UNIVERSIDADE FEDERAL DE ALAGOAS
INSTITUTO DE COMPUTAÇÃO
PROGRAMA DE PÓS GRADUAÇÃO EM INFORMÁTICA

Anderson Mendes dos Santos

UM MÉTODO PARA APOIAR DECISÕES DE PROJETO EM APLICAÇÕES WEB COM *STREAMING* DE MÍDIA VISANDO DESEMPENHO E ESCALABILIDADE

Maceió - AL
2016

ANDERSON MENDES DOS SANTOS

UM MÉTODO PARA APOIAR DECISÕES DE PROJETO EM APLICAÇÕES WEB COM *STREAMING* DE MÍDIA VISANDO DESEMPENHO E ESCALABILIDADE

Dissertação apresentada ao Programa de Pós-Graduação em Informática da Universidade Federal de Alagoas, como requisito para obtenção do grau de Mestre em Informática.

Orientador: Prof. Dr. Patrick H. S. Brito

Maceió - AL
2016

Catalogação na fonte
Universidade Federal de Alagoas
Biblioteca Central
Divisão de Tratamento Técnico
Bibliotecária Responsável: Helena Cristina Pimentel do Vale

S237u Santos, Anderson Mendes dos.
Um método para apoiar decisões de projeto em aplicações web com streaming de mídia visando desempenho e escalabilidade / Anderson Mendes dos Santos. – 2016.
112 f. : il.

Orientador: Patrick H. S. Brito.
Dissertação (mestrado em Modelagem Computacional de Conhecimento) – Universidade Federal de Alagoas. Instituto de Computação. Maceió, 2016.

Bibliografia: f. 76-79.
Apêndices: f. 80-112.

1. Streaming de Mídia. 2. Apoio ao desenvolvedor. 3. Throughput. 4. Tecnologias de desenvolvimento Web. 5. Linguagem de programação – Avaliação. I. Título.

CDU: 004.42

UNIVERSIDADE FEDERAL DE ALAGOAS/UFAL
Programa de Pós-Graduação em Informática – PpgI
Instituto de Computação
Campus A. C. Simões BR 104-Norte Km 14 BL 12 Tabuleiro do Martins
Maceió/AL - Brasil CEP: 57.072-970 | Telefone: (082) 3214-1401

Membros da Comissão Julgadora da Dissertação de Mestrado de Anderson Mendes dos Santos, intitulada: "Um Método para Apoiar Decisões de Projeto em Aplicações Web com Streaming de Mídia Visando Desempenho e Escalabilidade", apresentada ao Programa de Pós-Graduação em Informática da Universidade Federal de Alagoas em 16 de dezembro de 2016, às 14h15min, na Sala de Reuniões do Instituto de Computação da UFAL.

COMISSÃO JULGADORA

Prof. Dr. Patrick Henrique da Silva Brito
UFAL – Campus Arapiraca
Orientador

Prof. Dr. Baldoino Fonseca dos Santos Neto
UFAL – Instituto de Computação
Examinador

Prof. Dr. Aydano Pamponet Machado
UFAL – Instituto de Computação
Examinador

A Deus, o que seria de mim sem a fé que eu tenho nele, agradeço ao meu pai Anacleto, minha mãe Jucileide, ao meu irmão Alan, e ao meu amado e precioso irmão Alisson.

AGRADECIMENTOS

Meus agradecimentos a todos os familiares, amigos, professores, que direta ou indiretamente contribuíram para a realização deste trabalho. Em especial, dedico meus agradecimentos:

- A Deus, por ter me dado força e saúde para chegar até aqui;
- Ao meu orientador, Prof. Dr. Patrick Brito, por ter aceitando essa missão de me orientar e ajudar durante todo esse trabalho;
- Aos Professores Dr. Baldoíno e Dr. Aydano pelo acompanhamento na banca examinadora, sugestões e incentivo;
- Ao Prof. Msc. Fabrício Cabral pelas sugestões e ajuda na definição da pesquisa;
- Aos meus amigos e colegas da UFAL que de forma direta ou indiretamente me ajudaram nesse processo, em especial Michel Miranda, Sebastião Rogério e Tássio Gonçalves pela ajuda em diversos momentos, com incentivos e conselhos.

RESUMO

O avanço da arquitetura em que as aplicações são oferecidas aos clientes atuais, baseada em computação em nuvem, contribuiu para que as aplicações com *streaming* de mídia passassem a ser uma das formas mais utilizadas de entrega de conteúdo. O presente trabalho tem como finalidade a investigação de tecnologias de desenvolvimento Web que possam auxiliar projetos de aplicações baseadas em *streaming* de mídia. Para tal, foi proposto um método que é aplicado no contexto desse trabalho como um caso teste. Entre as diversas etapas, é feito um estudo comparativo das principais tecnologias (linguagem de programação e *framework*) utilizadas atualmente, destacando as duas mais promissoras pra uma comparação mais detalhada: uma tecnologia baseada em Java e outra baseada em Python. A seleção de uma arquitetura de referência a ser utilizada como base para o projeto e implementação das aplicações de teste. Também é utilizado um método de avaliação de desempenho e escalabilidade através *benchmarking*, onde foram definidos cenários com 10, 100 e 1000 usuários executando pelo período de 20 minutos em um ambiente controlado. As tecnologias foram avaliadas em termos de uso de CPU, uso de memória RAM, tempo de resposta e taxa de transferência com uso de pseudo *streaming*. Como resultado da aplicação do método, temos que Java tende a ter melhor desempenho em algumas métricas, à medida que a quantidade de usuários cresce; enquanto Python se mostra constante e uma boa solução em cenários com menos usuários.

Palavras-chaves: *Streaming* de Mídia. Apoio ao Desenvolvedor. Escolha de Tecnologia. Desempenho. *Throughput*. Avaliação.

ABSTRACT

The advancement of architecture in which applications are offered to current customers, based on cloud computing, contributed to applications with streaming media have become one of the most used forms of delivery content. This research has purpose to research web development technologies that can assist projects of web applications for streaming media. For this, a method was proposed that is applied in the context of this work as a test case. Among the several steps, a comparative study was done on several sources of which technologies (programming language and framework) are more promising and selected two: a technology based on Java and another based on Python. It selected a reference architecture to serve as the basis for the design and implementation of experiments. Through a performance evaluation method, benchmarking, the scenarios were defined with 10, 100 and 1000 users running the 20 minute period in a controlled environment. The technologies were evaluated in terms of CPU, RAM, response time and throughput with the use of pseudo streaming. As a result of the application of the method, we have that Java tends to perform better in some metrics, as the number of users grows; While Python is steady and a good solution in scenarios with fewer users.

Key-words: Streaming Media. Developer Support. Choice of Technology. Performance. Throughput. Evaluation.

LISTA DE ILUSTRAÇÕES

Figura 1 – *Streaming* Crescimento (CISCO, 2016). 12
Figura 2 – HTTP *Streaming* (BEGEN; AKGUL; BAUGHER, 2011). 23
Figura 3 – RTP (BEGEN; AKGUL; BAUGHER, 2011). 24
Figura 4 – Métricas de Desempenho. 26
Figura 5 – Funcionamento JSF (ORACLE, 2013) 31
Figura 6 – Método para Seleção da Tecnologia. 34
Figura 7 – TIOBE *Ranking* (TIOBE, 2016). 39
Figura 8 – CodeEval *Ranking* (CODEEVAL, 2016). 40
Figura 9 – RedMonk *Ranking*(REDMONK, 2016). 41
Figura 10 – IEEE Spectrum *Ranking*(IEEE SPECTRUM, 2016). 42
Figura 11 – BuiltWith *Ranking* por linguagem de programação (BUILTWITH, 2016). . . 43
Figura 12 – *Hotframeworks Ranking* (HOTFRAMEWORKS, 2016) 44
Figura 13 – Spotify 49
Figura 14 – Spotify Arquitetura (YANGGRATOKE et al., 2012) 50
Figura 15 – Netflix Web 51
Figura 16 – Netflix Arquitetura Web (ADHIKARI et al., 2012) 52
Figura 17 – Amazon Cloud *Web* (TAVIS; FITZSIMONS, 2012) 53
Figura 18 – Arquitetura da Aplicação Web 54
Figura 19 – Diagrama de Caso de Uso 56
Figura 20 – Diagrama de Classes 57
Figura 21 – Tela Inicial em Java 59
Figura 22 – Tela Inicial em Python 59
Figura 23 – *Benchmarking* CPU e Memória para 10 Usuários 65
Figura 24 – *Benchmarking* CPU e Memória para 100 Usuários 66
Figura 25 – *Benchmarking* CPU e Memória para 1000 Usuários 67
Figura 26 – *Benchmarking* Tempo de Resposta para 10 Usuários 68
Figura 27 – *Benchmarking* Tempo de Resposta 100 Usuários 69
Figura 28 – *Benchmarking* Tempo de Resposta 1000 Usuários 70
Figura 29 – *Benchmarking* Taxa de Transferência 10 Usuários 71
Figura 30 – *Benchmarking* Taxa de Transferência 100 Usuários 72
Figura 31 – *Benchmarking* Taxa de Transferência 1000 Usuários 73

LISTA DE TABELAS

Tabela 1 – Atividades do Projeto. 15
Tabela 2 – Classificação de Técnicas de Avaliação de Desempenho (JOHN, 2007). . . . 27
Tabela 3 – *Benchmarks* (JOHN, 2007). 29
Tabela 4 – Conversão *Frameworks* para Linguagens de Programação. 45
Tabela 5 – *Ranking* Hotframeworks Adaptado 45
Tabela 6 – *Ranking* Buildwith Adaptado 45
Tabela 7 – *Ranking* Resumido de Linguagens de Programação e *Frameworks*. 46
Tabela 8 – Ocorrência de Popularidade das Linguagens de Programação 46
Tabela 9 – Cenários de Execução 55
Tabela 10 – Requisitos do Sistema 56
Tabela 11 – Criação de Usuários 63

LISTA DE ABREVIATURAS E SIGLAS

AJAX	*Asynchronous Javascript and XML*
API	*Application Programming Interface*
AWS Cloud	*Amazon Web Service Cloud*
CDN	*Content Delivery Network*
CPU	*Central Processing Unit*
DAO	*Data Acess Objeto*
DASH	*Dynamic Adaptive Streaming over HTTP*
HD	*High Definition*
GWT	*Google Web Toolkit*
HTML	*Hypertext Markup Language*
HTTP	*Hypertext Transfer Protocol*
HLS	*HTTP Live Streaming*
HDS	*HTTP Dynamic Streaming*
IaaS	*Infrastructure as a Service*
IDE	*Integrated Development Environment*
JDBC	*Java Database Connectivity*
Java EE	*Java Enterprise Edition*
Java ME	*Java Micro Edition*
Java SE	*Java Standard Edition*
JSF	*Java Server Faces*
JSF	*Java Server Pages*
JVM	*Java Virtual Machine*
JIT	*just-in-time*
MSS	*Microsoft Smooth Streaming*

POO	Paradigma Orientada a Objetos
PaaS	*Plataform as a Service*
PSF	*Python Software Foundation*
PVM	*Python Virtual Machine*
RTP	*Real-time Transport Protocol*
SaaS	*Software as a Service*
SDP	Session Description Protocol
SO	Sistema Operacional
SOA	Service-Oriented Architecture Session Description Protocol
UI	*User Interface*
URL	Uniform Resource Locator
UTP	*Unshielded Twisted Pair*
WEB	*World Wide Web*

SUMÁRIO

1 INTRODUÇÃO 11
1.1 Objetivo da Pesquisa 13
1.2 Metodologia de Trabalho 14
1.3 Trabalhos Relacionados 15
1.4 Organização do Texto 18

2 STREAMING, AVALIAÇÃO DE DESEMPENHO E TECNOLOGIAS WEB . . . 20
2.1 Streaming 20
2.1.1 Mecanismos 20
2.1.2 Métodos e Protocolos 21
2.1.3 Content Delivery Network 24
2.2 Avaliação de Desempenho em Aplicações Web 25
2.2.1 Métricas 25
2.2.2 Técnicas 26
2.2.3 Workloads e Benchmark 27
2.3 Tecnologias Web 29
2.3.1 Tecnologias Baseadas em Java 29
2.3.2 Tecnologias Baseadas em Python 31

3 MÉTODO PARA SELEÇÃO DA TECNOLOGIA DE DESENVOLVIMENTO ADOTADA 34
3.1 Critérios Adotados 34
3.1.1 Identificar as Linguagens mais Populares 34
3.1.2 Definir Funcionalidade Alvo da Aplicação 35
3.1.3 Definir Arquitetura de Referência 35
3.1.4 Definir Cenários de Utilização 36
3.1.5 Implementar Aplicação de Teste 36
3.1.6 Executar Benchmarking e Analisar Resultados 37

4 SELEÇÃO DAS LINGUAGENS 38
4.1 Popularidade das Linguagens de Programação 38
4.2 Frameworks para Aplicações com Streaming de Mídia 42
4.3 Identificando as Tecnologias Mais Promissoras 44

5 ARQUITETURAS DE REFERÊNCIA E ESPECIFICAÇÃO DA APLICAÇÃO DO EXPERIMENTO 47

5.1 Objetivo do Sistema ***47***
5.2 Análise de Domínio ***47***
5.2.1 Spotify *48*
5.2.2 Netflix *50*
5.3 Arquitetura do Sistema ***54***
5.4 Cenários ***55***
5.5 Requisitos ***55***
5.6 Diagrama de Caso de Uso e Classe ***55***
5.7 Implementações das Aplicações ***58***

6 BENCHMARKING E RESULTADOS ***61***
6.1 Benchmark ***61***
6.2 Ambiente de Execução ***62***
6.3 Execução ***63***
6.4 Análise dos Resultados ***64***
6.4.1 Benchmarking de Uso de CPU e Memória *64*
6.4.2 Tempo de Resposta *66*
6.4.3 Taxa de Transferência *69*

7 CONCLUSÃO ***74***
7.1 Trabalhos futuros ***74***

Referências ***76***

APÊNDICES ***80***

APÊNDICE A – CÓDIGO DA IMPLEMENTAÇÃO JAVA ***81***

APÊNDICE B – CÓDIGO DA IMPLEMENTAÇÃO PYTHON ***89***

APÊNDICE C – PLANO DE TESTE JMETER ***96***

1 INTRODUÇÃO

A atual conjuntura dos sistemas computacionais aponta claramente para uma tendência de migração de um modelo cliente/servidor para um modelo de computação em nuvem (*cloud computing*). No primeiro, as informações são processadas em um local físico previamente conhecido, em que as aplicações são executadas em uma máquina remota, com comunicação via rede, normalmente uma rede local. No segundo, as informações e o processamento estão distribuídos geograficamente e anônimos, e as aplicações são executadas remotamente através da *World Wide Web* (Web), diversas tecnologias estão envolvidas nesse modelo, tais como, a própria Intenet, virtualização, computação em *gride*, serviços *on-demand*. Diversos fatores contribuíram para o avanço da computação em nuvem, e consequentemente da Web. A Internet passou a ser vista como uma plataforma para *software*, sendo centrada na tecnologia, usuário e negócio (CHOUDHURY, 2014). A tecnologia tem sua influência nesse avanço, além das citadas anteriormente, temos o desenvolvimento de aplicações Web*, Web Services, Service-Oriented Architecture* (SOA) *Asynchronous Javascript and XML* (AJAX), bancos de dados distribuídos, *streaming* de mídia, etc (ZHANG, 2008). Outras influências tecnológicas também consideradas fundamentais são o avanço na velocidades de conexão de Internet e a quantidade de novos tipos de dispositivos, tais como *smartphones* e *tablets* (TIMMERER; GRIWODZ, 2012). Já a influência relacionada aos usuários reside na facilidade de acesso a diversos *softwares* e recursos de onde quer que estejam, que tem se tornado cada mais natural e até mesmo essencial para os usuários. O mercado também não poderia deixar de influenciar, onde muitos negócios passaram a desenvolver aplicações orientadas a serviços que envolvem *streaming* de mídia, tais como Youtube, Vimeo, Spotify, Netflix e Google Player Music.

No aspecto dos serviços que são fornecidos através da arquitetura da computação em nuvem, podem ser disponibilizados em três níveis diferentes de abstração, chamados aqui de camadas: *Infrastructure as a Service* (IaaS), *Platform as a Service* (PaaS) e *Software as a Services* (SaaS). Na camada inferior tem-se o IaaS que provê o uso de partes físicas, como servidores e também é voltado para o desenvolvimento de aplicações proprietárias; na camada do meio a PaaS se utiliza de uma arquitetura virtual de *hardware* para oferecer um serviço. E a camada mais acima e que se destaca é a SaaS, ela pode prover acesso via Web de aplicações que permitem ao cliente adicionar novas funcionalidades de acordo com sua necessidade (LI, 2015). Essa camada permite que o cliente pague pelo uso do serviço em diversos dispositivos, como *smartphone, smartTV, desktop, tablet,* etc., não mais pelo produto instalado no dispositivo. Dessa forma, esse tipo de serviço tem a vantagem de não mais necessitar de instalação e gerenciamento do *software* em rede local, a não ser um cliente para acessar as informações contidas na base de dados, sem contar que é uma solução com

alta portabilidade, chegando a ser independente de sistema operacional (LI, 2015).

Fazendo uso das tecnologias que permeiam a computação em nuvem como citado anteriormente, um serviço que tem se destacado na atualidade é o serviço de *streaming* de mídia. Apesar de ser uma tecnologia concebida há muitos anos (FORTINO et al., 2002), diversas empresas têm emergido no cenário mundial com serviços baseados em *streaming* de mídia, que já aderem milhões de usuários, principalmente para serviços de áudio e vídeo, chegando a somar cerca de 50% do tráfego na Internet nos Estados Unidos. Para se ter uma ideia, só o Netflix é responsável por cerca de 30% desse tráfego (TIMMERER; GRIWODZ, 2012). E mais, existe uma grande tendência de que serviços de *streaming* de vídeo e áudio cresçam ainda mais no passar dos anos, conforme podemos observar no gráfico da Figura 1 publicado no relatório anual da Cisco (CISCO, 2016).

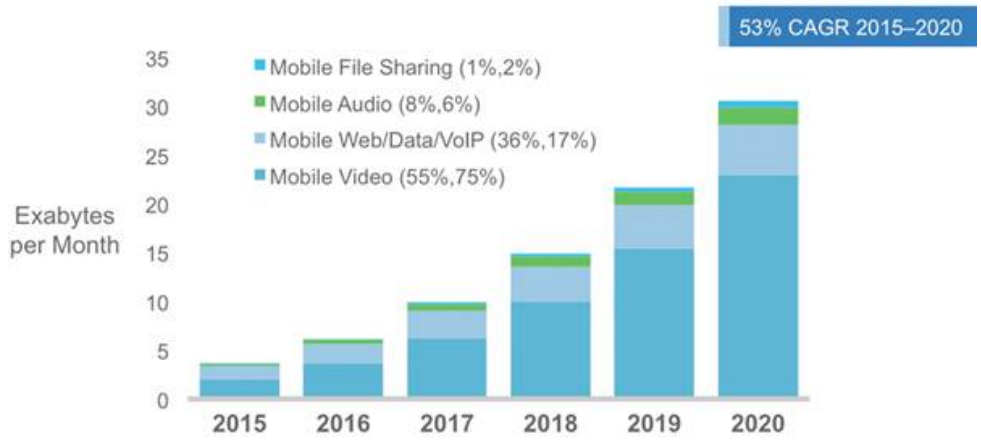

Figura 1 – *Streaming* Crescimento (CISCO, 2016).

Para trabalhar com esse tipo de serviço, podem surgir algumas dificuldades. Por exemplo, dados de arquivos multimídia, consistindo em imagem, áudio e vídeo, requerem grande quantidade de espaço de armazenamento e velocidades de transferências elevadas. Além disso, a consistência na sincronização é fundamental. A forma de representar o objeto do banco de dados com eficiência, a arquitetura e modelo do banco de dados, a eficiência na recuperação do arquivo de multimídia, também são outros fatores (KHAN; AHIRWAR, 2011). Pode-se também ter a situação de dispositivos móveis que recebem dados multimídia em diversos ambientes de conexão, sendo, em algumas situações, perdida a conexão levando a problemas de sincronização e consistência (KHAN; AHIRWAR, 2011). As dificuldades não param por aí. Na área de Engenharia de Software também existem desafios, alguns deles compartilhados com o desenvolvimento de aplicações tradicionais. Segundo (YAU; AN, 2011), foram encontrados sete problemas principais quando se fala em desenvolvimento de aplicações em nuvem, são eles: (1) confidencialidade e integridade; (2) confiança e disponibilidade; (3) segurança em um ambiente *multitenant*; (4) desconhecido perfil de risco; (5) monitoramento de QoS; (6) dinamismo da arquitetura fruto de computação móvel; e (7) questões legais.

O desempenho de uma aplicação também é fator relevante, podendo ser encarado como um problema em diversos domínios, quando falamos em aplicação Web de *streaming*, o desempenho pode variar de acordo com o domínio do negócio e as tecnologias adotadas para seu desenvolvimento, dentre elas, podemos destacar a linguagem de programação, *codecs*, mecanismo e método de *streaming* (JEON; JANG, 2008; TIMMERER; GRIWODZ, 2012).

Para desenvolver um projeto de uma aplicação Web diversos aspectos são considerados, além dos citados por Yau e An (2011) e Khan e Ahirwar (2011). Dois aspectos fundamentais são a escolha da arquitetura e de tecnologias adequadas ao tipo de projeto. Para isso, se faz necessário avaliar o impacto de tais tecnologias a fim de fornecer aos envolvidos no projeto informações sobre o comportamento delas e quais são melhores e mais adequadas de acordo com as características da aplicação a ser desenvolvida (JAIN, 1991). A escolha da tecnologia utilizada pode ser fundamental no sucesso do serviço, podendo influenciar várias decisões de projeto do sistema (JAIN, 1991).

O estudo proposto neste trabalho tem por finalidade propor um método para auxiliar o desenvolvedor de aplicações de *streaming* de mídia a selecionar tecnologias adequadas, que atendam às suas necessidades e acompanhem as tendências do mercado. O método proposto contempla a seleção de uma arquitetura de referência baseada nos casos de sucesso recentes, além de apoiar a escolha da linguagem de programação e do *framework* de desenvolvimento, a partir de critérios previamente definidos no método, como popularidade. Para isso, após a identificação de uma arquitetura de referência, foram projetadas e implementadas duas aplicações Web desenvolvidas em linguagens de programação e *frameworks* diferentes. Tais aplicações passaram por uma avaliação criteriosa em cenários realistas utilizando *benchmarking* com quantidade variável de usuários. As tecnologias foram avaliadas em termos de uso de CPU, uso de memória RAM, tempo de resposta e taxa de transferência com uso de pseudo *streaming*. Os resultados mostraram que a escolha da tecnologia mais adequada depende das características da linguagem, métrica, tempo e a quantidade de usuários simultâneos.

1.1 OBJETIVO DA PESQUISA

Objetivo Geral

O objetivo do trabalho proposto é auxiliar a tomada de decisões de projeto no desenvolvimento de aplicações Web com *streaming* de mídia, auxiliando desde o projeto arquitetural à seleção de tecnologias adequadas ao desenvolvimento do serviço.

Objetivo Específico

- Identificar uma arquitetura de referência para o desenvolvimento de aplicações Web com *streaming* de mídia;
- Identificar as principais tecnologias com maior potencial de mercado;
- Comparar as tecnologias selecionadas em termos de desempenho e escalabilidade, por meio de um experimento com aplicações reais.

1.2 METODOLOGIA DE TRABALHO

O projeto pretende investigar as arquiteturas de *software* e o desempenho de aplicações Web na camada SaaS, ou seja, através do navegador, que irão consumir dados via *streaming* de mídia, em diferentes cenários realistas, considerando inclusive o impacto tecnológico da adoção de diferentes linguagem de programação e *frameworks* Web. Para esta investigação, o projeto passará por diversas fases para alcançar o seu objetivo, conforme apresentado na Tabela 1.

Tabela 1 – Atividades do Projeto.

Atividade	Descrição
1	Realizar uma revisão da literatura sobre os estilos arquiteturais utilizados pelas principais aplicações de *streaming* de mídia atuais, a fim de propor uma arquitetura de referência para a aplicação dos experimentos.
2	Realizar uma revisão da literatura sobre as principais tecnologias utilizadas no desenvolvimento de aplicações Web, assim como, tecnologias de *streaming*. Esse estudo inicial é essencial para o aprofundamento sobre o domínio, fatores e sujeitos que rodeiam o contexto do projeto.
3	Realizar uma revisão da literatura sobre avaliação de desempenho e escalabilidade, para poder entender como funciona todo o processo de avaliação. E com isso entender como funciona a avaliação, e poder selecionar a melhor forma de avaliar.
4	Avaliar as arquiteturas de *software* das principais aplicações, e com base neles identificar a arquitetura de referência para as aplicações do experimento.
5	Selecionar tecnologias de acordo com o método proposto que tenha maior potencial de mercado (linguagem de programação e *framework*), dentre as soluções identificadas na revisão da literatura.
6	Projetar e implementar as aplicações Web que usem *streaming* de mídia, com base nas linguagens de programação previamente selecionadas.
7	Executar a avaliação de desempenho e escalabilidade das aplicações desenvolvidas, em relação ao consumo de CPU, uso de memória, tempo de respostas e taxa de transferência para quantidades variáveis de usuários em diferentes cenários realistas. E assim, analisar os pontos positivos e negativos de cada tecnologia.
8	Escrever artigos e dissertação de mestrado.

1.3 TRABALHOS RELACIONADOS

Na literatura existem diversos trabalhos que tratam de *benchmarking* de aplicações nos mais variados tipos de sistemas, Collier e Meyer (2000), Fourment e Gillings (2008) e Kreitz (2011) são exemplos. Alguns dentre eles comparam o impacto da linguagem de programação no desenvolvimento de determinado sistema (COLLIER; MEYER, 2000), buscando resultados em diversos aspectos, tais como consumo de memória, tempo de respostas, número de linhas de códigos, etc. No âmbito de linguagens de programação ainda existem *benchmarkings* mais especializados, como é o caso de aplicações Web (GROSS; EMMELMANN; WOLISZ, 2007), onde existe uma infraestrutura de *middleware* para viabilizar tal aplicação, como por exemplo um servidor Web específico para uma determinada linguagem de programação. Também existem trabalhos com foco em *benchmarking* para *streaming* de mídia, mas em menor quantidade (KREITZ, 2011). Quase que a totalidade dos trabalhos identificados que lidam com avaliação de sistemas de *streaming* não fazem comparação entre as tecnologias, mas sim em avaliar sistemas específicos. Além disso, os aspectos considerados na avaliação

se concentram principalmente em requisitos de redes, não considerando questões gerais de desempenho e escalabilidade. Mesmo assim, como apontam os trabalho de Kreitz (2011) e Adhikari et al. (2012), são poucas as pesquisas existentes na literatura que consideram avaliação de *streaming* de mídia.

Primeiro destacaremos os trabalhos que estão relacionados a *benchmarking* de linguagens de programação e *framworks*.

O trabalho de Collier e Meyer (2000) fez uma comparação empírica sobre diversas linguagens de programação, entre elas, C, C++, Java, Perl, Python, Rexx e Tlc, para os respectivos aspectos: tamanho do programa; esforço de programação; eficiência de execução; uso de memória e confiabilidade. Para isso as linguagens foram divididas em duas categorias: *scripting* e sem *scripting*. Foram feitas 80 implementações do mesmo conjunto de requisitos por diversos tipos de programadores, rodando na mesma configuração de *hardware*.

Com o estudo, Collier e Meyer (2000) chegaram as seguintes conclusões: projetar e escrever programas em Perl, Python, Rexx ou Tlc é quase metade do tempo que leva para escrever o mesmo programa em C, C++ ou Java; o consumo de memória de uma linguagem de *scripting* é duas vezes maior do que em C e C++; para fase de leitura inicial C e C++ é mais rápido 4 vezes em relação a Java e de 5 a 10 comparado com outras linguagens; na fase principal do programa C e C++ é 2 vezes mais rápido que Java e as linguagens de *scripting* tendem a ser mais rápidas que Java; dentro das linguagens de *scripting* Python e Perl são mais rápidas do que Tlc nas duas fases; a variabilidade de desempenho devido a diferentes programadores é maior que a variabilidade devido a diferentes linguagens.

Como resultado os estudos apontam alguns dados interessantes: C e C++ foram as implementações mais rápidas, com pouco uso de memória, porém são programas com mais linha de códigos; Java e C# estão compreendidas entre a flexibilidade de Perl e Python e o desempenho rápido de C e C++; O desempenho testado nas diversas linguagens não mudaram de acordo com sistema operacional.

O estudo de Gross, Emmelmann e Wolisz (2007) tem a finalidade de analisar o desempenho de aplicações Web, que necessitam de um conjunto de tecnologias para executar. Tendo em vista que aplicações Web tem muita interatividade, o bom ponto de partida é a análise desempenho, sendo um requisito chave.

Para o trabalho foram selecionadas quatro tecnologias: Java Servlets, Java Server Pages (JSP), CGI/C++ e FastCGI/C++. Os testes de comparação são feitos em dois casos: uma é um estudo de caso de uma aplicação Web real, desenvolvida considerando requisitos reais do sistema; a outra é uma aplicação trivial. A finalidade desses dois experimentos foi medir o desempenho tanto em uma aplicação mais complexa, quanto em uma aplicação simples. Como metodologia foi adotado o teste de *stress* e medida.

As duas suítes de teste apresentaram resultados importantes para o desempenho das

diferentes tecnologias avaliadas. A seguir estão as considerações do estudo realizado: Java Servlets teve o pior desempenho, porque a implementação e tecnologias envolvem vários processos até a execução final do programa, como compilação, intepretação, etc.; JSP tem uma melhora considerável em relação a Java Servlets, por não ter que passar por todo o processo até a execução real do programa; CGI/C++ tem resultados similares aos de Java Servlets, mas por executar de maneira otimizada tem um desempenho melhor; O FastCGI consegue diminuir mais o *overhead* do CGI/C++ o deixando com desempenho ainda melhor.

Os próximos trabalhos estão relacionados a avaliação de desempenho em sistemas de *streaming*, dentre eles apenas um está relacionado a *benchmarking* entre tecnologias em aplicação envolvendo *streaming* de mídia.

O estudo elaborado por Summers et al. (2012) busca criar ferramentas, *workloads* e *benchmarking* para aplicações que usem *streaming* de vídeo. Com base no que existe de *workloads* na área e nos propostos no trabalho, são criados exemplos de *benchmarking* para avaliar o desempenho em três servidores Web: Apache, nginx e userver. O trabalho tem como principal motivação a falta de critério na avaliação de desempenho de *Web servers* para *streaming* de mídia. O foco do trabalho é avaliar os servidores Web como fornecedores de *streaming*, em termos de *throughput*, desempenho de disco, falha nos *chunks*, e analisar quais os pontos benéficos no uso do *Hypertext Transfer Protocol* (HTTP).

