PHILLIP LUIZ VIANA

**Uma arquitetura de preservação
a longo prazo de Big Data
com gerenciamento de elasticidade em nuvem**

São Paulo
(2018)

PHILLIP LUIZ VIANA

**Uma arquitetura de preservação
a longo prazo de Big Data
com gerenciamento de elasticidade em nuvem**

Tese apresentada à Escola Politécnica da
Universidade de São Paulo para obtenção
do título de Doutor em Ciências

Orientador: Prof.^a Dr.^a Liria Matsumoto
Sato

São Paulo
(2018)

PHILLIP LUIZ VIANA

**Uma arquitetura de preservação
a longo prazo de Big Data
com gerenciamento de elasticidade em nuvem**

Tese apresentada à Escola Politécnica da
Universidade de São Paulo para obtenção
do título de Doutor em Ciências

Área de Concentração: Engenharia da
Computação

Orientador: Prof.^a Dr.^a Liria Matsumoto
Sato

São Paulo
(2018)

Autorizo a reprodução e divulgação total ou parcial deste trabalho, por qualquer meio convencional ou eletrônico, para fins de estudo e pesquisa, desde que citada a fonte.

Este exemplar foi revisado e corrigido em relação à versão original, sob responsabilidade única do autor e com a anuência de seu orientador.

São Paulo, _____ de _____ de _____

Assinatura do autor: _____

Assinatura do orientador: _____

Catálogo-na-publicação

VIANA, PHILLIP

Uma arquitetura de preservação a longo prazo de Big Data com gerenciamento de elasticidade em nuvem / P. VIANA -- versão corr. -- São Paulo, 2018.
135 p.

Tese (Doutorado) - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Computação e Sistemas Digitais.

1.COMPUTAÇÃO EM NUVEM 2.PRESERVAÇÃO DIGITAL
3.ARQUITETURA E ORGANIZAÇÃO DE COMPUTADORES 4.BIG DATA
I.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Computação e Sistemas Digitais II.t.

RESUMO

Com o crescimento exponencial do volume de dados estruturados e não estruturados (Big Data) em sistemas de armazenamento corporativos, aliado à também crescente demanda por preservação de tais dados devido a regulamentações e auditorias, surge o problema da preservação a longo prazo de Big Data, e mais especificamente o de como estender sistemas existentes ao longo do tempo. Pesquisas recentes contemplam arquiteturas de preservação de dados estruturados ou de arquivamento a curto prazo de Big Data, porém carecem de um modelo para arquiteturas que suportem a preservação a longo prazo de Big Data com elasticidade. Na presente tese, propõe-se uma arquitetura para o arquivamento, preservação a longo prazo e recuperação de Big Data com elasticidade. Um método de criação de arquiteturas de referência foi seguido e obteve-se como resultado uma arquitetura de preservação a longo prazo que é reproduzível e capaz de adaptar-se a uma demanda crescente, recebendo Big Data de fontes heterogêneas continuamente. A arquitetura é compatível com computação em nuvem e foi testada com diversas mídias de armazenamento, como mídias magnéticas, nuvem e de estado sólido. É feito também um comparativo entre a arquitetura desenvolvida e outras arquiteturas disponíveis.

Palavras-chave: Preservação. Retenção. Dados não estruturados. Big Data. Elasticidade.

ABSTRACT

With the exponential growth in the volume of structured and unstructured data (Big Data) in enterprise storage systems, along with the also increasing demand for preservation of such data due to regulations and audits, there arises the problem of long-term preservation of Big Data, and more specifically of how to extend existing systems with time. Recent research projects encompass architectures for the preservation of structured data or short term archiving of Big Data, however they lack a model for architectures that support long-term preservation of Big Data with elasticity. In the present thesis, we propose an architecture for the archiving, long-term preservation and retrieval of Big Data with elasticity. A method for creating reference architectures was followed and as a result a reproducible long-term preservation architecture was obtained, which is capable of adapting to a growing demand receiving Big Data continuously. The architecture is compatible with cloud computing and was tested against several storage media, such as magnetic media, cloud and solid state. A comparison between the architecture and other available architectures is also provided.

Keywords: Preservation. Retention. Big Data. Unstructured data. Elasticity.

LISTA DE ILUSTRAÇÕES

Figura 1 - Curva do ciclo de vida da informação.....	13
Figura 2 - Modelo funcional do OAIS.....	25
Figura 3 - Modelo dos elementos do formato PREMIS.....	26
Figura 4 - Representação de um volume AXF contendo arquivos AXF.....	29
Figura 5 - Mapeamento e redução.....	42
Figura 6 - Diagrama de estados de elasticidade.....	51
Figura 7 - Visão geral da arquitetura proposta.....	58
Figura 8 - Fluxo do monitor de <i>workload</i>	69
Figura 9 - Fluxo do controlador de elasticidade para provisionamento de recursos.....	71
Figura 10 - Fluxo do controlador de elasticidade para remoção de recursos.....	72
Figura 11 - Classes do OpenSIRF Core usadas para modelar os elementos SIRF.....	77
Figura 12 - Visão esquemática dos componentes REST e Core na arquitetura do protótipo.....	79
Figura 13 - Fluxo de objetos Java durante requisições.....	81
Figura 14 - Fluxo da instanciação do driver do contêiner a ser utilizado.....	85
Figura 15 - Esquema da elasticidade.....	92
Figura 16 - Processo de elasticidade para inserção de um novo contêiner de armazenamento.....	94
Figura 17 - Processo de elasticidade para remoção de um contêiner de armazenamento.....	96
Figura 18 - Interações do gateway de arquivamento com outros sistemas.....	98
Figura 19 - Envio de informações dos sistemas de Big Data para o gateway de arquivamento.....	100
Figura 20 - Fluxo dos testes com banco de dados relacional.....	103
Figura 21 - Número de cenários do experimento por mídia e operação de elasticidade.....	105
Figura 22 - Porcentagem de operações do experimento por política de distribuição.....	106
Figura 23 - Políticas de elasticidades e seus limites.....	107
Figura 24 - Operações sobre POs de acordo com o tipo de contêiner.....	108
Figura 25 - Cenários testados pelo tipo de dados.....	109

LISTA DE TABELAS

Tabela 1 - Comparativo dos modelos de preservação disponíveis.....	29
Tabela 2 - Tipos e valores de estado.....	35
Tabela 3 - Comparação entre SIRD e outros formatos abordados.....	38
Tabela 4 - Comandos da camada de acesso e operações SIRD.....	61
Tabela 5 - Comparação da proposta da tese com outras propostas.....	111
Tabela 6 - Características de reprodutibilidade.....	112

LISTA DE ABREVIATURA E SIGLAS

AIP	Archival Information Package
API	Application Programming Interface
AXF	Archive eXchange Format
CPU	Central Processing Unit
FTP	File Transfer Protocol
HIPAA	Health Insurance Portability and Accountability Act
HTML	Hypertext Markup Language
ISBN	International Standard Book Number
ISO	International Organization for Standardization
JSON	JavaScript Object Notation
OAIS	Open Archival Information System
ONG	Organização não-governamental
OSHA	Occupational Safety and Health Administration
PO	Preservation Object
RAID	Redundant Array of Independent Disks
REST	Representational State Transfer
SIRF	Self-contained Information Retention Format
SNIA	Storage Network Industry Association
TWG	Technical Working Group
SQL	Structured Query Language
XML	eXtensible Markup Language

SUMÁRIO

1	INTRODUÇÃO.....	8
1.1	OBJETIVO.....	10
1.2	JUSTIFICATIVA.....	11
1.3	METODOLOGIA DE PESQUISA.....	14
1.4	ESTRUTURA DA TESE.....	16
2	ESTADO DA ARTE.....	18
2.1	DIREÇÕES DA PRESERVAÇÃO DE BIG DATA.....	19
2.2	FORMATOS DE PRESERVAÇÃO.....	22
2.2.1	OAIS.....	23
2.2.2	PREMIS.....	24
2.2.3	OpenAXF.....	26
2.2.4	Análise.....	28
3	FUNDAMENTOS DE ARQUITETURAS DE BIG DATA E PRESERVAÇÃO	30
3.1	PRESERVAÇÃO DE DADOS.....	30
3.2	RECUPERAÇÃO E E-DISCOVERY.....	32
3.3	FORMATO DE PRESERVAÇÃO DE DADOS: SIRF.....	33
3.4	ARQUITETURAS DE BIG DATA.....	38
3.4.1	Ferramentas de processamento e análise de Big Data.....	39
3.4.2	Dados não-relacionais (NoSQL).....	42
3.4.3	Dados relacionais e Sistemas de arquivos.....	43
3.5	MÉTODO PARA CRIAÇÃO DE ARQUITETURAS.....	43
3.6	ARQUITETURAS RELACIONADAS EXISTENTES.....	46
3.7	COMPUTAÇÃO EM NUVEM, ESCALABILIDADE, EFICIÊNCIA E ELASTICIDADE.....	48
3.8	CONSIDERAÇÕES SOBRE OS TRABALHOS EXISTENTES APRESENTADOS.....	50
4	PROPOSTA DE ARQUITETURA DE PRESERVAÇÃO A LONGO PRAZO EM NUVEM COM ELASTICIDADE.....	53
4.1	CONCEPÇÃO DA PROPOSTA DE ARQUITETURA.....	54
4.2	VISÃO GERAL DA PROPOSTA DE ARQUITETURA.....	56
4.3	CAMADA DE DADOS.....	58

