, todavia, comparando a arquitetura A18 com A12, esta diferença foi de 32 horas (um dia e 8 horas). Salientando, que a arquitetura A12 não está contida na lista de arquiteturas de custo mais elevado, apresentadas a seguir. Então, analisando superficialmente, a arquitetura A12 tem sua participação na redução do *downtime*, com um custo menor. As vezes a arquitetura não aparenta ter um *downtime* alto, porém, o custo não compensa em relação a disponibilidade. Aconteceu com a arquitetura A5 que possui disponibilidade razoável com um custo elevado, de forma tal que ao ser comparada com a arquitetura A8 tem custo inferior. Apresentando *downtime* menor de 18 horas de diferença de 18 horas em relação à A5 e R$ 5.058,00 de investimento a menos.

Na Tabela 5.8, os custos de cada arquitetura foram calculados, a Figura 5.11 as ilustram, entre as de custo elevado estão: A11 (R$ 47.624,35), A5(R$ 48.289,80), A16(R$ 49.487,9), A15(R$ 50.818,80), A14(R$ 52.682,35) e A18(R$ 62.132,90). Realizando um comparativo entre o percentual de aumento do custo com base na arquitetura inicial (A1 = R$ 33.115,80), tem-se um aumento de investimento na arquitetura A11 de 43,81% para a redução do *downtime* de 84,10%, na arquitetura A5 o aumento é de 45,82% para redução de 50,35%, na A16 esse elevação do custo é de 49,43% para uma diminuição de 54,76% do *downtime*, na arquitetura A15 aumentando 53,45% do custo obtém redução de 79,69%, na arquitetura A14 ao aumentar

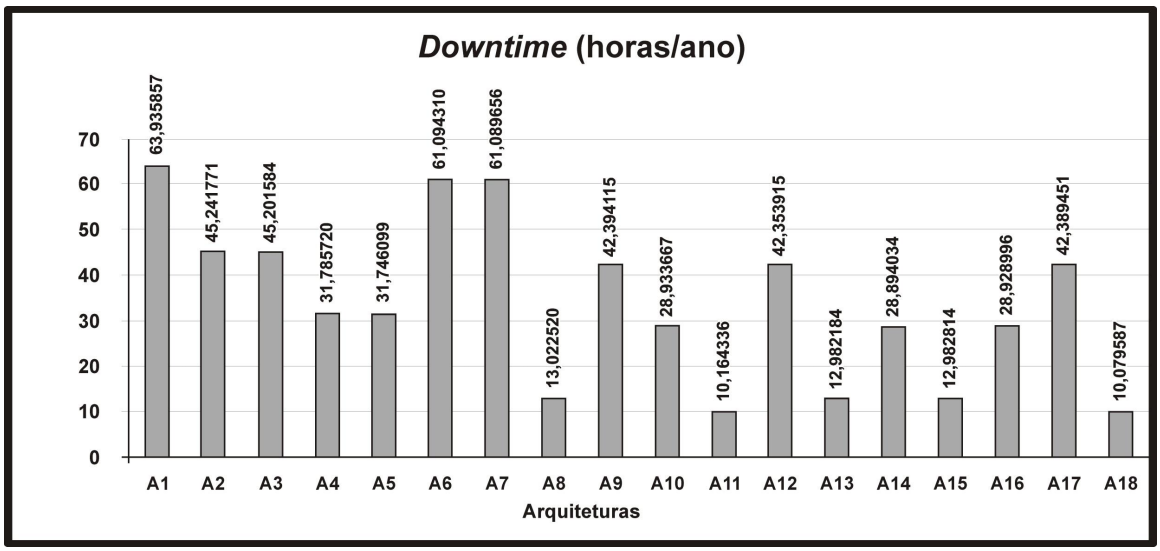


Figura 5.10: *downtime* das arquiteturas da empresa X

59,08% para a redução de 54,80% e por fim, a arquitetura A18 houve um investimento de 87,62% (equivale a R$ 29.017,10 a mais em relação à arquitetura A1) para reduzir 84,24% do tempo de inatividade.

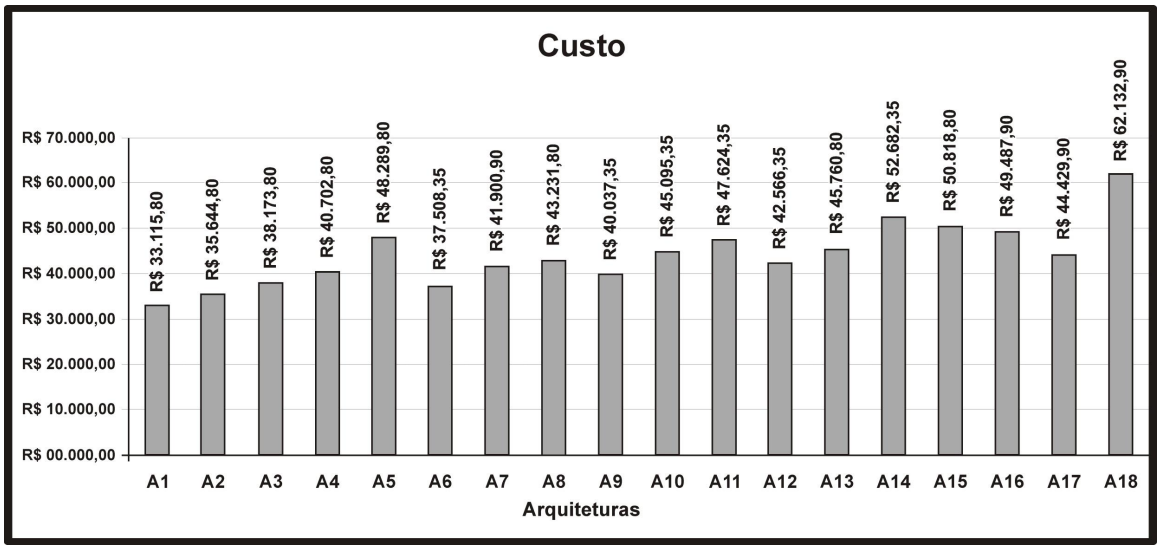


Figura 5.11: Custos das arquiteturas da empresa X

Após a análise dos dados no parágrafo anterior, a arquitetura A5 novamente apresentou-se em desvantagem em relação às demais. Porém, ela é acompanhada pelas arquiteturas A16 e A14, como opções inviáveis, considerando custo e *downtime*. Contudo, a arquitetura A11 destacou-se por quase equiparar o *downtime* da melhor arquitetura (A18) com o investindo R$ 14.508,55 a menos.