Como resultado, os autores chegaram à conclusão de que a escolha do *Web server* tem um impacto substancial no desempenho de *streaming* de vídeo, principalmente no que se refere ao tempo de acesso ao disco. Uma pequena modificação no servidor usever para acesso ao disco proporcionou um benefício considerável no desempenho no contexto geral.

Kreitz (2011) define que a eficiência de sistemas de *streaming* pode ser medida avaliando o servidor, o cliente, ou ambos, nos aspectos de consumo de CPU, uso de memória, armazenamento e em termo de requisitos de rede, como *throughput*, servidor, *link*, perca de pacotes, etc. Em seu trabalho ele avalia aplicações Web com *streaming* de áudio no contexto de uma aplicação específica, o Spotify da Suécia. São considerados aspectos relacionados a rede e o sistema de armazenamento. Não houve *benchmarking* comparativo entre diferentes aplicações, porém os testes buscam analisar aspectos relacionados ao Spotify. Entre as conclusões do trabalho está que a metodologia aplicada para alguns aspectos podem ser aplicado também para outros sistemas.

Por fim, o trabalho de Adhikari et al. (2012) apresenta uma visão geral da arquitetura de *streaming* do Netflix através de medidas para descobrir como funciona o seu sistema de *streaming*. Ele analisa e avalia os *Content Delivery Network* (CDN) utilizados pelo Netflix, chegando a análise que o Netflix mantem sua estrutura em camadas para cada usuário, fazendo com que a sua conta permaneça inalterada por alguns dias. O centro da investigação é baseada em quatros questões sobre o Netflix: qual a desempenho dos CDN em termos de largura de banda para demanda de vídeo em alta qualidade; a quanto um CDN é diferente de

outros em termos de desempenho e se existe transparência em um ser melhor que outro; a estratégia de atribuição atual da Netflix consegue ser "ótima" até que ponto; e se é possível alguma outra estratégia para garantir um melhor uso da largura de banda na entrega. Em posse dessas questões os autores executaram experimentos com métricas diferentes em diversos dias e chegarem a algumas conclusões, dentre que a método de escolha do CDN do Netflix melhora o consumo de banda em até 12% se comparado o abordagem de CDN estático.

Como pode-se observar alguns trabalhos de avaliação de desempenho comparam linguagens de programação em ambiente *desktop* e Web; esse último através de servidores Web. Tais *benchmarking* tem foco mais específico, como é o caso de Summers et al. (2012), e também mais genérico, como é caso do estudo de Gross, Emmelmann e Wolisz (2007) e Collier e Meyer (2000). Há pouco de avaliação de desempenho relacionado a *streaming* de mídia na literatura. Kreitz (2011) e Adhikari et al. (2012) foram um dos poucos que pesquisaram sobre o tema, e como falado em parágrafos anteriores, avalia no aspecto de uma aplicação em si, sem perceber o impacto do uso de diferentes tecnologias.

O presente trabalho busca avaliar o desempenho de uma maneira mais específica, além do impacto da linguagem de programação em aplicações Web com *streaming* de mídia, motivado pela ausência de *benchmarking* nesse campo de estudo, assim como foi a motivação do trabalho de Summers et al. (2012). Outra motivação é a carência de pesquisas no campo de eficiência de sistema de *streaming* apontado nos trabalhos de Kreitz (2011) e Adhikari et al. (2012). Summers et al. (2012) trabalha com comparação de alguns servidores Web, com foco no servidor de mídia HTTP que irá fornecer o conteúdo *streaming* para um cliente. Já a presente pesquisa tem o objetivo de avaliar o impacto da tecnologia, tais como linguagens de programação e *frameworks*, nas aplicações Web que usem *streaming* de mídia.

1.4 ORGANIZAÇÃO DO TEXTO

O restante desta dissertação está estruturado da seguinte forma:

- **Capítulo 2 – *Streaming*, Avaliação de Desempenho e Tecnologias Web** Nesse capítulo são descritas as principais tecnologias que estão inseridas no contexto de *streaming* de mídia. São abordados aspectos sobre avaliação de desempenho em sistemas, incluindo técnicas, métricas e procedimentos recomendados. Assim como as tecnologias de desenvolvimento Web que podem ser usadas para desenvolver aplicações com *streaming* de mídia.

- **Capítulo 3 – Método para Seleção da Tecnologia de Desenvolvimento Adotada.** Nessa parte do trabalho é apresentando com maior detalhe a composição do método adotado.

- **Capitulo 4 – Seleção das Linguagens.** Nesta capítulo é aplicado parte do método referentes as linguagens de programação e *frameworks* que estão em destaque na atualidade, identificando as mais promissoras, definindo os critérios de escolha, e então determinando quais farão parte da aplicação teste do trabalho.

- **Capitulo 5 – Arquiteturas de Referência e Especificação da Aplicação do Experimento.** Aqui são apresentadas as arquitetura dos principais sistemas Web de *streaming*, e definido a arquitetura de referência para as aplicações Web. São abordadas também algumas fases do desenvolvimento dos sistemas do experimento, tais como objetivo do sistema, levantamento de requisitos, diagrama de casos de uso, diagrama de classes de análise e implementação das aplicações.

- **Capítulo 6 – *Benchmarking* e Resultados.** Neste capítulo é apresentada a última etapa de aplicação do método, com a condução e execução dos experimentos, através de *benchmarking* entres os sistemas desenvolvidos, de acordo com os cenários definidos. Também são apresentados os resultados obtidos na avaliação de cada uma das aplicações em termos de consumo de CPU, uso de memória, tempo de resposta e taxa de transferência.

- **Capítulo 7 – Conclusão.** Por fim, são apresentadas algumas considerações finais do estudo em questão, assim como direcionamentos para trabalhos futuros.

2 *STREAMING*, AVALIAÇÃO DE DESEMPENHO E TECNOLOGIAS WEB

Neste capítulo é apresentada uma revisão da literatura sobre *streaming* de mídia e avaliação de desempenho em sistemas computacionais. Primeiro começando pelo contexto atual do uso de *streaming*, formas de acesso, o processo de *streaming* e quais são as tecnologias que são empregadas nesse tipo de aplicação. Em seguida são apresentados métodos de avaliação de desempenho, incluindo uma visão geral do procedimento de avaliação, métricas, técnicas, *workload* e *benchmarking*. Por fim, uma breve apresentação das tecnologias Web que estão no contexto do trabalho.

2.1 *STREAMING*

O uso de áudio e vídeo, ou melhor, de multimídia, via *streaming* teve seus primórdios no início de 1990, e teve uma evolução considerável no passar dos anos, se tornando uma das principais tecnologias da computação em nuvem da atualidade, principalmente para serviços Web de áudio e vídeo. Estatísticas afirmam que cerca de 350.000 horas de programação ao vivo e sob demanda são usadas através da Internet (CONKLIN et al., 2001). Nos jogos de inverno de *Vancouver* no Canadá, foram mais de 300 eventos para os telespectadores disponível na Web, entre *on live* e *on demand*, o que resultou em mais de 6.2 petabytes de tráfego de dados em duas semanas de jogos (BEGEN; AKGUL; BAUGHER, 2011).

Como o *streaming* passou a ser um dos tipos de aplicações Web mais importantes, veio a tona diversas questões, por exemplo, a eficiência que uma aplicação de *streaming* pode oferecer ao usuário final já que a carga adicionada à infraestrutura de internet é bem maior do que o normal. Assim, como construir um sistema de *streaming* eficiente é uma pergunta bem interessante dando espaço para pesquisas rica em seus resultados (KREITZ, 2011).

Quando falamos em sistema *streaming* é importante ressaltar que existem mecanismo e métodos para transmitir a mídia, que veremos com mais detalhes nas próximas seções. Os mecanismos se dividem em dois, *live streaming* e *on-demand*. Os Método para *streaming* são basicamente divididos em três: os que funcionam diretamente com a internet, ou melhor, com protocolo HTTP, pseudo *streaming* e HTTP *Streaming*; e por fim, temos outro método sem uso HTTP mas com protocolos nativo de *streaming* (GUO et al., 2005).

2.1.1 Mecanismos

Como discutido anteriormente, *streaming* de mídia tem dois mecanismos distintos de prover acesso aos dados através da rede, *live streaming* e *on-demand streaming*, que é a

forma em que os dados chegam aos clientes para serem executados por algum *player*. Os dois podem ser diferenciados pela categoria de disponibilidade da mídia que está sendo oferecida. Também, em geral, encontra-se na literatura sistemas que são focados em um dos dois mecanismos, porém existe possibilidade de haver uma aplicação que funcione com os dois mecanismos (KREITZ, 2011).

O *live streaming* cuida da entrega do conteúdo em tempo real, geralmente ocorrido em transmissões ao vivo. Neste caso o usuário não tem controle sobre o *streaming*, não podendo avançar, retroceder ou pausar ao seu critério, já que o conteúdo é ao vivo. Os usuários que usam esse mecanismo tem acesso ao mesmo objeto que todos os outros ao mesmo tempo. Esse processo é um pouco mais complexo do que *on-demand streaming*, pois precisa de alguns passos antes da mídia ser transmitida. É necessário um equipamento para capturar da fonte analógica, por exemplo, vídeo, para então realizar a codificação por algum *codec* previamente selecionado nesse processo, para em seguida passar a mídia codificada para o servidor de *streaming*, que irá gerenciar esse conteúdo e servir as requisições vindas das aplicações dos usuários (CONKLIN et al., 2001).

On-demand é o mecanismo para a entrega de *streaming* de mídia sob demanda, ou seja, o conteúdo já está codificado em arquivo e armazenado em alguma mídia de armazenamento em um servidor. O conteúdo é então enviado ao cliente quando houver a requisição. Funcionando dessa forma esse mecanismo tem duas características que diferem do mecanismo anterior: cada usuário tem acesso a uma instância da mídia, e controle sobre o *player* que está executando o *streaming*, podendo inclusive ir para qualquer parte do conteúdo (KREITZ, 2011). Um bom exemplo de uso desse tipo de mecanismo são aplicativos de TV e música, tais como Netflix e Spotify (CONKLIN et al., 2001).

2.1.2 Métodos e Protocolos

Como falado anteriormente os protocolos de avaliação de *streaming* são basicamente divididos em três. O primeiro método e mais antigo meio de fazer *streaming* é através de *download* progressivo ou pseudo *streaming* muito usado pelo YouTube (BEGEN; AKGUL; BAUGHER, 2011). Ele é um método tradicional e conhecido, onde o próprio arquivo vai sendo baixado para o computador local e executado pelo *player*, permitindo que seja executada a mídia enquanto são baixados mais fragmentos do arquivo, além de possibilitar avançar para determinada posição da mídia (GUO et al., 2005). A grande maioria dos *media player* dão suporte a esse tipo de *streaming*, quando falamos de *player* em navegador Web alguns não tem suporte nativo e para funcionar com o pseudo *streaming* é preciso que o servidor de mídia ou servidor Web tenha essa funcionalidade, uma exceção é o *player* do *Hypertext Markup Language* (HTML) 5 que nativamente dá suporte ao pseudo *streaming* (JW PLAYER, 2016). Normalmente, a mídia que está sendo executada e baixada fica armazenada em uma pasta temporária (HICKSON; HYATT, 2011) e é excluída após o fim da execução. Esse método

usa HTTP sobre TCP e a execução da mídia começa logo que tenha uma boa quantidade de conteúdo da mídia em *buffer*. Dessa forma, o conteúdo é entregue do servidor para o cliente, sem levar em consideração diversos aspectos que podem influenciar nessa transferência e consequentemente na experiência do usuário final. Tais aspectos fazem com que esse método encontre diversos problemas nos dias atuais, por exemplo, o usuário tem que previamente escolher a qualidade e formato da mídia, ou ainda, a ocorrência de travamento quando se tentar avançar ou retroceder para determinada parte da mídia, que ainda não tenha sido carregada.

O outro método de transmissão utilizado para *streaming* através do HTTP é híbrido, ou seja, ele usa HTTP usado no método de pseudo *streaming* mas agora levando em consideração diversos aspectos na transmissão dos dados. O uso do HTTP para essa finalidade é chamado de *Adaptive Streaming* ou *HTTP streaming*, e começou a ter uma atenção maior nesses últimos anos, por diversas razões: HTTP é simples e executado facilmente em um *browser*; sem estado; é mais fácil de atravessar *firewall*; menor custo na infraestrutura para montar um servidor de *streaming* (SUMMERS et al., 2012; TIMMERER; GRIWODZ, 2012).

O *HTTP Streaming* basicamente funciona com fragmento ou *chunks* de arquivo codificado de mídia, onde esses fragmentos podem conter dados de vídeo, áudio, legendas, informação do programa ou metadados que ajudam o *software* a entender características da mídia. São armazenados em algum servidor HTTP, e transferidos para o cliente quando é requisitado levando em consideração aspectos da transmissão, tais como banda de transmissão. Assim ele pode se adaptar às adversidades e mudar o que vai ser enviado para melhorar a experiencia do usuário final. Algumas das adaptações que o *HTTP streaming* pode fazer é melhorar a qualidade de um vídeo quando houver condições para isso, diferente do pseudo *streaming*, de forma transparente e sem dar travadas, que é muito comum no pseudo *streaming*. A Figura 2 mostra como acontece a comunicação com HTTP *Streaming*. Primeiro que a mídia é dividido em vários fragmentos de tempo curtos codificados em *bitrates* diferentes e mantidos no servidor de mídia, a aplicação cliente após ter solicitado o *manisfest* para mídia em questão, informa ao servidor que inicie o *streaming* de forma rápida, ou seja, aquele fragmento com menor qualidade, e conforme alguns requisitos de tráfego do *streaming* são favoráveis ou não, a aplicação solicita ao servidor para aumentar ou diminuir a qualidade da mídia de acordo com suas condições de rede.

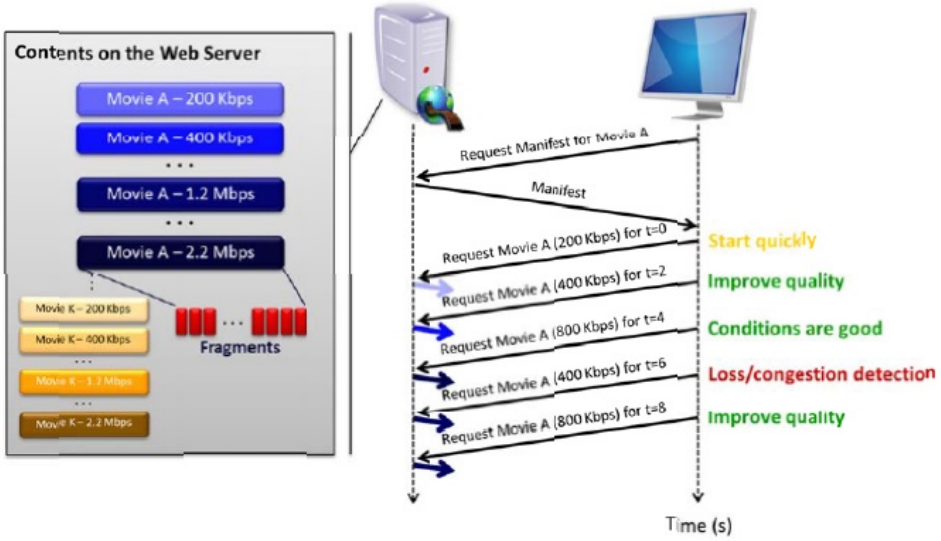

Figura 2 – HTTP *Streaming* (BEGEN; AKGUL; BAUGHER, 2011).

Existem basicamente quatro implementações do HTTP *streaming*, que são adotados por empresas de TV, *players* e outros serviços de *streaming*. São elas: HTTP *Live Streaming* (HLS) da Apple; *Microsoft Smooth Streaming* (MSS), da Microsoft; HTTP *Dynamic Streaming* (HDS), da Adobe; e *Dynamic Adaptive Streaming over* HTTP (DASH), do MPEG.

Deixando de lado o HTTP nós temos dois protocolos mais conhecidos projetados para funcionarem como método de transmissão de *streaming*, *Real-time Streaming Protocol* (RTSP) e *Real-time Transport Protocol* (RTP), que operam em cima do UDP. São protocolos específicos para *streaming* de mídia, ou seja, eles levam em consideração diversos aspectos para entregar o conteúdo para o usuário da melhor forma possível, como requisitos de rede, de usuário, de *player*, máquina, etc. Esse método permite o controle de fluxo dos pacotes graças ao UDP, mantem uma conexão com cliente e envia conteúdo de acordo com os requisitos de implementação tanto do cliente como do servidor, a partir do controle de fluxo os pacotes podem ser enviados com qualidades diferentes para manter uma melhor transmissão e execução do *streaming* (BEGEN; AKGUL; BAUGHER, 2011). Uma representação desse método pode ser vista na Figura 3, primeiro é a configuração da sessão, onde o servidor manda uma lista para o cliente dos *streams* e suas propriedades através do *Session Description Protocol* (SDP), o cliente por sua vez analisa seus requisitos e envia ao servidor informações através de RTSP do *stream* escolhido para o servidor poder enviar. Após o *streaming* está ocorrendo via RTP, em intervalos o cliente envia informações sobre seus requisitos e possíveis mudanças.

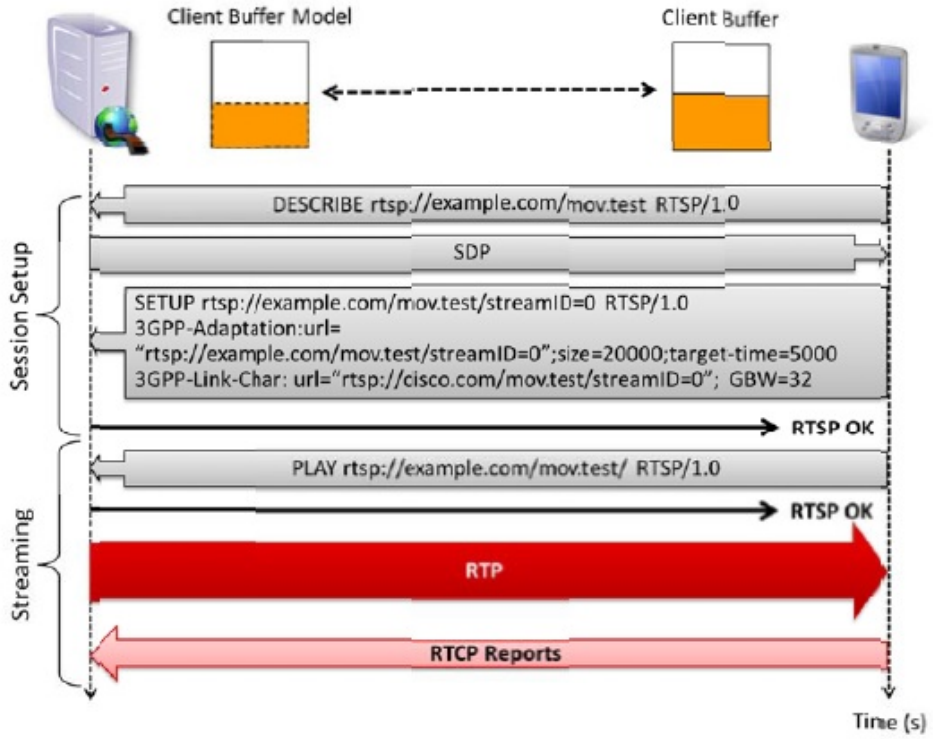

Figura 3 – RTP (BEGEN; AKGUL; BAUGHER, 2011).

2.1.3 *Content Delivery Network*

Content Delivery Network (CDN) vieram para ajudar a diminuir a carga de trabalho cada vez maior das aplicações Web, em relação a conteúdos dinâmicos geralmente multimídia, apesar de ser uma tecnologia que no passado foi usada para transmissão de conteúdo estático. Quando falamos em conteúdo estático, páginas visitas recentemente podem ser armazenadas em caches locais para serem usadas posteriormente por usuários na mesma rede, e assim fornecer o conteúdo de forma mais rápida, os primeiros CDN da década de 90 funcionam semelhantes a isso. Já conteúdos dinâmicos, com essa técnica, não tem tanta eficiência. Para conseguir algo semelhante aos caches para conteúdo dinâmicos os CDN reapareceram nos últimos anos, agora com novas características, tais como *streaming* e customização (MOLINA et al., 2004).

O funcionamento dos CDNs é baseada na localização geográfica do usuário, para isso os CDNs implementam técnicas que avaliam diversos aspectos para fornecer esse conteúdo, sempre visando a melhor experiência para o usuário, seja no sentido de desempenho, disponibilidade, escalabilidade, etc. Entre outras coisas os CDNs observam o perfil do usuário e a situação dos requisitos de rede para direcionar o usuário para obter os dados do CDN mais favorável no momento.

O mecanismo de direcionamento dos usuários é muito relevante para o CDN ser mais eficiente. Existem diversas possibilidades para o CDN direcionar, trabalhando em algumas

das camadas do TCP/IP e com alguns protocolos específicos. O TCP e UDP são protocolos antigos mas mesmo assim oferecem suporte suficiente para os CDNs desempenharem o seu trabalho. Outro protocolo importante para essa finalidade é o DNS, por tratar de resolução de nomes ele ajuda e muito a questão de roteamento entre a localização do usuário e o CDN que servirá o conteúdo em um determinado instante.

2.2 AVALIAÇÃO DE DESEMPENHO EM APLICAÇÕES WEB

Projetos de *software*, em suas fases inicias, tendem a sofrer influência das tecnologias que melhor se adaptam às necessidades do projeto, de acordo com seus requisitos. Para decidir o que deve ser utilizado, diversos parâmetros e características são considerados. No caso das tecnologias de *streaming*, pode-se realizar inclusive avaliação de desempenho para ter uma resposta em que situações que a tecnologia é adequada e então decidir qual solução escolher. Existem diversas técnicas e métricas de avaliação que podem ser selecionadas de acordo com a realidade de cada projeto para chegar a determinado objetivo (JAIN, 1991).

Avaliação de desempenho requer cuidado na sua execução, e para isso é importante ter conhecimento do sistema que foi projeto, assim como o cuidado na seleção apropriada de técnicas, métricas, *workloads* ou *benchmarks*, ferramentas e metodologias de avaliação de desempenho (JAIN, 1991).

2.2.1 Métricas

As métricas são, na verdade, critérios escolhidos pelos analistas de desempenho, de acordo com o projeto de avaliação, levando em consideração as características peculiares de cada sistema, e vão variar de acordo com esses fatores.

Para selecionar todas as métricas para avaliar, é preciso saber quais são os serviços oferecidos pelo sistema, e então selecionar quais métricas usar para a avaliação, lembrando que um serviço pode gerar diversos resultados. E esses resultados podem ser classificados em três categorias de acordo com a resposta do serviço. A Figura 4 apresenta um esquema dessa classificação. A primeira é que o serviço responde corretamente, o que também é chamado de velocidade, no qual é uma relação entre tempo, taxa e recurso, sendo para cada um desses uma medida, por exemplo, tempo de resposta, *throughput* e recursos disponíveis de um sistema funcionando em rede. Segundo, o serviço tem o resultado incorreto, que é chamado de confiabilidade, que pode ser a probabilidade do sistema executar um serviço com erro. E por último, o serviço está indisponível no momento da solicitação, chamado de disponibilidade, no qual podem ser medidos em um percentual de disponibilidade, com a palavra *down* quando o sistema está indisponível e *up* quando o sistema está disponível (JAIN, 1991).

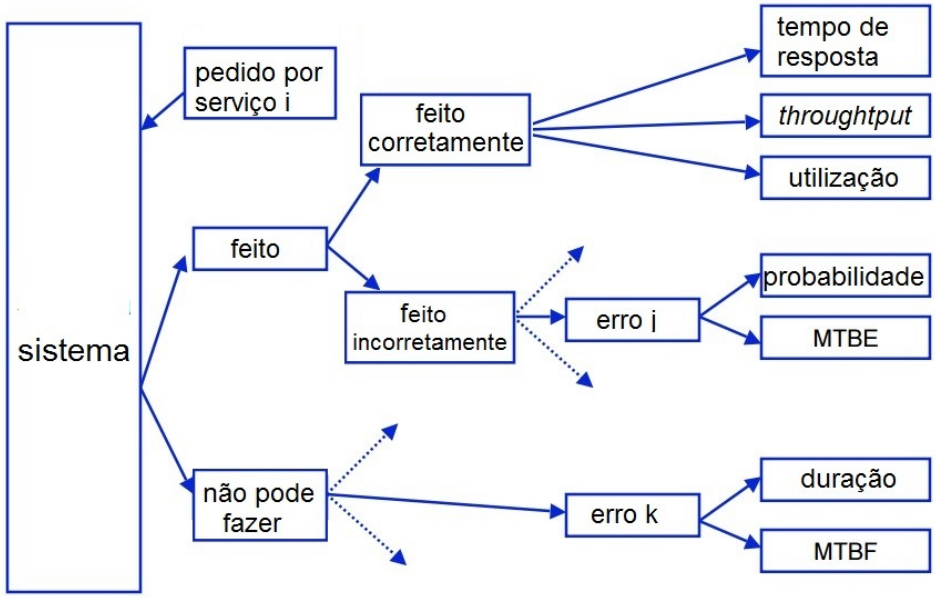

Figura 4 – Métricas de Desempenho.

2.2.2 Técnicas

Avaliação de desempenho pode estar ligada às várias fases do projeto de *software* (JAIN, 1991). Para cada fase existem técnicas de avaliação que melhor se adequam. Por exemplo, nas fases iniciais do desenvolvimento do *software* não se tem o *software* ou mesmo o *hardware* à disposição, e antecipar o desenvolvimento de tais recursos seria extremamente dispendioso nessas fases iniciais (JOHN, 2007). Por essa razão, nessas fases é recomendado o uso de técnicas baseadas em simulação. Em outras fases mais avançadas ou até quando o produto ou um protótipo funcional está pronto e à disposição, pode-se usar a técnica de medida de desempenho, para melhor entender o desempenho do sistema real já concebido. A Tabela 2 sintetiza as técnicas utilizadas para avaliação de desempenho, dependendo da disponibilidade ou não do produto final.

Tabela 2 – Classificação de Técnicas de Avaliação de Desempenho (JOHN, 2007).

Modelagem de Desempenho	Simulação	Simulação Dirigida por Trace Simulação Dirigida por Execução Simulação de Sistema Completo Simulação Dirigida por Eventos Perfis de *Software*
	Modelagem Analítica	Modelos Probabilísticos Modelos de Filas Modelos de Markov Modelos de Rede de Petri
Medida de Desempenho	Monitoramento de *Hardware on-chip* Monitoramento de *Hardware off-chip* Monitoramento de *Software* *Micro-coded* Instrumentação	

Como apresentado na Tabela 2, a técnica de *Modelagem de Desempenho* é dividida em duas: simulação e modelagem analítica. As duas técnicas podem ser aplicadas nas fases inicias, porém, cada uma delas tem aplicações específicas. A simulação geralmente é aplicada na avaliação de processadores, uma vez que a modelagem analítica não é precisa. Simuladores podem simular as funcionalidades e aspectos relacionados ao tempo em processadores, assim como todo o sistema ou componentes específicos. Em contrapartida, a modelagem analítica é indicada e usada geralmente em avaliação de desempenho de sistemas de grande porte, que incluem conjunto de *software* e *hardware*. O custo dessa técnica é relativamente baixo, por se basear em soluções matemáticas para criar modelos de sistemas. Exemplos de modelos matemáticos utilizados são (JOHN, 2007): métodos probabilísticos, modelos de filas, Markovs e Rede de Petri.

Já a técnica de *Medida de Desempenho* pode ser aplicada quando o produto está disponível para avaliação, com o sistema real ou protótipos, então é aplicada nas fases finais de um projeto. A medição consiste em submeter o sistema a um específico *workload*, e então monitorar o comportamento dele, com objetivo de beneficiar o seu desempenho (JAIN, 1991). Alguns *workload* tem sido usados para comparar sistemas, dependendo da classificação da medida de desempenho, entre eles pode-se citar: adição de instrução, mistura de instrução, *kernels*, programas sintéticos, *benchmarks* de aplicação.

2.2.3 *Workloads* e *Benchmark*

Workloads e *benchmarks* são termos que são usados na literatura com frequência e algumas vezes causa uma certa dúvida. *Workload* pode ser visto como uma lista de procedimentos que será submetida a um determinado sistema, e suas características irão variar de acordo com a técnica selecionada e o tipo de sistema a ser avaliado (JAIN, 1991). O termo *test workload* é também utilizado como sinônimo de *workload* em alguns trabalhos da

literatura, sendo classificados em duas categorias: *test workload* real e *test workload* sintético. O *test workload* real é aplicado em sistemas que estão rodando em tempo real, e os testes não podem ser repetidos, caracterizando assim uma limitação de aplicação. O *test workload* sintético é um modelo de *test workload* real, mas podem ser aplicados diversas vezes em um sistema dentro de um ambiente controlado. Por essa razão, os *test wokloads* sintéticos são adequados para comparação entre sistemas (JAIN, 1991). O termo *benchmark* é muitas vezes empregado na literatura e em relatórios de testes como sinônimo de *workload*, porém alguns *workloads* podem não ser considerados *benchmarks*; além disso, suítes de *software* que implementam um *workload* também são chamados de *benchmarks*. Já o termo *benchmarking* refere-se ao processo de comparação de dois sistemas ou mais, utilizando como critério os resltados mensurados em um *benchmark* comum (JAIN, 1991).

A Tabela 3 mostra vários *benchmarks* de acordo com o tipo de *workload* usado para o sistema.

Tabela 3 – *Benchmarks* (JOHN, 2007).

Categoria de *Workload*		**Exemplos de *Benchmark***
CPU *Benchmarks*	Uniprocessador	SPEC CPU 2000 Java Grande Forum Benchmarks SciMark ASCI
	Processador Paralelo	SPLASH NASPAR
Multimídia		MediaBench
Embutido		EEMBC Benchmarks
Sinal Digital		BDTI Benchmarks
Java	Lado Cliente	SPECjvm98 CaffeineMark
	Lado Servidor	SPECjBB2000 VolanoMark
	Científico	Java Grande Forum Benchmarks SciMark
Processamento de Transações	OLTP (Processamento de Transações *On-Line*)	TPC-C TPC-W
	DSS (Sistemas de Suporte à Decisão)	TPC-H TPC-R
Servidor Web		SPEC web99 TPC-W VolanoMark
Comércio Eletrônico	Com SGBD Comercial	TPC-W
	Sem SGBD Comercial	SPECjBB2000
Mail-server		SPECmail2000
Network File System		SPEC SFS 2.0
Desktop		SYSMARK Ziff Davis WinBench 3DMarkMAX99

2.3 TECNOLOGIAS WEB

2.3.1 Tecnologias Baseadas em Java

Java é uma linguagem que foi criada pela Sun Microsystem, na década de 90, sendo uma das linguagens mais populares da atualidade. Devido a essa popularidade, tornou-se uma das linguagens preferidas para desenvolvimento de aplicativos para rede, especialmente aplicações para Web. Isso tudo começou em 1991 quando James Gosling começou conduzir um projeto de pesquisa financiado pela Sun Microsystem, que resultou em uma linguagem de programação baseada em C++, mais tarde chamada de Java, o sucesso do projeto se deu pela oportunidade de aplicar Java a enorme explosão da popularidade da Web em 1993. Hoje o Java

tem três versões principais, *Java Standard Edition* (Java SE) voltada para aplicações *desktop* e servidores locais; *Java Enterprise Edition* (Java EE) que é adequada para desenvolvimento de sistemas Web em larga; *Java Micro Edition* (Java ME) voltada para sistema desenvolvidos para dispositivos limitados quanto à memória (DEITEL, 2010).