4.4	CAMADA DE ACESSO.....	60
4.5	CAMADA DE ARQUIVAMENTO.....	63
4.6	CAMADA DE ARMAZENAMENTO.....	64
4.7	COMPONENTES DE APRESENTAÇÃO E SERVIÇOS WEB.....	64
4.8	GERENCIAMENTO DE ELASTICIDADE.....	65
4.8.1	Política de elasticidade.....	66
4.8.2	Mecanismo de elasticidade e seus componentes.....	67
5	IMPLEMENTAÇÃO E RESULTADOS.....	74
5.1	INTRODUÇÃO AO OPENSIRF.....	74
5.2	OPENSIRF CORE.....	75
5.3	OPENSIRF SERVER.....	76
5.4	UTILIZAÇÃO DOS ELEMENTOS DO OPENSIRF CORE PELO OPENSIRF SERVER.....	77
5.5	IMPLEMENTAÇÃO DE MÚLTIPLOS CONTÊINERES DE ARMAZENAMENTO.....	81
5.6	MAPEAMENTO PARA O OPENSTACK SWIFT.....	85
5.7	MAPEAMENTO PARA SISTEMAS DE ARQUIVOS.....	88
5.8	IMPLEMENTAÇÃO DO MECANISMO DE ELASTICIDADE.....	89
5.9	IMPLEMENTAÇÃO DO GATEWAY DE ARQUIVAMENTO E CAMADA DE DADOS.....	96
5.9.1	Gateway de Arquivamento – Jenkins.....	96
5.9.2	Dados não estruturados e <i>Streaming</i> – Apache Spark e Twitter4s.....	97
5.9.3	Dados estruturados não-relacionais – mongoDB.....	100
5.9.4	Sistemas de arquivos.....	101
5.9.5	Dados estruturados relacionais – HammerDB/TPC-C com DB2.....	101
5.10	DADOS DO EXPERIMENTO.....	103
5.11	REPRODUÇÃO DA ARQUITETURA.....	108
6	DISCUSSÃO.....	110
7	CONCLUSÕES.....	113
7.1	Contribuições.....	114
7.2	Trabalhos futuros.....	115
	APÊNDICE A – INTERFACE REST.....	125

1 INTRODUÇÃO

O volume de dados armazenados em *data centers* corporativos vem crescendo a uma taxa de 35% a 50% ao ano (BEATH *et al.*, 2012). Segundo Tallon (2013), o crescimento exponencial de dados é a nova realidade. Grande parte desses dados são de natureza não estruturada, que em conjunto com os dados estruturados são comumente chamados na literatura jornalística de Big Data. Este crescimento exponencial no volume de dados leva a alguns desafios dos pontos de vista técnico, operacional e financeiro. Surgem os problemas de qual a melhor maneira e a melhor mídia para armazenar-se uma crescente quantidade de dados; como estender a quantidade e capacidade de recursos em tempo de execução para que um volume ainda maior de dados possa ser armazenado; e, finalmente, os custos associados. O crescimento dos dados não estruturados em particular está exacerbando esses problemas (TALLON, 2013) e é uma tendência crescente para o futuro devido a inovações no mundo digital tais como a internet das coisas, que tendem a incorrer em um volume ainda maior de dados.

Aliada a esse crescimento do volume de dados, vem aumentando também a demanda pela *preservação* desses dados. Dadas regulamentações governamentais, como por exemplo, HIPAA (ESTADOS UNIDOS DA AMÉRICA, 1996), Sarbanes-Oxley (ESTADOS UNIDOS DA AMÉRICA, 2002) e OSHA (ESTADOS UNIDOS DA AMÉRICA, 1970 e 2004) que definem períodos de preservação específicos para finalidades médicas, jurídicas ou criminalísticas, passa a ter mais importância do que nunca a manutenção de uma quantidade de dados em mídia por longos períodos de tempo – meses, anos, décadas ou mesmo para sempre. A não-preservação de dados pode incorrer penalidades financeiras ou jurídicas e, sendo assim, a preservação é justificada se o custo de armazenamento e manutenção são menores do que o custo de oportunidade de não ter os dados disponíveis para análise futura (TALLON, 2013).

A convergência dessas duas tendências, isto é, do crescimento do volume de dados e do aumento da demanda por preservação, implica que os sistemas de

armazenamento precisam dar suporte à preservação de dados a muito longo prazo (FACTOR *et al.*, 2007).

Além disso, quando os dados preservados são de natureza não-estruturada, é comum que o aumento no volume de dados se dê tão rapidamente que é necessário aumentar a capacidade do sistema em tempo de execução para acomodar a nova demanda. Com o advento da computação em nuvem essa nova demanda pode ser acomodada através da alocação de recursos sob demanda. Por outro lado, é necessário gerenciar a alocação de tais recursos, visando a utilizar a menor quantidade de recursos possível para reduzir o custo e ao mesmo tempo manter os requisitos de qualidade do sistema (LEHRIG, EIKERLING e BECKER, 2015a). Desta forma, em sistemas na nuvem faz-se necessário um mecanismo que seja capaz de controlar o número e a alocação de recursos de maneira a minimizar os custos. Herbst, Kounev e Reussner (2013) definem o conceito de elasticidade como “o grau em que a camada da nuvem adapta a capacidade ao *workload* com o tempo”. Dado esse conceito, a elasticidade pode ser utilizada para adaptar a quantidade de recursos utilizados por um sistema de preservação de dados cuja demanda pode aumentar ou diminuir rapidamente, visando otimizar a alocação de recursos sempre que possível.

Diversas pesquisas abordam o arquivamento, a preservação e o descobrimento (*E-Discovery*) de dados estruturados. Algumas pesquisas abordam os aspectos econômicos de Big Data e seu armazenamento. No entanto, após extensa pesquisa, não foram encontradas arquiteturas especificamente focadas na preservação a longo prazo de Big Data, capazes de adaptar-se a uma demanda crescente em tempo de execução, adicionando novas mídias de armazenamento e novos servidores com capacidade de processamento de maneira autônoma. Sistemas de armazenamento de Big Data possuem a característica de geração de grandes volumes de dados em curtos intervalos de tempo do ponto de vista de processamento e armazenamento. Assim, são necessárias técnicas para garantir que o sistema de armazenamento esteja disponível com capacidade suficiente e consiga armazenar os dados de maneira suficientemente rápida para que não haja perdas, além de ser capaz de adaptar suas capacidades em tempo de execução. Tal característica tem implicações de desempenho e escalabilidade, o que impacta

diretamente na arquitetura do sistema de armazenamento a longo prazo, levando à necessidade de uma abordagem dedicada a sistemas de Big Data, capaz de satisfazer os requisitos de escalabilidade e elasticidade, até o momento não cobertos por nenhuma pesquisa na revisão bibliográfica realizada.

A principal hipótese deste trabalho é que, utilizando-se elasticidade em uma arquitetura de preservação a longo prazo de Big Data, é possível ter um sistema capaz de não apenas preservar os dados como também crescer ao longo do tempo de maneira automática. A hipótese tem como fundamento as seguintes premissas: (1) o volume de dados gerados e preservados nas últimas décadas tem crescido e, portanto, é provável que continue crescendo no longo prazo; (2) a capacidade de armazenamento também tem crescido nas últimas décadas, de maneira que volumes atuais de dados não poderiam ser preservados em sistemas de armazenamento do passado, e como consequência (3) é necessária a criação de arquiteturas escaláveis para que se possa efetivamente fazer preservação de dados a longo prazo ou para sempre.

1.1 OBJETIVO

O objetivo desta tese é propor uma arquitetura para arquivamento, preservação a longo prazo e recuperação de dados estruturados e não estruturados (Big Data). A arquitetura deve prover mecanismos para o arquivamento, o posicionamento de dados em diferentes mídias físicas (fita, discos magnéticos, armazenamento em nuvem, entre outros), a busca, recuperação e o *E-Discovery*, além de mecanismos de metadados. Dada a natureza de sistemas de Big Data, onde dados são criados e consumidos em tempo real, os objetivos principais da arquitetura são a *escalabilidade* e *elasticidade*, ou seja, respectivamente “a habilidade de um sistema de aumentar sua capacidade ao expandir a quantidade de serviços das camadas inferiores” e “a capacidade do sistema de autonomamente adaptar sua capacidade de acordo com ao workload ao longo do tempo” (LEHRIG, EIKERLING e BECKER, 2015). Para que o requisito da elasticidade seja satisfeito é utilizada computação em

nuvem, de maneira que novos sistemas computacionais e mídias de armazenamento possam ser adicionados em tempo de execução.