Visualmente é complexo determinar a arquitetura mais eficiente ao comparar custo e *downtime*. Por isso, a distância euclidiana foi utilizada para determinar a arquitetura que terá um custo benefício aceitável. Os cálculos dessa distância são referenciados ao ponto de origem (0). Entretanto, pelas duas métricas possuírem magnitudes diferentes. Realizou-se a normatização do custo e do *downtime* de cada uma das arquiteturas, a fim de igualar as escalas. As normatizações são obtidas pela Equação 5.2 e os resultados exibidos na Tabela 5.9:

$$NormatizarXY = \frac{NumX}{MaxNumX - MinNumX} \tag{5.2}$$

Onde,
X = {Custo, *downtime*}
Y = {Arquiteturas: A1 .. A18}
MinNumX = {Menor valor das arquiteturas}
MaxNumX = {Maior valor das arquiteturas}

Por exemplo, a normatização do *downtime* (N.D.) da arquitetura A1 é resultante da operação de 63,935857 horas (*downtime* de A1) divido pela subtração de 63,935857 (valor máximo) por 10,079587 (valor mínimo), totalizando 1,187157. A normatização do custo (N.C.) da A1 é realizada igualmente, sendo a divisão de R$ 33.115,80 (custo de A1) pela subtração de R$ 33.115,80 por R$ 62,132,90, resultando 1,141251. Após a normatização dos dados, a etapa seguinte foi calcular a distância de cada uma das arquiteturas em relação à origem (0), através da distância euclidiana, expressa na Equação 5.3.

$$DistanciaZ = \sqrt{NC^2 + ND^2} \tag{5.3}$$

Onde,
Z = {Arquitetura: A1 .. A18}
NC = {Normatização do custo}
ND = {Normatização do *downtime*}

No caso, o resultado 1,646753 da distância de A1 foi obtido pela raiz quadrada da soma entre $1,141251^2$ (NC) e $1,187157^2$ (ND), ambos elevados ao quadrado.

Com o resultado das distâncias de todas as arquiteturas realizou-se a ordenação decrescente que é visualizada na Tabela 5.10. A arquitetura A2 foi apontada como eficiente, destacada em vermelho na Figura 5.12. Ao ser relacionada com a arquitetura inicial percebeu-se uma redução de 30% do *downtime*, mediante um aumento no custo de 7%. Ela apresenta o melhor custo benefício considerando pesos iguais de importância entre Custo e *Downtime*.

Caso a arquitetura A2 não for apreciada pelo administrador. Deve-se retirar a arquitetura

Tabela 5.8: Apresentação do resultados da disponibilidade, *downtime* e custo das Arquiteturas

Arquiteturas	Disponibilidade	Downtime	Custo (R$)
$A1$	99,270139%	63,935857	33.115,80
$A2$	99,483541%	45,241770	35.644,80
$A3$	99,484000%	45,201583	38.173,80
$A4$	99,637149%	31,785720	40.702,80
$A5$	99,637602%	31,746099	48.289,80
$A6$	99,302576%	61,094309	37.508,35
$A7$	99,302630%	61,089655	41.900,90
$A8$	99,851341%	13,022520	43.231,80
$A9$	99,516049%	42,394114	40.037,35
$A10$	99,669707%	28,933667	45.095,35
$A11$	99,883969%	10,164336	47.624,35
$A12$	99,516508%	42,353914	42.566,35
$A13$	99,851802%	12,982184	45.760,80
$A14$	99,670159%	28,894033	52.682,35
$A15$	99,851794%	12,982814	50.818,80
$A16$	99,669760%	28,928996	49.487,90
$A17$	99,516102%	42,389450	44.429,90
$A18$	99,884936%	10,079587	62.132,90

A2 e repetir o processo das Equações 5.2 e 5.3, desta forma, encontrará outra arquitetura ideal. Deve-se destacar que esta técnica irá considerar como ideal somente o resultado com o menor caminho em relação à origem.

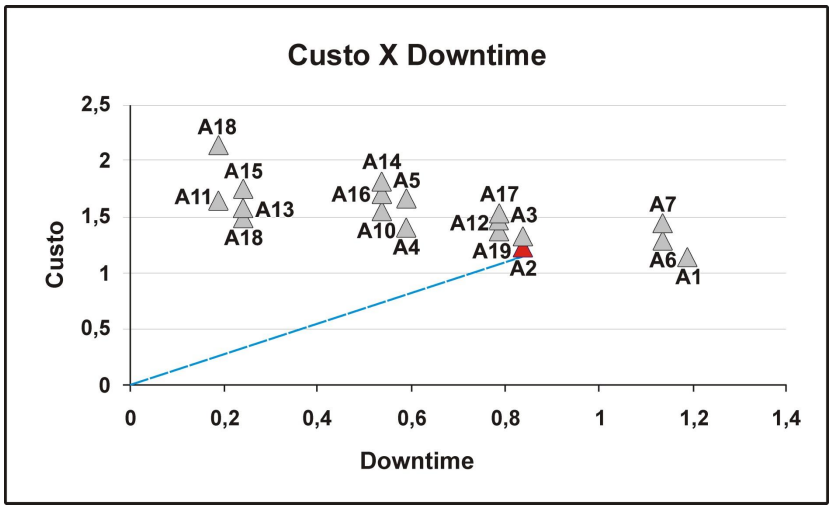


Figura 5.12: Apresentação das arquiteturas com prioridades iguais

Havendo a definição de prioridade por parte da empresa em garantir um *downtime*

Tabela 5.9: Normatização e distância euclidiana nas arquiteturas

Arquiteturas	N. D.	N. C.	Distância Euclidiana
$A1$	$1,187157$	$1,141251$	$1,646753$
$A2$	$0,840046$	$1,228407$	$1,488174$
$A3$	$0,839300$	$1,315562$	$1,56049$
$A4$	$0,590195$	$1,402718$	$1,521824$
$A5$	$0,589460$	$1,664184$	$1,765495$
$A6$	$1,134395$	$1,292629$	$1,719809$
$A7$	$1,134309$	$1,444007$	$1,83625$
$A8$	$0,241801$	$1,489873$	$1,509367$
$A9$	$0,787171$	$1,379785$	$1,588535$
$A10$	$0,537239$	$1,554096$	$1,644335$
$A11$	$0,188731$	$1,641251$	$1,652067$
$A12$	$0,786425$	$1,466940$	$1,664445$
$A13$	$0,241052$	$1,577029$	$1,595345$
$A14$	$0,536503$	$1,815562$	$1,893172$
$A15$	$0,241064$	$1,751340$	$1,767853$
$A16$	$0,537152$	$1,705474$	$1,788064$
$A17$	$0,787085$	$1,531163$	$1,721616$
$A18$	$0,187157$	$2,141251$	$2,149415$