Um dos grandes trunfos do Java foi a chamada portabilidade, com sua frase clássica "*Write one, run anywhere*", ou seja, escreva o código uma única vez e rode em diferentes sistemas operacionais e plataformas. Essa possibilidade acontece graças a *Java Virtual Machine* (JVM).

Antes do conceito da máquina virtual, o código fonte deveria ser compilado em código de máquina para poder ser executado em determinado sistema operacional, com o código escrito para utilizar as bibliotecas daquele sistema, como era caso das linguagem C. Isso era um dos grandes problemas do desenvolvimento de *software*, porém com advento do Java, esse paradigma começou a mudar. Através da JVM um aplicativo pode ser executado em sistemas operacionais distintos.

Para utilizar a JVM, primeiro o código fonte é compilado em *byte codes* pelo compilador Java, e então interpretado pela JVM que está instalada em determinado sistema operacional. Dessa forma, existe o isolamento entre a aplicação e o sistema operacional.

No tocante a ferramentas de suporte para desenvolvimento Java, existem diversos *frameworks*, com as mais diversas finalidades. Em os vários podemos destacar os mais conhecidos e importantes: Hibernate, *Google Web Toolkit* (GWT), *Spring*, *Struts*, *VRaptor*, *Java Server Faces* (JSF).

Nós vamos fazer uso no desenvolvimento de nossos experimentos de alguns *frameworks* destinados ao desenvolvimento de aplicações Web. Para Java faremos uso do JSF, que é produtivo e bem documentado *framework* Web Java (DEITEL, 2010). Os critérios adotados para a seleção do JSF foram empíricos e seus detalhes são apresentados no Capítulo 4.

De maneira geral, o JSF possui uma API para representar e manipular estados da interface de usuário (UI[1]). Entre esses componentes pode-se destacar o validador *server-side*, o conversor de data, o manipulador de eventos, etc. O modelo de programação bem estruturado do JSF e as chamadas *tag library* fornecem ao desenvolvedor uma ferramenta poderosa de criação de interface intuitiva e com *design* visual agradável, com um esforço relativamente pequeno (ORACLE, 2013).

Na prática, o JSF trabalha em conjunto com outros *frameworks*, como por exemplo o *Java Server Page* (JSP), que é outra tecnologia Java para processar conteúdo estático e dinâmico no servidor Web, através de uma requisição do navegador Web. Na Figura 5 é possível visualizar como funciona uma interação com JSF, o cliente solicita a página `myfacelet.xhtml` ao servidor, que é construída no Web *container* através *tags* JSF e adicionada a *view* myUI

[1] do inglês *user interface*

que é uma representação da página Web no servidor, após isso a página enviada ao cliente para ser renderizada pelo seu navegador.

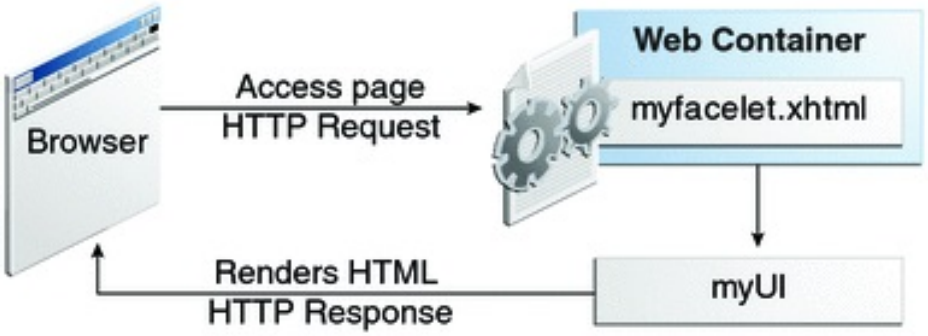

Figura 5 – Funcionamento JSF (ORACLE, 2013)

Com toda essa estrutura, existe alguns benefícios que o JSF proporciona. Entre eles, pode-se destacar a separação entre a camada de apresentação e a implementação das funcionalidades em si (ORACLE, 2013).

2.3.2 Tecnologias Baseadas em Python

Python é uma linguagem de programação criada em 1990, por Guido van Rossum, no Instituto Nacional de Pesquisa para Matemática e Ciência da Computação da Holanda. Python é considerada uma linguagem de altíssimo nível, derivada de outras linguagens existentes na época, assim como uma boa parte das linguagens mais modernas (BORGES, 2014).

Python é uma linguagem de programação que suporta vários paradigmas, entre eles o conhecido Paradigma Orientada a Objetos (POO). Considerada uma linguagem híbrida, também oferece a facilidade de uso de uma linguagem de *scripting* e com o seu vasto campo de domínio, pode ser aplicada em sistemas de propósitos distintos; é portátil, poderosa, e fácil de usar. Também é muito utilizada para criação de *scripting* em vários *softwares* atuais, como LibreOffice, GIMP e PostgresSQL (LUTZ; ASCHER, 2007).

Python tem sua licença compatível com a *General Public License* (GPL), sendo um *software* aberto e podendo, inclusive, ser incorporado a outros produtos proprietários. A sua especificação é mantida pela *Python Software Foundation* (PSF), que é uma organização que tem como missão promover, proteger e avançar no que diz respeito a linguagem de programação Python, além de dar suporte ao crescimento de uma comunidade diversificada e internacional de programadores com um suporte atuante.

Uma coisa bastante importante e interessante que o Python oferece é a facilidade de integração a outras linguagens de programação, por exemplo, C, Java e Fortran (BOR-

GES, 2014). Dessa forma, já podemos ver o quanto a linguagem pode se tornar poderosa, permitindo ao desenvolvedor fazer coisas mais elaboradas com essa integração.

A implementação principal do Python é baseada na linguagem C, e é conhecida como CPython, mantida pelo site www.python.org. Existem outras variações de sua implementação para atividades mais específicas, como é o caso de Jython, que é a implementação do Python baseado no Java, que permite a intregração entre as duas linguagens. O Python.NET é outra implementação alternativa do Python visando a integração com produtos da plataforma .NET.

A execução do Python passa por algumas etapas até chegar à execução final. Semelhante a outras linguagens de programação, como é caso de Java, o processo de compilação de código fonte não gera diretamente código de máquina, mas sim *byte code*, que pode ser executado pela *Python Virtual Machine* (PVM). A PVM utiliza a interpretação dos *byte code* em sua execução, gerando portabilidade nas aplicações em Python (LUTZ; ASCHER, 2007).

Em relação à máquina virtual usada pelo Python, existe um compilador *just-in-time* (JIT) conhecido como sistema Psyco, que é uma melhoria da PVM, que basicamente é encarregado de colher informações sobre os tipos de objetos, e substitui a parte de *byte code* em uma equivalente em código de máquina. O Psyco dessa forma melhora o desempenho de tempo de execução global do programa (LUTZ; ASCHER, 2007).

Ademais, existem diversas vantagens técnicas que contribuíram para que o Python seja uma das linguagens mais utilizadas e populares que temos nos dias atuais (LUTZ; ASCHER, 2007):

- É uma linguagem compatível com o paradigma orientado a objetos (POO), porém o seu diferencial reside na facilidade de aplicar o POO com sua estrutura de sintaxe e tipagem simples.
- É gratuita, sem qualquer restrição para copiar, integrar ou distribuir com seus produtos.
- A comunidade do Python é muito atuante, mesmo sendo o *software* gratuito, quando há dúvidas as respostas são em tempo bastante hábil, que deixa o Python à frente de muitos *software* proprietários.
- É portátil e pode ser usada em diversos tipos de dispositivos: *desktop*, Web, celular, PDA, etc. Além disso, a forma de execução de Python em *byte code* e depois em sua máquina virtual o deixa portátil entre plataformas e sistemas operacionais.
- É bastante poderoso, com diversas características de linguagem de *scripting* e de linguagem tradicional, tais como: tipagem dinâmica; gerenciamento de memória de forma automática; suporte para programação de grande escala; tipos de objetos incorporados; ferramentas incorporadas; bibliotecas; utilitários de outros fornecedores.

- Oferece facilidades para integrar a outras linguagens de programação.
- Facilidade de depuração, que possibilita construção do *software* passo-a-passo, permitindo a execução das linhas de código à medida que são escritas.
- Curva de aprendizado é muito boa se comparado com outras linguagens de programação, sendo utilizada inclusive como primeira linguagem de programação.

Python conta com uma variedade de ferramentas para ajudar os desenvolvedores em sua tarefas de programação, como *frameworks* Web e implementações alternativas para finalidades específicas (PYTHON SOFTWARE FOUNDATION, 2016). Exemplos de implementações para finalidades específicas são: *Stackless Python*, implementação do CPython com ênfase em concorrências, é utilizado pelo Nintendo DS; MicroPython, o Python designado para micro controladores; Parakeet, um compilador de tempo de execução para trabalhar com cálculos numéricos.

São muitos os *frameworks* Web projetados para o Python, alguns com incremento maior de funcionalidade, outros com uma abrangência menor de funcionalidade.Também há aqueles que são indicados para iniciantes ou programadores mais experientes. De qualquer forma, é uma ferramenta que muito ajuda na produtividade em desenvolver aplicações Web. Podemos citar alguns *frameworks* que são bem conhecidos na atualidade, como Django, Pyramid, Web2py, Flask, etc.

Flask é o *framework* utilizado nos nossos experimentos em Python, para aumentar a produtividade e por ser uma ferramenta mais indicada para tarefas simples, como é o caso do nosso sistema utilizado para avaliação de desempenho. O Flask na verdade é considerado um *microframework* em contraste com Django, isso significa que ele tem o núcleo bastante simples, oferecendo ao desenvolvedor a capacidade de poder estender suas funções. Seus objetivos são pautados justamente na produtividade e simplicidade tornando o desenvolvimento mais rápido. De um modo geral, o desenvolvedor tem a opção de usar módulos mais específicos de acordo com a sua necessidade, como por exemplo conexão com banco de dados e até mapeamento objeto-relacional (RONACHER, 2016). Os critérios adotados para a seleção do Flask foram empíricos e seus detalhes são apresentados no Capítulo 4.

3 MÉTODO PARA SELEÇÃO DA TECNOLOGIA DE DESENVOLVIMENTO ADOTADA

Este capítulo tem o objetivo de detalhar os passos adotados no método para seleção da tecnologia de desenvolvimento Web para *streaming* de mídia. Descrevendo as etapas iniciais de escolha das linguagens de programação, *framework*, arquitetura, etc., até a etapa final de aplicação do *benchmarking*.

3.1 CRITÉRIOS ADOTADOS

Um assunto bastante complexo é a tomada de decisão relativa a escolha da linguagem de programação mais adequada às necessidades de um determinado projeto, de acordo com a plataforma que será usada (Web, *desktop*, *mobile*, etc.), que tenha melhor desempenho em determinado requisito de qualidade, campo de atuação, etc. (JAIN, 1991). O nosso método buscar ajudar os projetistas nessa escolha, dando subsídios para determinar qual a linguagem mais adequada para desenvolver uma determinada aplicação Web com *streaming*.

Os critérios utilizados para avaliar esse domínio da aplicação, ou seja, a instância de *software* que é executada através do navegador, leva em consideração consumo de CPU, memória RAM, tempo de respostas e taxa de transferência.

A Figura 6 apresenta os passos adotados pelo método proposto. As Seções 3.1.1 a 3.1.6 apresentam uma descrição das atividades com instruções gerais de como cada uma deve ser executada.

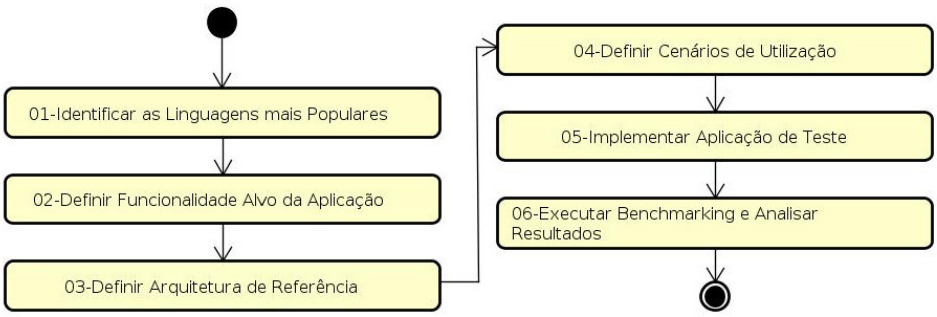

Figura 6 – Método para Seleção da Tecnologia.

3.1.1 Identificar as Linguagens mais Populares

Para selecionar as linguagens de programação que são mais usadas na atualidade, deve ser feita uma pesquisa em diversas fontes que publicam periodicamente quais as linguagens

de programação, *frameworks*, *Integrated Development Environment* (IDE), banco de dados, etc., que estão sendo mais usadas na atualidade, algumas dessas fontes tem sua base em repositórios de código fonte, sites de ajuda a desenvolvedores, empresas de consultorias em TI, etc. Além disso, a pesquisa não ficará restrita as linguagens, mas também levará em consideração os *framewoks* mais utilizados de determinada linguagem, tendo visto que grande maioria do desenvolvimento das aplicações Web fazem uso dessa tecnologia, para isso será extraído do *framework* a sua respectiva linguagem.

Neste primeiro passo são adotados dois critérios principais para a escolha da linguagem: popularidade e *framework* especializado.

Escolher uma linguagem de programação que está sendo muito usada e tenha uma comunidade ativa na atualidade pode trazer benefícios no tocante a atualizações de tecnologias e acompanhamento de tendências; além da facilidade para se conseguir suporte, aceitação no mercado, especialistas, programadores, curva de aprendizado menor, etc. O critério de existência de *framework* foi considerado importante, já que o uso de *frameworks* é uma prática habitual que aumenta consideravelmente a produtividade e a qualidade do *software* desenvolvido (HOTFRAMEWORKS, 2016). Além de que a grande maioria das aplicações Web atualmente foram desenvolvida com o auxílio de algum tipo de framework.

Após definida as fontes de pesquisa e montando uma relação com as linguagens mais populares, é definido os critérios de escolha das linguagens que farão parte de todo o processo de avaliação de desempenho e escalabilidade. Exemplo de aplicação dessa etapa do método é apresentados no Capítulo 4.

3.1.2 Definir Funcionalidade Alvo da Aplicação

Essa etapa visa a definição da funcionalidade alvo da aplicação com os seus objetivo principais, para servir de subsídio para avaliar as arquiteturas existentes e projetar as aplicações de teste.

Aqui também é definido o tipo de tecnologia de *streaming* que a aplicação será baseada, ou seja, qual protocolo de streaming será utilizado. Essa definição se torna importante porque a tecnologia poderá influenciar no desempenho da aplicação.

Detalhes da aplicação desse passo do método é apresentado no Capítulo 5

3.1.3 Definir Arquitetura de Referência

A arquitetura de um projeto de *software* é parte fundamental para o sucesso do mesmo, com ela é definido diversos aspectos relacionados às funcionalidades e qualidades que são primordiais ao domínio da aplicação em questão. Com a evolução do desenvolvimento de *software*, arquiteturas pré existentes podem ser utilizadas para dar maior produtividade, e

isso é prática bastante usual nos dias atuais, e assim ajudar a se definir uma arquitetura de referência.

Para se definir a arquitetura de referência, deve ser analisado sistemas semelhantes de mesmo domínio com funcionalidades que sejam de interesse da finalidade da aplicação do método. Oferecendo diretrizes para definição da arquitetura, através de análise dos pontos positivos e negativos das principais arquiteturas existentes.

Através do objetivo da aplicação e de sua arquitetura de referência, deve ser definido a seus requisitos mais relevantes, criando conforme o caso, diagramas UML que sejam importantes para representar a aplicação, dando origem um pequeno projeto da aplicação teste.

Detalhes da aplicação desse passo do método é apresentado no Capítulo 5

3.1.4 Definir Cenários de Utilização

É importante também definir os cenários de acordo com o objetivo do projeto, com uma projeção que possa remeter o mais próximo da realidade. A definição de cenários através de número de acessos simultâneos é a principal atividade aqui, já que uma aplicação Web baseada em *streaming*, pela sua próprio natureza, tem a consequência de seu desempenho e escalabilidade afetada pela quantidade de usuários que fazem acesso ao mesmo tempo.

Aqui pode ser definido um número específico de quantidade de usuários para saber até que ponto a aplicação Web consegue escalar bem com aquela quantidade.

Detalhes da aplicação dessa etapa do método é apresentado no Capítulo 5

3.1.5 Implementar Aplicação de Teste

Com base na funcionalidade alvo da aplicação de teste essa etapa do método visa a implementação da aplicação de teste, conforme definido nas fases anteriores do método. Aqui nós temos a descrição mais detalhada de como foram implementados as aplicações, fazendo um paralelo entre as linguagens de programação selecionadas e as tecnologias de desenvolvimento adotadas.

A forma que foi implementada a aplicação também deve ser informada, no que diz respeito à estruturação das camadas da aplicação, e como se comporta cada tecnologia nessas camadas.

O código fonte dos principais componentes podem ser detalhados, da mesma forma que foi feito com as camadas, fazendo um paralelo entre os códigos das linguagens de programação selecionada.

Ainda aqui podem ser agregados algumas imagens de como ficaram a implementação em cada linguagem adotada, mesmo não sendo parte relevante para a avaliação de desempe-

nho, mas ajuda a entender onde cada componente fica e onde o *software* de *benchmark* fará suas requisições.

Detalhes sobre a aplicação dessa etapa e da aplicação teste são apresentados no Capítulo 5

3.1.6 Executar *Benchmarking* e Analisar Resultados

Esse passo é relacionado à execução do *benchmarking* para avaliação de desempenho e escalabilidade. Primeiro deve ser definido qual *software* de *benchmarking* será usado nos experimentos, com a descrição de suas funcionalidades e quais serão úteis para aplicação do teste, assim como a sua configuração.

O ambiente de execução do *benchmarking* também deve ser descrito, tomando cuidado para dar as especificações das máquinas e estruturas utilizadas, definindo à qual componente cada máquina vai servir, observando sempre que necessário o componente da arquitetura que sofrerá mais carga, assim necessitando de um computador melhor.

A execução do *benchmarking* com a dinâmica de requisições é parte importante do método, levando consideração a quantidade de usuários definido, deve ser explicado como funcionará essa dinâmica. Deve ser observado como será feito as requisições à aplicação Web de *streaming*. Também deve ser explicitado com está configurada todas as tecnologias que farão parte do experimento, como navegador Web, *benchmark*, SO, servidor Web, etc.

Por fim, temos a análise dos resultados, que tem por finalidade avaliar as linguagens de programação selecionadas. Através dos dados coletados podem ser inferidas semelhanças e diferenças entre elas, além de ponto mais relvantes de cada linguagem, nos mais variados cenários.

O Capítulo 6 aborda essa última etapa do método da execução do *benchmarking* .

4 SELEÇÃO DAS LINGUAGENS

Este capítulo tem por finalidade apresentar as primeiras etapas de aplicação do método. Aborda as principais tecnologias de desenvolvimento disponíveis para desenvolver aplicações Web com *streaming* de mídia. Dentre as tecnologias consideradas, estão: (1) linguagem de programação; e (2) *framework* de desenvolvimento. Após a apresentação das tecnologias, foram escolhidas aquelas com maior potencial de uso, a partir da análise das tendências de mercado. As duas tecnologias mais promissoras foram escolhidas para uma comparação mais detalhada, cuja aplicação teste é apresentada no Capítulo 5 e a execução do *benchmarking* é apresentada no Capítulo 6.

4.1 POPULARIDADE DAS LINGUAGENS DE PROGRAMAÇÃO

Algumas empresas de TI, que trabalham com acesso a milhares de códigos, como repositório, análise de qualidade de código, consultoria, etc., divulgam dados periódicos com informações sobre as tecnologias de desenvolvimento que estão sendo mais usadas na atualidade, tais como linguagens de programação. Vale ressaltar que as linguagens de programação tem características particulares, como compilador e interpretador, servidor Web, tipos de dados, acesso à memória, etc. Levando em consideração a forma de executar do programa, as linguagens são divididas em dois grandes grupos (COLLIER; MEYER, 2000): as linguagens de *scripting* e as linguagens tradicionais.

As linguagens de *scripting* se caracterizam por serem interpretadas e, na maioria das vezes, não terem declarações de variáveis no decorrer do seu desenvolvimento. Nesse grupo temos: Perl, Python, Ruby, PHP, etc. As linguagens tradicionais são compiladas em algum momento. Nesse segundo grupo de linguagens tradicionais temos: Java, C, C++, C#, Objective-C, etc. (FOURMENT; GILLINGS, 2008).

Para a obtenção dos dados de popularidade das linguagens de programação, uma das fontes de informação utilizadas foi a TIOBE (TIOBE, 2016), empresa que oferece um serviço pago para análise de qualidade de código-fonte, com um sistema de avaliação baseado na ISO 25010. Empresas de desenvolvimento de *software* que desejarem podem submeter seus códigos para serem analisados para eventualmente ser atribuído um selo de qualidade denominado *TIOBE Quality Indicator*. A TIOBE publica a cada mês um índice com as linguagens mais populares no mundo, a partir dos projetos avaliados por ela, como pode ser visto na Figura 7.

Jul 2016	Jul 2015	Change	Programming Language
1	1		Java
2	2		C
3	3		C++
4	5	▲	Python
5	4	▼	C#
6	7	▲	PHP
7	9	▲	JavaScript
8	8		Visual Basic .NET
9	11	▲	Perl
10	12	▲	Assembly language

Figura 7 – TIOBE *Ranking* (TIOBE, 2016).

Segundo essa publicação da TIOBE, no mês de abril de 2016, as linguagens tradicionais estão nas 3 primeiras posições, com Java (em 1^o), C (em 2^o), C++ (em 3^o) e C# (em 4^o), seguida pelas linguagens de *scripting* Python (em 4^o), PHP (em 6^o) e JavaScript (em 7^o).

A segunda fonte de informações utilizada para consultar a popularidade das linguagens de programação foi o Codeeval (CODEEVAL, 2016), que é uma plataforma com diversas características, visando dar oportunidade a desenvolvedores de mostrar suas habilidades na qualidade da codificação. O repositório da Codeeval pode ser acessado gratuitamente ou com alguns planos pagos que disponibilizam algumas funcionalidades adicionais da plataforma. Foi publicado um *ranking* com as linguagens mais populares de 2016, como apresentado na Figura 8.

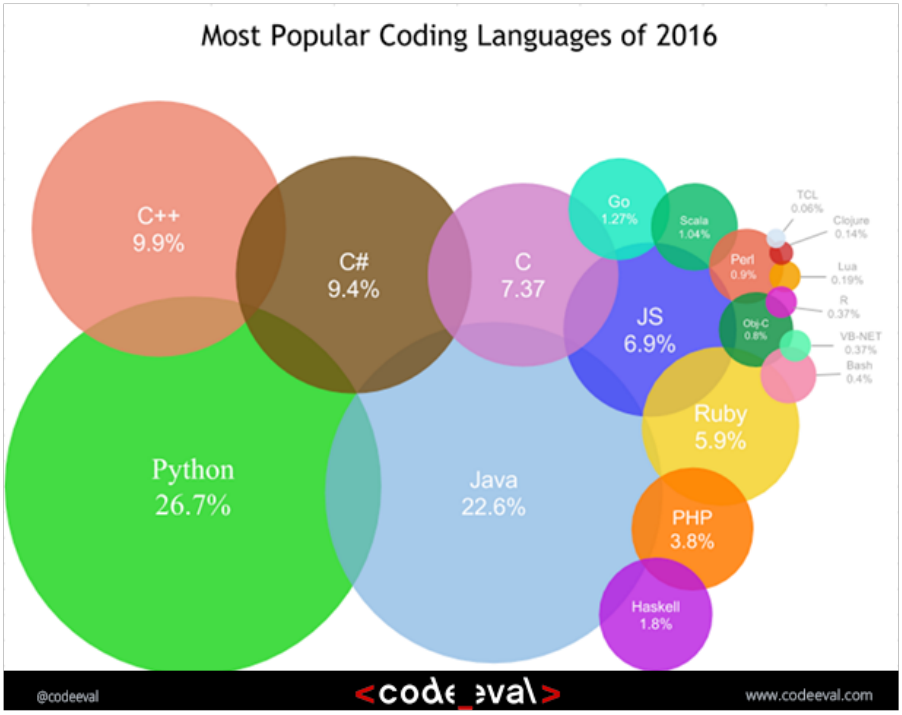

Figura 8 – CodeEval *Ranking* (CODEEVAL, 2016).

Segundo o *ranking* do Codeeval, existe utilização massiva tanto de linguagens de *scripting*, quanto de linguagens tradicionais, com Python (em 1^o), Java (em 2^o), C++ (em 3^o) e C# (em 4^o).

A terceira fonte de dados utilizada vem da empresa de consultoria RedMonk (REDMONK, 2016), que baseado em informações vindas dos repositórios Stack Overflow (STACK OVERFLOW, 2016) e GitHub (GITHUB, 2016), publicam um *ranking* periódicos com as linguagens que estão sendo mais usadas e discutidas no mundo, e assim conseguem filtrar as informações que precisam para fazer as publicações. A figura 9, mostra como anda o *ranking* de acordo com RedMonk.

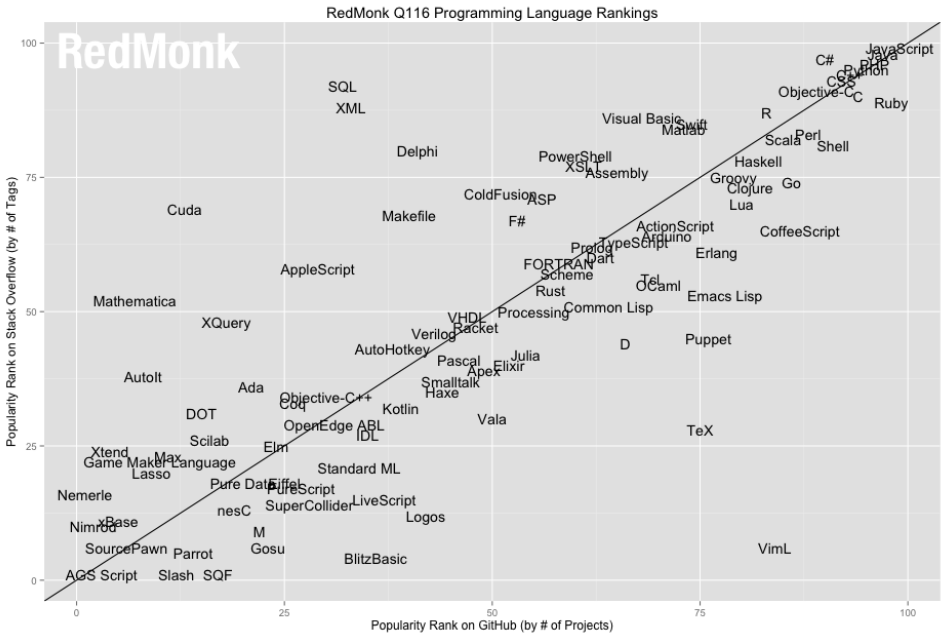

Figura 9 – RedMonk *Ranking*(REDMONK, 2016).

Conforme essa publicação, as linguagens de *scripting* estão mais populares do que as linguagens tradicionais, com JavaScripting (em 1^o), Java (em 2^o), PHP (em 3^o), Python (em 4^o) e C# (em 5^o). Os dados apresentados pela RedMonk refletem bem a tendência atual, uma vez que se baseia em repositórios que contém projetos recentes e ativos.

Por fim, temos o *ranking* da IEEE Spectrum, que esse ano de 2016 está em suas terceira publicação, a composição do *ranking* é derivada de 10 fontes online e 12 métricas. O *ranking* é bem interessante e fornece um aplicativo que dar a possibilidade de filtrar o *ranking* de acordo com a aplicação da linguagens, e essa aplicação é dividida em quatro tipo: linguagens usadas para aplicações para site Web, *mobile*, *desktop* e microchip (IEEE SPECTRUM, 2016). A Figura 10 , mostra o *ranking* geral de 2016 do IEEE Spectrum sem levar em consideração o filtro abordado anteriormente.

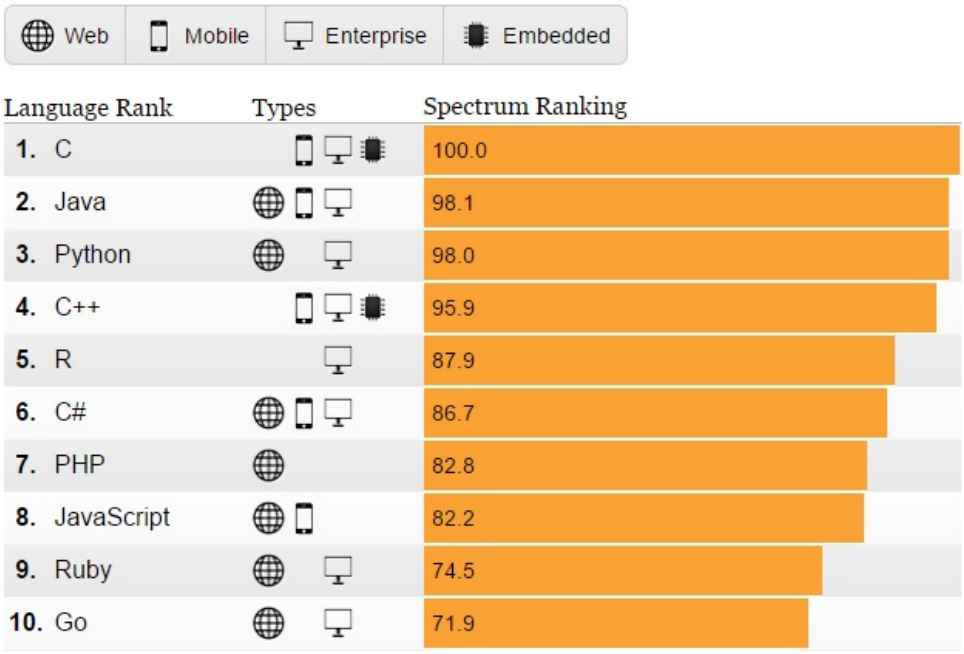

Figura 10 – IEEE Spectrum *Ranking*(IEEE SPECTRUM, 2016).