Além disso a arquitetura também deve ser facilmente reproduzível, de modo que pesquisadores e praticantes do domínio da preservação a longo prazo possam se basear na proposta de arquitetura para criar seus próprios sistemas.

1.2 JUSTIFICATIVA

Foi realizada uma revisão bibliográfica, partindo-se de uma busca pelas principais publicações da área de arquivamento com o objetivo de buscar todo o material recente possivelmente relacionado com o objeto da tese.

A concepção da arquitetura proposta partiu de modelos já existentes de arquivamento e preservação para dados estruturados (FACTOR *et al.*, 2007A; FACTOR, NAOR E RABINOVICI-COHEN, 2007; NGUYEN, LAKE E HUBER, 2010; STRICOT, 2013; NGUYEN E LAKE, 2011; RABINOVICI-COHEN *et al.*, 2013; HAN, SUN E WANG, 2013; WALLACE E PORDESH, 2013), e foram estudadas as características relacionadas aos dados não estruturados. Também partiu-se de formatos de metadados específicos para a preservação a longo prazo (RABINOVICI-COHEN *et al.*, 2011; LIBRARY OF CONGRESS, 2016). Pease e Rabinovici-Cohen (2012) enunciam e discutem o problema da preservação de Big Data, particularmente focando em armazenamento em fitas magnéticas, citando a escalabilidade como um dos desafios da área de preservação a longo prazo de Big Data.

Beath *et al.* (2012), Tallon (2013), Merrill (2011), Reichman (2011) e Alnafoosi e Steinbach (2013) estudaram os aspectos econômicos do armazenamento de Big Data. Particularmente Tallon (2013) abordou com profundidade os aspectos de governança e financeiros de Big Data, como custo, valor, retorno de investimento, custos de oportunidade para preservação e/ou descarte de dados; bem como a curva do ciclo de vida da informação. Essa curva está reproduzida na Figura 1 e

contém o tempo no eixo X e o valor e utilização dos dados no eixo Y. Na região (I) os dados estão sendo criados e portanto tem valor e uso crescentes. Nessa fase os dados são armazenados em mídias relativamente rápidas e mais caras. Quando os dados estão prontos para serem consumidos eles tendem a ser acessados regularmente, portanto o seu valor aumenta consideravelmente e eles ficam armazenados em uma camada de armazenamento em mídia ainda mais rápida. Essa fase está mostrada na região (II).

À medida que o tempo vai passando e os dados já foram consumidos, o seu valor e utilização começam a decair. Nesse ponto os dados podem ser movidos para mídia menos rápida, o que ocorre na região (III) do gráfico. Os dados então atingem o chamado limite de preservação no limiar entre regiões (III) e (IV), e a partir desse ponto eles são arquivados e preservados ou removidos. Se os dados são preservados, novos picos de utilização podem ocorrer no futuro quando um usuário busca pelos dados, o que temporariamente aumenta o seu valor.

Figura 1. Curva do ciclo de vida da informação

Fonte: Tallon (2013)

O foco da tese está no que acontece a partir do limite de preservação, isto é, a região (IV), levando em consideração que os dados são preservados e não removidos. O interesse encontra-se no arquivamento, preservação a longo prazo e recuperação de informações presentes na região (IV).

Ao propor-se uma arquitetura de preservação a longo prazo de Big Data, propõe-se por consequência a preservação de um *workload* muitas vezes contínuo e com

grande volume de dados. Albonico *et al.* (2014) enunciam que, para lidar com grandes *workloads*, as aplicações precisam ser escaladas, e que o gerenciamento de tal escalabilidade não é uma tarefa trivial, sendo difícil de ser conseguido manualmente. É aí que entram a computação em nuvem e os conceitos de escalabilidade e elasticidade, permitindo que recursos sejam alocados e desalocados automaticamente de acordo com a demanda. Em Lehrig, Eikerling e Becker (2015) define-se os conceitos de capacidade e escalabilidade, enquanto Herbst, Kounev e Reussner (2013) utilizam esses mesmos conceitos para definir eficiência e elasticidade. A partir desses conceitos, foram criadas propostas para mecanismos de elasticidade de acordo com flutuações de *workload* (TOEROE, PAWAR e KHENDEK, 2014).

Há propostas de formatos de gerenciamento de metadados focadas especificamente na região (IV). O SIRF (*Self-contained Information Retention Format*) (RABINOVICI-COHEN *et al.*, 2011) é um formato focado na preservação a longo prazo de dados, com características compatíveis com Big Data e armazenamento em nuvem. É um formato projetado para ser independente de fabricante (deve funcionar independentemente do modelo de disco rígido, fita ou qualquer outro dispositivo de armazenamento), independente de mídia (deve funcionar igualmente em mídias magnéticas, de estado sólido, em nuvem, entre outras), extensível e autocontido (não necessita de outros formatos ou um software para ser entendido). O SIRF foi criado após a execução da pesquisa investigativa do “100 year archive”, ou arquivo de 100 anos (SNIA, 2007).

A preservação a longo prazo de Big Data se diferencia da preservação a longo prazo tradicional, de dados puramente estruturados, dadas algumas características típicas de *workload* de sistemas de Big Data. Particularmente, sistemas de Big Data possuem dados não estruturados que podem crescer de volume rapidamente. Além disso, muitas vezes esses dados estão distribuídos em diversos servidores diferentes, o que traz desafios no tocante à busca dos dados preservados.

Na revisão realizada constatou-se que se carece de uma arquitetura que sirva como base para sistemas de preservação a longo prazo com foco também em dados não estruturados e sua típica demanda crescente. Yan *et al.* (2017) apontam que os

dados preservados a longo prazo tem se tornado uma fonte primária de Big Data. Segundo os autores, dados acumulados todos os dias provindos de fontes como redes sociais e Internet das coisas também devem ser preservados a longo prazo já que tais dados podem prover valiosas informações quando analisados por tecnologias emergentes de análise de Big Data. Nguyen e Lake (2011) sugerem que um sistema de preservação a longo prazo na nuvem “poderia oferecer elasticidade no provisionamento de capacidade computacional e de armazenamento”, sem no entanto implementar mecanismos que possibilitam tal elasticidade. Pesquisas posteriores também não contemplaram o problema. Essas duas pesquisas deixam o problema em aberto e indicam que a hipótese considerada nesta pesquisa é válida. Além disso, com a tendência de armazenamento de dados em nuvem, a utilização de computação em nuvem com mecanismos de elasticidade também torna-se relevante para a tese. Embora tenha-se encontrado arquiteturas para a preservação de dados estruturados, bem como pesquisas sobre os aspectos de governança de Big Data, o conjunto atual de pesquisas não possui propostas de arquiteturas de referência ou arquiteturas básicas para preservação de longo prazo de Big Data capazes de satisfazer os requisitos de escalabilidade e elasticidade.

Desta forma, a tese debruça-se sobre três grandes tópicos: Big Data, preservação a longo prazo e arquiteturas em nuvem com elasticidade. Sucintamente, o objetivo é criar uma arquitetura para a preservação a longo prazo de Big Data com elasticidade, e então implementar e disponibilizar a arquitetura de maneira que ela possa ser reproduzida. A reprodução é possível pois usou-se um método para criação de arquiteturas de referência.

1.3 METODOLOGIA DE PESQUISA

Inicialmente foi feita uma busca nos anais dos principais congressos e artigos das principais revistas da área de armazenamento, procurando-se por palavras-chave tais como *preservação*, *retenção*, *longo prazo*, *dados não estruturados* e *big data*, com diversas combinações diferentes, concatenados ou não. O objetivo dessa primeira busca foi fazer uma varredura na área, buscar material relacionado e,

eventualmente, identificar que a proposta de arquitetura é de fato uma contribuição relevante para a área de pesquisa.

De posse dos resultados dessa primeira pesquisa, foram lidas as publicações encontradas, bem como as principais referências das mesmas relevantes para a tese. Desse ponto em diante foi-se formando uma teia de referências partindo das primeiras publicações encontradas, suas referências, referências das referências, e assim por diante. Além da teia de referências, repetiu-se a pesquisa pelas palavras-chave de maneira iterativa para garantir que novas propostas relacionadas fossem encontradas desde o início da pesquisa. À medida que novas propostas foram encontradas, seu conteúdo foi incorporado caso tenha apresentado relevância para a tese. As propostas mais relevantes e recentes foram adicionadas ao capítulo sobre o estado da arte.

Encontrou-se propostas de métodos de geração de arquiteturas de referência e também diversas propostas relacionadas à preservação a longo prazo de dados, arquiteturas de Big Data e padrões de preservação a longo prazo. De posse dessas pesquisas, instanciou-se um método escolhido para a criação de arquiteturas de referência, tendo em mente a preservação a longo prazo.