Tabela 5.10: *Ranking* das Arquiteturas

Arquitetura	Distância em relação a origem
A2	1,488174
A8	1,509367
A4	1,521824
A3	1,56049
A9	1,588535
A13	1,595345
A10	1,644335
A1	1,646753
A11	1,652067
A12	1,664445
A6	1,719809
A17	1,721616
A5	1,765495
A15	1,767853
A16	1,788063
A7	1,83625
A14	1,893172
A18	2,149415

menor ou investir pouco. É possível adequar a Equação 5.3, da distância com a multiplicação das variáveis NC e ND, por exemplo, ao multiplicar o NC$\times$0,7 e ND$\times$0,3 significar que o administrador definiu a prioridade do custo com 70% e 30% de prioridade no *downtime*. Tal como o oposto seria NC$\times$0,3 e ND$\times$0,7, agora sendo o custo com 30% de prioridade contra 70% *downtime*.

Com a delimitação de pesos na prioridade entre custo e *downtime*, considerando um percentual, na Tabela 5.11, estão os resultados do *ranking* dos cinco menores caminhos respeitando a política estabelecida no parágrafo anterior. Na primeira aplicando peso de 70% para *downtime* e 30% para custo foi apontado a arquitetura A8 como eficiente e seja replicado os componente *Frontend* e *Nodes*. Este arquitetura possuiu um dos menores *downtime* entre todas as arquiteturas e com um custo razoável, significa que para reduzir o *downtime* em 80% é necessário investir 30%, diferença de R$ 10.116,00.

70% Downtime / 30% Custo		***Igual***		***30% Downtime / 70% Custo***	
A8	0,840740	A2	1,488174	A2	1,126051
A13	0,887006	A8	1,509367	A1	1,155215
A11	0,912713	A4	1,521824	A3	1,192820
A4	0,913300	A3	1,560490	A4	1,217305
A10	0,962602	A9	1,588535	A9	1,232297

Tabela 5.11: Comparação das arquiteturas considerando prioridades

Para a segunda política de prioridade, desta vez invertendo os pesos, a arquitetura apontada foi a mesma do resultado com prioridades iguais, a arquitetura A2. O fato ocorre pelo custo ter maior relevância, sendo a arquitetura A2 o de menor custo e com redução de *downtime* interessante.

Portanto, este segundo estudo apresentou diversos resultados envolvendo uma arquitetura um pouco mais robusta que o sistema doméstico. Nesse estudo, utilizou-se outras duas técnicas (*Availability Importance* e distância euclidiana) que foram capazes de apresentar resultados que apoiam a decisão de gestores ou administradores, quando o objetivo é aumentar a disponibilidade do *Video Surveillance as a Service*.

6

Conclusões e Trabalhos Futuros

"Se hoje fosse o último dia da minha vida, queria fazer o que vou fazer hoje? E se a resposta fosse não muitos dias seguidos, sabia que precisava mudar algo".

—STEVE JOBS

Os vídeos de vigilância estão contribuindo para crescimento dos *big datas*, apresentando uma fatia importante no universo digital. O serviço *VSaaS* foi recentemente introduzido como uma alternativa para o desenvolvimento e a gestão de sistemas de vigilância (XIONG et al., 2014). O modelo *VSaaS* permite maior escalabilidade dos recursos a um baixo custo, pela concentração dos recursos computacionais na nuvem. Esta infraestrutura permite atender a mais de um cliente e oferece mais qualidade e confiabilidade.

Para entender a grandeza desse serviço, uma pesquisa da *International Data Corporation* (IDC, 2013), evidencia que é esperado em 2015 o percentual de 65% de todos os materiais importantes a serem analisados no mundo digital sejam de vídeos de vigilância. Pelo diagnóstico publicado em (MARKETSANDMARKETS, 2012), espera-se uma taxa composta de crescimento anual do mercado de *VSaaS* de 31,5% de 2012 até 2017 resultando em $2.390,90 milhões em 2017.

Neste contexto, o presente trabalho trouxe um estudo sobre avaliação de disponibilidade em ambientes de *VSaaS*. Investigou dois sistemas: doméstico e empresarial. Destes, mais de 20 arquiteturas foram geradas com a premissa de aumentar a disponibilidade. Para avaliar as arquiteturas, criaram-se dois modelos hierárquicos utilizando RBD com CTMC no primeiro sistema e RBD com SPN no segundo. As redundâncias de componentes foram baseadas em análises e conceitos publicados. No sistema empresarial foi discutida a relação do custo das arquiteturas e o *downtime*.

Inicialmente, o primeiro estudo de caso apresentou uma arquitetura básica, sem redundância, com a finalidade foi apresentar os modelos analíticos que representasse o comportamento dessa arquitetura e obter o resultado de disponibilidade total desse serviço, através dos parâmetros de falha e reparo de cada um dos componentes que formam essa arquitetura sem redundância. Essa teve o seu modelo validado por meio de experimentos de injeção de falhas, guiados pelo método de KEESEE (1965). Constatou-se que o resultado do modelo está contido no intervalo de confiança de 95%, adotado no estudo. Portanto, no modelo proposto não tem evidências para ser refutado como modelo que representa o comportamento do ambiente real, pois ambos estão consistentes entre si. Por coerência, os mesmos valores de tempo de falha foram reduzidos. Os mesmos parâmetros de entrada (tempos de falhas e reparos) usados durante o experimento na injeção de falhas serviram como insumo para os parâmetros dos modelos avaliados neste trabalho.

Este estudo de caso foi composto por três arquiteturas: a primeira sem redundância (arquitetura 1) e as outras duas houve a inclusão de redundância (arquiteturas 2 e 3). Os resultados mostraram um aumento significativo na disponibilidade de 99,50% da arquitetura 1 para 99,68% da arquitetura 3 quando a redundância foi atribuída às Câmeras, à rede de comunicação, ao *Node*, ao *Frontend* e ao serviço *LiveVideo*. Comparado com a métrica de *downtime*, houve redução de 36,89% entre as arquiteturas 1 e 3, uma diferença de 16 horas (significa a metade de um dia com o serviço interrompido). Considerando essa interrupção para serviços críticos, tais como transportadoras de valores, bancos, museus, poderiam ocasionar prejuízos financeiros severos.