De acordo com o esse *ranking*, nós temos C (em 1º), Java (em 2º), Python (em 3º), C++ (em 4º) e R (em 5º). Observe que nesse caso, as linguagens de *scripting* não aparecem como sendo as mais populares, mas sim as linguagens ditas tradicionais.

4.2 *FRAMEWORKS* PARA APLICAÇÕES COM *STREAMING* DE MÍDIA

A seção anterior teve como foco a popularidade das linguagens de programação na atualidade. Essa seção trata do segundo critério adotado para a escolha da tecnologia mais adequada, tendo como foco a identificação dos *frameworks* que estão sendo usados com mais frequência no cenário atual, extraindo a linguagem de programação relacionado ao *framework*.

Frameworks são projetados para uma domínio específico com a finalidade de abstrair certos aspectos, essas abstrações criam funcionalidades já prontas, tornando-as simples. Assim ajudam os desenvolvedores, uma vez que oferecem estrutura e funcionalidades previamente planejadas, além de recursos específicos implementados e testados, deixando que o desenvolvedor se preocupe com aspectos mais específicos da aplicação sendo desenvolvida. Dessa forma, um dos principais benefícios que um framework para desenvolvimento pode fornecer é o há reúso de *software*, tanto do ponto de vista funcional, quanto estrutural (RIEHLE, 2000). Podemos citar diversos exemplos, como *frameworks* que podem implementar o padrão de projeto MVC (REENSKAUG; COPLIEN, 2009), outros apenas alguma camada do MVC, como visão, alguns oferecem recursos visuais e de gerenciamento, ou para persistência

em banco de dados. São diversos os recursos que um *frameworks* pode oferecer e tudo isso ajuda a aumentar a produtividade e a adoção de boas práticas de projeto e desenvolvimento. A seguir temos as pesquisas que mostram os *frameworks* que estão em destaque atualmente.

A BuiltWith(BUILTWITH, 2016) é uma empresa que divulga tendências de tecnologias Web em geral e outros serviços de análise de dados. Ela disponibiliza, através do seu site[1], um percentual de uso de *frameworks* que foram utilizados no desenvolvimento de vários sites. Tais *frameworks* também são agrupados de acordo com a linguagem de programação nos quais foram construídos. A Figura 11 mostra um quadro que sintetiza essa informação.

Name	10k	100k	Million	Entire Web
ASP.NET	⬆2,359	⬇22,528	⬇281,463	⬆42,967,599
PHP	⬇3,733	⬆36,018	⬇606,945	⬆41,672,384
Shockwave Flash Embed	⬇532	⬆4,485	⬇141,498	⬆5,019,267
J2EE	⬇1,334	⬇6,789	⬇72,253	⬆3,593,262
Classic ASP	⬇306	⬇3,633	⬇23,909	⬆3,159,197
Adobe Dreamweaver	⬇358	⬇3,881	⬇75,327	⬆2,609,074
Ruby on Rails	⬆651	⬆3,786	⬆33,345	⬆1,055,584
ASP.NET MVC	⬆705	⬆3,865	⬇22,675	⬆977,076
ASP.NET Ajax	⬇785	⬇7,787	⬇74,182	⬆894,849
DAV	⬆137	⬆1,321	⬇17,560	⬆874,366

Figura 11 – BuiltWith *Ranking* por linguagem de programação (BUILTWITH, 2016).

Como visto na Figura 11, grande parte de sites trabalham com *frameworks* ASP.NET (compatível com C#), uma outra boa parte são projetados para trabalhar com a linguagem PHP, seguido Shockwave Flash Embed e logo após vários *frameworks* da plataforma J2EE (compatível com Java).

Já o site `www.hotframeworks.com` mantém uma lista totalmente dinâmica com os *frameworks* mais populares nos repositórios GitHub (GITHUB, 2016) e Stack Overflow (STACK OVERFLOW, 2016). A Figura 12 mostra o estado desse *ranking* no dia da escrita deste capítulo.

Conforme a Figura 12, o ASP.NET tem os *frameworks* mais populares, mesmo sem ter escore do GitHub(GITHUB, 2016). Ruby on Rails também tem uma popularidade boa, ficando com 96 pontos na média. A linguagem de *scripting* do lado cliente JavaScprit aparece com os *frameworks* AngularJS (90) e Meteor (85). Ainda tem o Django, da linguagem Python, com 90 pontos, seguidos do Laravel do PHP com 86, e por último Spring de Java com 84.

[1] Endereço: `builtwith.com`, acessado em setembro de 2016

Framework	Github Score	Stack Overflow Score	Overall Score
ASP.NET		100	100
AngularJS	100	96	98
Ruby on Rails	95	98	96
ASP.NET MVC		93	93
Django	90	93	91
Laravel	92	84	88
Meteor	96	79	87
Spring	82	90	86
Express	93	80	86
CodeIgniter	85	85	85

Figura 12 – *Hotframeworks Ranking* (HOTFRAMEWORKS, 2016) .

4.3 IDENTIFICANDO AS TECNOLOGIAS MAIS PROMISSORAS

Dando sequência com os passos de aplicação do método deste trabalho, se faz necessário identificar as tecnologias mais promissoras, no nosso caso duas, para então realizar um experimento comparativo detalhado, envolvendo experimentos e execução de *benchmarking*. Esta seção apresenta os critérios utilizados para a escolha das duas tecnologias a serem comparadas. Os dados da aplicação piloto utilizada no experimento são apresentados no Capítulo 5, enquanto a execução dos *benchmarking* é apresentada no Capítulo 6.

A escolha das tecnologias para fazerem parte dos experimentos e posterior *benchmarking* são derivadas das pesquisas feitas nesse capítulo, e levará em consideração não apenas as linguagens de programação mais populares, mas também os *frameworks*.

O critério utilizado será uma pontuação aplicado a cada linguagem de programação, aplicado 1 ponto se ele estiver nas nas quatro primeiras posições de de cada um dos seis *rankings* considerados (quatro de linguagens de programação e dois de *frameworks*). Lembrando que os dois *rankings* relativos aos *frameworks* levará em consideração as linguagens de programação em que eles são baseados, para isso iremos converter alguns *frameworks* em sua respetivas linguagens, conforme apresentado na Tabela 4. O *framework* DAV não está relacionado às linguagens programação mas sim funções oferecidas aos desenvolvedores, como extensões para o protocolo HTTP, nesse caso será mantido o nome do *framework* (WEBDAV, 2016). Também no caso do Adobe Dreamweaver que tem suportes à várias linguagens de programação será mantido o nome do *framework* (ADOBE SYSTEMS INCORPORATED, 2016).

Tabela 4 – Conversão *Frameworks* para Linguagens de Programação.

Frameworks	**Linguagem de Programação**
ASP.NET	C#
ASP.NET MVC	C#
ASP.NET AJAX	C#
Classic ASP	ActionScript
Shockwave Flash Embed	Visual Basic
J2EE	Java
Ruby on Rails	Ruby
Django	Python
Laravel	PHP
Express	JavaScripting
CodeIgniter	PHP
AngularJS	JavaScripting
Meteor	JavaScripting
Spring	Java

Após essa conversão é criado um *ranking* adaptado somando a pontuação de cada linguagem de programação, os *rankings* adaptados podem ser verificados nas Tabelas 5 e 6. No caso do *framework* .NET e suas variações será levado em consideração C# por ter sido a linguagem desenvolvido junto com o projeto do .NET (ASSOCIATION et al., 2006).

Tabela 5 – *Ranking* Hotframeworks Adaptado

	Linguagem	**Pontuação**
1º	JavaScripting	271
2º	C#	191
3º	PHP	173
4º	Ruby	96
5º	Python	91
6º	Java	86

Tabela 6 – *Ranking* Buildwith Adaptado

	Linguagem	**Pontuação**
1º	C#	44,839,524
2º	PHP	41,672,384
3º	ActionScript	5,019,267
4º	Java	3,593,262
5º	Visual Basic	3,159,197
6º	Adobe Dreamweaver	2,609,074
7º	Ruby	1,055,584
8º	DAV	874,366

O *ranking* resumindo de pontuação até a 4ª posição pode ser verificada na Tabela 7, as divergências que podem ser verificadas nos *rankings*, estão provavelmente relacionados a natureza do *ranking*, ou seja, alguns ranking estão mais relacionados a parte acadêmica ou empresa.

Tabela 7 – *Ranking* Resumido de Linguagens de Programação e *Frameworks*.

	TIOBE	**Codeeval**	**RedMonk**	**IEEE Spectrum**	**BuildWith**	**Hotframeworks**
1º	Java	Python	JavaScripting	C	C#	JavaScripting
2º	C	Java	Java	Java	PHP	C#
3º	C++	C++	PHP	Python	ActionScript	PHP
4º	Python	C#	Python	C++	Java	Ruby

Com base nos critérios definidos, a Tabela 8 mostra a pontual final de cada linguagem de programação.

Tabela 8 – Ocorrência de Popularidade das Linguagens de Programação

Linguagem de Programação	**Pontuação**
Java	5
Python	4
C#	3
C++	3
PHP	3
JavaScripting	2
C	2
Ruby	1
ActionScript	1

Como é possível observar, segundo o critério utilizado, a classificação entre as linguagens tradicionais e linguagens de *scripting* são similares. Java e Python são as linguagens mais populares, sendo Java uma linguagem tradicional e Python uma linguagem de *scripting*. Assim, por serem as duas mais utilizadas segundo o nosso método aplicado, foram as selecionadas para fazerem parte da aplicação teste para avaliação de desempenho e escalabilidade em aplicação Web com *streaming* de mídia.

5 ARQUITETURAS DE REFERÊNCIA E ESPECIFICAÇÃO DA APLICAÇÃO DO EXPERIMENTO

Este capítulo está relacionado as etapas centrais do método. Desmostra o desenvolvimento do sistema que servirá de experimento, descrevendo o objetivo do sistema de acordo com o domínio da aplicação, funcionalidade alvo do sistema e tecnologia de *streaming*. Análise de domínio para subsidiar a etapa de levantamento dos requisitos, para isso será analisado aplicações de mesmo domínio de acordo com as metas estabelecidas. A definição da arquitetura de referência do sistema ajudará na elicitação do requisitos, deixando de forma mais clara as interações entre os componentes. Por fim, o projeto da aplicação com seus diagramas de caso de uso e classes e a descrição da implementação do experimento nas duas linguagens previamente selecionadas.

5.1 OBJETIVO DO SISTEMA

Para atender o objetivo da pesquisa e do experimento, é necessário a definição de um tipo de sistema Web funcionando com *streaming* de mídia. No nosso caso, foi adotado um sistema que deve permitir um *player* de *streaming* de música através de um navegador Web. Apesar das pesquisas acadêmicas darem enfase em *streaming* de vídeo não fazendo distinção entre áudio e vídeo, essa diferença existe, em relação ao consumo dos recursos computacionais como apontado por Kreitz (2011), a escolha vem do fato que *streaming* de áudio requerem menos esforço computacional do que de vídeos, permitindo que nossos experimentos sejam executados de forma mais simplista para coleta dos resultados. O sistema vai fornecer aos usuários a possibilidade de acessar músicas utilizando um sistema com o conceito de SaaS, onde os dados dos arquivos de músicas estarão dispostos na nuvem. Também optamos por usar o método de pseudo *streaming* já que os *browsers* HTML5 dão suporte nativamente a esse método de *streaming* e outro protocolos (FIELDING et al., 2009).

Esse objetivo forma a base para utilização da técnica de análise de domínio, permitindo um refinamento mais detalhado de quais serão os requisitos essências para o sistema ora proposto levando em consideração o contexto de *streaming*.

5.2 ANÁLISE DE DOMÍNIO

O processo de desenvolvimento de *software* é complexo à medida que o contexto do sistema também é. E uma das fases consideradas por muitos como sendo críticas no desenvolvimento de *software* é o levantamento de requisitos, é uma das primeiras etapas mas muito importante. Os requisitos do sistema pode vim de diversas fontes de informações e

coletado por diversas técnicas, como usuários, proprietários, entrevistas, observação, sistemas existentes, etc. Por ser uma tarefa complexa, a depender do contexto do sistemas, essas fontes e técnicas podem ser combinadas (ZOWGHI; COULIN, 2005). O nosso contexto não requer um processo complexo de desenvolvimento de *software*, porém, ainda requer o mínimo de planejamento e execução de algumas fases importantes, como o levantamento dos requisitos, e para isso usaremos a técnica de análise de domínio.

A análise de domínio consiste em analisar documentação, manual, aplicações, componentes, conceitos, etc. de um determinado domínio. Todas essas informações ajudam na definição dos requisitos de um sistema, podendo ser usado em conjunto com outras técnicas e informações. Essa abordagem podem gerar a linha de base consistente para um sistema, com analogias entres os sistemas analisados e o sistema que será desenvolvido (ZOWGHI; COULIN, 2005). Como ela é uma técnica que observa outros sistemas de determinado contexto, ela se encaixa perfeitamente na proposta do experimento, que são sistemas relativamente simples para avaliação de desempenho em ambiente Web com *streaming* de mídia.

Os sistema escolhidos para análise de domínio são dois: o primeiro é uma empresa nova mas com uma quantidade bem considerável de usuário, que é o Spotify; E o segundo é uma empresa que no passado teve o seu negócio baseado em aluguel de filmes online, porém com a mudança na forma de ofertar serviços online passou a oferecer filmes sob demanda usando *streaming*, na atualidade conta com uma quantidade de usuários muito alta.

5.2.1 Spotify

O Spotify é um serviço de *streaming*, onde em 2012 contava com cerca de 16 milhões de músicas para mais de 10 milhões de usuários, esses dados constam em um artigo publicado por Yanggratoke et al. (2012) . O Spotify atualmente conta com três tipos de assinatura: *free*, sendo totalmente gratuita mas com algumas funcionalidades limitadas, como executar apenas músicas aleatórias em alguns dispositivos; *Premium*, que adiciona uma melhoria na qualidade do som, modo *off-line*, e agora permitindo selecionar qualquer música naqueles dispositivos que na versão *free* executava apenas aleatoriamente; E *Family*, que oferece a possibilidade de adicionar até 6 contas na mesma assinatura. O Spotify é repleto de uma grande variedades de músicas e ainda conta com aplicativos para *desktop*, *mobile* e Web. A sua interface Web é bem intuitiva e fácil de usar como qualquer *player* Web, conforme mostra a Figura 13, ela gera algumas sugestões para os usuários de acordo com as preferências de músicas de cada usuário, além de permitir que o usuário selecione a música, álbum, artista, banda e *playlist* criadas por outros usuários (KREITZ; NIEMELÄ, 2010). Uma coisa que pode ser encarada como um ponto negativo no Spotify, é que você não pode fazer *upload* daquela sua música predileta e obscura, e que não vai encontrar em produtora de música, em contra partida você pode criar sua *playlist* favorita e outros usuários podem adicioná-la a sua.

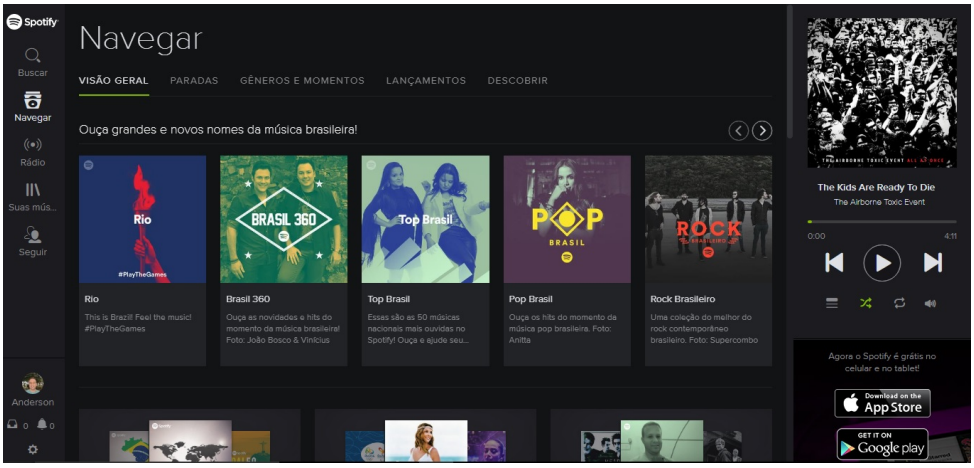

Figura 13 – Spotify

A arquitetura do Spotify mostrada na Figura 14 é bastante interessante, os servidores *back-end* principais estão localidades em três locais diferentes, Estocolmo, na Suécia, Londres, no Reino Unido, e Ashburn, na Vírginia. Cada *back-end* tem uma *layouts* semelhante, em geral executando serviços de login, busca de músicas, administração de *playlist*, e funções de rede social, além do *streaming* de música. O seu servidor *master* fica distribuído em infraestrutura de terceiros, e nesse caso prove apenas *streaming* quando as outras fontes não podem prover a mídia. O interessante na sua arquitetura é no momento de fornecer o *streaming* de áudio para o usuário final, onde é usado em conjunto com seus servidores o sistema de *peer-to-peer*. Quando o usuário vai executar uma música em seu cliente, o Spotify faz uma combinação entre o *cache* local do usuário no qual executou a música recentemente e ainda tem dados da mesma, dos seus servidores *back-end*, e pelo sistema *peer-to-peer* onde usuários que executaram de alguma forma a música podem fornecer dados para esse cliente, para essas duas últimas combinações o Spotify levando em consideração os *back-end* e *peer-to-peer* mais próximos do cliente, em último caso o servidor *master* é acionado. Essa arquitetura implementada pelo Spotify consegue fazer uma mistura de pedaços de dados das três fontes, diminuindo assim a latência de execução (YANGGRATOKE et al., 2012). A oferta de *streaming* na arquitetura do Spotify, a princípio, é através de RTSP, não foi possível identificar se todas as aplicações do Spotify usam esse protocolo.

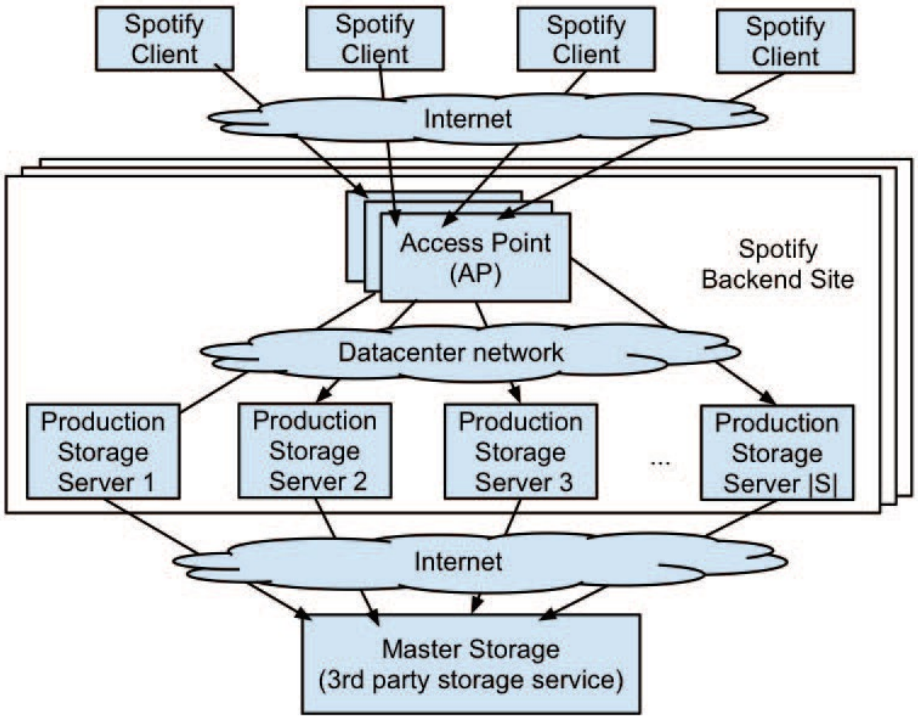

Figura 14 – Spotify Arquitetura (YANGGRATOKE et al., 2012)

Apesar de ter um sistema interessante de *streaming*, o Spotify peca em não ter em infraestrutura *master* alguns servidores próprios, onde poderia ter um controle maior sobre o conteúdo. O Spotify também não faz uso de HTTP *Streaming*, o que podia melhorar um pouco a transmissão de conteúdo em momentos inesperados, para algumas aplicações do Spotify.

5.2.2 Netflix

O Netflix, diferentemente do Spotify tem seu negócio pautado em *streaming* de vídeo, filmes, séries e desenhos, com conteúdo podendo chegar em *High Definition* (HD). Dessa forma faz uso do mecanismos chamado de *on-demand* visto nos capítulos anteriores, já que a mídia está atrelada a um usuário e este tem acesso ao conteúdo quando quiser. Hoje em dia a grande parte do tráfego da internet pelo mundo vem das assinaturas do Netflix, principalmente nos Estado Unidos e Canadá, com cerca de 23 Milhões de assinaturas (ADHIKARI et al., 2012). O Netflix tem alguns tipos de assinatura, não existe uma versão *free* como o Spotify mas eles oferecem um teste grátis por um período, versão 1 *screen* para apenas um usuário por simultâneo com resolução *standard*, 2 *screen* + HD para até dois usuários ao mesmo tempo agora com conteúdo em HD, e 4 *screen* + Ultra HD permitindo até quatro usuário simultâneos com conteúdo podendo chegar em *Ultra* HD. O Netflix tem diversas parcerias com diversas empresas, produtoras para disponibilizar os conteúdos para seus assinantes, fabricantes de aparelhos que em conjunto disponibilizam *software* em seus

aparelhos como Playstation 3, Xbox, Samsung, iPhone, etc. Com o crescimento do Netflix no passar dos anos, ele conseguiu criar um tremendo diferencial que foi produção própria de filmes e séries. A sua aplicação *web* é de fácil uso, com um algoritmo de recomendação de conteúdo na página principal para diversas categorias, filmes assistidos e sugestões de acordo com o perfil do usuário. O *player* Web conforme mostrado na Figura15, é bem básico mas com alguns conteúdos adicionais além das funções básicas de *player*, ele permite que seja modificada a legenda e áudio, e ainda permite que sejam selecionados episódios no caso de uma série. O *player* até pouco tempo atrás usava uma aplicação Microsoft Silverlight que era baixado toda vez que o usuário ia executar um vídeo, porém hoje em dia o Netflix usa HTML5 que tem suporte para alguns protocolos de *streaming*, e o protocolo usado para *streaming* é o DASH (ADHIKARI et al., 2012).

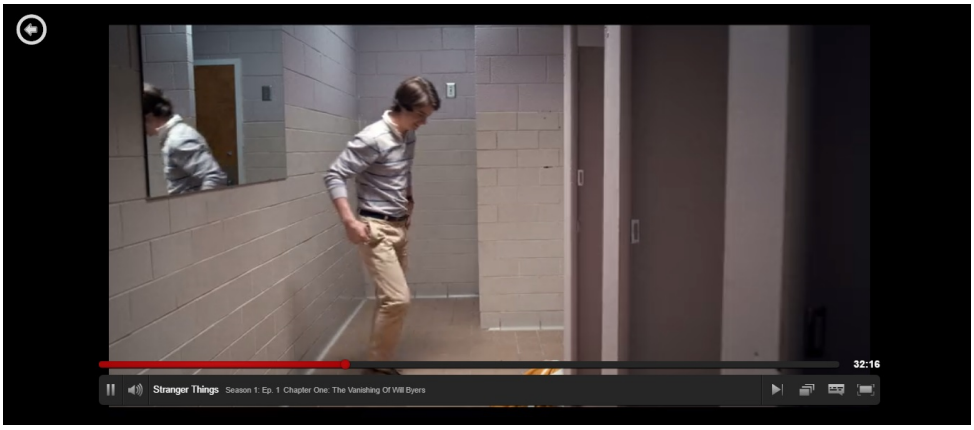

Figura 15 – Netflix Web

A arquitetura do Netflix recentemente foi reorganizada para trabalhar com serviços de nuvem devido a necessidade constante de escalabilidade, fazendo uso da infraestrutura de nuvem da *Amazon Web Service Cloud* (AWS Cloud) apresentada na Figura 17. O Netflix usam diversas soluções tecnológicas para compor sua arquitetura, entre elas o CDN é usado para fornecer *streaming* para seus clientes com interação entre os múltiplos CDN e as outras tecnologias de forma transparente para o usuário. Como mostrado na Figura 16 a arquitetura do Netflix é composta de quatro componentes principais: o *data center* do Netflix, o *player* Web, múltiplos CND e AWS Cloud. Inicialmente o Netflix fazia uso de três CDN de terceiros, como apresentado no estudo de Adhikari et al. (2012), Akamai, LimeLight, and Level-3. Depois do crescimento do Netflix e da necessidade de ter sua infraestrutura globalmente localizada, o Netflix passou a ter seus próprios CDN (NGINX, 2015). O segredo em manter o vídeo executando em picos baixo de banda vem do arquivo *manisfest*, ele é o arquivo que contem muitas informações valiosas sobre tudo o contexto de execução do vídeo. Informações como a localização do cliente para fazer *streaming* do ponto que terá o melhor desempenho para ao usuário. Qual o aparelho, aplicativo e sistema operacional que está solicitando o vídeo, fazendo com que o Netflix selecione o arquivo de áudio e vídeo com formatos e *bitrates* que podem ser executados de forma satisfatória (ADHIKARI et al., 2012).

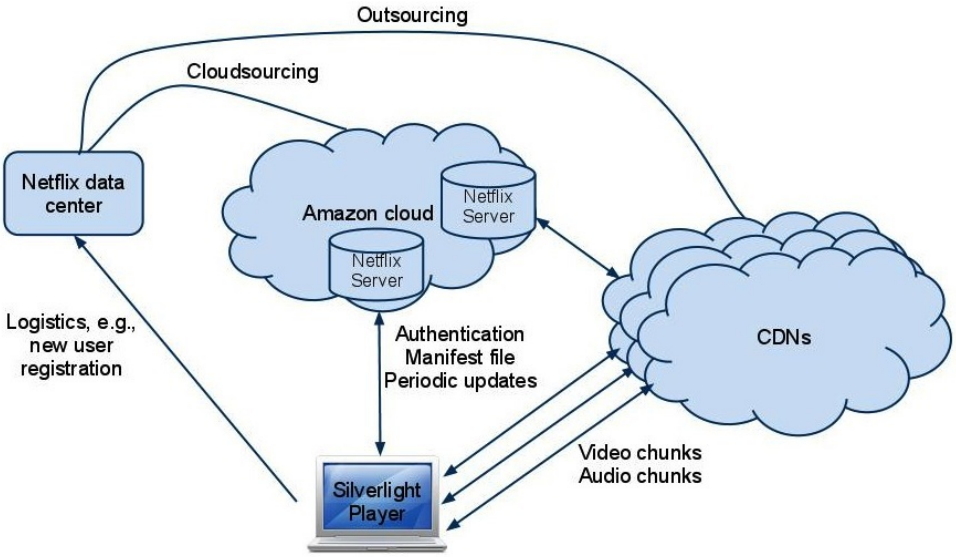

Figura 16 – Netflix Arquitetura Web (ADHIKARI et al., 2012)

O Netflix usa AWS Cloud para compor sua arquitetura, armazenando partes de seus servidores. Por exemplo, o servidor `www.netflix.com` é hospedado pelo servidor central do Netflix, já os servidores `agmoviecontrol.netflix.com` e `movies.netflix.com` são hospedados pela arquitetura da Amazon. Os CDN's utilizados pelos Netflix não são da AWS Cloud, mas sim seus próprios CDN's.

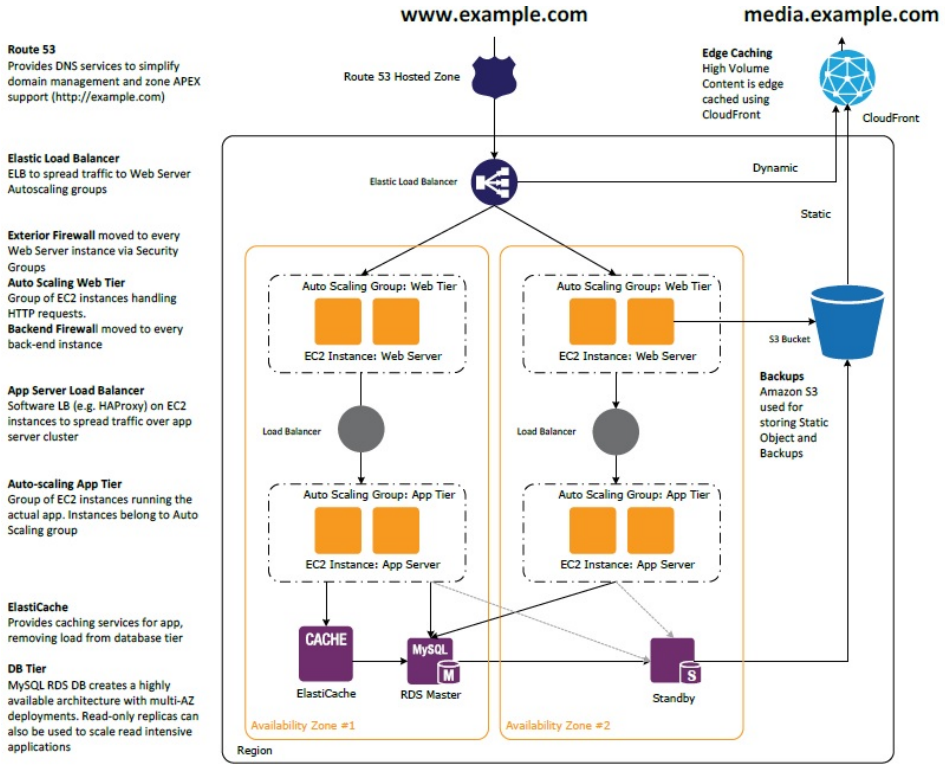

Figura 17 – Amazon Cloud *Web* (TAVIS; FITZSIMONS, 2012)

No mais, o Netflix não faz uso de todo o pontencial do HTTP *Streaming*, onde poderia utilza-se dele para conseguir enviar streaming simultanemante de diversos CNDs, o que poderia melhorar a o desempenho geral da aplicação.

5.3 ARQUITETURA DO SISTEMA

Conforme o objetivo do experimento em ser uma aplicação Web de *streaming* de áudio e da análise de domínio realizada, essa seção tem a finalidade de apresentar a arquitetura para o sistema de *streaming* aqui proposto para duas versões, *Java* e *Python*.