Após a definição da arquitetura básica de preservação a longo prazo (ainda sem elasticidade), iniciou-se a implementação de um protótipo. Inicialmente este protótipo possuía implementadas exclusivamente as funcionalidades de preservação de dados. Entretanto, dada a natureza de sistemas de Big Data de gerar uma quantidade crescente de dados, ou seja, ter um *workload* contínuo e com grande volume de dados, exaurir a capacidade de recursos, em especial espaço livre em disco, torna-se uma questão de tempo. Dada a necessidade de escalar os recursos do sistema (ALBONICO *et al.*, 2014), buscou-se alternativas que pudessem resolver o problema. A computação em nuvem e a elasticidade foram encontradas como uma potencial solução, em especial pelo fato do problema da elasticidade em sistemas de preservação a longo prazo ter sido enunciado porém não resolvido (NGUYEN e LAKE, 2011). Portanto, iniciou-se uma nova busca, desta vez por publicações relativas a computação em nuvem, elasticidade e, em especial, como aplicar elasticidade em sistemas com workloads contínuos e com grande volume de dados.

A pesquisa por publicações que relacionam a computação em nuvem à preservação de longo prazo de Big Data se deu de maneira similar à pesquisa inicial sobre preservação mencionada nos parágrafos anteriores: inicialmente foi feita uma busca nos anais dos principais congressos e artigos das principais revistas da área de computação em nuvem, procurando-se por palavras-chave tais como escalabilidade, elasticidade, cloud, preservação, longo prazo, Big Data e dados não estruturados, com diversas combinações diferentes, de maneira que pelo menos uma das palavras-chave relativas ao gerenciamento dos recursos (escalabilidade, elasticidade, cloud) e uma das palavras-chave relacionadas à arquitetura de preservação a longo prazo de Big Data (preservação, longo prazo, Big Data e dados não estruturados) estavam presentes em todas as buscas. Foram lidas as publicações encontradas, bem como suas principais referências. Desse ponto em diante foi-se formando uma teia de referências partindo das primeiras publicações encontradas, suas referências, referências das referências, e assim por diante. Além da teia de referências, repetiu-se a pesquisa pelas palavras-chave de maneira iterativa para garantir que novas propostas relacionadas fossem encontradas desde o início da pesquisa. As propostas mais relevantes e recentes foram adicionadas ao capítulo sobre o estado da arte.

1.4 ESTRUTURA DA TESE

A tese está dividida nas seguintes seções:

- Capítulo 1 – Apresenta a introdução e contextualização da área de arquivamento e preservação de Big Data, bem como justificativa, metodologia e objetivo.
- Capítulo 2 – É apresentado o estado da arte das arquiteturas de preservação a longo prazo, bem como as pesquisas mais recentes relacionando a preservação com computação na nuvem.

- Capítulo 3 – É apresentada a análise feita das pesquisas existentes de arquivamento e preservação de Big Data. As propostas principais que deram origem à tese são abordadas.
- Capítulo 4 – É mostrada a proposta da tese e a arquitetura proposta é detalhada.
- Capítulo 5 – Explica-se como a arquitetura foi instanciada como um protótipo, os limites experimentais da tese e como a aplicação experimental foi realizada.
- Capítulo 6 – Discute-se e analisa-se os resultados da aplicação experimental.
- Capítulo 7 – São apresentadas as conclusões obtidas através da pesquisa, bem como futuras melhorias que podem ser implementadas.
- REFERÊNCIAS – São listadas em ordem alfabética de sobrenome do primeiro autor as referências bibliográficas usadas no desenvolvimento da tese.

2 ESTADO DA ARTE

Nesta seção são apresentadas algumas pesquisas que estabeleceram os fundamentos do estado da arte. A área de pesquisa de preservação a longo prazo por natureza possui em grande parte os mesmos conceitos e objetivos de anos ou até décadas atrás, já que nos dias atuais se está preservando dados passados e no futuro se estará preservando dados atuais. Sendo assim, grande parte das pesquisas passadas continuam relevantes e por isso são apresentadas como base fundamental das pesquisas atuais que compõem o estado da arte.

Factor *et al.* (2007) dão a definição de preservação de dados e estabelecem alguns dos fundamentos das pesquisas atuais. São definidas a preservação *lógica*, que é a preservação do formato e significado dos dados, bem como sua legibilidade; e *binária* ou *física*, que é a preservação (não-corrupção) dos dados em mídia, geralmente magnética. Essa pesquisa dá início aos aspectos técnicos do arquivamento e preservação digitais a longo prazo. Fujita, Naono e Hanai (2012) apresentam uma pesquisa sob a perspectiva do *E-Discovery*, isto é, no descobrimento e recuperação da informação após os dados já estarem retidos. Rabinovici-Cohen *et al.* (2011) introduzem o conceito de objeto de preservação e definem o formato SIRF, específico para a preservação a longo prazo de dados. Além disso, definem aspectos de implementação do SIRF para mídia magnética (discos e fitas) bem como para armazenamento na nuvem.

Através de extensa pesquisa bibliográfica foram encontrados diversos estudos que tratam do arquivamento de dados estruturados (BAKER, KEETON E MARTIN, 2005; CCSDS, 2002; HAN, SUN E WANG, 2013; WALLACE E PORDESH, 2013), preservação de dados estruturados (FACTOR *et al.*, 2007A; BAKER, KEETON E MARTIN, 2005; FACTOR, NAOR E RABINOVICI-COHEN, 2007; NGUYEN, LAKE E HUBER, 2010; STRICOT, 2013; NGUYEN E LAKE, 2011; RABINOVICI-COHEN *et al.*, 2013; HAN, SUN E WANG, 2013; WALLACE E PORDESH, 2013) e recuperação de dados estruturados (FACTOR, NAOR E RABINOVICI-COHEN, 2007; NGUYEN, LAKE E HUBER, 2010; FUJITA, NAONO E HANAI, 2012; BUTLER *et al.*, 2009; RIM *et al.*, 2012; EDRM, 2014). Foram encontradas diversas propostas que abordavam

os aspectos financeiros, legais e técnicos do arquivamento e preservação a longo prazo de dados.

No tocante a Big Data, alguns estudos (BEATH *et al.*, 2012; TALLON, 2013; MERRILL, 2011; ALNAFOOSI e STEINBACH, 2013; REICHMAN, 2011) abordam os aspectos econômicos de Big Data e seu armazenamento. Há também propostas de arquiteturas de referência de Big Data (GEERDINK, 2013; DOSHI *et al.*, 2013). Tallon (2013) abordou governança e aspectos financeiros de Big Data, bem como a curva do ciclo de vida da informação apresentada na Figura 1.

Foram encontradas também arquiteturas de referência para a preservação a longo prazo de dados estruturados (HUHNLEIN *et al.*, 2009; BUTLER *et al.*, 2009; EDRM, 2014). O método seguido para gerar tais arquiteturas foi a base usada nesta tese para garantir que a arquitetura proposta possa ser reproduzida.

Outros trabalhos não citados anteriormente também foram utilizados e receberam seções ou capítulos inteiros dedicados ao seu detalhamento. Na seção 2.2 são detalhados os formatos PREMIS, OAIS e OpenAXF, que tem sido usados como implementações de arquivamento e preservação há diversos anos e tendem a continuar a ser utilizados. No capítulo 3 é apresentado em detalhes o formato SIRF que foi estendido neste trabalho e serviu como base da tese.

2.1 DIREÇÕES DA PRESERVAÇÃO DE BIG DATA

Com o advento de métodos analíticos focados em dados não-estruturados e a identificação de Big Data como uma área a ser explorada pela academia e pelo mercado, pesquisas começaram a apontar na direção da preservação a longo prazo de Big Data. Particularmente destaca-se o foco atual em conteúdos multimídia (vídeo, áudio, textos de redes sociais) como forma de preservação da memória e tendências da humanidade. A curadoria digital de tais artefatos, bem como a mineração e análise dos dados não-estruturados em busca de padrões, aparece como uma tendência atual de pesquisa.

Pease e Rabinovici-Cohen (2012) são os primeiros a explicitamente cobrir o problema da preservação de Big Data, apontando a escalabilidade como um dos desafios para a preservação. Desde então diversas pesquisas tem focado na preservação de Big Data. Khan *et al.* (2016) abordam a preservação de longo prazo de notícias de jornais ou revistas online e apontam a necessidade de extração de várias fontes de notícias online e migração para um formato normalizado como um desafio. Dado que há constante mudança em tecnologias usadas para a publicação de histórias e notícias, as fontes utilizam formatos diferentes para a publicação de seu conteúdo e, portanto, técnicas de análise de Big Data, como mapeamento, redução, filtro, entre outros, são algumas das formas de se normalizar o conteúdo. Além disso, em muitos casos a preservação pode consistir na digitalização de conteúdo em formatos de imagem como TIFF ou JPEG2000, o que torna o texto de natureza ainda mais não-estruturada e com ausência de metadados, tornando a análise ainda mais importante.