Para a realização da análise de sensibilidade paramétrica, adotou-se a arquitetura 3 do modelo analítico do sistema doméstico. Os resultados mostraram que os parâmetros de falha e reparo do *Node* têm o maior impacto na disponibilidade, sendo uma boa opção indicada para receber melhorias e consequentemente aumentar a disponibilidade. Outros componentes relevantes que contribuem na disponibilidade são o *Frontend* e o *Streaming Transmitter*. Porém, os componentes do subsistema LiveVideo registraram pequeno impacto, este justificado pela redundância *warm-standby* aplicada. Os componentes câmera, 3G e *WiFi* tiveram comportamentos semelhantes.

No segundo estudo de caso, o empresarial, mais robusta que as arquiteturas do sistema doméstico. Nesse estudo, iniciou-se pela a definição da estratégia de disponibilidade mais coesa, optando-se por quatro das cinco câmeras funcionarem por andar. A escolha foi motivada por ela proporcionar maior sensação de segurança e a existência de uma diferença de *downtime* pequena em relação às outras estratégias, não ultrapassando as três horas anuais.

Após a definição dessa estratégia, aplicou-se a técnica de *Availability Importance* para en-

contrar os componentes de maior impacto na disponibilidade. Usado com a finalidade semelhante a da análise de sensibilidade, na detecção de gargalos e pontos críticos. Entre os componentes avaliados e apontados a serem replicados foram a *CameraLiveVideo*, o *Node* e o *Frontend*. Com esta orientação foram geradas 18 arquiteturas com dois níveis a mais de redundância para os três primeiros componentes do *ranking* do *Availability Importance*. Visualmente, algumas arquiteturas destacam-se como boas escolhas conforme a métrica de disponibilidade (A8, A11, A13, A15 e A18) e de *downtime* as arquiteturas descartadas (A1, A5 e A6).

Para determinar uma arquitetura eficaz, entre todas, foi aplicada a técnica de distância euclidiana, considerando a relação custo e *downtime*. Desse modo, houve necessidade de transformar os valores em grandeza proporcionalmente iguais, por meio da normatizados dos dados. O resultado indicou a arquitetura A2 como a de maior aumento na disponibilidade em relação ao menor investimento. Com o investimento 7%, obteve-se 30% na redução do *downtime* anual. Somente a replicação do componente *Frontend* alcançou um resultado satisfatório. Contudo, isso não significa ser o melhor, porque dependerá das pretensões da empresa ao considerar o capital de investimento disponível e o nível exigido de tolerância a falha do sistema.

Para auxiliar a decisão dos gestores, dois pesos de prioridade foram determinados: primeiro priorizando 70% na redução do *downtime* e o outro 70% no baixo custo. O primeiro peso apontou a arquitetura A8, que por meio da replicação do *Frontend* e do *Node* reduziu 80% do *downtime* com R$ 10.116,00 a mais que a arquitetura básica, em torno de 30%. No segundo peso, a arquitetura A2 foi indicada como viável financeiramente, inclusive é a mesma indicada no resultado com prioridades iguais para custo e *downtime*.

Portanto, os resultados obtidos e a abordagem utilizada durante esta pesquisa podem auxiliar outros sistemas de vigilância em nuvem, porém as conclusões dependerão do nível de disponibilidade desejado e os recursos financeiros destinados à política de redundância aplicada aos componentes. Ressalta-se quanto maior o número de componentes replicados da arquitetura, maior será o custo e a disponibilidade.

Além das conclusões obtidas dos estudos, podem-se destacar algumas contribuições específicas desse trabalho. Essas são listadas a seguir:

- Desenvolvimento de uma metodologia para auxiliar o processo de avaliação do modelo de disponibilidade de *VSaaS*, com modelos CTMC, SPN e RBD. Com a realização de experimentos de injeção de falhas para validação de modelos;
- Modelos de arquiteturas de sistemas domésticos e empresariais para a plataforma de *VSaaS*, considerando os componentes presentes no cliente e na infraestrutura da nuvem;
- Utilização da análise de sensibilidade paramétrica e o *Availability Importance* em arquiteturas de *VSaaS*, para detectar pontos críticos e indicar os componentes a serem replicados;

- Aplicação do conceito de distância euclidiana na discutir o *trade-off* entre custo versus *downtime*.

Ao término dessa pesquisa, verificou-se que a mesma possui extensões para atividades futuras. Dentre as quais, podem-se destacar:

- Estender a modelagem das arquiteturas incluindo a entrega do serviço ao usuário final, verificando as disponibilidades dos dispositivos (*notebook*, *tablets* e *smartphones*) e os softwares de navegação, bem como, modelar o serviço de armazenamento ocorrendo na nuvem, localmente ou de maneira híbrida;
- Verificar se há aumento de disponibilidade com o uso de câmeras com detecção de movimento, considerando a viabilização do custo;
- Considerar balanceamento de carga e mecanismos de *auto scaling* nas VM do *hosts*, a fim de avaliar o seu impacto sobre a disponibilidade do *VSaaS*;
- Implementar um mecanismo de aviso automático quando o sistema estiver indisponível e realizar uma análise confiabilidade;
- Utilizar um *middleware* no cliente com o objetivo de detectar falha no provedor de nuvem e direcionar o processamento do serviço para um provedor ativo.

Referências

ABDALLAH, H.; HAMZA, M. On the sensitivity analysis of the expected accumulated reward. **Performance Evaluation**, [S.l.], v.47, n.2, p.163–179, 2002.

AJMONE MARSAN, M.; CONTE, G.; BALBO, G. A Class of Generalized Stochastic Petri Nets for the Performance Evaluation of Multiprocessor Systems. **ACM Trans. Comput. Syst.**, New York, NY, USA, v.2, p.93–122, May 1984.

ARAUJO, J. et al. Software rejuvenation in eucalyptus cloud computing infrastructure: a method based on time series forecasting and multiple thresholds. In: SOFTWARE AGING AND REJUVENATION (WOSAR), 2011 IEEE THIRD INTERNATIONAL WORKSHOP ON. **Anais...** [S.l.: s.n.], 2011. p.38–43.

ARAúJO, C. J. M. **Avaliaçãoo e modelagem de desempenho para planejamento de capacidade do sistema de transferência eletrônica de fundos utilizando tráfego em rajada**. 2009. Tese (Doutorado em Ciência da Computação) — Dissertação, Universidade Federal de Pernambuco, Centro de Informática.

ARLAT, J. et al. Fault Injection and Dependability Evaluation of Fault-Tolerant Systems. **Computers, IEEE Transactions on**, [S.l.], v.42, n.8, p.913–923, 1993.

AVIZIENIS, A. et al. **Fundamental Concepts of Dependability**. [S.l.]: University of Newcastle upon Tyne, Computing Science, 2001.