Como aplicação será bem mais simples que uma aplicações de serviço de *streaming* comercial como *Netflix* e *Spotify* pela natureza do experimento, a arquitetura do sistema foi projetada para ajudar no entendimento da solução e para ajudar na obtenção dos requisitos conforme Figura 18. Não iremos fazer uso de CDN como outras aplicações comerciais de grande porte, pois não é o foco da nossa investigação ter um experimento complexo.

A arquitetura projetada é simples, consiste de um servidor de mídia, onde ficará armazenado as músicas que servirão os usuário, um servidor Web executando a aplicação principal de nosso experimento que será implementada para fornecer um *player* de música Web para os usuário, e por fim, os clientes que acessam essa aplicação do servidor Web para executaram o *streaming* de áudio.

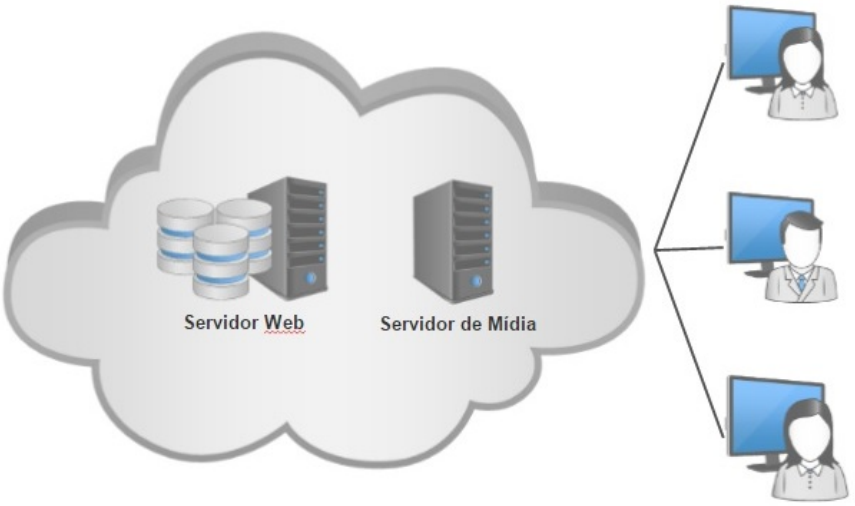

Figura 18 – Arquitetura da Aplicação Web

5.4 CENÁRIOS

Com os cenários tentamos criar ambientes que possam nos dar resultados em diversas condições de uso, assumindo quantidades variáveis de usuários simultâneos; com isso, espera-se obter dados importantes para estimar a escalabilidade de cada solução. A Tabela 9 mostra um resumo dos cenários de execução; é importante ressalvar que a quantidade de usuários usada no experimento representa acessos simultâneos, ou seja, o número de usuários que estão acessando a aplicação de *streaming* ao mesmo tempo, em paralelo. É importante destacar também que essa quantidade máxima de usuários foi definido pelas limitações de *hardware* que temos disponíveis no servidor Web e servidor de mídia.

Tabela 9 – Cenários de Execução

Cenário	**Número Usuário**	**Tempo de Execução**
1	10	20min
2	100	20min
3	1000	20min

5.5 REQUISITOS

A partir da análise de domínio e da arquitetura definida, foram levantadas as principais funcionalidade do sistema, por exemplo, com o cadastro do usuário as músicas e informações estarão vinculados aquele usuário. Então, com bases nas informações obtidas, foram identificadas funcionalidades ditas essências: recuperação das músicas, sair do sistema, entrar no sistema, cadastrar novos usuários. Além das funcionalidades ditas essências, a instância do *software* Web, deverá conter um *player* de músicas via *streaming* dentro do próprio sistema Web, permitindo ao usuário escutar suas músicas de forma online.

Na Tabela 10, temos a descrição dos requisitos do sistema:

5.6 DIAGRAMA DE CASO DE USO E CLASSE

Para dar proseguimento ao experimento foram criados diagramas de caso de uso e diagrama de classes para melhor entendimento da aplicação Web. Aqui nos faremos uma breve descrição sobre os dois diagramas.

O diagrama de caso de uso tem a finalidade de mostrar de forma visual como é o comportamento geral do sistema em um nível mais alto de abstração, sem entrar em detalhes de implementação(BOOCH; RUMBAUGH; JACOBSON, 2006).

Como pode ser visto no diagrama da Figura 19, o ator **Usuário** interage com o ator **Sistema** acionando os diversos casos de uso, que são as funcionalidade do sistema. O caso de uso **Fazer Login no Sistema** aciona outro caso de uso Visualizar Música que recupera a

Tabela 10 – Requisitos do Sistema

Requisito	**Prioridade**	**Funcionalidade / Característica de Qualidade**	**Ator**
NE01 – Possibilitar ao usuário entrar no sistema	Essencial	F01 – Solicitar *login* no sistema	Usuário
		F02 – Autenticar usuário	Sistema
NE02 – Possibilitar ao usuário sair do sistema	Essencial	F01 - Solicitar saída do sistema	Usuário
		F02 – Fazer *logoff* do usuário	Sistema
NE03 – Permitir que o sistema cadastre novos usuário	Essencial	F01 – Solicitar cadastro	Usuário
		F02 – Cadastrar usuário na base de dados do sistema	Sistema
NE04 – Visualizar todas as músicas armazenadas em sua conta	Essencial	F01 – Mostrar músicas	Sistema
NE05 – Fornecer um player Web de *streaming*	Essencial	F01 – Iniciar streaming da música	Sistema

lista de músicas de acordo com o usuário. O caso de uso **Streaming Música** faz com que a aplicação Web solicite ao servidor de mídia o arquivo em questão para posterior execução.

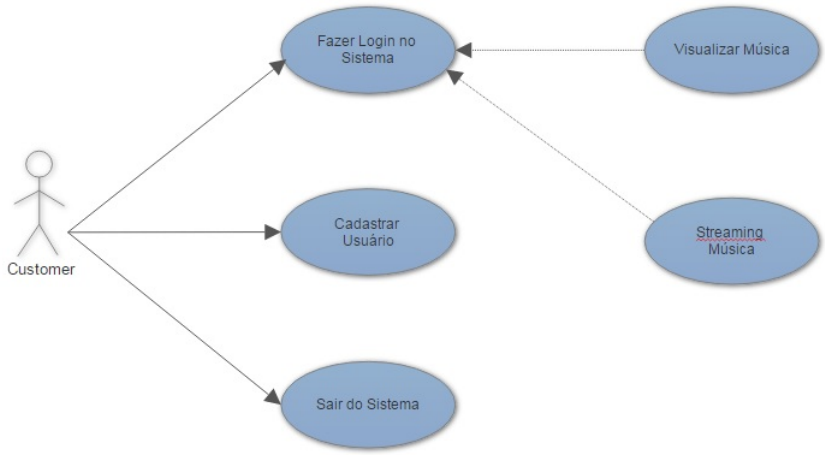

Figura 19 – Diagrama de Caso de Uso

Os diagramas de classe do sistema tem a finalidade de dar uma visão geral das classes do modelo da aplicação (BOOCH; RUMBAUGH; JACOBSON, 2006).

O sistema tem por base duas classes principais, que representam as regras do negócio definidas para o experimento, com isso, temos duas classes para representar as regras do sistema, **Usuario** e **Musica**, e cada com suas características que fazem determinado objeto ser

único a medida que são instanciados. As músicas estão relacionadas a determinado usuário, fazendo com que cada usuário tem aquela quantidade de músicas associadas a sua conta.

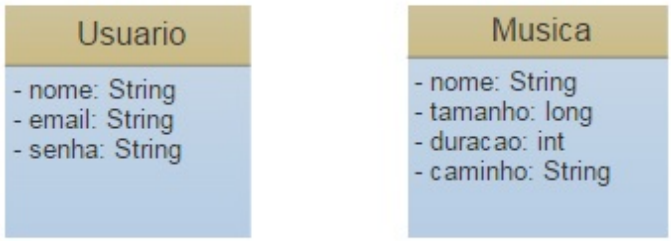

Figura 20 – Diagrama de Classes

5.7 IMPLEMENTAÇÕES DAS APLICAÇÕES

Conforme falamos no capítulo anterior usaremos alguns *framework* para desenvolvimento das aplicações que servirão de experimento. No caso da aplicação em Java vamos usar o JSF na sua versão 2.2. Usaremos o Eclipse como IDE para ajudar no desenvolvimento, testes, etc. Também vamos fazer uso de servidor Web de aplicação Java, nesse caso o Apache Tomcat versão 7. Para a aplicação desenvolvida em Python faremos uso da versão 2.7 do Python, e também do *framework* Web Flask na versão 0.11 que já incluí os *frameworks* Jinja 2 e Werkzeug . Como servidor de aplicação Web para Python vamos usar o Apache versão 2.2 com a instalação do módulo WSGI, que é voltado especificamente para aplicações Web em Python. Para a base de dados do sistema para duas implementações faremos uso do mesmo banco de dados e será utilizado o MySQL em sua versão 5.6.

De acordo com o projeto da aplicação, foram criadas camadas para dividir as responsabilidades de parte do sistema. Uma camada para as regras de negócio da aplicação, ou seja, todas aquelas classes que estão diretamente relacionada com música e usuário. Uma segunda camada de controle, nela as classes farão a comunicação e controle entre a interface e as demais camadas, em Java representado por uma camada onde temos várias classes Controller através do *framework* JSF e em Python temos o arquivo app.py que implementa a lógica de controle pelo *framework* Flask para encaminhamento para as *views*, que são os *templates* HTML. Por fim, a camada de persistência responsável por pelo armazenamento dos dados, incluindo dados dos usuários e suas respectivas músicas, na verdade não é armazenado a música em banco de dados, mas o *link* com a referência de onde a aplicação vai carregar a música. Para fazer a conexão com o MySQL, em Java foi utilizado o *Java Database Connectivity* (JDBC) um drive que fornece várias classes para instruções SQL, e em Python foi utilizado o MySQLdb que é uma interface de comunicação entre o Python e o MySQL.

A interface em Java pode ser vista na Figura 21 e a interface em Python pode ser verificada na Figura 22, porém o que mais no interessa é como a aplicação foi desenvolvida nas duas linguagens, ou seja, o código de *backend*.

Para a lógica implementada em Java junto com JSF, quando o usuário acessa o sistema é direcionado para página de *login*, e o usuário faz *login* em sua conta através da classe LoginBean que é responsável por implementar a lógica de autenticação e redirecionado para tela principal da aplicação. A classe MusicaBean é responsável por carregar todas as músicas que estão relacionadas ao usuário através das classes *Data Acess Objeto* (DAO) que acessam o banco de dados para esse fim, e colocadas em uma lista de objetos da classe do modelo Musica, esses objetos são listados na interface principal para serem executadas quando for pressionado o botão de *play* pelo usuário. Quando o usuário executa a música a classe de controle MusicaBean se carrega de fazer a comunicação para que a música seja executada pelo *player* do HTML 5 através de *streaming* do servidor de mídia pelo *Uniform Resource Locator* (URL) da música.

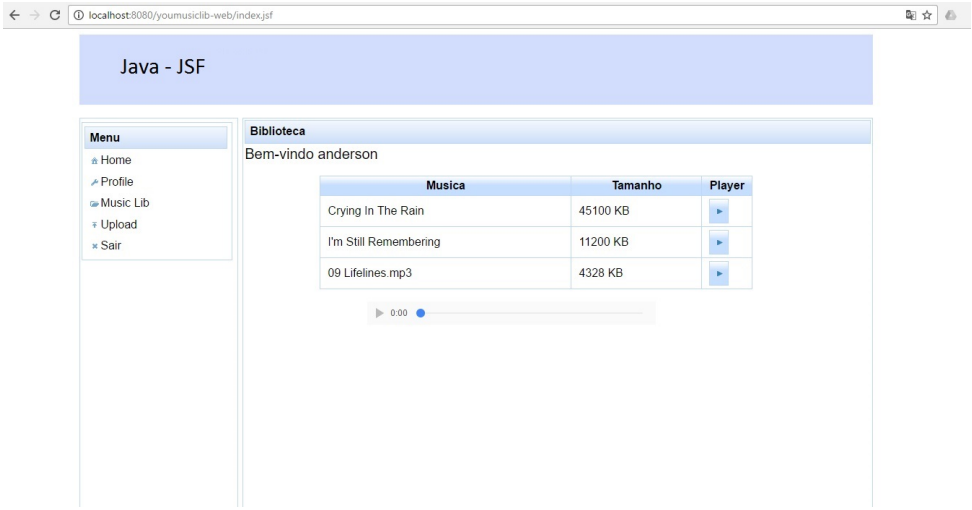

Figura 21 – Tela Inicial em Java

A aplicação em Python com a utilização do Flask, o arquivo app.py é o coração da aplicação, onde acontece todo o roteamento entre as *views* e a lógica da aplicação. De início ele captura a requisição inicial fazendo o roteamento para da página de login. Quando o usuário faz a requisição de *login* o pedido é retornado para o app.py que usa a lógica semelhante a aplicação em Java, é criado um objeto do tipo DAO da classe que tem função de acesso ao banco de dados para então fazer acesso ao sistema. Depois do *login* o usuário é direcionado para a página principal da aplicação, quando isso é feito é passado também como parâmetros uma lista de objetos da classe Música com as músicas associadas aquele usuário que foram recuperadas do banco de dados, assim como acontece em Java. Por fim, esses objetos são colocados na interface como uma lista para que o usuário possa acionar o *play* e então a música ser recuperada do servidor de mídia para ser executada pelo *player* HTML5.

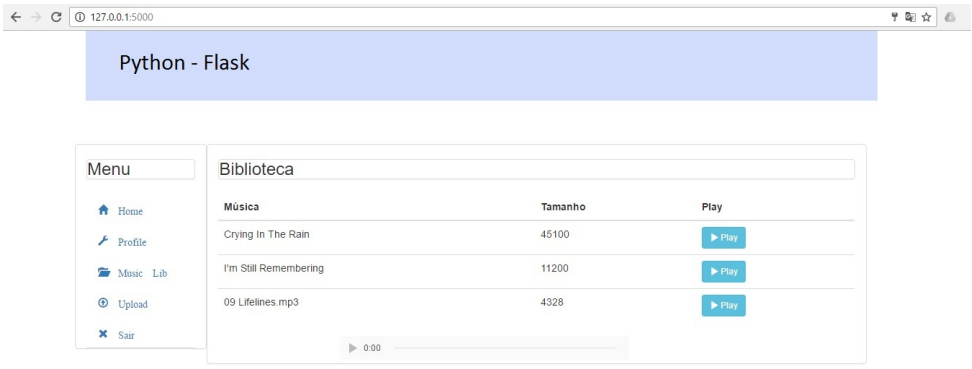

Figura 22 – Tela Inicial em Python

O código fonte das principais classes do experimento em Java podem ser vista

no Apêndice A e o código fonte da aplicação desenvolvida em Pyhton pode ser vista no Apêndice B, neles podem ser verificados com mais detalhes como ficaram os códigos das implementações.

6 *BENCHMARKING* E RESULTADOS

Este capítulo apresenta as etapas finais do método, trabalhando em cima das métricas que foram utilizados para guiar os experimentos realizados, assim como quais os *softwares* utilizados para esta finalidade. A configuração do ambiente de realização dos experimentos também é descrita no que diz respeito aos equipamentos e suas configurações. Também é apresentada a execução dos experimentos em suas duas aplicações previamente selecionadas: uma baseada em Java e outra em Python. Por fim, são apresentados os resultados obtidos e a posterior análise comparativa.

6.1 *BENCHMARK*

O objetivo do trabalho é avaliar o desempenho e escalabilidade das duas implementações da aplicação baseada em *streaming* de mídia. Destacando que o método escolhido para *streaming* das aplicações foi o pseudo *streaming*.

A categoria do *workload* adotado será de servidor Web e as métricas buscam aferir desempenho e escalabilidade; foram consideradas: uso de CPU, uso de memória, tempo de resposta e taxa do transferência do servidor Web. Tais métricas serão calculadas para cada uma das implementações da aplicação, a fim de gerar dados para analisar de forma comparativa. Para isso foram realizadas seis execuções, considerando cada um dos três cenários apresentados na Tabela 9 em cada uma das implementações. A execução do *benchmarking* para obtenção das métricas foi realizada utilizando o Apache JMeter (APACHE SOFTWARE FOUNDATION, 2016).

O JMeter é um *benchmark* mantido pelo Apache e escrito totalmente em Java para execução de testes de carga em servidores Web de várias linguagens e tecnologias, tais como Java, PHP, Python, SOAP, REST, HTTP (APACHE SOFTWARE FOUNDATION, 2016). Ele é bem completo e pode usar configurações para realizar os testes de carga, permitindo que sejam geradas cargas de requisições utilizando os protocolos HTTP, TCP, etc. Para criar planos de teste para simulação ele usa o conceito de *thread* para representar usuários que fazem requisição simultâneas a um servidor Web. Para coleta de informações da simulação é necessário configurar um *Listener*, que coleta os dados relacionados aos recursos de rede utilizados entre servidor Web e o JMeter. Outros dados podem ser coletados através de *plugins*. No nosso caso as métricas obtidas, relacionadas aos requisitos de rede são: tempo de resposta e taxa de transferência da aplicação para o JMeter. As métricas obtidas através de *plugin* externo são: uso de CPU e uso de memória (JMETER PLUGINS, 2016).

O *plugin* utilizado para a coleta de dados dos recursos locais do servidor Web foi o *PerfMon Metrics Collector*. O referido *plugin* possui uma arquitetura cliente-servidor, que

coleta dados locais do servidor através de um agente de *software*, instalado na máquina do servidor Web. Esses dados são enviados em seguida para o módulo cliente, integrado ao JMeter. Além das funcionalidades de coleta de dados locais, o mesmo *plugin* possui a funcionalidade de plotar os gráficos das amostras coletadas.

6.2 AMBIENTE DE EXECUÇÃO

O ambiente que serão executados os experimentos segue a arquitetura do sistema definida na Figura 18 (Seção 5.5). Trata-se de um ambiente controlado sem interferências externas de tráfego de dados, como é o caso de comunicação com servidores distribuídos na Internet. Nesse sentido, a única interferência possível é a interferência eletromagnética entre os cabos *Unshielded Twisted Pair* (UTP) que irão conectar as máquinas ao *switch*, que na nossa hipótese foi considerada desprezível. A adoção de um ambiente controlado e isolado de interferências externas tem o objetivo de verificar a interferência da tecnologia de desenvolvimento utilizada, uma vez que ambas as aplicações possuem as mesmas funcionalidades e são executadas no mesmo ambiente.

Conforme pode-se observar na Figura 18 (Seção 5.5), temos uma arquitetura com dois servidores e uma quantidade variável de usuários finais. A ligação entre eles será através de um *switch* com 4 portas *Fast Ethernet* com cabos UTP nas ligações entre os servidores e clientes.

O servidor Web com as aplicações será executado em um *notebook* com um processador Intel Core 2 Duo com frequência de 2.26GHz, memória RAM de 4GB, sendo o Sistema Operacional Windows 7 de 64 bits. Para o experimento da aplicação em Java será utilizado o servidor de aplicação Apache Tomcat 7. Para o experimento da aplicação em Python será o utilizado o servidor Web Apache 2.2 configurado para funcionar como o Python e o Flask.

O servidor de mídia, em ambas as aplicações, foi hospedado em um *notebook* com processador Intel Pentium com uma frequência de 2.13 GHz e memória RAM de 2GB, funcionando com SO Windows 7 de 64 bits. O servidor Web utilizado para armazenar as mídia de áudio foi o Apache 2.2.

O sistema de monitoração JMeter também foi o mesmo nas duas aplicações, sendo executado em um *notebook* com processador Intel Core i7 com frequência de 1.8GHz, memória RAM de 8GB, e executando o SO Windows 7 de 64 bits. A razão para ter optado por uma máquina de melhor configuração é o fato dessa máquina ter a necessidade de iniciar múltiplas *threads* para simular múltiplos clientes simultâneos.

Tabela 11 – Criação de Usuários

Usuários	**Quantidade de Criação**	**Intervalo de Criação**	**Tempo de Criação**	**Tempo de Finalização**
10	1	10 segundos	10 segundos	10 segundos
100	10	10 segundos	10 segundos	10 segundos
1000	100	10 segundos	10 segundos	10 segundos

6.3 EXECUÇÃO

De acordo com o que foi definido nas Seções 6.1 e 6.2 o agente do *PerfMon Metrics Collector* ficará na mesma máquina que hospedou o servidor Web do experimento, para então coletar os dados. Como mencionado, as mídias de áudio foram hospedadas em outro servidor. Como dito anteriormente, a máquina que simula os clientes simultâneos é a mesma que hospeda o JMeter, que executará a aplicação de *streaming* desenvolvida em cada uma das *threads*.

As aplicações Web executam em um navegador com suporte a HTML5 que tem suporte ao *player* de mídia. Um navegador HTML5 segue a RFC2616 (FIELDING et al., 2009) no que diz respeito a transferência de mídia, que define que o agente (navegador Web), pode definir um intervalo do fragmento do arquivo em *bytes* (*chunks*) que vai ser solicitado ao servidor. Como existe a possibilidade do tamanho do *chunk* variar de acordo com o servidor Web utilizado, no experimento o navegador Web foi configurado para solicitar sempre o tamanho padrão do DASHEnconder, que é de 2000 *bytes* (LEDERER; MÜLLER; TIMMERER, 2012). No JMeter configuramos os HTTP *Request* para fazer iteração em *chunks* de 2000 *bytes* conforme explicado acima, e assim simulando um pseudo *streaming*. A Tabela 11 apresenta a dinâmica de criação das *threads* que simulam os clientes do serviço de *streaming*. Como pode ser observado, para 10 usuários definimos a criação e finalização de 1 usuário a cada 10 segundo, para 100 usuários a criação de 10 usuários a cada 10 segundos com tempo de criação em 10 segundos, e finalmente para os 1000 usuários definimos a criação de 100 usuários a cada 10 segundo com tempo de criação de 10 segundos.

Em relação à configuração do tempo que uma conexão TCP pode ficar operando (*KeepAliveTime*), configuramos nas máquinas com Windows para o tempo padrão de 7,000,00 milissegundos ou duas horas, já que as máquinas não estão com versões do Windows Server e essa configuração não vem habilitada por padrão (MICROSOFT, 2016). Já o *KeepAliveTime* do HTTP faz uso de apenas uma conexão TCP para fazer várias requisições e *MaxKeepAliveRequests* diz quantas requisições HTTP podem ser feitas por conexão TCP (MAXCDN, 2016). Nesse caso configuramos tanto no Apache 2.2 para a aplicação em Python como no Apache Tomcat 7 para a aplicação em Java o tempo de 60 segundos no *KeepAliveTime* e conexões infinitas no *MaxKeepAliveRequests*. Essa decisão decorre da natureza da aplicação com *streaming*, já que são feitas várias requisições HTTP para diferentes *chunks* da mídia.

6.4 ANÁLISE DOS RESULTADOS

Após a execução dos experimentos passamos a analisar os resultados do *benchmarking* nas duas aplicações desenvolvidas. Antes de continuar com a análise dos dados é importante frisar algumas observações na fase de execução. Um fato interessante que foi observado no JMeter, que dependendo da quantidade de usuários, o tempo de simulação vai além do tempo definido para que o JMeter possa finalizar todos os usuários virtuais. Porém, como nossos cenários são de 20 minutos de execução, foram considerados nos gráficos apenas o período determinado. Além disso, há intervalos de tempo em algumas métricas em que o JMeter não consegue coletar nenhum dado ou coleta de forma incoerente, observado através do arquivo CSV gerado por ele. Por exemplo, para a métrica de uso de CPU e memória em Java, nos intervalos entre 463 e 668 segundos de simulação o JMeter não conseguiu coletar dados, e assim repetindo nesse período o último dado coletado.

6.4.1 *Benchmarking* de Uso de CPU e Memória

Essa seção analisa a comparação em termos de uso de CPU e memória, que são consideradas métricas importantes para avaliar a escalabilidade de cada tecnologia em função dos recursos locais utilizados nas máquinas. A seguir estão dispostos em gráficos os resultados dos cenários para essas métricas.

O primeiro cenário considera 10 usuário em execução simultânea. No gráfico da Figura 23 pode-se observar que Java tem um consumo de CPU maior que Python durante todo o teste de carga e também em alguns períodos menores de tempo, chegando a picos em torno de 60% a 70% de consumo com uma média de 44.59%, apresentando consumo médio de 11.42% da CPU. Já Python obteve picos em torno de 40% de memória RAM e uma média de 40.06%, com um consumo médio de CPU em torno de 5.90%. Para ambas aplicações, esse consumo se mantem estável durante todos os 20 minutos do teste.

No cenário com 100 usuário simultâneos os resultados mudam um pouco. O consumo

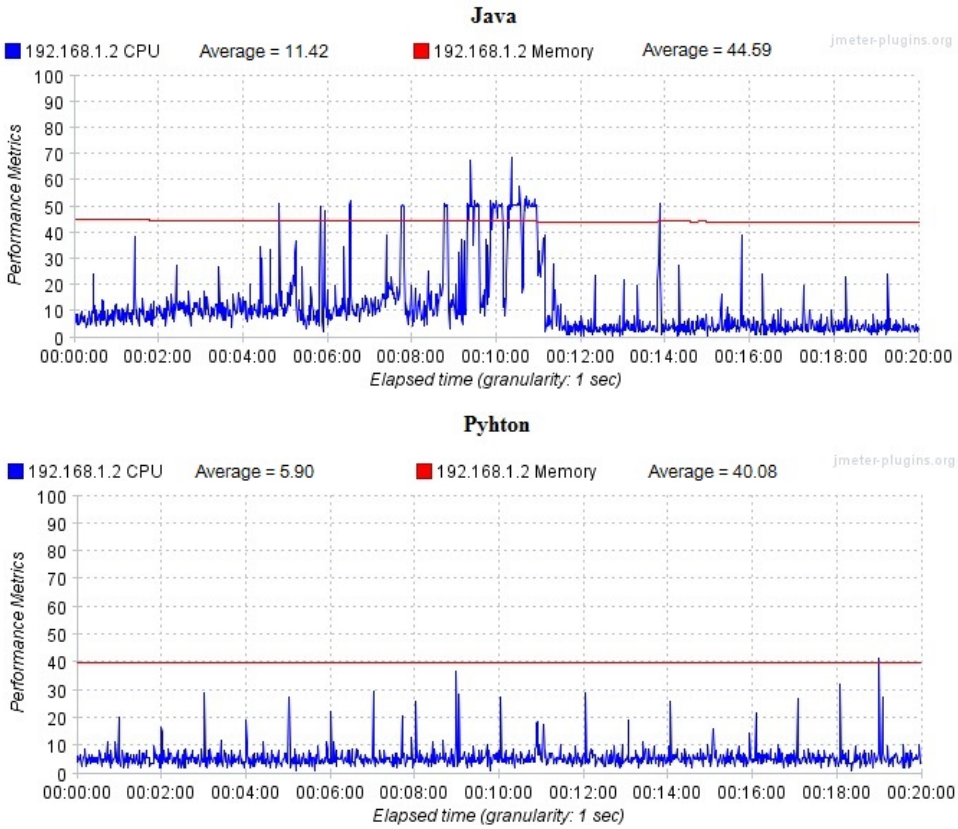

Figura 23 – *Benchmarking* CPU e Memória para 10 Usuários

de CPU se torna muito semelhante nas duas aplicações, com consumo de memória RAM e CPU bastante próximos, com Java tendo um pico de consumo levemente maior em certo intervalo de tempo. As médias de consumo de memória foram 44.17% em Java e 40.12% em Python. Já as médias de uso de CPU foram 8.67% em Java e 6.58% em Python. Vale observar que o aumento comparativo entre os dois cenários foi muito mais acentuado para a solução baseada em Python, o que pode representar um comportamento nocivo em termos de escalabilidade. Em Java foi percebida até uma pequena redução, o que pode ser interpretado como manutenção da média, tendo em vista as pequenas variações que podem ocorrer entre execuções.

No cenário com 1000 usuários simultâneos, os resultados são bastante interessantes. O uso de CPU em Java se torna um pouco menor que em Python, a primeira com uma média de 8.32 e a segunda com 8.75. O Consumo da memória RAM também segue o mesmo padrão, mas ambas com médias próximas. Esse padrão de consumo ratifica a tendência da solução baseada em Java ser ligeiramente mais escalável que a solução baseada em Python. Contudo ambas as aplicações apresentam períodos sem monitoração de dados, o que nos leva a crer que foi devido à alta quantidade de requisições simultâneas aos servidores Web, fazendo com que o JMeter não conseguisse coletar de forma constante os dados do servidor.

Com a análise de todos os cenários, podemos inferir que Java consegue melhorar seu

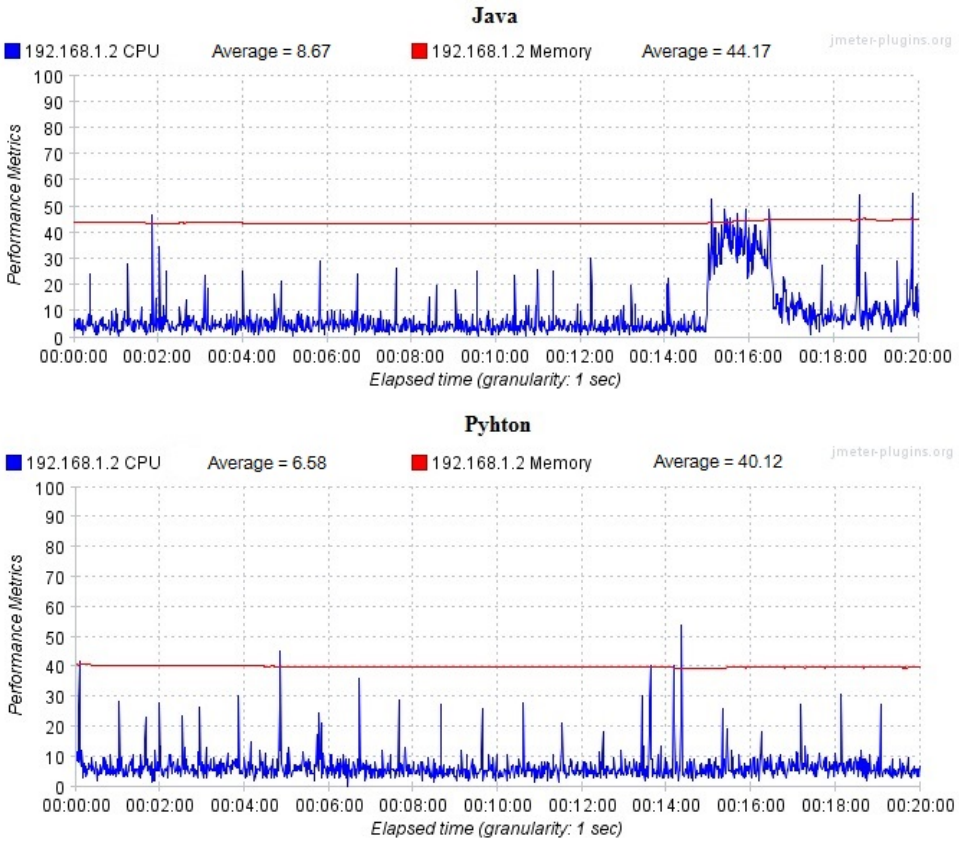

Figura 24 – *Benchmarking* CPU e Memória para 100 Usuários

consumo de acordo com uma quantidade maior de usuário acessando a aplicação, tanto em termos de uso de CPU e memória RAM. E Pyhton por sua vez, à medida que a quantidade de usuários cresce o consumo aumenta linearmente, ainda que de forma sutil. Isso é fato muito importante na escolha da tecnologia a ser adotada de acordo com a regra de negócio do projeto, já que com uma quantidade cada vez maior de usuários as tecnologias se comportam de forma diferente. Então como em uma aplicação de *streaming* um dos pontos relevantes é a quantidade de usuários acessando a aplicação ao mesmo tempo, de acordo com os resultados obtidos nesse trabalho, é interessante a escolha de tecnologias Java quando se tem preocupação com o consumo de CPU e memória RAM à medida que a quantidade de usuário cresce fortemente.