Bruns e Weller (2016) seguem uma direção parecida ao identificar o Twitter como fonte histórica e um artefato cultural que deve ser preservado. De fato, a biblioteca do congresso americano possui uma parceria com o Twitter para a preservação de todos os *tweets* já submetidos como forma de preservar os pensamentos da sociedade para uso por futuros historiadores. Segundo Bruns e Weller (2016), a distribuição viral de notícias no Twitter não constitui uma transmissão puramente linear de informação para um círculo de espectadores cada vez mais abrangente; ao contrário, a cada passo do processo a informação tende a engatilhar mais respostas individuais e discussões interpessoais. Isso gera um volume grande de dados. Os autores também identificam como desafio a preservação de elementos não-textuais como informação visual e audiovisual, conexões a conteúdo fora do Twitter e várias funcionalidades embutidas na interface de usuário. Trata-se de dados não-estruturados e, portanto, mais uma vez a preservação se relaciona com Big Data.

Yan *et al.* (2017) mencionam que tecnologias emergentes de análise de Big Data permitem revelar o valor de um crescente volume de dados que seria descartado diariamente, mas que agora está sendo considerado para preservação de longo prazo. Aplicações analíticas que dependem da mineração (*mining*) de dados

históricos dependem da acessibilidade a esses dados preservados a longo prazo, de maneira semelhante ao acesso a bancos de dados e sistemas de arquivos. Os autores também apontam que os dados preservados a longo prazo tem se tornado uma fonte primária de Big Data.

A área de Big Data apresenta desafios próprios de preservação, dados os formatos heterogêneos que são utilizados e a presença de diversos tipos de dados não-estruturados, tais como vídeos, áudio e texto puro. Como o volume de informação é crescente, ferramentas analíticas tornam-se interessantes para resumir as informações e preservá-las de maneira estruturada e acessível.

Hirano e Ogawa (2016) focam em preservação de longo prazo para processos forenses em sistemas distribuídos em vários servidores e mídias de armazenamento, focando em restaurar um dispositivo de blocos virtual (disco rígido) a um ponto arbitrário no tempo. Lagos e Vion-Dury (2016) apresentaram um trabalho sobre a preservação lógica, particularmente sobre a adição do contexto de documentos para aumentar a inteligibilidade e governança dos documentos preservados, e apresentam uma comparação entre o seu método de preservação e outros métodos que não levam em conta o contexto.

Shankar (2016) apresenta uma pesquisa sobre curadoria digital, ressaltando a diferença entre fazer backups tradicionais de dados e uma estratégia real de preservação. Aponta-se que arquivos, bibliotecas e museus investem fortemente em desenvolver coleções digitais que incluem digitalização de coleções baseadas em papel, o que mais uma vez aponta para a necessidade de gerenciamento de metadados e uma estruturação de informações não-estruturadas contidas nessas obras digitalizadas e, portanto, relaciona-se com preservação de Big Data.

Da Silva e Bores (2016) também apresentam pesquisa sobre curadoria digital e ressaltam as diferenças entre a preservação física e lógica, tal como fizeram Lagos e Vion-Dury (2016) ao abordar os aspectos de contexto de preservação. O modelo apresentado por Silva e Bores (2016) destaca alguns elementos do modelo PREMIS, tal como a fixidez, além de apresentar uma técnica de emulação de

sistemas para interpretar documentos originais cujas tecnologias não mais se encontram disponíveis (preservação lógica).

Clua e Feldgen (2017) apresentam uma pesquisa sobre digitalização de manuscritos para pesquisas históricas. Catálogos digitais com metadados estão sendo criados através da análise do conteúdo não-estruturado digitalizado, e para isso foram utilizadas técnicas de extração de palavras-chave significativas. A pesquisa aponta a necessidade de utilização de um padrão internacional que permita a integração de diversos formatos, ou seja, a conversão para um único modelo global com interoperabilidade.

McGovern (2017) apresentou uma pesquisa focada em dados ameaçados de extinção, como por exemplo dados sobre mudanças climáticas que poderiam ser destruídos por motivações políticas. A pesquisa envolveu atividades para garantir que o conteúdo digital continuasse significativo e acessível através de diversas gerações de tecnologias.

Khan *et al.* (2016) criaram uma arquitetura para coletar notícias relacionadas a um mesmo tópico de diversas fontes online e arquivá-las. Um dos principais desafios apontados na pesquisa é a heterogeneidade de formatos, metadados e tecnologias, o que aponta para a necessidade de normalização dos dados, que pode ser feita através de técnicas de análise de Big Data.

2.2 FORMATOS DE PRESERVAÇÃO

Nesta seção são apresentados os principais formatos utilizados no domínio de preservação de dados. Mais uma vez ressalta-se que, dado o requisito de longa duração de preservação, tais formatos continuam relevantes mesmo após a sua criação há muitos anos ou até décadas, já que continuam em uso e dando suporte a repositórios de preservação a longo prazo criados há muitos anos.

2.2.1 OAIS

O OAIS (*Open Archival Information System*) é um modelo de sistema cuja tarefa é preservar informações a longo prazo, inicialmente projetado para servir o domínio de agências espaciais. O OAIS contém um modelo de referência de um ambiente OAIS. Um ambiente OAIS é composto de consumidores, que utilizam as informações do arquivo; fornecedores, que criam as informações do arquivo; gerenciamento, que estabelece as políticas e objetivos do arquivo, e finalmente o arquivo em si.

O modelo OAIS define o conceito de comunidade designada, que é um subconjunto de indivíduos consumidores cuja base de conhecimento é necessária para a interpretação dos objetos preservados. Um objeto preservado no modelo OAIS nem sempre poderá ser interpretado apenas com a base de conhecimento da comunidade designada, necessitando frequentemente de informações adicionais contidas no objeto (metadados) que ajudam a interpretar o conteúdo do objeto preservado.

Define-se também o conceito de pacote de informações (*information package*), que contém quatro tipos de objetos de informação: informações de conteúdo (o conteúdo do objeto preservado em si); de descrição de preservação, que são as informações necessárias para garantir a preservação do objeto, incluindo informações de proveniência e *checksums*; de empacotamento, e de descrição, que facilita a busca e acesso do objeto no arquivo. Três tipos de pacotes de informações existem: o AIP, ou *archival information package* (pacote de informações do arquivo), que é o pacote de informação armazenado pelo arquivo; o SIP, ou *submission information package* (pacote de informações de submissão), que é enviado de um fornecedor para o arquivo; e, finalmente, o DIP, ou *dissemination information package* (pacote de informações de disseminação), que é enviado do arquivo para um consumidor.

A Figura 2 mostra o modelo funcional do OAIS. Primeiramente um fornecedor de dados a serem arquivados cria um pacote de submissão (SIP) e entrega o pacote para o serviço de ingestão, cuja responsabilidade é receber o pacote e prepará-lo para o arquivamento. O serviço de ingestão transforma o pacote SIP em um pacote

AIP (de arquivamento) e passa as informações descritivas para o serviço de gestão de dados. A função de armazenamento de arquivo faz o armazenamento em mídia, manutenção (migração de mídia, checagem de erros) e recuperação dos dados arquivados, ou seja, entrega dos dados para o serviço de acesso. A função de administração é responsável pelos aspectos operacionais do arquivo, tais como controle de acesso, auditorias, políticas e padrões, enquanto a função de gerenciamento de dados coordena os dados descritivos dos AIPs, sendo responsável por encontrar informações de acordo com as requisições vindas da função de acesso e retornar as informações necessárias. Finalmente, a função de acesso é a interface utilizada pelo consumidor para requerer informações presentes no arquivo. A função de acesso retorna um pacote de disseminação (DIP) que é então utilizado pelo consumidor.

2.2.2 PREMIS

O PREMIS é um formato de metadados de preservação que compreende um conjunto de *unidades semânticas* que podem ser utilizadas para funções de preservação. As unidades semânticas são definidas no *dicionário de dados* do formato PREMIS, que é independente de formato de dados.

A Figura 3 mostra um modelo dos elementos do formato PREMIS: entidades intelectuais, direitos, objetos, agentes e eventos.

Figura 3. Modelo dos elementos do formato PREMIS

Adaptado de: Library of Congress (2016)

Entidades intelectuais são elementos com conteúdo, tais como livros, mapas, fotografias e bancos de dados. Diversos tipos e padrões de metadados existem para entidades intelectuais, como por exemplo ISBN para livros. Um objeto de preservação do PREMIS é associado a uma entidade intelectual através de um identificador nos metadados do objeto.