AVIZIENIS, A. et al. Basic Concepts and Taxonomy of Dependable and Secure Computing. **Dependable and Secure Computing, IEEE Transactions on**, [S.l.], v.1, n.1, p.11–33, 2004.

AVIŽIENIS, A. et al. Basic Concepts and Taxonomy of Dependable and Secure Computing. **Dependable and Secure Computing, IEEE Transactions on**, [S.l.], v.1, n.1, p.11–33, 2004.

BAIER, C. et al. Model-checking algorithms for continuous-time Markov chains. **Software Engineering, IEEE Transactions on**, [S.l.], v.29, n.6, p.524–541, 2003.

BAKSHI, K. Cisco Cloud Computing-Data Center Strategy, Architecture, and Solutions. **DOI= http://www. cisco. com/web/strategy/docs/gov/CiscoCloudComputing_WP. pdf**, [S.l.], 2009.

BALBO, G. Introduction to Stochastic Petri Nets. In: **Lectures on Formal Methods and PerformanceAnalysis**. [S.l.]: Springer, 2001. p.84–155.

BASSANEZI, R. C. **Ensino-aprendizagem com modelagem matemática**: uma nova estratégia. [S.l.]: Editora Contexto, 2002.

BLAKE, J. T.; REIBMAN, A. L.; TRIVEDI, K. S. Sensitivity analysis of reliability and performability measures for multiprocessor systems. In: ACM SIGMETRICS PERFORMANCE EVALUATION REVIEW. **Anais...** [S.l.: s.n.], 1988. v.16, n.1, p.177–186.

BOLCH, G. et al. **Queuing Networks and Markov Chains**: modeling and performance evaluation with computer science applications. [S.l.]: John Wiley & Sons, 2006.

BUYYA, R.; YEO, C. S.; VENUGOPAL, S. Market-Oriented Cloud Computing: vision, hype, and reality for delivering it services as computing utilities. In: HIGH PERFORMANCE COMPUTING AND COMMUNICATIONS, 2008. HPCC'08. 10TH IEEE INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2008. p.6.

CALHEIROS, R. N. et al. CLOUDSIM: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. **Software: Practice and Experience**, [S.l.], v.41, n.1, p.23–50, 2011.

CALLOU, G. et al. Estimating Sustainability Impact of High Dependable Data Centers: a comparative study between brazilian and us energy mixes. **Computing**, [S.l.], v.95, n.12, p.1137–1170, 2013.

CARVALHO, L. S. de. Modelagem e simulação: poderosa ferramenta para a otimização de operações logísticas. , [S.l.], 2003.

CHAN, H. A. Accelerated Stress Testing for Both Hardware and Software. In: RELIABILITY AND MAINTAINABILITY, 2004 ANNUAL SYMPOSIUM-RAMS. **Anais...** [S.l.: s.n.], 2004. p.346–351.

CHANG, S.-F.; VETRO, A. Video Adaptation: concepts, technologies, and open issues. **Proceedings of the IEEE**, [S.l.], v.93, n.1, p.148–158, 2005.

CHEN, X. et al. Understanding, Modelling, and Improving the Performance of Web Applications in Multicore Virtualised Environments. In: ACM/SPEC INTERNATIONAL CONFERENCE ON PERFORMANCE ENGINEERING, 5. **Proceedings...** [S.l.: s.n.], 2014. p.197–207.

CIARDO, G. Petri Nets with Marking-Dependent Arc Cardinality: properties and analysis. In: **Application and Theory of Petri Nets 1994**. [S.l.]: Springer, 1994. p.179–198.

COOPER, T.; FARRELL, R. Value-Chain Engineering of a Tower-Top Cellular Base Station System. In: VEHICULAR TECHNOLOGY CONFERENCE, 2007. VTC2007-SPRING. IEEE 65TH. **Anais...** [S.l.: s.n.], 2007. p.3184–3188.

D-LINK. **D-link Wireless N150 Router**. [S.l.: s.n.], 2012.

DANTAS, J. et al. Models for Dependability Analysis of Cloud Computing Architectures for Eucalyptus Platform. **International Transactions on Systems Science and Applications**, [S.l.], v.8, p.13–25, 2012.

DANTAS, J. R. **Modelos para Análise de Dependabilidade de Arquiteturas de Computação em Nuvem**. 2012. Dissertação (Mestrado em Ciência da Computação) — Universidade Federal de Pernambuco, Recife, PE.

DAS, O. **Performance and Dependability Analysis of Fault-Tolerant Layered Distributed Systems**. 1998. Tese (Doutorado em Ciência da Computação) — Carleton University Ottawa.

FIGUEIRÊDO, J. Jair Cavalcante de; ROMERO MARTINS MACIEL, P. O. Análise de dependabilidade de sistemas data center baseada em índices de importância. , [S.l.], 2011.

FRANK, P. M. **Introduction to System Sensitivity Theory**. [S.l.]: Academic press New York, 1978. v.11.

GERMAN, R. **Performance Analysis of Communication Systems with Non-Markovian Stochastic Petri Nets**. [S.l.]: John Wiley & Sons, Inc., 2000.

GOEL, A. L. Software Reliability Models: assumptions, limitations, and applicability. **Software Engineering, IEEE Transactions on**, [S.l.], n.12, p.1411–1423, 1985.

GOMES, C. N. **Estudo do Paradigma Computação em Nuvem**. 2012. Tese (Doutorado em Ciência da Computação) — INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA.

HAGEN, S.; SEIBOLD, M.; KEMPER, A. Efficient Verification of it Change Operations or: how we could have prevented amazon's cloud outage. In: NETWORK OPERATIONS AND MANAGEMENT SYMPOSIUM (NOMS), 2012 IEEE. **Anais...** [S.l.: s.n.], 2012. p.368–376.

HAMBY, D. A Review of Techniques for Parameter Sensitivity Analysis of Environmental Models. **Environmental Monitoring and Assessment**, [S.l.], v.32, n.2, p.135–154, 1994.

HAVERKORT, B. R. **Performance of Computer Communication Systems**: a model-based approach. [S.l.]: Wiley, 1998.

HERBERT, L.; ERICKSON, J. The ROI of Cloud Apps. **A Total Economic Impact™ Analysis Uncovers Long-Term Value In Cloud Apps, Forrester**, [S.l.], 2011.

IDC. **The Digital Universe in 2020**: big data, bigger digital shadows, and biggest growth in the far east. [S.l.: s.n.], 2013. IDC.

INTRANSA. **High Availability Concepts for Video Surveillance**. [S.l.: s.n.], 2010. INTRANSA.