6.4.2 Tempo de Resposta

Outro critério de comparação considerado foi o tempo de resposta das aplicações. Foi considerado um item relevante para aplicações Web, que estão em locais geograficamente distintos. A utilização de um ambiente controlado nos experimentos oferece uma maior precisão de como as aplicações se comportam com esse tipo de métrica, uma vez que não sofre interferências externas, como por exemplo oscilação de velocidade de conexão com a Internet.

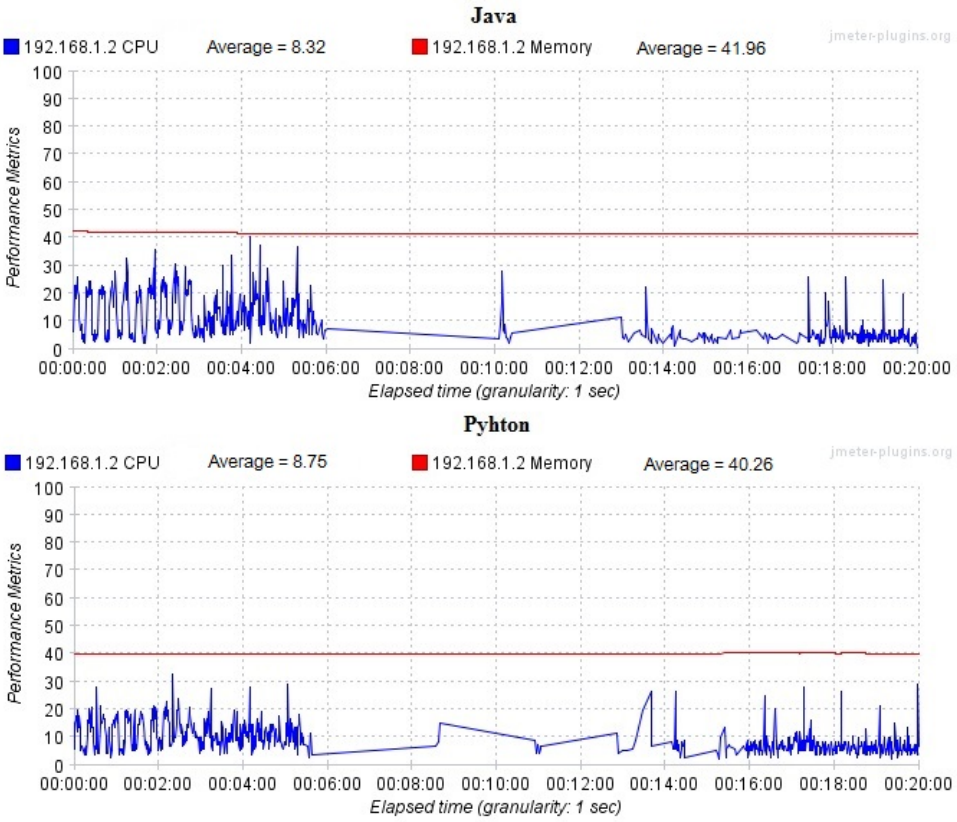

Figura 25 – *Benchmarking* CPU e Memória para 1000 Usuários

No cenário com 10 usuários simultâneos, conforme gráfico da Figura 26, percebe-se uma grande diferença entre as duas aplicações avaliadas. A aplicação baseada em Java possui tempo de resposta muito acima dos registrados para a aplicação baseada em Python. Para se ter ideia, o pico mais alto que a solução em Python apresentou foi de 2.400 milissegundos, enquanto que a solução em Java apresentou picos de quase 800.000 milissegundos. No decorrer do tempo de simulação as aplicações se comportaram de forma distinta, com picos altos e baixos em intervalos diferentes.

No cenário com 100 usuário simultâneos as discrepâncias foram reduzidas, quando comparado com a simulação com 10 usuários, conforme gráfico da Figura 27. A solução baseada em Java apresentou picos levemente mais altos que a solução baseada em Python, com o pico máximo de 400.000 milissegundos e média de 20.615 milissegundos; enquanto a solução baseada em Python apresentou um tempo de resposta menor, com média em torno de 9.060 milissegundos. Apesar disso, foi registrado um pico de tempo maior que a solução em Java, chegando próximo dos 700.000 milissegundos.

No cenário envolvendo 1000 usuários simultâneos (Figura 28), foi percebido um padrão semelhante ao cenário anterior. A solução baseada em Java apresentou tempo de resposta levemente maior que a solução baseada em Python, com média de 82.943 milisegundos contra 67.452 milisegundos. Mas dessa vez o pico de tempo de resposta

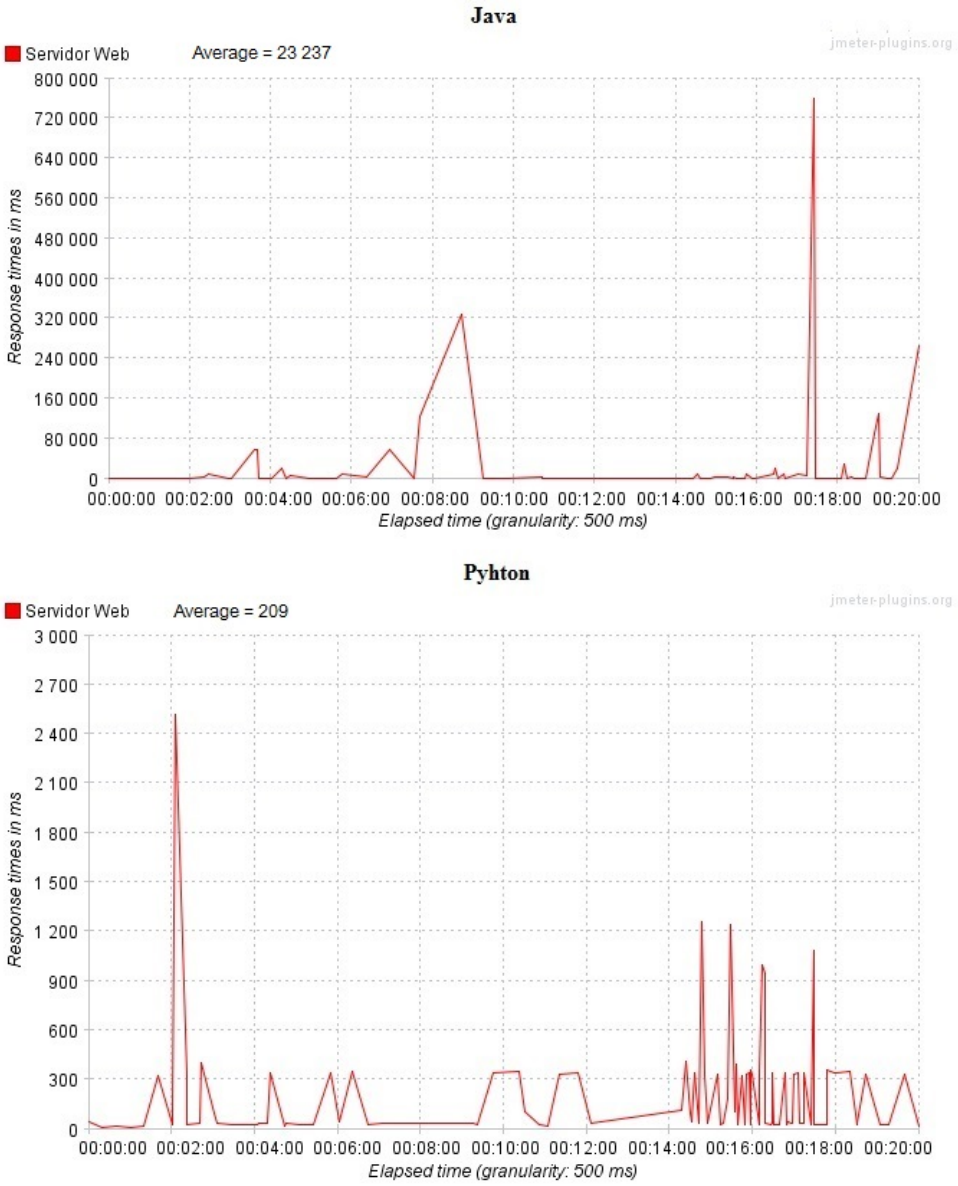

Figura 26 – *Benchmarking* Tempo de Resposta para 10 Usuários

registrado também se manteve contrário à solução baseada em Java.

No geral, apesar da grande discrepância percebida no primeiro cenário, foi percebido que ambas as soluções oferecem um tempo de resposta satisfatório, com vantagem para a solução baseada em Python. Porém, o resultado reforça mais uma vez a hipótese de que a solução baseada em Java possui uma menor taxa de degeneração do sistema; isto é, à medida que o número de usuários cresce, a diferença entre a solução baseada em Java e a solução baseada em Python diminui. No cenário com 10 usuários simultâneos, Java se mostrou 11518,5% mais lento; com 100 usuários simultâneos, essa diferença caiu para 127,54% e com 1000 usuários simultâneos, caiu para 22,97%. Esses resultados nos levam a crer que, dependendo do número de acessos simultâneos esperado para a aplicação, há uma forte tendência que a escolha da tecnologia no quesito tempo de resposta tenda para Python,

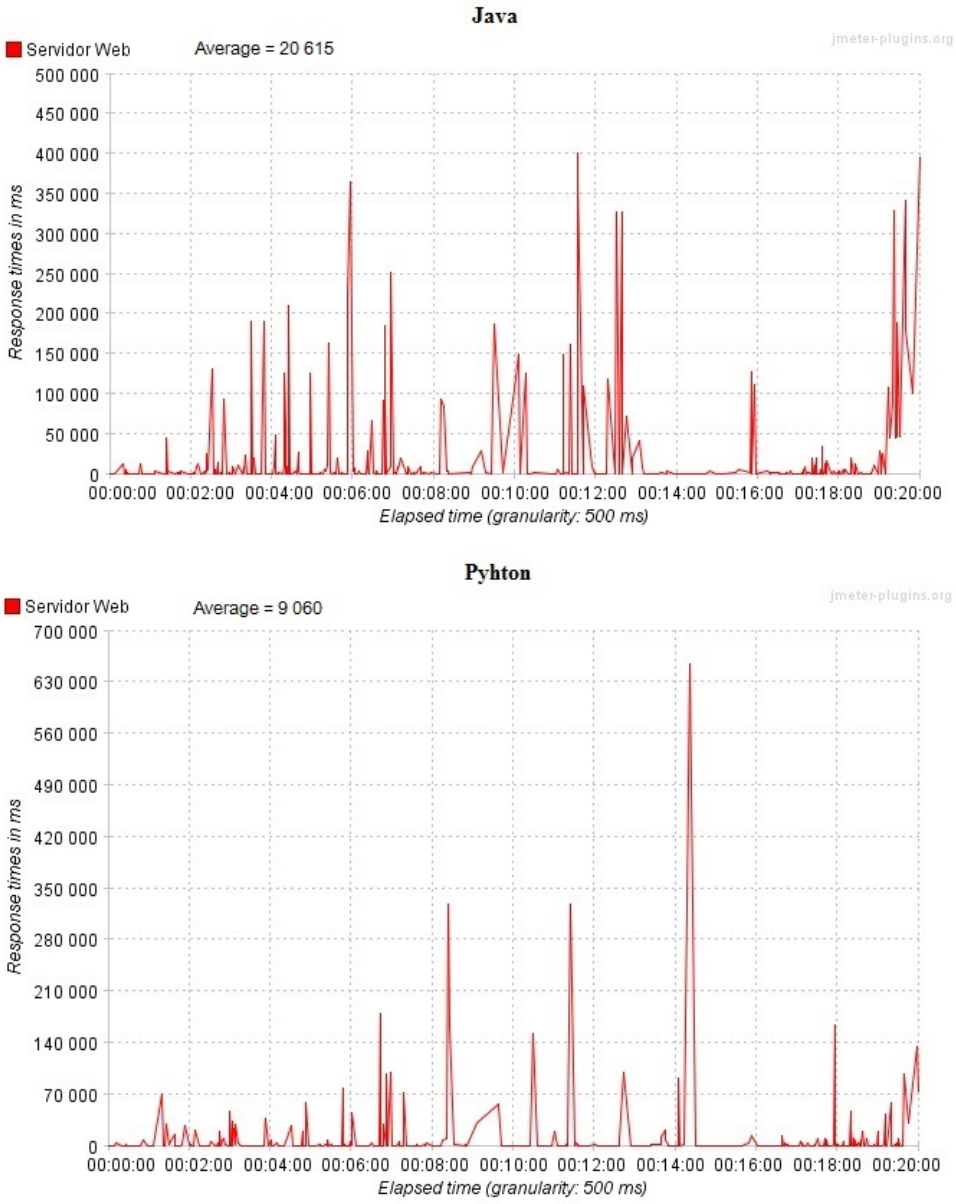

Figura 27 – *Benchmarking* Tempo de Resposta 100 Usuários

apesar que a solução em Java diminui a degeneração do sistema em relação à Python à medida que a quantidade de usuários cresce, não sendo possível no experimento identificar um ponto onde a solução em Java escala melhor.

6.4.3 Taxa de Transferência

O último fator de comparação entre as duas aplicações foi a taxa de transferência, importante métrica para saber o volume de dados que cada aplicação trafegou durante o experimento, conforme as requisições de *streaming* vão acontecendo durante o período da simulação.

A carga em *bytes* gerada no primeiro teste da métrica para 10 usuários simultâneos

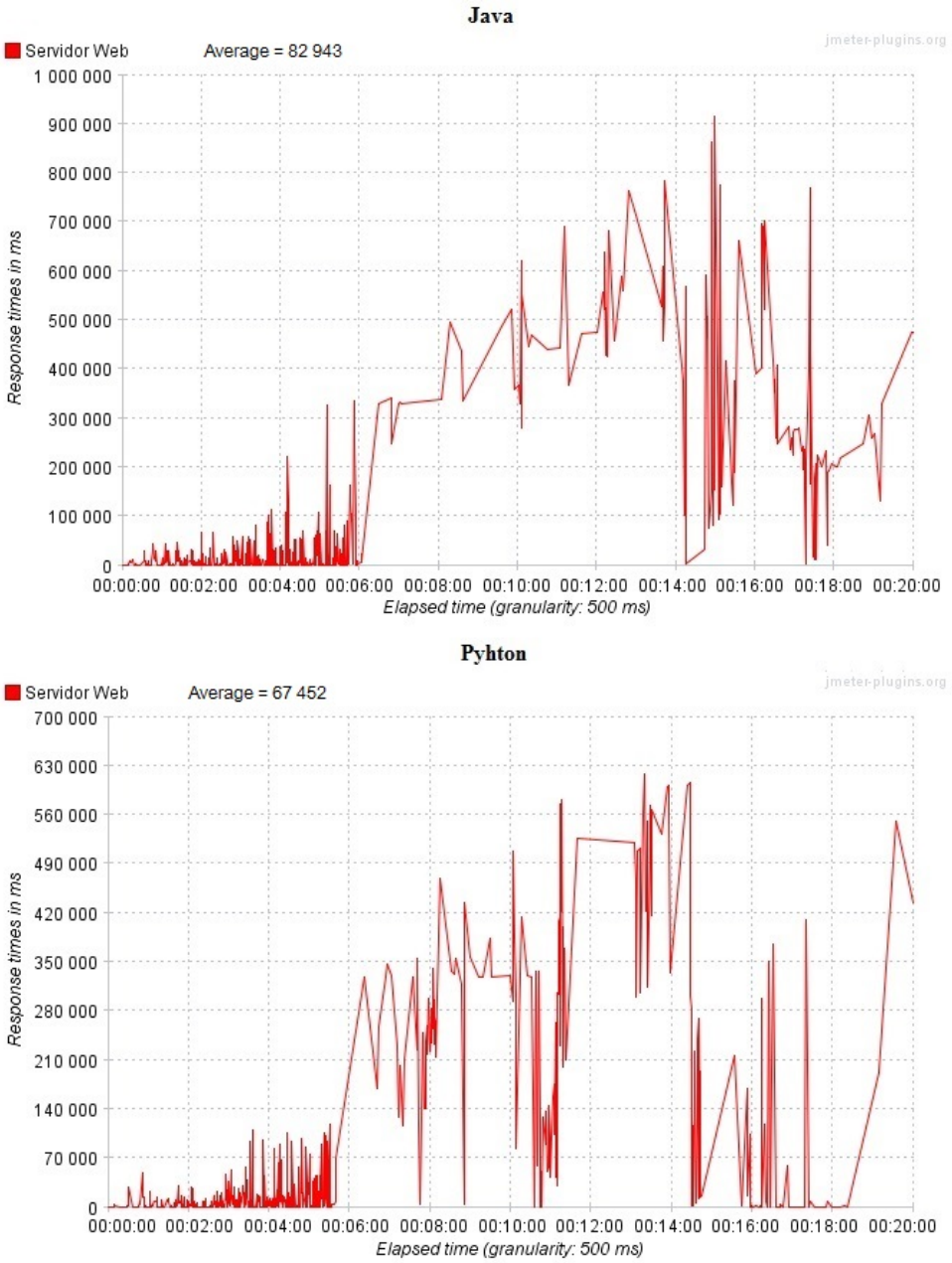

Figura 28 – *Benchmarking* Tempo de Resposta 1000 Usuários

é praticamente igual para as duas implementações, conforme o gráfico da Figura 29 e suas respectivas médias, Java com 13.766.199 *bytes* e Python com 13.426.115 *bytes*. No decorrer da simulação a quantidade de tráfego aumenta na mesma proporção para as duas aplicações, provavelmente mais uma vez relacionado à quantidade de usuários que fazem requisição ao mesmo tempo, mesmo em um ambiente com apenas 10 usuários concorrentes.

O padrão de tráfego de dados permaneceu uniforme quando considerado o cenário com 100 usuários simultâneos. O gráfico da Figura 30 mostra que a taxa de transferência em *bytes* é praticamente igual, Java com média de 13.302.749 *bytes* e Python com 13.314.665 *bytes*. E no decorrer do tempo o aumento da taxa permanece uniforme para duas aplicações.

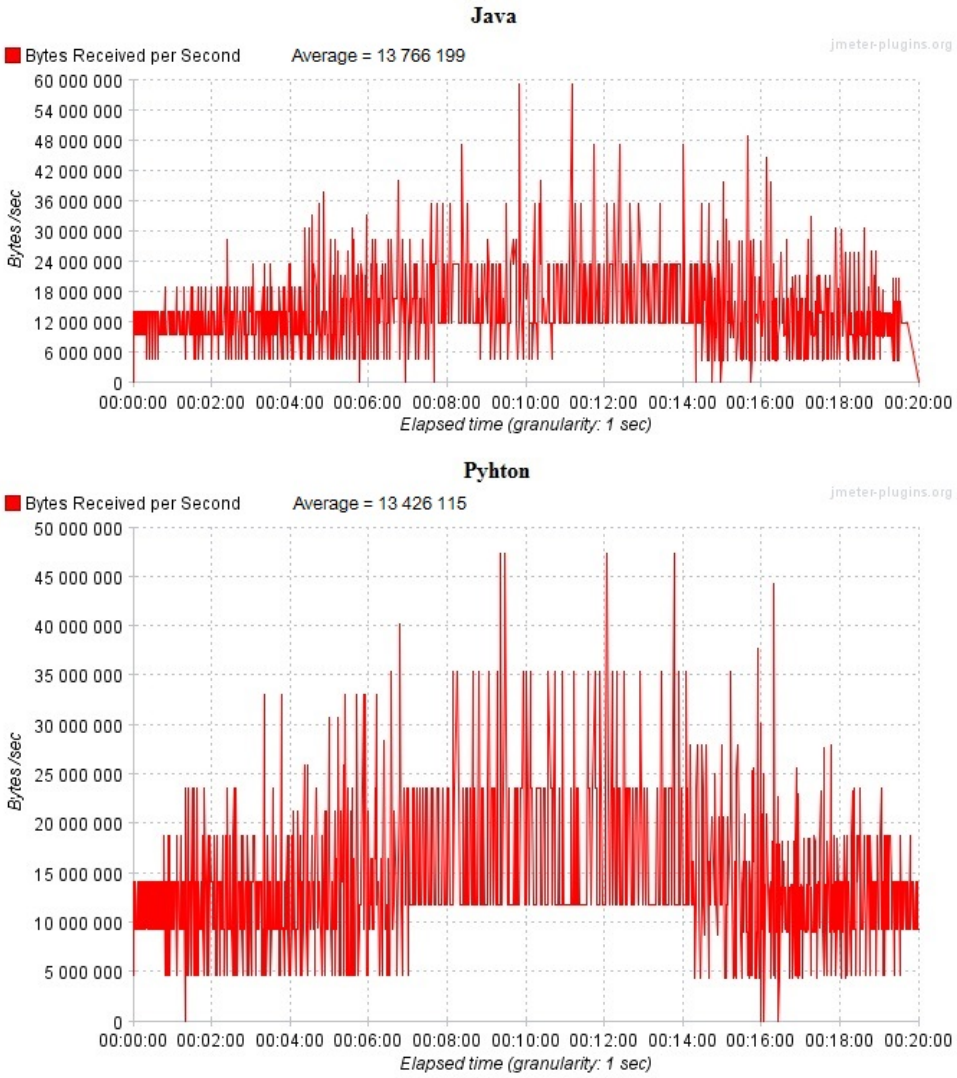

Figura 29 – *Benchmarking* Taxa de Transferência 10 Usuários

No cenário envolvendo 1000 usuários simultâneos, os resultados apresentados no gráfico da Figura 31 não diferem consideravelmente dos dois cenários anteriores. Porém, a taxa de transferência da solução baseada em Python se torna um pouco maior que Java, apresentando média de 11.029.361 bytes. No decorrer do tempo há um fato interessante a se observar. No período de 6 a 12 minutos, não foi registrado envio de dados por parte da aplicação em Java. Já a solução baseada em Python, apesar de também haver períodos sem registro de envio de dados nesse período de 6 a 12 minutos, foram registrados alguns envios de dados. Os resultados nos levam a crer que o período de 6 a 12 minutos foi o período com maior número de requisições simultâneas de clientes e por essa razão, o JMeter ficou sobrecarregado e não conseguiu monitorar satisfatoriamente o tráfego da rede.

Com os resultados apresentados nessa métrica verificamos que a taxa de transferência em Java com JSF e Python com Flask são similares. A solução baseada em Python apresentou um melhor comportamento em situações diversas, conforme apresentado anteriormente

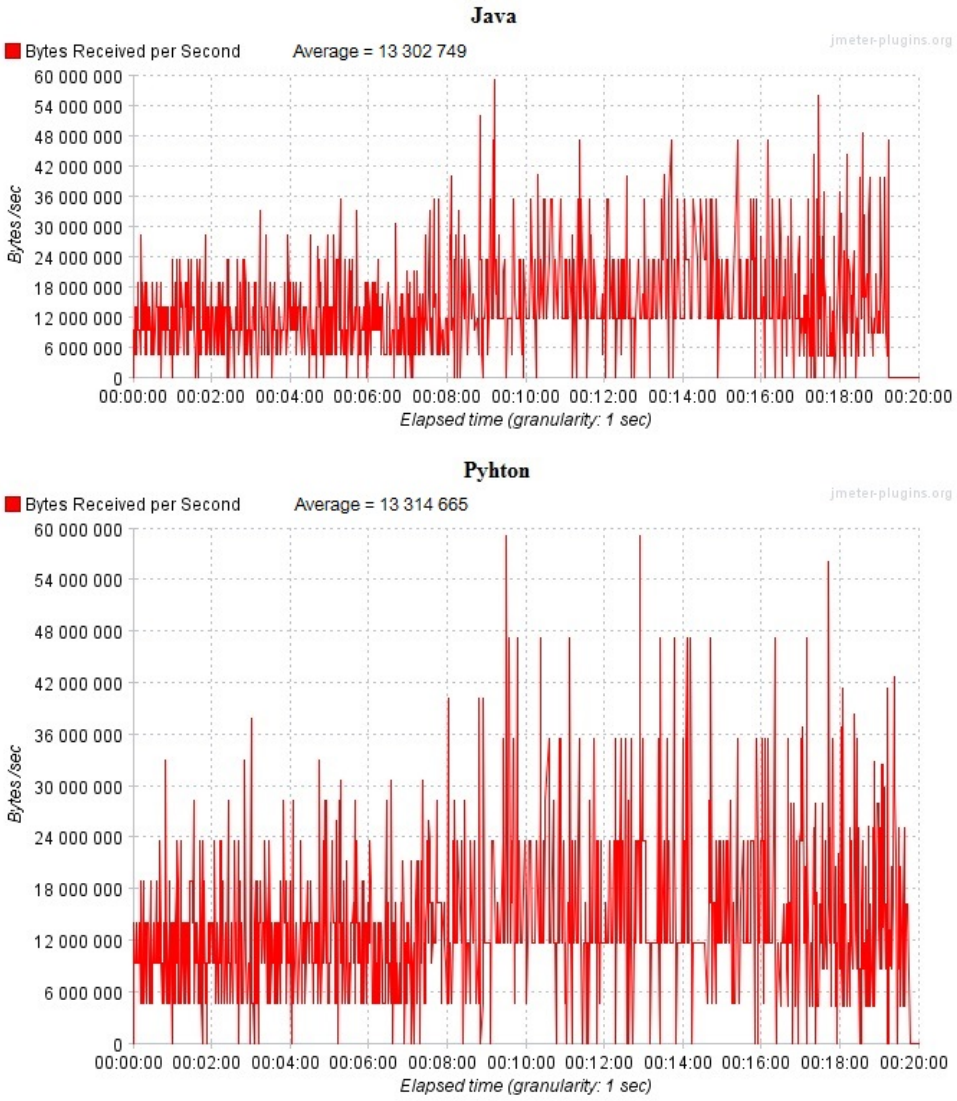

Figura 30 – *Benchmarking* Taxa de Transferência 100 Usuários

no cenário com 1000 usuários, mesmo assim a solução em Java teve uma média de tempo de resposta menor nesse cenário de 1000 usuários. De acordo com os 3 cenários não é possível afirmar que alguma das duas linguagens se torna mais eficiente que outra quando a quantidade de usuários é aumentada ou existe uma quantidade determinada. O que se pode afirmar, de acordo com o resultados, é que a escolha de uma das duas tecnologias analisadas no experimento não exerce influência significativa no quesito taxa de transferência. Esse resultado já era esperado, uma vez que em ambos os casos, o servidor de conteúdo era o mesmo, conforme apresentado na Seção 6.2, tanto em termos de *hardware*, quanto em termos de software (Apache 2.2).

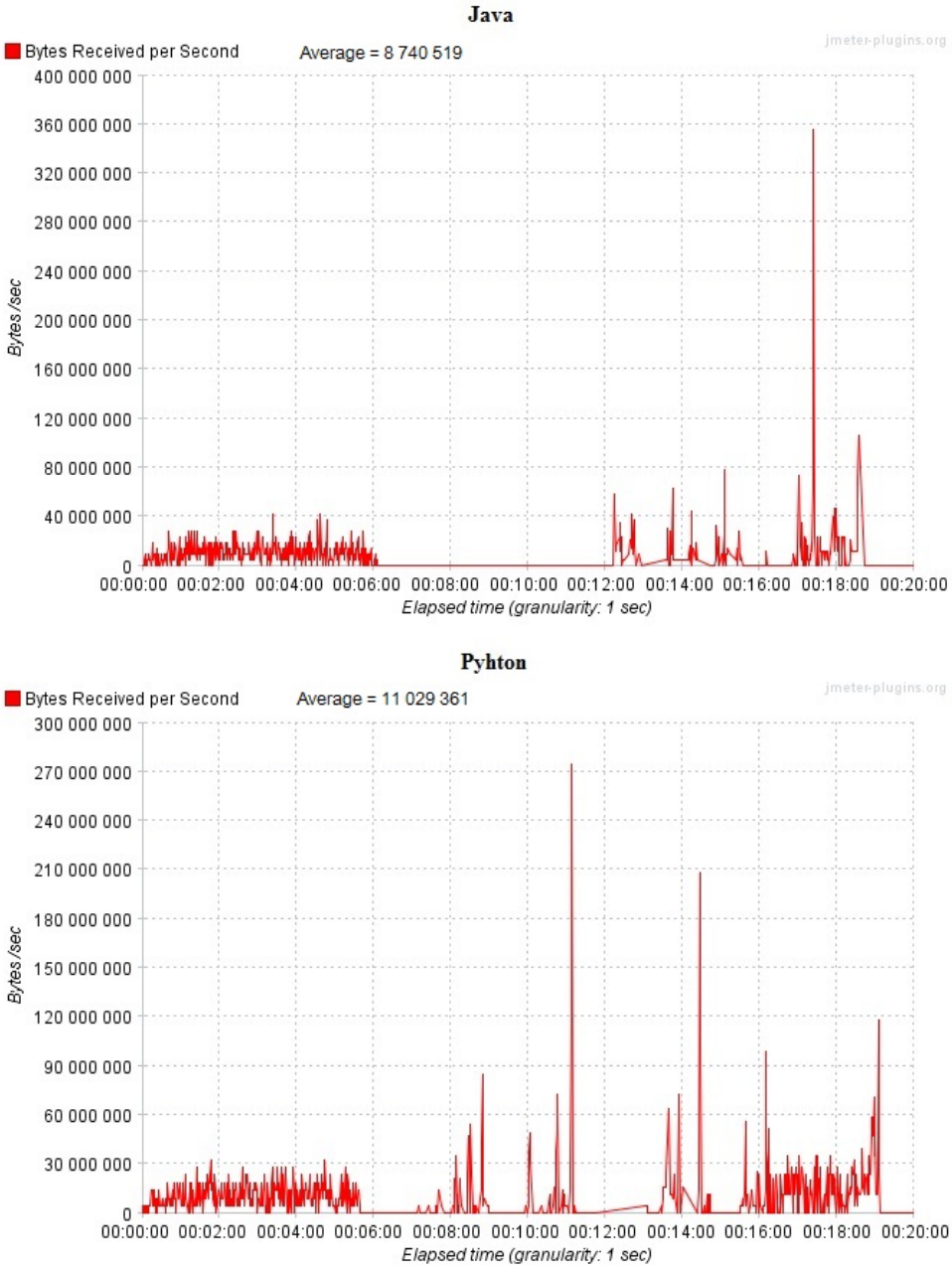

Figura 31 – *Benchmarking* Taxa de Transferência 1000 Usuários

7 CONCLUSÃO

Esta dissertação apresentou um método para apoiar decisões de projeto no desenvolvimento de aplicações Web que usam *streaming* de mídia no que se refere a escolha das tecnologias, no caso de linguagens de programação. Para isso, inicialmente foram avaliados repositórios ativos, com o intuito de identificar as tendências atuais de tecnologias de desenvolvimento de *software* para identificar as mais promissoras, no caso duas: uma baseada em Java e outra baseada em Python. Prosseguindo no método, foi definida uma arquitetura de referência para servir de bases para aplicações. Em seguida, foi executado um experimento para realizar uma comparação sistemática entre as duas tecnologias. Para isso, conforme o que foi definido nas etapas anteriores do método, foi desenvolvido um projeto simples de uma aplicação Web teste baseada em *streaming* de áudio para o desenvolvimento das aplicações nas duas tecnologias a serem comparadas e posterior execução dos experimentos. Os resultados obtidos foram através da técnica de medida de desempenho para as métricas de uso de CPU, uso de memória RAM, tempo de resposta e taxa de transferência.