Objetos são elementos do mundo real que de fato são preservados no repositório. Três tipos de objetos são definidos:

- *Arquivo*, que são arquivos de computadores como documentos do Word, imagens e vídeos.
- *Representação*, que são conjuntos de objetos necessários para interpretar uma entidade intelectual, por exemplo, uma página web é composta por mais de um arquivo, com CSS, HTML, JavaScript, etc.
- *Bitstream*, que são pedaços de um arquivo que possuem propriedades comuns para preservação mas que não podem existir sem um cabeçalho ou estrutura externa. Por exemplo, num arquivo de vídeo único que contém vídeo e áudio combinados, pode-se ter um *bitstream* para o vídeo e outro para o áudio.

Eventos são elementos responsáveis por manter informações de proveniência e autenticidade de um objeto através do registro de ações que afetam os objetos no repositório. A criação, migração e destruição de um objeto são exemplos de registro de ações. Um evento também possui informações sobre o(s) agente(s) envolvido(s) nas ações que afetaram os objetos.

Direitos são elementos que representam restrições e permissões relativas à cópia e reprodução dos objetos preservados. Dado que repositórios de preservação geralmente envolvem criação de cópias ou versões derivadas de um objeto, frequentemente as leis de copyright são uma preocupação em repositórios de preservação de dados. No PREMIS as informações de direito declaram as ações que são permitidas em um repositório e a base legal que permite essas ações.

Agentes são participantes do repositório que se possuem direitos e são capazes de atuar no repositório, gerando eventos. Agentes podem ser pessoas, organizações ou aplicações que criam eventos. Um agente pode ter diversos papéis, e quando um agente é referenciado em uma declaração de direito ou de evento o papel do agente também é declarado.

2.2.3 OpenAXF

O OpenAXF é um formato de preservação a longo prazo que utiliza a compressão para empacotar dados e metadados sendo preservados. Cada volume do OpenAXF compreende um cabeçalho, um identificador e um conjunto de objetos AXF, abstraindo os contêineres de armazenamento de outros elementos do sistema. Trata-se, na prática, de uma coleção de arquivos compactados usando o algoritmo zip. O formato OpenAXF suporta armazenamento em nuvem.

O OpenAXF foi concebido com os seguintes objetivos:

- Migração de dados entre diferentes sistemas de arquivos

- Flexibilidade na mudança de fornecedores de gestão de arquivos
- Suporte à automação de metadados e inclusão de metadados opcionais

A Figura 4 mostra um diagrama que representa um volume do OpenAXF com diversos objetos AXF. Cada objeto AXF é autocontido e possui uma coleção de arquivos e metadados. Um objeto AXF sempre possui conteúdo equivalente através de diversas mídias, ou seja, o conteúdo do objeto é o mesmo em uma fita, disco rígido ou mídia óptica. Um objeto compreende:

- Um cabeçalho com metadados descrevendo o conteúdo do objeto, como data de criação, descrição, permissões, entre outros.
- Um contêiner de metadados que podem ser de qualquer tipo, específico do fornecedor do objeto, respeitar ou não um determinado padrão, etc. É um espaço genérico reservado para o curador ou empresa curadora do objeto um marcador de início de payload, que identifica que o conteúdo cru dos arquivos está a seguir.
- Um conjunto de arquivos que incluem um rodapé com dados de tamanho e *checksum*; um marcador de fim de payload e, finalmente, um rodapé para o objeto AXF que na prática repete o conteúdo do cabeçalho do objeto AXF.

O identificador AXF possui informações sobre a mídia. A implementação desse cabeçalho varia de acordo com o tipo de mídia sendo utilizado. Nota-se que objetos AXF são independentes de mídia, mas não necessariamente os metadados contidos fora dos objetos, como é o caso do identificador AXF. O identificador permite que sistemas que suportam o OpenAXF compreendam imediatamente as informações necessárias para acessar os objetos.

O formato AXF também suporta o mecanismo de *spanning*, em que um objeto de preservação pode ocupar mais de um contêiner de preservação.

2.2.4 Análise

Dentre as propostas de padrões e arquiteturas de preservação apresentados, tem-se modelos de preservação com foco em independência de fabricante de armazenamento, suporte a migração entre mídias, autodescrição e suporte a futuras

Figura 4. Representação de um volume AXF contendo arquivos AXF

Adaptado de: Weiss (2014)

mídias (incluindo a nuvem). Nenhuma dessas propostas, no entanto, dá suporte à preservação de Big Data, o que é um ponto a ser explorado segundo Yan *et al.* (2017), já que dados acumulados diariamente de fontes heterogêneas tem provido informações valiosas quando analisados por tecnologias de análise de Big Data. Pesquisas atuais de preservação focam na curadoria digital de artefatos históricos, ciência forense e domínios específicos que exigem preservação de longo prazo, muitas vezes envolvendo a digitalização de objetos concretos que são por natureza não-estruturados e portanto podem ser analisados futuramente por ferramentas analíticas de Big Data. De fato, quando se trata de curadoria digital é impossível não se pensar em Big Data, dado o grande volume de dados não-estruturados

proveniente de mídias digitais e digitalizadas. Desta forma, o problema da preservação de Big Data encontra-se com margem para melhorias e avanços.

Como enunciado por Pease e Rabinovici-Cohen (2012) a escalabilidade (aumento da capacidade ao expandir a quantidade de serviços) é um dos desafios da preservação de Big Data, e dentre as propostas cobertas no estado da arte apenas o formato OpenAXF possui essa característica. No entanto, em sistemas de computação em nuvem é possível ir além da escalabilidade através da implementação de uma arquitetura elástica, isto é, que adapta *autonomamente* a capacidade ao workload com o tempo, desta maneira garantindo que a inatividade de recursos seja mínima. Nguyen e Lake (2011) sugerem a elasticidade para provisionamento de capacidade computacional e de armazenamento para sistemas de preservação a longo prazo.

A Tabela 1 mostra um comparativo dos modelos de preservação disponíveis de acordo com características importantes de tais sistemas baseado nos trabalhos encontrados na pesquisa do estado da arte.

Tabela 1. Comparativo dos modelos de preservação disponíveis

Propostas	Características	Estratégia de implementação	Independente de fabricante	Migração para outros arquivos	Spanning	Versionamento de objetos	Autodescrito	Sup. futuras mídias	Suporte a fita	Escalabilidade	Armazenamento em nuvem	Concebida independente de domínio	Suporte a Big Data	Preservação	Gerenciamento de elasticidade
OpenAXF		X	X	X	X		X	X	X	X	X			X	
PREMIS			X			X	X							X	
OAIS			X	X			X					X		X	

3 FUNDAMENTOS DE ARQUITETURAS DE BIG DATA E PRESERVAÇÃO

Nesse capítulo são detalhados o levantamento realizado da literatura existente publicada na área de preservação a longo prazo e o método de criação de arquiteturas de referência que serviu de base para a elaboração da tese.

3.1 PRESERVAÇÃO DE DADOS

Factor *et al.* (2007) dão a definição de preservação de dados como a capacidade de "garantir que os dados digitais sendo armazenados hoje possam ser lidos e interpretados muitos anos (dezenas ou centenas de anos) de agora". Nesta mesma pesquisa são identificados dois aspectos da preservação: a preservação *física* (*bit preservation*) e a preservação *lógica* (*logical preservation*). Enquanto a preservação física tem o objetivo de garantir que os bits presentes em mídia poderão ser lidos no futuro, a preservação lógica visa garantir que a informação continue compreensível e interpretável, independentemente de eventuais alterações nos bits.

Rabinovici-Cohen *et al.* (2011) foi um trabalho de fundamental importância, já que a presente tese debruçou-se sobre o formato proposto para a elaboração da arquitetura desejada. Nesse trabalho é dada uma definição de preservação lógica de dados: "consiste em preservar a inteligibilidade e usabilidade dos dados, apesar das mudanças que acontecerão em servidores, sistemas operacionais, produtos de gerenciamento de dados, aplicações e até mesmo os usuários a longo prazo". Garantir apenas a integridade física de um conjunto de bits não é suficiente se as ferramentas responsáveis por decodificar os dados ou interpretá-los tornaram-se obsoletas ou não estão mais disponíveis. O aspecto físico tipicamente recebe mais atenção por parte de pesquisadores, e por esse motivo o problema da preservação costuma ser visto como uma questão de confiabilidade. Alnafoosi e Steinbach (2013) por exemplo identificam a confiabilidade como a habilidade de recuperar dados *sem perdas*.

Embora grande parte dos trabalhos na área de preservação de dados tem proposto estratégias para garantir a preservação física de dados, o modelo de referência *Open Archival Information System*, ou OAIS (CCSDS, 2002), define a base para o aspecto de preservação lógica. Um OAIS consiste de uma organização de pessoas e sistemas com a habilidade de preservar dados para uma comunidade específica por um longo período de tempo, ou até mesmo pra sempre. O modelo de referência OAIS possui como elemento principal o *Archival Information Package*, ou AIP. Um AIP contém dados e metadados. Os metadados possuem um identificador único (informação de referência), a origem do objeto (chamada *informação de proveniência*), a relação do objeto com outros objetos (chamada *informação de contexto*), além de outras informações cuja função é garantir a autenticidade e integridade dos dados.