KARIMAA, A. Video Surveillance in the Cloud: dependability analysis. In: PROCEEDING OF THE 4TH INTERNATIONAL CONFERENCE ON DEPENDABILITY (DEPEND 11). **Anais...** [S.l.: s.n.], 2011. p.92–95.

KARTSON, D. et al. **Modeling with Generalized Stochastic Petri Nets**. [S.l.]: John Wiley & Sons, Inc., 1994.

KEESEE, W. **A Method of Determining a Confidence Interval for Availability**. [S.l.]: DTIC Document, 1965.

KIM, D. S.; GHOSH, R.; TRIVEDI, K. S. A Hierarchical Model for Reliability Analysis of Sensor Networks. In: DEPENDABLE COMPUTING (PRDC), 2010 IEEE 16TH PACIFIC RIM INTERNATIONAL SYMPOSIUM ON. **Anais...** [S.l.: s.n.], 2010. p.247–248.

KIM, D. S.; MACHIDA, F.; TRIVEDI, K. S. Availability Modeling and Analysis of a Virtualized System. In: DEPENDABLE COMPUTING, 2009. PRDC'09. 15TH IEEE PACIFIC RIM INTERNATIONAL SYMPOSIUM ON. **Anais...** [S.l.: s.n.], 2009. p.365–371.

KLEINROCK, L. Queueing systems, volume I: theory. , [S.l.], 1975.

KUO, W.; ZUO, M. J. **Optimal Reliability Modeling**: principles and applications. [S.l.]: Wiley. com, 2003.

LAVENBERG, S. **Computer Performance Modeling Handbook**. [S.l.]: Elsevier, 1983. v.4.

LAWTON, G. Users take a close look at visual analytics. **Computer**, [S.l.], n.2, p.19–22, 2009.

M, R. Users take a close look at visual analytics. **Smartvue White Paper**, [S.l.], p.1–3, 2012.

MACIEL, P. R. et al. Dependability Modeling. **Performance and Dependability in Service Computing: Concepts, Techniques and Research Directions**, [S.l.], v.1, p.53–97, 2011.

MACIEL, P. R. et al. Dependability Modeling. **Performance and Dependability in Service Computing: Concepts, Techniques and Research Directions**, [S.l.], p.53–97, 2012.

MACIEL, P. R.; LINS, R. D.; CUNHA, P. R. **Introdução às Redes de Petri e Aplicações**. [S.l.]: UNICAMP-Instituto de Computacao, 1996.

MALHOTRA, M.; TRIVEDI, K. S. A Methodology for Formal Expression of Hierarchy in Model Solution. In: PETRI NETS AND PERFORMANCE MODELS, 1993. PROCEEDINGS., 5TH INTERNATIONAL WORKSHOP ON. **Anais...** [S.l.: s.n.], 1993. p.258–267.

MARKETSANDMARKETS. **Global Video Surveillance as a Service Market worth $2,390.9 Million by 2017**. [S.l.: s.n.], 2012. MarketsandMarkets.

MARRANGHELLO, N. Redes de Petri: conceitos e aplicações. **São Paulo: DCCE/IBILCE/UNESP**, [S.l.], 2005.

MARSAN, M. A.; BALBO, G.; CONTE, G. Performance Models of Multiprocessor Systems. , [S.l.], 1986.

MARSAN, M. A. et al. **Modelling with Generalized Stochastic Petri Nets**. [S.l.]: John Wiley & Sons, Inc., 1994.

MARSTON, S. et al. Cloud Computing - The Business Perspective. **Decision Support Systems**, [S.l.], v.51, n.1, p.176–189, 2011.

Matos Júnior, R. S. **An Automated Approach for Systems Performance and Dependability Improvement Through Sensitivity Analysis of Markov Chains**. 2011. Dissertação (Mestrado em Ciência da Computação) — Universidade Federal de Pernambuco, Recife, Brazil.

MATOS, R. et al. Sensitivity Analysis of Server Virtualized System Availability. **Reliability, IEEE Transactions on**, [S.l.], v.61, n.4, p.994–1006, 2012.

MELL, P.; GRANCE, T. The NIST Definition of Cloud Computing. **National Institute of Standards and Technology Special Publication**, [S.l.], p.800–145, 2014.

MELO, M. D. **Modelos de Disponibilidade para Nuvens Privadas: rejuvenescimento de software habilitado por agendamento de migração de vms**. 2014. Dissertação (Mestrado em Ciência da Computação) — Universidade Federal de Pernambuco, Recife, PE.

MELO, M. et al. Availability study on cloud computing environments: live migration as a rejuvenation mechanism. In: DEPENDABLE SYSTEMS AND NETWORKS (DSN), 2013 43RD ANNUAL IEEE/IFIP INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2013. p.1–6.

MENASCE, D. A. et al. **Performance by design**: computer capacity planning by example. [S.l.]: Prentice Hall Professional, 2004.

MILLER, M. **Cloud Computing**: web-based applications that change the way you work and collaborate online. [S.l.]: Que publishing, 2008.

MURATA, T. Petri Nets: properties, analysis and applications. **Proceedings of the IEEE**, [S.l.], v.77, n.4, p.541–580, 1989.

NEAL, D.; RAHMAN, S. Video Surveillance in the Cloud? **The International Journal of Cryptography and Information Security (IJCIS)**, [S.l.], v.2, n.3, 2012.

PEARSON, S. Taking Account of Privacy When Designing Cloud Computing Services. In: ICSE WORKSHOP ON SOFTWARE ENGINEERING CHALLENGES OF CLOUD COMPUTING, 2009. **Proceedings...** [S.l.: s.n.], 2009. p.44–52.

PETRI, C. A. Kommunikation mit automaten. , [S.l.], 1962.

PICCONI, F.; MASSOULIÉ, L. Is There a Future for Mesh-Based Live Video Streaming? In: PEER-TO-PEER COMPUTING, 2008. P2P'08. EIGHTH INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2008. p.289–298.

PRADO, G. Desertesejo: um projeto de ambiente virtual de multiusuário na web. **Catálogo da Electronic Art Exhibition, SIBGRAPI**, [S.l.], 2000.

SABAHI, F. Cloud Computing Security Threats and Responses. In: COMMUNICATION SOFTWARE AND NETWORKS (ICCSN), 2011 IEEE 3RD INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2011. p.245–249.

SALEH, N. A.; MAHMOUD, M. A.; ABDEL-SALAM, A.-S. G. The Performance of the Adaptive Exponentially Weighted Moving Average Control Chart with Estimated Parameters. **Quality and Reliability Engineering International**, [S.l.], v.29, n.4, p.595–606, 2013.

SALES, A. H. C. de. Um Estudo sobre Redes de Petri Estocásticas Generalizadas. , [S.l.], 2002.