Umas das contribuições deste trabalho é a identificação e análise das principais arquiteturas de aplicações com *streaming* de mídia utilizadas pelas principais empresas nesse segmento. Além de identificar quais as tecnologias que estão mais em evidência quando do desenvolvimento da pesquisa desse trabalho.

Por fim, a maior contribuição vem do método apresentado através da aplicação de suas etapas, principalmente dos resultados dos experimentos da aplicações, que é a finalidade do método. No caso em questão, da aplicação no contexto do trabalho, foi possível inferir em quais aspectos as tecnologias escolhidas se saem melhor. Um dos principais achado diz respeito ao fato de que a solução baseada em Java se mostrou com maior potencial de escalabilidade, quando comparada com a solução baseada em Python. Isso foi percebido comparando os resultados das métricas à medida que o número e usuários simultâneos aumentada. Também foi observado que em relação ao tempo de resposta, Python tem uma melhor performance, tendo um tempo mais baixo que Java em todas as simulações. Porém, a diferença tende a cair à medida que o número de usuários simultâneos cresce. Em relação à taxa de transmissão, as tecnologias analisadas se comportaram de forma semelhante, apresentando a mesma taxa de *bytes* gerados pela aplicação e enviado para os clientes.

Ademais, com a apresentação e aplicação do método, espera-se ajudar novos projetos, no tocante à seleção do conjunto de tecnologias para aplicação Web com *streaming* de mídia.

7.1 *TRABALHOS FUTUROS*

Como trabalho futuro temos algumas sugestões a destacar:

- A aplicação do método, através de *benchmarking*, para avaliar o desempenho envolvendo diferentes protocolos de *streaming*;
- A aplicação do método, definindo como foco a avaliação individual dos principais *frameworks* das linguagens de programação consideradas;
- Por último, a aplicação do método no tocante a avaliação de desempenho e escalabilidade entre servidores Web de uma mesma linguagem de programação.

REFERÊNCIAS

ADHIKARI, V. K. et al. Unreeling netflix: Understanding and improving multi-cdn movie delivery. In: IEEE. **INFOCOM, 2012 Proceedings IEEE**. [S.l.], 2012. p. 1620–1628.

ADOBE SYSTEMS INCORPORATED. **General information about coding in Dreamweaver**. 2016. Disponível em: <https://helpx.adobe.com/dreamweaver/using/general-information-coding-dreamweaver.html>. Acesso em: 03/10/2016.

APACHE SOFTWARE FOUNDATION. **Apache JMeter**. 2016. Disponível em: <http://jmeter.apache.org/>. Acesso em: 03/10/2016.

ASSOCIATION, E. C. M. et al. **Standard ECMA-334: C# Language Specification**. [S.l.]: ECMA, 2006.

BEGEN, A.; AKGUL, T.; BAUGHER, M. Watching video over the web: Part 1: Streaming protocols. **IEEE Internet Computing**, IEEE, v. 15, n. 2, p. 54–63, 2011.

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. **UML: guia do usuário**. [S.l.]: Elsevier Brasil, 2006.

BORGES, L. E. **Python para desenvolvedores**. [S.l.]: Novatec Editora, 2014.

BUILTWITH. **Framework Usage Statistics**. 2016. Disponível em: <http://trends.builtwith.com/framework>. Acesso em: 04/07/2016.

CHOUDHURY, N. World Wide Web and Its Journey from Web 1 . 0 to Web 4.0. **International Journal of Computer Science and Information Technologies (IJCSIT)**, v. 5, n. 6, p. 8096–8100, 2014.

CISCO, C. V. N. I. Global mobile data traffic forecast update, 2015–2020. **White paper**, 2016.

CODEEVAL. **Most Popular Coding Languages of 2016**. 2016. Disponível em: <http://blog.codeeval.com/>. Acesso em: 04/07/2016.

COLLIER, D. a.; MEYER, S. M. An empirical comparison of C, C++, Java, Perl, Python, Rexx, and Tcl. **International Journal of Operations & Production Management**, v. 20, n. 6, p. 705–729, 2000. ISSN 0144-3577.

CONKLIN, G. J. et al. Video Coding for Streaming Media Delivery on the Internet. **IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY**, v. 11, n. 3, 2001.

DEITEL, H. **M.; DEITEL, Paul J. Java: Como programar. 8ª Edição**. [S.l.]: São Paulo: Prentice Hall, 2010.

FIELDING, R. et al. Rfc 2616, hypertext transfer protocol–http/1.1, 1999. **URL http://www. rfc. net/rfc2616. html**, 2009.

FORTINO, G. et al. A Java-based adaptive media streaming on-demand platform. **Proceedings of EUROMEDIA, Society for Computer Simulation**, 2002.

FOURMENT, M.; GILLINGS, M. R. A comparison of common programming languages used in bioinformatics. **BMC bioinformatics**, BioMed Central, v. 9, n. 1, p. 1, 2008.

GITHUB. 2016.

GROSS, J.; EMMELMANN, M.; WOLISZ, A. Performance Comparison of Dynamic Web Platforms. **Focus**, n. November, p. 1–13, 2007.

GUO, L. et al. Analysis of multimedia workloads with implications for internet streaming. In: ACM. **Proceedings of the 14th international conference on World Wide Web**. [S.l.], 2005. p. 519–528.

HICKSON, I.; HYATT, D. Html5: A vocabulary and associated apis for html and xhtml. **W3C Working Draft, May**, v. 25, 2011.

HOTFRAMEWORKS. **Find your new favorite web framework**. 2016. Disponível em: <http://hotframeworks.com/#rankings>. Acesso em: 04/07/2016.

IEEE SPECTRUM. **The 2016 Top Programming Languages**. 2016. Disponível em: <http://spectrum.ieee.org/computing/software/the-2016-top-programming-languages>. Acesso em: 28/09/2016.

JAIN, R. **The art of computer system performance analysis: techniques for experimental design, measurement, simulation and modeling**. [S.l.: s.n.], 1991. ISBN 0471503363.

JEON, Y.-h.; JANG, J.-s. Design and Implementation of Java-based Software Streaming Services with Enhanced Performance. v. 8, n. 2, p. 72–81, 2008.

JMETER PLUGINS. **JMeter Plugins**. 2016. Disponível em: <https://jmeter-plugins.org/>. Acesso em: 03/10/2016.

JOHN, L. Performance Evaluation : Techniques , Tools and Benchmarks. **Digital Systems and Applications**, p. 1–20, 2007.

JW PLAYER. **JW6: Pseudo Streaming in Flash**. 2016. Disponível em: <https://support.jwplayer.com/customer/portal/articles/1430518-pseudo-streaming-in-flash>. Acesso em: 19/08/2016.

KHAN, A.; AHIRWAR, K. Mobile cloud computing as a future of mobile multimedia database. **Database**, v. 2, n. 1, p. 219–221, 2011.

KREITZ, G. Aspects of secure and efficient streaming and collaboration. KTH Royal Institute of Technology, 2011.

KREITZ, G.; NIEMELÄ, F. Spotify–large scale, low latency, p2p music-on-demand streaming. In: **Peer-to-Peer Computing**. [S.l.: s.n.], 2010. p. 1–10.

LEDERER, S.; MÜLLER, C.; TIMMERER, C. Dynamic adaptive streaming over http dataset. In: ACM. **Proceedings of the 3rd Multimedia Systems Conference**. [S.l.], 2012. p. 89–94.

LI, P. Research on key technologies of cloud computing and its application in telecom industry. 2015.

LUTZ, M.; ASCHER, D. **Aprendendo Python, 2**. [S.l.]: Bookman, 2007.

MAXCDN. **What is Keep-Alive?** 2016. Disponível em: <https://www.maxcdn.com/one/visual-glossary/keep-alive/>. Acesso em: 06/10/2016.

MICROSOFT. **Things that you may want to know about TCP Keepalives**. 2016. Disponível em: <https://blogs.technet.microsoft.com/nettracer/2010/06/03/things-that-you-may-want-to-know-about-tcp-keepalives/>. Acesso em: 03/10/2016.

MOLINA, B. et al. On content delivery network protocols and applications. Citeseer, 2004.

NGINX. **Why Netflix Chose NGINX as the Heart of Its CDN**. 2015. Disponível em: <https://www.nginx.com/blog/why-netflix-chose-nginx-as-the-heart-of-its-cdn/>. Acesso em: 19/08/2016.

ORACLE. **The Java EE 6 Tutorial**. 2013. Disponível em: <http://docs.oracle.com/javaee/6/tutorial/doc/bnapk.html>. Acesso em: 18/11/2016.

PYTHON SOFTWARE FOUNDATION. **Python Implementations**. 2016. Disponível em: <https://wiki.python.org/moin/PythonImplementations>. Acesso em: 27/07/2016.

REDMONK. **The RedMonk Programming Language Rankings: June 2016**. 2016. Disponível em: <http://redmonk.com/sogrady/category/programming-languages/>. Acesso em: 04/07/2016.

REENSKAUG, T.; COPLIEN, J. O. The dci architecture: A new vision of object-oriented programming. **An article starting a new blog:(14pp) http://www. artima. com/articles/dci_vision. html**, 2009.

RIEHLE, D. **Framework design**. Tese (Doutorado) — Diss. Technische Wissenschaften ETH Zürich, Nr. 13509, 2000, 2000.

RONACHER, A. **Flask Documentation**. 2016. Disponível em: <http://flask.pocoo.org/>. Acesso em: 28/09/2016.

STACK OVERFLOW. 2016.

SUMMERS, J. et al. Methodologies for generating HTTP streaming video workloads to evaluate web server performance. **Proceedings of the 5th Annual International Systems and Storage Conference on - SYSTOR '12**, p. 1–12, 2012. Disponível em: <http://dl.acm.org/citation.cfm?doid=2367589.2367602>.

TAVIS, M.; FITZSIMONS, P. **Web application hosting in the AWS cloud**. 2012.

TIMMERER, C.; GRIWODZ, C. Dynamic adaptive streaming over HTTP. **Proceedings of the 20th ACM international conference on Multimedia - MM '12**, p. 1533, 2012. Disponível em: <http://dl.acm.org/citation.cfm?doid=2393347.2396553>.

TIOBE. **TIOBE Index for July 2016**. 2016. Disponível em: <http://www.tiobe.com/tiobe_index>. Acesso em: 04/07/2016.

WEBDAV. **WebDAV**. 2016. Disponível em: <http://www.webdav.org/>. Acesso em: 03/10/2016.

YANGGRATOKE, R. et al. Predicting response times for the spotify backend. In: INTERNATIONAL FEDERATION FOR INFORMATION PROCESSING. **Proceedings of the 8th International Conference on Network and Service Management**. [S.l.], 2012. p. 117–125.

YAU, S.; AN, H. Software engineering meets services and cloud computing. **Computer**, v. 44, n. 10, p. 47–53, 2011. ISSN 00189162.

ZHANG, L.-J. Eic editorial: Introduction to the body of knowledge areas of services computing. **IEEE Transactions on services computing**, v. 1, n. 2, p. 62–74, 2008.

ZOWGHI, D.; COULIN, C. Requirements elicitation: A survey of techniques, approaches, and tools. In: **Engineering and managing software requirements**. [S.l.]: Springer, 2005. p. 19–46.

Apêndices

APÊNDICE A – CÓDIGO DA IMPLEMENTAÇÃO JAVA

```
##############################
Autor: Anderson Mendes
Arquivo: MusicaBean.java
##############################

import java.util.ArrayList;
import java.util.List;

import javax.annotation.PostConstruct;
import javax.faces.bean.ManagedBean;
import javax.faces.context.FacesContext;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpSession;

import br.com.youmusiclib.dao.MusicaDao;
import br.com.youmusiclib.dao.factory.MusicaDaoFactory;
import br.com.youmusiclib.modelo.Musica;
import br.com.youmusiclib.modelo.Usuario;

@ManagedBean
public class MusicaBean {

   private List<Musica> listaMusica = new ArrayList<Musica>();
   private String caminho;
   private String midiaServer = "http://localhost/";

   public String getCaminho() {
         return midiaServer+caminho;
   }
   public void setCaminho(String caminho) {
      this.caminho = midiaServer+caminho;
   }

   public MusicaBean(){
   }
```

```
   public List<Musica> getListaMusica() {
      return listaMusica  ;
   }
   public void setListaMusica(List<Musica> listaMusica) {
      this.listaMusica = listaMusica;
   }
   @PostConstruct
   public void criarListaDeMusica(){

      HttpServletRequest request = (HttpServletRequest)
FacesContext.getCurrentInstance().getExternalContext().getRequest();
      HttpSession session = request.getSession();
      Usuario usuario = (Usuario)session.getAttribute("usuario");

      MusicaDao daoMusica = MusicaDaoFactory.createMusicaDao();

      listaMusica = daoMusica.listarTudo(usuario);
   }
   public void executarMusica(Musica musica){
      this.caminho = musica.getCaminho();
   }
}
```

```
================================================================================
```

```
##############################
Autor: Anderson Mendes
Arquivo: LoginBean.java
##############################

import java.io.File;
import java.io.IOException;

import javax.annotation.PostConstruct;
import javax.faces.application.FacesMessage;
import javax.faces.bean.ManagedBean;
import javax.faces.bean.SessionScoped;
import javax.faces.context.FacesContext;
import javax.servlet.http.HttpSession;
```

```
import br.com.youmusiclib.dao.UsuarioDao;
import br.com.youmusiclib.dao.factory.UsuarioDaoFactory;
import br.com.youmusiclib.modelo.Usuario;

@ManagedBean
@SessionScoped
public class LoginBean {

private Usuario usuario;
private String login;
private String senha;

public LoginBean(){

}

public String getLogin() {
return login;
}

public void setLogin(String login) {
this.login = login;
}

public String getSenha() {
return senha;
}

public void setSenha(String senha) {
this.senha = senha;
}

@PostConstruct
public void init(){
usuario = new Usuario();
}

public Usuario getUsuario() {
```

```
return usuario;
}

public void setUsuario(Usuario usuario) {
this.usuario = usuario;
}

public void submit() throws IOException{
FacesContext context = FacesContext.getCurrentInstance();

UsuarioDao dao = UsuarioDaoFactory.createUsuarioDao();

usuario = dao.buscarPorEmail(login);

try{
if          (usuario.getEmail().equals(login)&&
            usuario.getSenha().equals(senha)) {
context.addMessage(null,new FacesMessage("Login efetuado
                com sucesso"));
try {

HttpSession session =
                    (HttpSession)context.getExternalContext().
                    getSession(false);
session.setAttribute("usuario", usuario);
context.getExternalContext().redirect("index.jsf");
} catch (IOException e) {
e.printStackTrace();
}
}
else{
context.addMessage(null,new FacesMessage("Login falhou"));
}

}catch(NullPointerException e){
context.addMessage(null,new FacesMessage("Login falhou"));
}
}
```

```
}

================================================================================

##############################
Autor: Anderson Mendes
Arquivo: Musica.java
##############################

package br.com.youmusiclib.modelo;

public class Musica {

   private String nome;
   private long tamanho;
   private int duracao;
   private String caminho;

   public String getCaminho() {
      return caminho;
   }
   public void setCaminho(String caminho) {
      this.caminho = caminho;
   }
   public String getNome() {
      return nome;
   }
   public void setNome(String nome) {
      this.nome = nome;
   }
   public long getTamanho() {
      return tamanho;
   }
   public void setTamanho(long tamanho) {
      this.tamanho = tamanho;
   }
   public int getDuracao() {
      return duracao;
   }
```

```
   public void setDuracao(int duracao) {
      this.duracao = duracao;
   }
}

================================================================================

################################
Autor: Anderson Mendes
Arquivo: Usuario.java
################################

public class Usuario{

   private String nome;
   private String email;
   private String senha;

   public Usuario(){
   }
   public String getSenha() {
      return senha;
   }
   public void setSenha(String senha) {
     this.senha = senha;
   }
   public String getNome() {
      return nome;
   }
   public void setNome(String nome) {
      this.nome = nome;
   }
   public String getEmail() {
      return email;
   }
   public void setEmail(String email) {
      this.email = email;
   }
```

```
}

================================================================================

##############################
Autor: Anderson Mendes
Arquivo: index.xhtml
##############################

<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:p="http://primefaces.org/ui">

    <ui:composition template="/template.xhtml">

        <ui:define name="conteudo">

         <h:form id="form">
            <h:outputText dir="ltr" value="Bem-vindo #
            {loginBean.usuario.nome}"
            style="font-size:20px"/>
            <br/>
            <br/>
            <p:dataTable var="musica" value="#{musicaBean.listaMusica}"
            style="width:600px;margin-left:12%">
                <p:column headerText="Musica">
                   <h:outputText value="#{musica.nome}"/>
                </p:column>
                <p:column headerText="Tamanho">
                    <h:outputText value="#{musica.tamanho} KB"/>
                </p:column>

                <p:column headerText="Player" style="width:30px">
```

```
                    <p:commandButton  icon="ui-icon-play" actionListener="
                    #{musicaBean.executarMusica(musica)}" ajax="false"/>

                </p:column>

              </p:dataTable>
            <br/>

             <audio controls="controls" style="width:400px;margin-
             left:170px"
             autoplay="autoplay">
                   <source src="#{musicaBean.caminho}"  type="audio/mp3" />
            </audio>
            </h:form>
          </ui:define>
    </ui:composition>
</html>
```

APÊNDICE B – CÓDIGO DA IMPLEMENTAÇÃO PYTHON

```
##############################
Autor: Anderson Mendes
Arquivo: app.py
##############################

from flask import Flask, render_template, request, redirect, url_for
from dao.UsuarioDAO import UsuarioDAO
from modelo.Usuario import Usuario
from dao.MusicaDAO import MusicaDAO
from modelo.Musica import Musica

app = Flask(__name__)


@app.route("/index")
def index():
    return render_template("index.html", teste ="gay", title = "sou foda")

@app.route("/cadastrar")
def cadastrar():
    return render_template("cadastrar.html")

@app.route('/main')
def main():
    try:
        return render_template("main.html")
    except Exception, e:
        return str(e)


@app.route("/connect")
def test_connect():
    try:
        usuario = Usuario()
```

```
        dao = UsuarioDAO()
        usuario = dao.buscar_por_email("anderson.mends@gmail.com")
        return render_template("index.html")

    except Exception, e:
        return str(e)

@app.route("/", methods=["GET", "POST"])
def login():

    if request.method== "GET":
        return render_template("login.html")

    if request.method=="POST":

        # return request.form["cadastrar"]
        # return request.form["email"]

        try:
            if request.form["cadastrar"] is not None:
                return redirect(url_for('cadastrar'))
            else:
                return redirect(url_for("index"))

        except Exception, e:

            dao = UsuarioDAO()
            usuario = dao.buscar_por_email(request.form["email"])

            if usuario.email == request.form["email"] and usuario.senha ==
            request.form["senha"]:
                musicas = []
                dao = MusicaDAO()
                musicas = dao.listar_musicas(usuario)
                return render_template("index.html", musicas=musicas)
                # return redirect(url_for("index", musicas = "teste"))
            else:
                return "Usuario nao cadastrado"
```

```
if __name__ == "__main__":
    app.run()

================================================================================

################################
Autor: Anderson Mendes
Arquivo: Musica.py
################################

class Musica():

    def __init__(self):
        self.nome = None
        self.tamanho = None
        self.duracao = None
        self.caminho = None

================================================================================

################################
Autor: Anderson Mendes
Arquivo: Usuario.py
################################

class Usuario():

    def __init__(self):
        self.nome = None
        self.email = None
        self.senha = None

================================================================================

################################
Autor: Anderson Mendes
Arquivo: index.html
################################
```

```
<!DOCTYPE html>
<html lang="en" xmlns="http://www.w3.org/1999/html">
<head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-
    scale=1">

    <title>Bootstrap 3, from LayoutIt!</title>

    <meta name="description" content="Source code generated using
    layoutit.com">
    <meta name="author" content="LayoutIt!">

    <link href="static/css/bootstrap.min.css" rel="stylesheet">
    <link href="/static/css/style.css" rel="stylesheet">
    <script src="/static/js/jquery.min.js"></script>


</head>
<body>

<div class="container-fluid" style="min-width:1100px;max-
width:1100px;min-height:600px;margin-left:100px">
    <div style="background-
    image:url('static/topo.jpg');width:1100px;height:97px">

    </div>
    <br/>
    <br/>
    <br/>

    <div class="row">
        <div class="col-md-2 panel panel-default">
            <ul class="nav nav-stacked nav-tabs">
                <h3 class="panel panel-default">
                    Menu
                </h3>
                <li>
```

```
                <a href="#" class="glyphicon glyphicon-home">
                Home</a>
            </li>
            <li>
                <a href="#" class="glyphicon glyphicon-wrench">
                Profile</a>
            </li>
            <li>
                <a href="#" class="glyphicon glyphicon-folder-open">
                Music
                Lib</a>
            </li>
            <li>
                <a href="#" class="glyphicon glyphicon-upload">
                Upload</a>
            </li>
            <li>
                <a href="#" class="glyphicon glyphicon-remove">
                Sair</a>
            </li>
        </ul>
    </div>
    <div class="col-md-10 panel panel-default">

        <h3 class="panel panel-default">
            Biblioteca
        </h3>
        <table class="table">
            <thead>
            <tr>
                <th>
                    Música
                </th>
                <th>
                    Tamanho
                </th>
                <th>
                    Play
                </th>
```

```
    </tr>
    </thead>
    <tbody>
    {% for musica in musicas %}
        <tr>
            <td>
                {{ musica.nome }}
            </td>
            <td>
                {{ musica.tamanho }}
            </td>
            <td>
                <button type="button" class="btn btn-info
                btn-sm" value="{{
                "http://localhost/"+musica.caminho }}">
               <span class="glyphicon glyphicon-play"></span>
               Play
                </button>
            </td>
        </tr>
    {% endfor %}
    </tbody>
</table>

<br/>

<audio controls="controls" style="width:400px;margin-
left:170px" id="audio" autoplay="autoplay" loop="loop">
    <source src="" type="audio/mp3"/>
</audio>

<script>
    $('button').click(function (event) {

        var eventCategory = $(this).attr("value");
        $("#audio").attr("src", eventCategory);
        var t = $("#audio").attr("src");
        $("p").text(t);
```

```
                });

            </script>


        </div>
    </div>
</div>

<body/>
```