Além do AIP, o modelo de referência OAIS identifica tipos de migração de dados, como a cópia de dados de uma mídia para uma mídia nova do mesmo tipo (*refreshment*), a cópia de uma mídia para uma mídia nova de outro tipo (*replication*), o rearranjo dos componentes de dados dentro de um pacote (*repackaging*) e a cópia de dados combinada com alterações de formato (*transformation*). O modelo OAIS, embora focado no arquivamento e não necessariamente na preservação a longo prazo, é sempre referenciado por propostas de arquiteturas de preservação a longo prazo. Sua referência na presente tese se dá em caráter informativo e introdutório, no entanto não é um modelo completo ou mesmo suficiente para garantir-se a preservação a longo prazo de dados.

Weiss (2014) introduz o formato OpenAXF, que iniciou-se devido a requisitos da Sociedade de Engenheiros de Cinema e Televisão (SMPTE – *Society of Motion Picture & Television Engineers*). O trabalho iniciou-se a partir de um levantamento feito com usuários do domínio, tendo em vista os desafios de migração de sistemas de arquivamento entre diferentes mídias e o suporte a metadados. O formato OpenAXF toma como base diversos metadados existentes no modelo OAIS.

O modelo OAIS também serve como base para o formato *Self-contained Information Retention Format*, SIRF (SNIA, 2014), da *Storage Network Industry Association*, ou SNIA (associação da indústria de redes de armazenamento). A SNIA é uma

associação sem fins lucrativos de empresas da área de armazenamento com mais de 400 membros, entre eles os principais fabricantes e vendedores de tecnologia de armazenamento. Sua missão é manter diversos *Technical Work Groups*, ou TWGs (grupos de trabalho técnico) especializados em diversas áreas relacionadas ao armazenamento. Um desses TWGs é o grupo de retenção a longo prazo de dados.

A principal contribuição do TWG de retenção foi o formato SIRF, que é em grande parte baseado no modelo OAIS. Mais detalhes sobre o SIRF são dados na seção 3.3.

3.2 RECUPERAÇÃO E E-DISCOVERY

Uma vez que os dados são preservados, em algum momento no futuro parte deles será recuperada para alguma finalidade – são os picos de valor e uso mostrados na região (IV) da Figura 1. Sendo assim, é necessário garantir que os dados possam ser buscados e encontrados. Essa garantia passa tanto pela preservação física quanto pela preservação lógica dos dados preservados. O estudo e aplicação de técnicas de preservação física de dados não são novidade e são bem conhecidos. Algumas técnicas, por exemplo, são os *arrays* tipo RAID¹, duplicação de dados e espelhamento. Já para a preservação lógica de dados, é necessário buscar um correto gerenciamento de metadados, já que a maior parte das informações que tem de ser preservadas é composta de dados de referência e, portanto, muda muito pouco (FACTOR *et al.*, 2007a).

A busca por dados e metadados é de especial interesse nesta tese dado o fato de que grande parte dos dados é composta de Big Data. Dado um grande volume de dados, muitas vezes não estruturados e não indexados, a busca por uma determinada informação pode ser custosa do ponto de vista computacional.

Ainda dentro do campo da recuperação de dados preservados, duas das principais aplicações da preservação são para fins de auditoria legal e para fins de

1 Redundant Array of Independent Disks. <https://searchstorage.techtarget.com/definition/RAID>

investigação judicial ou forenses. De acordo com Tallon (2013), o valor de dados preservados nesses casos "pode aumentar temporariamente não porque os dados ajudam a melhorar a tomada de decisão organizacional, mas porque não produzir os dados poderia desencadear em danos legais". Neste caso, a recuperação da informação arquivada é comumente conhecida como *E-Discovery*.

Desta forma, identifica-se que uma proposta de arquitetura de preservação a longo prazo deve possuir ao menos um componente focado na reobtenção de dados que foram preservados, a fim de garantir que as informações arquivadas e preservadas sejam encontradas adequadamente mediante um mecanismo de busca.

3.3 FORMATO DE PRESERVAÇÃO DE DADOS: SIRF

O formato SIRF desenvolvido pela SNIA foi criado com o propósito de servir como um formato autocontido para a preservação a longo prazo de dados. Trata-se de um formato que pode ser representado através de linguagens de marcação simples, como o XML ou o JSON, e que define um conjunto de metadados projetados com o objetivo de garantir a preservação de dados a muito longo prazo.

Segundo SNIA (2014), o custo para manter a integridade de dados é significativo, tanto em termos de esforço quanto em termos de custo financeiro. A utilização de contêineres de armazenamento como o SIRF adiciona um pequeno custo adicional enquanto aumenta consideravelmente a sustentabilidade dos dados.

O objetivo do formato SIRF é servir como um formato livre, independente de fabricante, já que deve funcionar com qualquer modelo de dispositivo de armazenamento; independente de mídia, pois deve funcionar igualmente com mídia magnética, de estado sólido, nuvem ou qualquer mídia que venha a surgir no futuro; e autocontido, pois não deve depender de nenhum software para ser lido. Devido a essas características, um documento em formato SIRF é facilmente interpretável por sistemas de preservação no futuro, reduzindo assim os custos associados à preservação digital.

Define-se um contêiner de armazenamento como um dispositivo, partição, volume lógico, diretório, sistema de arquivos ou contêiner de armazenamento na nuvem, ou seja, um pedaço ou o todo de uma mídia de armazenamento físico que esteja disponível para armazenar dados. Um contêiner SIRF é um contêiner de armazenamento dedicado exclusivamente a armazenar objetos SIRF.

O elemento central do SIRF é o *catálogo*, de modo que cada contêiner SIRF possui um catálogo, além de um *objeto mágico* com metadados e *POs* (*Preservation Objects*, objetos de preservação). De acordo com Rabinovici-Cohen *et al.* (2011), um objeto de preservação é "um objeto de informação digital que inclui os dados brutos a serem preservados mais metadados adicionais necessários para permitir a sustentabilidade da informação codificada nos dados brutos por décadas. O objeto de preservação é a unidade básica no armazenamento de um sistema de preservação". O catálogo contém metadados de preservação para cada PO e para todo o contêiner. Normalmente os POs são arquivos (binários ou de texto), o catálogo é um arquivo JSON e o objeto mágico é um arquivo JSON ou um conjunto de pares chave-valor nos metadados do contêiner de armazenamento subjacente, por exemplo, metadados do contêiner do OpenStack Swift (OPENSTACK, 2015).

O catálogo inclui todos os metadados relacionados com o contêiner e os metadados relativos a cada PO contido no contêiner. Ambos os tipos de metadados são organizados em categorias.

Os metadados do contêiner incluem os seguintes elementos:

Specification (especificação): inclui informações sobre a especificação SIRF usada no contêiner.

Container ID (identificador do contêiner): o identificador único do contêiner, que pode estar associado ao identificador único do armazenamento, por exemplo, o ID da fita ou o ID do contêiner na nuvem.

State (estado): o estado do contêiner indica o progresso atual das atividades. Por exemplo, se um contêiner está sendo migrado para outro dispositivo de armazenamento, se está sendo inicializado ou se está pronto para receber novos POs. A Tabela 2 mostra os possíveis tipos e valores do estado de um contêiner.

Tabela 2. Tipos e valores de estado

Tipo	Valores	Descrição
INITIALIZING	TRUE	<u>TRUE</u> quando o contêiner está sendo inicializado
READY	TRUE FALSE	<u>TRUE</u> quando o contêiner está pronto para receber novos POs ou alterações de catálogo. <u>FALSE</u> quando há uma falha que não pode ser especificada por outro tipo de estado
MIGRATING	TRUE	<u>TRUE</u> quando os dados do contêiner estão sendo movidos de/para outro contêiner
DESTROYED	TRUE	<u>TRUE</u> quando o contêiner está fechado e não pode receber mais novos POs

Provenance (proveniência): a proveniência é um conjunto de metadados descrevendo o histórico de informações de um contêiner SIRF. Esse metadado não possui um formato definido, podendo aceitar diversos valores distintos. É possível ter mais de um metadado de proveniência.

Audit Log (log de auditoria): o log de auditoria contém informações sobre como o contêiner foi acessado e modificado. É importante ressaltar que há o log de auditoria do contêiner e também os logs de auditoria dos objetos de preservação. O conteúdo dos logs de auditoria depende da implementação e do domínio que está sendo preservado. Desta forma, contêineres usados para preservação de dados médicos devem, por exemplo, conter logs de auditoria específicos para a área médica.

Os metadados para cada PO incluem as seguintes categorias:

Object IDs (identificadores do objeto): cada PO possui um ou mais identificadores, que podem ser de quatro tipos diferentes. Os identificadores são utilizados para identificar um PO e referenciá-lo em outros POs.