SARANYA, S. M.; VIJAYALAKSHMI, M. Interactive Mobile Live Video Learning System in Cloud Environment. In: RECENT TRENDS IN INFORMATION TECHNOLOGY (ICRTIT), 2011 INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2011. p.673–677.

SCHROEDER, B.; GIBSON, G. A. Disk Failures in the Real World: what does an mttf of 1, 000, 000 hours mean to you? In: FAST. **Anais...** [S.l.: s.n.], 2007. v.7, p.1–16.

SHARMA, C. M.; KUMAR, H. Architectural framework for implementing visual surveillance as a service. In: COMPUTING FOR SUSTAINABLE GLOBAL DEVELOPMENT (INDIACOM), 2014 INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2014. p.296–301.

SIA. **Mercado Formal de Segurança Eletrônica no Brasil Deve Crescer 20,6% até 2017**. [S.l.: s.n.], 2014. SIA.

SILVA, B. et al. ASTRO: an integrated environment for dependability and sustainability evaluation. **Sustainable Computing: Informatics and Systems**, [S.l.], 2012.

SONG, B.; TIAN, Y.; ZHOU, B. Design and Evaluation of Remote Video Surveillance System on Private Cloud. In: BIOMETRICS AND SECURITY TECHNOLOGIES (ISBAST), 2014 INTERNATIONAL SYMPOSIUM ON. **Anais...** [S.l.: s.n.], 2014. p.256–262.

SOUSA, E. d. **Avaliação do impacto de uma política de manutenção na performabilidade de sistemas de transferência eletrônica de fundos**. 2009. Tese (Doutorado em Ciência da Computação) — Dissertação, Universidade Federal de Pernambuco, Centro de Informática.

SOUSA, E. et al. Evaluating Eucalyptus Virtual Machine Instance Types: a study considering distinct workload demand. In: CLOUD COMPUTING 2012, THE THIRD INTERNATIONAL CONFERENCE ON CLOUD COMPUTING, GRIDS, AND VIRTUALIZATION. **Anais...** [S.l.: s.n.], 2012. p.130–135.

SOUZA, D. et al. A Tool for Automatic Dependability Test in Eucalyptus Cloud Computing Infrastructures. **Computer and Information Science**, [S.l.], v.6, n.3, p.57–67, 2013.

STEWART, W. J. **Introduction to the Numerical Solutions of Markov Chains**. [S.l.]: Princeton Univ. Press, 1994.

SUBRAMANIAN, K. Hybrid Clouds. **Access from http://emea. trendmicro. com/imperia/md/content/uk/cloud-security/wp01_hybridcloud-krish_110624us. pdf**, [S.l.], 2011.

SUN, D. et al. A Dependability Model to Enhance Security of Cloud Environment Using System-Level Virtualization Techniques. In: PERVASIVE COMPUTING SIGNAL PROCESSING AND APPLICATIONS (PCSPA), 2010 FIRST INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2010. p.305–310.

SUTOR, S. et al. Large-Scale Video Surveillance Systems: new performance parameters and metrics. In: INTERNET MONITORING AND PROTECTION, 2008. ICIMP'08. THE THIRD INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2008. p.23–30.

SUVONVORN, N. A Video Analysis Framework for Surveillance System. In: MULTIMEDIA SIGNAL PROCESSING, 2008 IEEE 10TH WORKSHOP ON. **Anais...** [S.l.: s.n.], 2008. p.867–871.

TERRY, D. Acm Tech Pack on Cloud Computing. **ACM Tech Pack Committee on Cloud Computing**, [S.l.], 2011.

TRIVEDI, K. S. **Probability and Statistics with Reliability, Queueing and Computer Science Applications**. [S.l.]: John Wiley & Sons, 2002.

TRIVEDI, K. S. et al. Reliability Analysis Techniques Explored Through a Communication Network Example. , [S.l.], 1996.

TRIVEDI, K. S. et al. Modeling High Availability. In: DEPENDABLE COMPUTING, 2006. PRDC'06. 12TH PACIFIC RIM INTERNATIONAL SYMPOSIUM ON. **Anais...** [S.l.: s.n.], 2006. p.154–164.

VACARO, J. C.; WEBER, T. S. Injeção de Falhas na Fase de Teste de Aplicações Distribuídas. , [S.l.], 2006.

VAQUERO, L. M. et al. A Break in the Clouds: towards a cloud definition. **ACM SIGCOMM Computer Communication Review**, [S.l.], v.39, n.1, p.50–55, 2008.

WEBER, T. S. Tolerância a Falhas: conceitos e exemplos. **Apostila do Programa de Pós-Graduação–Instituto de Informática-UFRGS. Porto Alegre**, [S.l.], 2003.

XIONG, Y.-H. et al. Design and Implementation of a Prototype Cloud Video Surveillance System. **Journal ref: Journal of Advanced Computational Intelligence and Intelligent Informatics**, [S.l.], v.18, n.1, p.40–47, 2014.

ZHANG, Q.; CHENG, L.; BOUTABA, R. Cloud Computing: state-of-the-art and research challenges. **Journal of internet services and applications**, [S.l.], v.1, n.1, p.7–18, 2010.

Apêndice

A

Scripts de Injeção de falha do Cliente

```
1  #!/bin/bash
2
3  echo "Fault injection Cliente...."
4  echo "# Event GT Date" >> ClienteFault.log
5  C1=0
6
7  while [ $C1 -le 49 ] ;
8  do
9          C1=$(($C1+1))
10         TIME=$(Rscript expFailCliente.r | awk '{print $2}')
11         sleep $TIME
12         sshpass -p "livevideo" ssh magno/192.168.56.109 "sudo -S /etc
               /init.d/rtsp stop"
13         echo "Fault injected"
14         echo  $C1 "Fault" $TIME $(date | awk '{ print $2, $3, $4 }')
               >> ClienteFault.log
15
16         TIMER=$(Rscript expRepairCliente.r | awk '{print $2}')
17         sleep $TIMER
18         sshpass -p "livevideo" ssh magno/192.168.56.109 "sudo -S /etc
               /init.d/rtsp start"
19         echo "Repair Injected"
20         echo $C1 "Repair" $TIMER $(date | awk '{ print $2, $3, $4 }')
                >> ClienteFault.log
21 done
22
23 echo "Finish Cliente"
```