APÊNDICE C – PLANO DE TESTE JMETER

```
<?xml version="1.0" encoding="UTF-8"?>
<jmeterTestPlan version="1.2" properties="2.9" jmeter="3.0 r1743807">
  <hashTree>
    <TestPlan guiclass="TestPlanGui" testclass="TestPlan" testname="Test Plan - Dow
      <stringProp name="TestPlan.comments">Author: www.havecomputerwillcode.com - U
      <boolProp name="TestPlan.functional_mode">false</boolProp>
      <boolProp name="TestPlan.serialize_threadgroups">false</boolProp>
      <elementProp name="TestPlan.user_defined_variables" elementType="Arguments" g
        <collectionProp name="Arguments.arguments"/>
      </elementProp>
      <stringProp name="TestPlan.user_define_classpath"></stringProp>
    </TestPlan>
    <hashTree>
      <kg.apc.jmeter.threads.SteppingThreadGroup guiclass="kg.apc.jmeter.threads.St
        <stringProp name="ThreadGroup.on_sample_error">continue</stringProp>
        <stringProp name="ThreadGroup.num_threads">1000</stringProp>
        <stringProp name="Threads initial delay">0</stringProp>
        <stringProp name="Start users count">100</stringProp>
        <stringProp name="Start users count burst">100</stringProp>
        <stringProp name="Start users period">10</stringProp>
        <stringProp name="Stop users count">100</stringProp>
        <stringProp name="Stop users period">10</stringProp>
        <stringProp name="flighttime">920</stringProp>
        <stringProp name="rampUp">10</stringProp>
        <elementProp name="ThreadGroup.main_controller" elementType="LoopController
          <boolProp name="LoopController.continue_forever">false</boolProp>
          <intProp name="LoopController.loops">-1</intProp>
        </elementProp>
      </kg.apc.jmeter.threads.SteppingThreadGroup>
      <hashTree>
        <HTTPSamplerProxy guiclass="HttpTestSampleGui" testclass="HTTPSamplerProxy"
          <elementProp name="HTTPsampler.Arguments" elementType="Arguments" guiclas
            <collectionProp name="Arguments.arguments"/>
          </elementProp>
```

```
      <stringProp name="HTTPSampler.domain">192.168.1.2</stringProp>
      <stringProp name="HTTPSampler.port">80</stringProp>
      <stringProp name="HTTPSampler.connect_timeout"></stringProp>
      <stringProp name="HTTPSampler.response_timeout"></stringProp>
      <stringProp name="HTTPSampler.protocol"></stringProp>
      <stringProp name="HTTPSampler.contentEncoding"></stringProp>
      <stringProp name="HTTPSampler.path">/index</stringProp>
      <stringProp name="HTTPSampler.method">GET</stringProp>
      <boolProp name="HTTPSampler.follow_redirects">true</boolProp>
      <boolProp name="HTTPSampler.auto_redirects">false</boolProp>
      <boolProp name="HTTPSampler.use_keepalive">true</boolProp>
      <boolProp name="HTTPSampler.DO_MULTIPART_POST">false</boolProp>
      <stringProp name="HTTPSampler.implementation">Java</stringProp>
      <boolProp name="HTTPSampler.monitor">false</boolProp>
      <stringProp name="HTTPSampler.embedded_url_re"></stringProp>
    </HTTPSamplerProxy>
    <hashTree/>
    <RunTime guiclass="RunTimeGui" testclass="RunTime" testname="Controlador de
      <stringProp name="RunTime.seconds">120</stringProp>
    </RunTime>
    <hashTree>
      <LoopController guiclass="LoopControlPanel" testclass="LoopController" te
        <boolProp name="LoopController.continue_forever">true</boolProp>
        <stringProp name="LoopController.loops">2000</stringProp>
      </LoopController>
      <hashTree>
        <HTTPSamplerProxy guiclass="HttpTestSampleGui" testclass="HTTPSamplerPr
          <elementProp name="HTTPsampler.Arguments" elementType="Arguments" gui
            <collectionProp name="Arguments.arguments"/>
          </elementProp>
          <stringProp name="HTTPSampler.domain">192.168.1.3</stringProp>
          <stringProp name="HTTPSampler.port">80</stringProp>
          <stringProp name="HTTPSampler.connect_timeout"></stringProp>
          <stringProp name="HTTPSampler.response_timeout"></stringProp>
          <stringProp name="HTTPSampler.protocol"></stringProp>
          <stringProp name="HTTPSampler.contentEncoding"></stringProp>
          <stringProp name="HTTPSampler.path">\youmusiclib\anderson.mends@gmail
          <stringProp name="HTTPSampler.method">GET</stringProp>
          <boolProp name="HTTPSampler.follow_redirects">true</boolProp>
```

```
        <boolProp name="HTTPSampler.auto_redirects">false</boolProp>
        <boolProp name="HTTPSampler.use_keepalive">true</boolProp>
        <boolProp name="HTTPSampler.DO_MULTIPART_POST">false</boolProp>
        <stringProp name="HTTPSampler.implementation">Java</stringProp>
        <boolProp name="HTTPSampler.monitor">false</boolProp>
        <stringProp name="HTTPSampler.embedded_url_re"></stringProp>
      </HTTPSamplerProxy>
      <hashTree/>
    </hashTree>
  </hashTree>
  <HTTPSamplerProxy guiclass="HttpTestSampleGui" testclass="HTTPSamplerProxy"
    <elementProp name="HTTPsampler.Arguments" elementType="Arguments" guiclas
      <collectionProp name="Arguments.arguments"/>
    </elementProp>
    <stringProp name="HTTPSampler.domain">192.168.1.2</stringProp>
    <stringProp name="HTTPSampler.port">80</stringProp>
    <stringProp name="HTTPSampler.connect_timeout"></stringProp>
    <stringProp name="HTTPSampler.response_timeout"></stringProp>
    <stringProp name="HTTPSampler.protocol"></stringProp>
    <stringProp name="HTTPSampler.contentEncoding"></stringProp>
    <stringProp name="HTTPSampler.path">/index</stringProp>
    <stringProp name="HTTPSampler.method">GET</stringProp>
    <boolProp name="HTTPSampler.follow_redirects">true</boolProp>
    <boolProp name="HTTPSampler.auto_redirects">false</boolProp>
    <boolProp name="HTTPSampler.use_keepalive">true</boolProp>
    <boolProp name="HTTPSampler.DO_MULTIPART_POST">false</boolProp>
    <stringProp name="HTTPSampler.implementation">Java</stringProp>
    <boolProp name="HTTPSampler.monitor">false</boolProp>
    <stringProp name="HTTPSampler.embedded_url_re"></stringProp>
  </HTTPSamplerProxy>
  <hashTree/>
  <RunTime guiclass="RunTimeGui" testclass="RunTime" testname="Controlador de
    <stringProp name="RunTime.seconds">115</stringProp>
  </RunTime>
  <hashTree>
    <LoopController guiclass="LoopControlPanel" testclass="LoopController" te
      <boolProp name="LoopController.continue_forever">true</boolProp>
      <stringProp name="LoopController.loops">2000</stringProp>
    </LoopController>
```

```
      <hashTree>
        <HTTPSamplerProxy guiclass="HttpTestSampleGui" testclass="HTTPSamplerPr
          <elementProp name="HTTPsampler.Arguments" elementType="Arguments" gui
            <collectionProp name="Arguments.arguments"/>
          </elementProp>
          <stringProp name="HTTPSampler.domain">192.168.1.3</stringProp>
          <stringProp name="HTTPSampler.port">80</stringProp>
          <stringProp name="HTTPSampler.connect_timeout"></stringProp>
          <stringProp name="HTTPSampler.response_timeout"></stringProp>
          <stringProp name="HTTPSampler.protocol"></stringProp>
          <stringProp name="HTTPSampler.contentEncoding"></stringProp>
          <stringProp name="HTTPSampler.path">\youmusiclib\anderson.mends@gmail
          <stringProp name="HTTPSampler.method">GET</stringProp>
          <boolProp name="HTTPSampler.follow_redirects">true</boolProp>
          <boolProp name="HTTPSampler.auto_redirects">false</boolProp>
          <boolProp name="HTTPSampler.use_keepalive">true</boolProp>
          <boolProp name="HTTPSampler.DO_MULTIPART_POST">false</boolProp>
          <stringProp name="HTTPSampler.implementation">Java</stringProp>
          <boolProp name="HTTPSampler.monitor">false</boolProp>
          <stringProp name="HTTPSampler.embedded_url_re"></stringProp>
        </HTTPSamplerProxy>
        <hashTree/>
      </hashTree>
    </hashTree>
    <HTTPSamplerProxy guiclass="HttpTestSampleGui" testclass="HTTPSamplerProxy"
      <elementProp name="HTTPsampler.Arguments" elementType="Arguments" guiclas
        <collectionProp name="Arguments.arguments"/>
      </elementProp>
      <stringProp name="HTTPSampler.domain">192.168.1.2</stringProp>
      <stringProp name="HTTPSampler.port">80</stringProp>
      <stringProp name="HTTPSampler.connect_timeout"></stringProp>
      <stringProp name="HTTPSampler.response_timeout"></stringProp>
      <stringProp name="HTTPSampler.protocol"></stringProp>
      <stringProp name="HTTPSampler.contentEncoding"></stringProp>
      <stringProp name="HTTPSampler.path">/index</stringProp>
      <stringProp name="HTTPSampler.method">GET</stringProp>
      <boolProp name="HTTPSampler.follow_redirects">true</boolProp>
      <boolProp name="HTTPSampler.auto_redirects">false</boolProp>
      <boolProp name="HTTPSampler.use_keepalive">true</boolProp>
```

```
      <boolProp name="HTTPSampler.DO_MULTIPART_POST">false</boolProp>
      <stringProp name="HTTPSampler.implementation">Java</stringProp>
      <boolProp name="HTTPSampler.monitor">false</boolProp>
      <stringProp name="HTTPSampler.embedded_url_re"></stringProp>
    </HTTPSamplerProxy>
    <hashTree/>
    <RunTime guiclass="RunTimeGui" testclass="RunTime" testname="Controlador de
      <stringProp name="RunTime.seconds">296</stringProp>
    </RunTime>
    <hashTree>
      <LoopController guiclass="LoopControlPanel" testclass="LoopController" te
        <boolProp name="LoopController.continue_forever">true</boolProp>
        <stringProp name="LoopController.loops">2000</stringProp>
      </LoopController>
      <hashTree>
        <HTTPSamplerProxy guiclass="HttpTestSampleGui" testclass="HTTPSamplerPr
          <elementProp name="HTTPsampler.Arguments" elementType="Arguments" gui
            <collectionProp name="Arguments.arguments"/>
          </elementProp>
          <stringProp name="HTTPSampler.domain">192.168.1.3</stringProp>
          <stringProp name="HTTPSampler.port">80</stringProp>
          <stringProp name="HTTPSampler.connect_timeout"></stringProp>
          <stringProp name="HTTPSampler.response_timeout"></stringProp>
          <stringProp name="HTTPSampler.protocol"></stringProp>
          <stringProp name="HTTPSampler.contentEncoding"></stringProp>
          <stringProp name="HTTPSampler.path">\youmusiclib\anderson.mends@gmail
          <stringProp name="HTTPSampler.method">GET</stringProp>
          <boolProp name="HTTPSampler.follow_redirects">true</boolProp>
          <boolProp name="HTTPSampler.auto_redirects">false</boolProp>
          <boolProp name="HTTPSampler.use_keepalive">true</boolProp>
          <boolProp name="HTTPSampler.DO_MULTIPART_POST">false</boolProp>
          <stringProp name="HTTPSampler.implementation">Java</stringProp>
          <boolProp name="HTTPSampler.monitor">false</boolProp>
          <stringProp name="HTTPSampler.embedded_url_re"></stringProp>
        </HTTPSamplerProxy>
        <hashTree/>
      </hashTree>
    </hashTree>
    <HTTPSamplerProxy guiclass="HttpTestSampleGui" testclass="HTTPSamplerProxy"
```

```
      <elementProp name="HTTPsampler.Arguments" elementType="Arguments" guiclas
        <collectionProp name="Arguments.arguments"/>
      </elementProp>
      <stringProp name="HTTPSampler.domain">192.168.1.2</stringProp>
      <stringProp name="HTTPSampler.port">80</stringProp>
      <stringProp name="HTTPSampler.connect_timeout"></stringProp>
      <stringProp name="HTTPSampler.response_timeout"></stringProp>
      <stringProp name="HTTPSampler.protocol"></stringProp>
      <stringProp name="HTTPSampler.contentEncoding"></stringProp>
      <stringProp name="HTTPSampler.path">/index</stringProp>
      <stringProp name="HTTPSampler.method">GET</stringProp>
      <boolProp name="HTTPSampler.follow_redirects">true</boolProp>
      <boolProp name="HTTPSampler.auto_redirects">false</boolProp>
      <boolProp name="HTTPSampler.use_keepalive">true</boolProp>
      <boolProp name="HTTPSampler.DO_MULTIPART_POST">false</boolProp>
      <stringProp name="HTTPSampler.implementation">Java</stringProp>
      <boolProp name="HTTPSampler.monitor">false</boolProp>
      <stringProp name="HTTPSampler.embedded_url_re"></stringProp>
    </HTTPSamplerProxy>
    <hashTree/>
    <RunTime guiclass="RunTimeGui" testclass="RunTime" testname="Controlador de
      <stringProp name="RunTime.seconds">296</stringProp>
    </RunTime>
    <hashTree>
      <LoopController guiclass="LoopControlPanel" testclass="LoopController" te
        <boolProp name="LoopController.continue_forever">true</boolProp>
        <stringProp name="LoopController.loops">2000</stringProp>
      </LoopController>
      <hashTree>
        <HTTPSamplerProxy guiclass="HttpTestSampleGui" testclass="HTTPSamplerPr
          <elementProp name="HTTPsampler.Arguments" elementType="Arguments" gui
            <collectionProp name="Arguments.arguments"/>
          </elementProp>
          <stringProp name="HTTPSampler.domain">192.168.1.3</stringProp>
          <stringProp name="HTTPSampler.port">80</stringProp>
          <stringProp name="HTTPSampler.connect_timeout"></stringProp>
          <stringProp name="HTTPSampler.response_timeout"></stringProp>
          <stringProp name="HTTPSampler.protocol"></stringProp>
          <stringProp name="HTTPSampler.contentEncoding"></stringProp>
```

```
        <stringProp name="HTTPSampler.path">\youmusiclib\anderson.mends@gmail
        <stringProp name="HTTPSampler.method">GET</stringProp>
        <boolProp name="HTTPSampler.follow_redirects">true</boolProp>
        <boolProp name="HTTPSampler.auto_redirects">false</boolProp>
        <boolProp name="HTTPSampler.use_keepalive">true</boolProp>
        <boolProp name="HTTPSampler.DO_MULTIPART_POST">false</boolProp>
        <stringProp name="HTTPSampler.implementation">Java</stringProp>
        <boolProp name="HTTPSampler.monitor">false</boolProp>
        <stringProp name="HTTPSampler.embedded_url_re"></stringProp>
      </HTTPSamplerProxy>
      <hashTree/>
    </hashTree>
  </hashTree>
  <HTTPSamplerProxy guiclass="HttpTestSampleGui" testclass="HTTPSamplerProxy"
    <elementProp name="HTTPsampler.Arguments" elementType="Arguments" guiclas
      <collectionProp name="Arguments.arguments"/>
    </elementProp>
    <stringProp name="HTTPSampler.domain">192.168.1.2</stringProp>
    <stringProp name="HTTPSampler.port">80</stringProp>
    <stringProp name="HTTPSampler.connect_timeout"></stringProp>
    <stringProp name="HTTPSampler.response_timeout"></stringProp>
    <stringProp name="HTTPSampler.protocol"></stringProp>
    <stringProp name="HTTPSampler.contentEncoding"></stringProp>
    <stringProp name="HTTPSampler.path">/index</stringProp>
    <stringProp name="HTTPSampler.method">GET</stringProp>
    <boolProp name="HTTPSampler.follow_redirects">true</boolProp>
    <boolProp name="HTTPSampler.auto_redirects">false</boolProp>
    <boolProp name="HTTPSampler.use_keepalive">true</boolProp>
    <boolProp name="HTTPSampler.DO_MULTIPART_POST">false</boolProp>
    <stringProp name="HTTPSampler.implementation">Java</stringProp>
    <boolProp name="HTTPSampler.monitor">false</boolProp>
    <stringProp name="HTTPSampler.embedded_url_re"></stringProp>
  </HTTPSamplerProxy>
  <hashTree/>
  <RunTime guiclass="RunTimeGui" testclass="RunTime" testname="Controlador de
    <stringProp name="RunTime.seconds">3</stringProp>
  </RunTime>
  <hashTree>
    <LoopController guiclass="LoopControlPanel" testclass="LoopController" te
```

```
        <boolProp name="LoopController.continue_forever">true</boolProp>
        <stringProp name="LoopController.loops">2000</stringProp>
      </LoopController>
      <hashTree>
        <HTTPSamplerProxy guiclass="HttpTestSampleGui" testclass="HTTPSamplerPr
          <elementProp name="HTTPsampler.Arguments" elementType="Arguments" gui
            <collectionProp name="Arguments.arguments"/>
          </elementProp>
          <stringProp name="HTTPSampler.domain">192.168.1.3</stringProp>
          <stringProp name="HTTPSampler.port">80</stringProp>
          <stringProp name="HTTPSampler.connect_timeout"></stringProp>
          <stringProp name="HTTPSampler.response_timeout"></stringProp>
          <stringProp name="HTTPSampler.protocol"></stringProp>
          <stringProp name="HTTPSampler.contentEncoding"></stringProp>
          <stringProp name="HTTPSampler.path">\youmusiclib\anderson.mends@gmail
          <stringProp name="HTTPSampler.method">GET</stringProp>
          <boolProp name="HTTPSampler.follow_redirects">true</boolProp>
          <boolProp name="HTTPSampler.auto_redirects">false</boolProp>
          <boolProp name="HTTPSampler.use_keepalive">true</boolProp>
          <boolProp name="HTTPSampler.DO_MULTIPART_POST">false</boolProp>
          <stringProp name="HTTPSampler.implementation">Java</stringProp>
          <boolProp name="HTTPSampler.monitor">false</boolProp>
          <stringProp name="HTTPSampler.embedded_url_re"></stringProp>
        </HTTPSamplerProxy>
        <hashTree/>
      </hashTree>
    </hashTree>
    <HTTPSamplerProxy guiclass="HttpTestSampleGui" testclass="HTTPSamplerProxy"
      <elementProp name="HTTPsampler.Arguments" elementType="Arguments" guiclas
        <collectionProp name="Arguments.arguments"/>
      </elementProp>
      <stringProp name="HTTPSampler.domain">192.168.1.2</stringProp>
      <stringProp name="HTTPSampler.port">80</stringProp>
      <stringProp name="HTTPSampler.connect_timeout"></stringProp>
      <stringProp name="HTTPSampler.response_timeout"></stringProp>
      <stringProp name="HTTPSampler.protocol"></stringProp>
      <stringProp name="HTTPSampler.contentEncoding"></stringProp>
      <stringProp name="HTTPSampler.path">/index</stringProp>
      <stringProp name="HTTPSampler.method">GET</stringProp>
```

```
      <boolProp name="HTTPSampler.follow_redirects">true</boolProp>
      <boolProp name="HTTPSampler.auto_redirects">false</boolProp>
      <boolProp name="HTTPSampler.use_keepalive">true</boolProp>
      <boolProp name="HTTPSampler.DO_MULTIPART_POST">false</boolProp>
      <stringProp name="HTTPSampler.implementation">Java</stringProp>
      <boolProp name="HTTPSampler.monitor">false</boolProp>
      <stringProp name="HTTPSampler.embedded_url_re"></stringProp>
    </HTTPSamplerProxy>
    <hashTree/>
    <RunTime guiclass="RunTimeGui" testclass="RunTime" testname="Controlador de
      <stringProp name="RunTime.seconds">10</stringProp>
    </RunTime>
    <hashTree>
      <LoopController guiclass="LoopControlPanel" testclass="LoopController" te
        <boolProp name="LoopController.continue_forever">true</boolProp>
        <stringProp name="LoopController.loops">2000</stringProp>
      </LoopController>
      <hashTree>
        <HTTPSamplerProxy guiclass="HttpTestSampleGui" testclass="HTTPSamplerPr
          <elementProp name="HTTPsampler.Arguments" elementType="Arguments" gui
            <collectionProp name="Arguments.arguments"/>
          </elementProp>
          <stringProp name="HTTPSampler.domain">192.168.1.3</stringProp>
          <stringProp name="HTTPSampler.port">80</stringProp>
          <stringProp name="HTTPSampler.connect_timeout"></stringProp>
          <stringProp name="HTTPSampler.response_timeout"></stringProp>
          <stringProp name="HTTPSampler.protocol"></stringProp>
          <stringProp name="HTTPSampler.contentEncoding"></stringProp>
          <stringProp name="HTTPSampler.path">\youmusiclib\anderson.mends@gmail
          <stringProp name="HTTPSampler.method">GET</stringProp>
          <boolProp name="HTTPSampler.follow_redirects">true</boolProp>
          <boolProp name="HTTPSampler.auto_redirects">false</boolProp>
          <boolProp name="HTTPSampler.use_keepalive">true</boolProp>
          <boolProp name="HTTPSampler.DO_MULTIPART_POST">false</boolProp>
          <stringProp name="HTTPSampler.implementation">Java</stringProp>
          <boolProp name="HTTPSampler.monitor">false</boolProp>
          <stringProp name="HTTPSampler.embedded_url_re"></stringProp>
        </HTTPSamplerProxy>
        <hashTree/>
```

```
      </hashTree>
    </hashTree>
    <HTTPSamplerProxy guiclass="HttpTestSampleGui" testclass="HTTPSamplerProxy"
      <elementProp name="HTTPsampler.Arguments" elementType="Arguments" guiclas
        <collectionProp name="Arguments.arguments"/>
      </elementProp>
      <stringProp name="HTTPSampler.domain">192.168.1.2</stringProp>
      <stringProp name="HTTPSampler.port">80</stringProp>
      <stringProp name="HTTPSampler.connect_timeout"></stringProp>
      <stringProp name="HTTPSampler.response_timeout"></stringProp>
      <stringProp name="HTTPSampler.protocol"></stringProp>
      <stringProp name="HTTPSampler.contentEncoding"></stringProp>
      <stringProp name="HTTPSampler.path">/index</stringProp>
      <stringProp name="HTTPSampler.method">GET</stringProp>
      <boolProp name="HTTPSampler.follow_redirects">true</boolProp>
      <boolProp name="HTTPSampler.auto_redirects">false</boolProp>
      <boolProp name="HTTPSampler.use_keepalive">true</boolProp>
      <boolProp name="HTTPSampler.DO_MULTIPART_POST">false</boolProp>
      <stringProp name="HTTPSampler.implementation">Java</stringProp>
      <boolProp name="HTTPSampler.monitor">false</boolProp>
      <stringProp name="HTTPSampler.embedded_url_re"></stringProp>
    </HTTPSamplerProxy>
    <hashTree/>
    <RunTime guiclass="RunTimeGui" testclass="RunTime" testname="Controlador de
      <stringProp name="RunTime.seconds">5</stringProp>
    </RunTime>
    <hashTree>
      <LoopController guiclass="LoopControlPanel" testclass="LoopController" te
        <boolProp name="LoopController.continue_forever">true</boolProp>
        <stringProp name="LoopController.loops">2000</stringProp>
      </LoopController>
      <hashTree>
        <HTTPSamplerProxy guiclass="HttpTestSampleGui" testclass="HTTPSamplerPr
          <elementProp name="HTTPsampler.Arguments" elementType="Arguments" gui
            <collectionProp name="Arguments.arguments"/>
          </elementProp>
          <stringProp name="HTTPSampler.domain">192.168.1.3</stringProp>
          <stringProp name="HTTPSampler.port">80</stringProp>
          <stringProp name="HTTPSampler.connect_timeout"></stringProp>
```

```
        <stringProp name="HTTPSampler.response_timeout"></stringProp>
        <stringProp name="HTTPSampler.protocol"></stringProp>
        <stringProp name="HTTPSampler.contentEncoding"></stringProp>
        <stringProp name="HTTPSampler.path">\youmusiclib\anderson.mends@gmail
        <stringProp name="HTTPSampler.method">GET</stringProp>
        <boolProp name="HTTPSampler.follow_redirects">true</boolProp>
        <boolProp name="HTTPSampler.auto_redirects">false</boolProp>
        <boolProp name="HTTPSampler.use_keepalive">true</boolProp>
        <boolProp name="HTTPSampler.DO_MULTIPART_POST">false</boolProp>
        <stringProp name="HTTPSampler.implementation">Java</stringProp>
        <boolProp name="HTTPSampler.monitor">false</boolProp>
        <stringProp name="HTTPSampler.embedded_url_re"></stringProp>
      </HTTPSamplerProxy>
      <hashTree/>
    </hashTree>
  </hashTree>
  <HTTPSamplerProxy guiclass="HttpTestSampleGui" testclass="HTTPSamplerProxy"
    <elementProp name="HTTPsampler.Arguments" elementType="Arguments" guiclas
      <collectionProp name="Arguments.arguments"/>
    </elementProp>
    <stringProp name="HTTPSampler.domain">192.168.1.2</stringProp>
    <stringProp name="HTTPSampler.port">80</stringProp>
    <stringProp name="HTTPSampler.connect_timeout"></stringProp>
    <stringProp name="HTTPSampler.response_timeout"></stringProp>
    <stringProp name="HTTPSampler.protocol"></stringProp>
    <stringProp name="HTTPSampler.contentEncoding"></stringProp>
    <stringProp name="HTTPSampler.path">/index</stringProp>
    <stringProp name="HTTPSampler.method">GET</stringProp>
    <boolProp name="HTTPSampler.follow_redirects">true</boolProp>
    <boolProp name="HTTPSampler.auto_redirects">false</boolProp>
    <boolProp name="HTTPSampler.use_keepalive">true</boolProp>
    <boolProp name="HTTPSampler.DO_MULTIPART_POST">false</boolProp>
    <stringProp name="HTTPSampler.implementation">Java</stringProp>
    <boolProp name="HTTPSampler.monitor">false</boolProp>
    <stringProp name="HTTPSampler.embedded_url_re"></stringProp>
  </HTTPSamplerProxy>
  <hashTree/>
  <RunTime guiclass="RunTimeGui" testclass="RunTime" testname="Controlador de
    <stringProp name="RunTime.seconds">30</stringProp>
```

```
</RunTime>
<hashTree>
  <LoopController guiclass="LoopControlPanel" testclass="LoopController" te
    <boolProp name="LoopController.continue_forever">true</boolProp>
    <stringProp name="LoopController.loops">2000</stringProp>
  </LoopController>
  <hashTree>
    <HTTPSamplerProxy guiclass="HttpTestSampleGui" testclass="HTTPSamplerPr
      <elementProp name="HTTPsampler.Arguments" elementType="Arguments" gui
        <collectionProp name="Arguments.arguments"/>
      </elementProp>
      <stringProp name="HTTPSampler.domain">192.168.1.3</stringProp>
      <stringProp name="HTTPSampler.port">80</stringProp>
      <stringProp name="HTTPSampler.connect_timeout"></stringProp>
      <stringProp name="HTTPSampler.response_timeout"></stringProp>
      <stringProp name="HTTPSampler.protocol"></stringProp>
      <stringProp name="HTTPSampler.contentEncoding"></stringProp>
      <stringProp name="HTTPSampler.path">\youmusiclib\anderson.mends@gmail
      <stringProp name="HTTPSampler.method">GET</stringProp>
      <boolProp name="HTTPSampler.follow_redirects">true</boolProp>
      <boolProp name="HTTPSampler.auto_redirects">false</boolProp>
      <boolProp name="HTTPSampler.use_keepalive">true</boolProp>
      <boolProp name="HTTPSampler.DO_MULTIPART_POST">false</boolProp>
      <stringProp name="HTTPSampler.implementation">Java</stringProp>
      <boolProp name="HTTPSampler.monitor">false</boolProp>
      <stringProp name="HTTPSampler.embedded_url_re"></stringProp>
    </HTTPSamplerProxy>
    <hashTree/>
  </hashTree>
</hashTree>
<HTTPSamplerProxy guiclass="HttpTestSampleGui" testclass="HTTPSamplerProxy"
  <elementProp name="HTTPsampler.Arguments" elementType="Arguments" guiclas
    <collectionProp name="Arguments.arguments"/>
  </elementProp>
  <stringProp name="HTTPSampler.domain">192.168.1.2</stringProp>
  <stringProp name="HTTPSampler.port">80</stringProp>
  <stringProp name="HTTPSampler.connect_timeout"></stringProp>
  <stringProp name="HTTPSampler.response_timeout"></stringProp>
  <stringProp name="HTTPSampler.protocol"></stringProp>
```

```
      <stringProp name="HTTPSampler.contentEncoding"></stringProp>
      <stringProp name="HTTPSampler.path">/index</stringProp>
      <stringProp name="HTTPSampler.method">GET</stringProp>
      <boolProp name="HTTPSampler.follow_redirects">true</boolProp>
      <boolProp name="HTTPSampler.auto_redirects">false</boolProp>
      <boolProp name="HTTPSampler.use_keepalive">true</boolProp>
      <boolProp name="HTTPSampler.DO_MULTIPART_POST">false</boolProp>
      <stringProp name="HTTPSampler.implementation">Java</stringProp>
      <boolProp name="HTTPSampler.monitor">false</boolProp>
      <stringProp name="HTTPSampler.embedded_url_re"></stringProp>
    </HTTPSamplerProxy>
    <hashTree/>
    <RunTime guiclass="RunTimeGui" testclass="RunTime" testname="Controlador de
      <stringProp name="RunTime.seconds">7</stringProp>
    </RunTime>
    <hashTree>
      <LoopController guiclass="LoopControlPanel" testclass="LoopController" te
        <boolProp name="LoopController.continue_forever">true</boolProp>
        <stringProp name="LoopController.loops">2000</stringProp>
      </LoopController>
      <hashTree>
        <HTTPSamplerProxy guiclass="HttpTestSampleGui" testclass="HTTPSamplerPr
          <elementProp name="HTTPsampler.Arguments" elementType="Arguments" gui
            <collectionProp name="Arguments.arguments"/>
          </elementProp>
          <stringProp name="HTTPSampler.domain">192.168.1.3</stringProp>
          <stringProp name="HTTPSampler.port">80</stringProp>
          <stringProp name="HTTPSampler.connect_timeout"></stringProp>
          <stringProp name="HTTPSampler.response_timeout"></stringProp>
          <stringProp name="HTTPSampler.protocol"></stringProp>
          <stringProp name="HTTPSampler.contentEncoding"></stringProp>
          <stringProp name="HTTPSampler.path">\youmusiclib\anderson.mends@gmail
          <stringProp name="HTTPSampler.method">GET</stringProp>
          <boolProp name="HTTPSampler.follow_redirects">true</boolProp>
          <boolProp name="HTTPSampler.auto_redirects">false</boolProp>
          <boolProp name="HTTPSampler.use_keepalive">true</boolProp>
          <boolProp name="HTTPSampler.DO_MULTIPART_POST">false</boolProp>
          <stringProp name="HTTPSampler.implementation">Java</stringProp>
          <boolProp name="HTTPSampler.monitor">false</boolProp>
```

```
          <stringProp name="HTTPSampler.embedded_url_re"></stringProp>
        </HTTPSamplerProxy>
        <hashTree/>
      </hashTree>
    </hashTree>
  </hashTree>
  <kg.apc.jmeter.vizualizers.CorrectedResultCollector guiclass="kg.apc.jmeter.v
    <boolProp name="ResultCollector.error_logging">false</boolProp>
    <objProp>
      <name>saveConfig</name>
      <value class="SampleSaveConfiguration">
        <time>true</time>
        <latency>true</latency>
        <timestamp>true</timestamp>
        <success>true</success>
        <label>true</label>
        <code>true</code>
        <message>true</message>
        <threadName>true</threadName>
        <dataType>true</dataType>
        <encoding>false</encoding>
        <assertions>true</assertions>
        <subresults>true</subresults>
        <responseData>false</responseData>
        <samplerData>false</samplerData>
        <xml>false</xml>
        <fieldNames>true</fieldNames>
        <responseHeaders>false</responseHeaders>
        <requestHeaders>false</requestHeaders>
        <responseDataOnError>false</responseDataOnError>
        <saveAssertionResultsFailureMessage>true</saveAssertionResultsFailureMe
        <assertionsResultsToSave>0</assertionsResultsToSave>
        <bytes>true</bytes>
        <threadCounts>true</threadCounts>
        <idleTime>true</idleTime>
      </value>
    </objProp>
    <stringProp name="filename"></stringProp>
    <longProp name="interval_grouping">1000</longProp>
```

```
      <boolProp name="graph_aggregated">false</boolProp>
      <stringProp name="include_sample_labels"></stringProp>
      <stringProp name="exclude_sample_labels"></stringProp>
      <stringProp name="start_offset"></stringProp>
      <stringProp name="end_offset"></stringProp>
      <boolProp name="include_checkbox_state">false</boolProp>
      <boolProp name="exclude_checkbox_state">false</boolProp>
    </kg.apc.jmeter.vizualizers.CorrectedResultCollector>
    <hashTree/>
    <kg.apc.jmeter.perfmon.PerfMonCollector guiclass="kg.apc.jmeter.vizualizers.P
      <boolProp name="ResultCollector.error_logging">false</boolProp>
      <objProp>
        <name>saveConfig</name>
        <value class="SampleSaveConfiguration">
          <time>true</time>
          <latency>true</latency>
          <timestamp>true</timestamp>
          <success>true</success>
          <label>true</label>
          <code>true</code>
          <message>true</message>
          <threadName>true</threadName>
          <dataType>true</dataType>
          <encoding>false</encoding>
          <assertions>true</assertions>
          <subresults>true</subresults>
          <responseData>false</responseData>
          <samplerData>false</samplerData>
          <xml>false</xml>
          <fieldNames>true</fieldNames>
          <responseHeaders>false</responseHeaders>
          <requestHeaders>false</requestHeaders>
          <responseDataOnError>false</responseDataOnError>
          <saveAssertionResultsFailureMessage>true</saveAssertionResultsFailureMe
          <assertionsResultsToSave>0</assertionsResultsToSave>
          <bytes>true</bytes>
          <threadCounts>true</threadCounts>
          <idleTime>true</idleTime>
        </value>
```

```
      </objProp>
      <stringProp name="filename"></stringProp>
      <longProp name="interval_grouping">1000</longProp>
      <boolProp name="graph_aggregated">false</boolProp>
      <stringProp name="include_sample_labels"></stringProp>
      <stringProp name="exclude_sample_labels"></stringProp>
      <stringProp name="start_offset"></stringProp>
      <stringProp name="end_offset"></stringProp>
      <boolProp name="include_checkbox_state">false</boolProp>
      <boolProp name="exclude_checkbox_state">false</boolProp>
      <collectionProp name="metricConnections">
        <collectionProp name="1321850483">
          <stringProp name="55965974">192.168.1.2</stringProp>
          <stringProp name="1600768">4444</stringProp>
          <stringProp name="66952">CPU</stringProp>
          <stringProp name="0"></stringProp>
        </collectionProp>
        <collectionProp name="-978863838">
          <stringProp name="55965974">192.168.1.2</stringProp>
          <stringProp name="1600768">4444</stringProp>
          <stringProp name="-1993889503">Memory</stringProp>
          <stringProp name="0"></stringProp>
        </collectionProp>
      </collectionProp>
    </kg.apc.jmeter.perfmon.PerfMonCollector>
    <hashTree/>
    <kg.apc.jmeter.vizualizers.CorrectedResultCollector guiclass="kg.apc.jmeter.v
      <boolProp name="ResultCollector.error_logging">false</boolProp>
      <objProp>
        <name>saveConfig</name>
        <value class="SampleSaveConfiguration">
          <time>true</time>
          <latency>true</latency>
          <timestamp>true</timestamp>
          <success>true</success>
          <label>true</label>
          <code>true</code>
          <message>true</message>
          <threadName>true</threadName>
```

```
            <dataType>true</dataType>
            <encoding>false</encoding>
            <assertions>true</assertions>
            <subresults>true</subresults>
            <responseData>false</responseData>
            <samplerData>false</samplerData>
            <xml>false</xml>
            <fieldNames>true</fieldNames>
            <responseHeaders>false</responseHeaders>
            <requestHeaders>false</requestHeaders>
            <responseDataOnError>false</responseDataOnError>
            <saveAssertionResultsFailureMessage>true</saveAssertionResultsFailureMe
            <assertionsResultsToSave>0</assertionsResultsToSave>
            <bytes>true</bytes>
            <threadCounts>true</threadCounts>
            <idleTime>true</idleTime>
          </value>
        </objProp>
        <stringProp name="filename"></stringProp>
        <longProp name="interval_grouping">500</longProp>
        <boolProp name="graph_aggregated">false</boolProp>
        <stringProp name="include_sample_labels"></stringProp>
        <stringProp name="exclude_sample_labels"></stringProp>
        <stringProp name="start_offset"></stringProp>
        <stringProp name="end_offset"></stringProp>
        <boolProp name="include_checkbox_state">false</boolProp>
        <boolProp name="exclude_checkbox_state">false</boolProp>
      </kg.apc.jmeter.vizualizers.CorrectedResultCollector>
      <hashTree/>
    </hashTree>
  </hashTree>
</jmeterTestPlan>
```