B

Scripts de Injeção de falha do Frontend

```
1  #!/bin/bash
2
3  echo "Fault injection Frontend...."
4  echo "# Event GT Date" >> FrontendFault.log
5  C1=0
6
7  while [ $C1 -le 49 ] ;
8  do
9          C1=$(($C1+1))
10         TIME=$(Rscript expFailFrontend.r | awk '{print $2}')
11         sleep $TIME
12         ssh matheus/192.168.56.1 "sudo -S shutdown -h 0 < ~/FILE.txt"
13         echo "Fault injected"
14         echo  $C1 "Fault" $TIME $(date | awk '{ print $2, $3, $4 }')
               >> FrontendFault.log
15
16         TIMER=$(Rscript expRepairFrontend.r | awk '{print $2}')
17         sleep $TIMER
18         wakeonlan 192.168.56.1
19         echo "Repair Injected"
20         echo $C1 "Repair" $TIMER $(date | awk '{ print $2, $3, $4 }')
                >> FrontendFault.log
21 done
22
23 echo "Finish Frontend"
```

C

Scripts de Injeção de falha do Node

```
#!/bin/bash

echo "Fault injection Node...."
echo "# Event GT Date" >> NodeFault.log
C1=0

while [ $C1 -le 49 ] ;
do
        C1=$(($C1+1))
        TIME=$(Rscript expFailNode.r | awk '{print $2}')
        sleep $TIME
        ssh matheus/192.168.56.2 "sudo -S shutdown -h 0 < ~/FILE.txt"
        echo "Fault injected"
        echo  $C1 "Fault" $TIME $(date | awk '{ print $2, $3, $4 }')
           >> NodeFault.log

        TIMER=$(Rscript expRepairNode.r | awk '{print $2}')
        sleep $TIMER
        wakeonlan 192.168.56.2
        echo "Repair Injected"
        echo $C1 "Repair" $TIMER $(date | awk '{ print $2, $3, $4 }')
             >> NodeFault.log
done

echo "Finish Node"
```

D

Scripts de Injeção de falha do LiveVideo

```
1  #!/bin/bash
2
3  echo "Fault injection LiveVideo...."
4  echo "# Event GT Date" >> LiveVideoFault.log
5  C1=0
6
7  while [ $C1 -le 49 ] ;
8  do
9          C1=$(($C1+1))
10         TIME=$(Rscript expFailLiveVideo.r | awk '{print $2}')
11         sleep $TIME
12         sshpass -p "opennebula" ssh matheus/192.168.56.117 "sudo -S /
               etc/init.d/apache2 stop"
13         echo "Fault injected"
14         echo  $C1 "Fault" $TIME $(date | awk '{ print $2, $3, $4 }')
               >> LiveVideoFault.log
15
16         TIMER=$(Rscript expRepairLiveVideo.r | awk '{print $2}')
17         sleep $TIMER
18         sshpass -p "opennebula" ssh matheus/192.168.56.117 "sudo -S /
               etc/init.d/apache2 start"
19         echo "Repair Injected"
20         echo $C1 "Repair" $TIMER $(date | awk '{ print $2, $3, $4 }')
                >> LiveVideoFault.log
21 done
22
23 echo "Finish LiveVideo"
```

E

Scripts de Monitoramento do Cliente

```
#!/bin/bash

C=0

echo "Start" $(date | awk '{ print $2, $3, $4 }') >> "
    TesteAvailCliente".txt
echo "# StatusCliente "  >> "TesteAvailCliente".txt

while [ True ]
do
        C=$(($C+1))
        STATUS=$(nmap -p 554 192.168.56.109 | grep 554/tcp | awk '{
            print $2}')
        if [ "$STATUSS" == "UNKNOWN"] ; then
                        echo $C "Cliente" "down" $(date | awk '{
                            print $2, $3, $4 }') >> "TesteAvailCliente
                            ".txt
        else
                echo $C "Cliente" "up" $(date | awk '{ print $2, $3,
                    $4 }') >> "TesteAvailCliente".txt
        fi
        sleep 1
done

echo "End" $(date | awk '{ print $2, $3, $4 }') > "TesteAvailCliente"
    .txt
```

F

Scripts de Monitoramento do Frontend

```
1  #!/bin/bash
2
3  C=0
4
5  echo "Start" $(date | awk '{ print $2, $3, $4 }') >> "
       TesteAvailFrontend".txt
6  echo "# StatusFrontend"  >> "TesteAvailFrontend".txt
7
8  while [ True ]
9  do
10         C=$(($C+1))
11         if [ping -c 5 192.168.56.1 > /dev/null] ; then
12                 echo $C "up" $(date | awk '{ print $2, $3, $4 }') >>
                       "TesteAvailFrontend".txt
13         else
14                 echo $C "down" $(date | awk '{ print $2, $3, $4 }')
                       >> "TesteAvailFrontend".txt
15         fi
16         sleep 1
17 done
18 echo "End" $(date | awk '{ print $2, $3, $4 }') > "TesteAvailFrontend
       ".txt
```

G

Scripts de Monitoramento do Node

```
1  #!/bin/bash
2
3  C=0
4
5  echo "Start" $(date | awk '{ print $2, $3, $4 }') >> "TesteAvailNode"
       .txt
6  echo "# StatusNode"  >> "TesteAvailNode".txt
7
8  while [ True ]
9  do
10         C=$(($C+1))
11         if [ping -c 5 192.168.56.2 > /dev/null] ; then
12                 echo $C "up" $(date | awk '{ print $2, $3, $4 }') >>
                       "TesteAvailNode".txt
13         else
14                 echo $C "down" $(date | awk '{ print $2, $3, $4 }')
                       >> "TesteAvailNode".txt
15         fi
16         sleep 1
17 done
18
19 echo "End" $(date | awk '{ print $2, $3, $4 }') > "TesteAvailNode".
       txt
```

H

Scripts de Monitoramento do LiveVideo

```
1  #!/bin/bash
2
3  C=0
4
5  echo "Start" $(date | awk '{ print $2, $3, $4 }') >> "
       TesteAvailLiveVideo".txt
6  echo "# StatusLiveVideo "  >> "TesteAvailLiveVideo".txt
7
8  while [ True ]
9  do
10         C=$(($C+1))
11         STATUS=$(nmap -p 80 192.168.56.118 | grep 80/tcp | awk '{
               print $2}')
12         if [ "$STATUSS" == "(1" ] ; then
13                         echo $C $STATUS "up" $(date | awk '{ print $2
                               , $3, $4 }') >> "TesteAvailLiveVideo".txt
14         else
15                 echo $C "LiveVideo" "down" $(date | awk '{ print $2,
                       $3, $4 }') >> "TesteAvailLiveVideo".txt
16         fi
17         sleep 1
18 done
19
20 echo "End" $(date | awk '{ print $2, $3, $4 }') > "TesteAvailApache".
       txt
```