Dentre os identificadores há um identificador universal, que deve ser único não apenas no contêiner, mas em qualquer contexto (portanto, um identificador único e universal, ou UUID). O formato SIRF permite a redundância de alguns identificadores e também possibilita que se registre as mudanças e versões do PO, desta forma auxiliando a preservação a longo prazo dos dados.

Os tipos de identificadores são:

- Nome do objeto: é o nome (ou um dos nomes) do objeto de preservação, e serve para identificar o objeto de maneira complementar, não servindo como um identificador único ou universal. Pode ser, por exemplo, o nome do arquivo ou um nome que identifique o que o arquivo contém de maneira legível para um ser humano. É um campo opcional e pode ser repetido.
- Identificador lógico do objeto: em diversas ocasiões um objeto de preservação precisa ser versionado devido a alterações. O identificador lógico é o identificador único de todas as versões originárias de um mesmo objeto de preservação original. É um campo obrigatório e não pode ser repetido.
- Identificador da versão do objeto: é o identificador (ou são os identificadores) único e universal da versão do objeto. Cada identificador lógico deve possuir um ou mais identificadores de versão, portanto este campo é obrigatório. Tipicamente o identificador de versão é composto pelo identificador lógico mais um número que identifica a versão do objeto.
- Identificador parental do objeto: quando um objeto é derivado de outro objeto de preservação, deve-se identificar o objeto do qual se derivou. Este elemento é opcional, mas caso exista deve ser único.

Related objects (objetos relacionados): esse metadado permite que se referencie um outro PO através do seu UUID. Pode-se referenciar tanto POs internos quanto externos.

Dates (datas): garantem que se mantenham registrados a data e horário da última modificação, o último acesso e da criação do PO. As datas devem estar de acordo com o padrão ISO 8601:2004 (ISO, 2004).

Packaging Format (formato de empacotamento): é usado para especificar o formato de empacotamento do PO. Tipicamente POs são empacotados em formatos de arquivamento, como por exemplo o AIP do OAIS.

Fixity (fixidez): à medida que o tempo passa num contêiner SIRF é necessário que o conteúdo dos POs seja checado periodicamente para garantir que dados não estão sendo perdidos, por exemplo, devido a um decaimento da mídia física. Esse metadado tem como função armazenar um *checksum* do arquivo e a data em que o mesmo foi obtido.

Audit Log (log de auditoria): assim como o log de auditoria do contêiner, o log de auditoria do PO contém informações sobre como o PO foi acessado e modificado. O conteúdo dos logs de auditoria depende da implementação e do domínio que está sendo preservado. Desta forma, POs usados para preservação de dados médicos devem, por exemplo, conter logs de auditoria específicos para a área médica.

Extension (extensão): trata-se de uma área reservada para informações específicas do domínio ou do contêiner. Cada organização que se utiliza do formato SIRF pode usar esse metadado para processamento de informações extras que não são cobertas por outros metadados do formato.

Na Tabela 3 adicionou-se o formato SIRF à comparação feita na Tabela 1. Nota-se que o SIRF possui um conjunto de características próprio, mas que podem ser encontradas isoladamente em outros formatos de preservação abordados no capítulo sobre o estado da arte. O formato SIRF não possui suporte a Big Data ou gerenciamento de elasticidade.

Tabela 3. Comparação entre SIRF e outros formatos abordados

Carac- terísticas	Estratégia de implementação	Independente de fabricante	Migração para outros arquivos	Spanning	Versionamento de objetos	Autodescrito	Suporte a futuras mídias	Suporte a fita	Escalabilidade	Armazenamento em nuvem	Concebida independente de domínio	Suporte a Big Data	Preservação	Gerenciamento de elasticidade
Propostas														
OpenAXF	X	X	X	X		X	X	X	X	X			X	
PREMIS		X			X	X							X	
OAIS		X	X			X					X		X	
SIRF	X	X	X		X	X	X	X		X	X		X	

3.4 ARQUITETURAS DE BIG DATA

Os sistemas de Big Data são um dos focos principais da tese. Sistemas de Big Data apresentam alguns desafios particulares para a preservação a longo prazo, tais como:

- Os dados podem ser estruturados ou não estruturados, o que dificulta na geração e seleção dos metadados adequados para a preservação.
- O volume de dados pode crescer muito rapidamente, e é necessário varrer, processar e catalogar esses dados em uma velocidade suficiente para garantir que dados não serão perdidos.
- Muitas vezes os dados em sistemas de Big Data estão distribuídos em diversos servidores, o que traz desafios no tocante à catalogação e recuperação dos dados preservados.

- No caso de alguns sistemas de *streaming* de dados, os dados podem existir apenas por uma fração de tempo até que sejam descartados, o que exige um desempenho maior na catalogação e preservação dos dados.

Dadas essas características, nota-se que numa arquitetura de preservação a longo prazo de Big Data é necessária a definição e o projeto de um componente com adequada capacidade de processamento e desempenho, capaz de obter os dados de sistemas de Big Data e preservá-los em tempo suficiente para garantir que não haverá perdas qualitativas ou quantitativas nos dados.

Foram estudados sistemas de dados estruturados e não estruturados que podem necessitar de preservação de dados. Particularmente alguns tipos de sistemas de Big Data foram identificados e portanto foram objetos de estudo da tese:

- Sistemas gerenciadores de bancos de dados relacionais – SQL
- Sistemas gerenciadores de bancos de dados não relacionais – NoSQL²
- Sistemas de arquivos (distribuídos ou não)
- Sistemas de *streaming* de dados em tempo real

3.4.1 Ferramentas de processamento e análise de Big Data

Ferramentas de processamento e armazenamento de dados em larga escala são componentes típicos de arquiteturas de Big Data. O Apache Hadoop³ e Apache Spark⁴ são exemplos de *frameworks* utilizados com frequência para o processamento paralelo em larga escala de Big Data, já que ambos utilizam a tecnologia de MapReduce.

Tipicamente, tecnologias MapReduce (mapeie-reduza) são utilizadas para processamento de Big Data. Dado um conjunto de dados não-estruturados

² <https://www.mongodb.com/nosql-explained>

³ <https://hadoop.apache.org/>

⁴ <https://spark.apache.org/>

(tipicamente texto puro extraído de fontes heterogêneas), sem metadados capazes de explicar as características ou semântica dos dados, uma tecnologia MapReduce irá mapear palavras-chave para um conjunto de pares <chave,valor> e então reduzir os pares que contêm a mesma chave. Uma vez que os dados são mapeados e reduzidos, pode-se aplicar outras operações como ordenamento, geração de grafos, aprendizado de máquina, entre outros.

A Figura 5 representa um exemplo de mapeamento e redução. Tem-se primeiramente (à esquerda) um conjunto de dados não estruturados em formato de texto puro. Os caracteres X, Y e Z representam palavras no texto, posteriormente usadas como palavras-chave. Primeiramente os dados são divididos em pedaços de tamanho aproximadamente igual (de maneira que possam ser distribuídos em um conjunto de servidores capazes de fazer mapeamentos e reduções). Em sequência ocorre o mapeamento, como indicado na figura, onde os dados são processados e então transformados em pares chave-valor. No exemplo da Figura 5 os pares chave-valor possuem como chave a palavra-chave vinda dos dados estruturados e como valor o número de ocorrências no pedaço do texto analisado. Nota-se, por exemplo, que o primeiro pedaço contendo “XXY” é mapeado para dois pares chave-valor (X,2) e (Y,1), indicando que a palavra-chave X teve duas ocorrências e a palavra-chave Y teve uma ocorrência no pedaço analisado. Vale salientar que a contagem de incidências de uma palavra-chave no texto é apenas um dos vários mapeamentos possíveis. Uma vez que o mapeamento é realizado para todos os pedaços, é feita a redução, que é a combinação de todos os mapeamentos de acordo com a chave. No exemplo, para a chave X houve três mapeamentos: (X,2), (X,1) e (X,1), que foram reduzidos como (X,5). Finalmente, as reduções são combinadas para gerar um resultado final, ou seja, um conjunto de pares chave-valor com a soma das ocorrências das palavras-chave.

O Apache Spark e o Apache Hadoop são tecnologias diferentes que suportam o processamento paralelo de tarefas de MapReduce. A tecnologia Hadoop é uma das pioneiras em MapReduce e portanto mais antiga, e possui suporte a processamento em um passo único com mapeamento e redução. Computações de múltiplos passos devem ser encadeadas em uma série de tarefas de MapReduce e realizadas em sequência, com o resultado de cada passo sendo armazenado no sistema de

arquivos do Hadoop, o HDFS. Já o Spark possui suporte nativo a computações de

Figura 5. Mapeamento e redução

Fonte: autor

múltiplos passos através da utilização de grafos acíclicos direcionados, ou *directed acyclic graphs* (DAGs). Também é possível utilizar o sistema de arquivos do Hadoop (HDFS) com o Spark.

O Spark também possui suporte a *streaming* de dados em tempo real, além de APIs em diversas linguagens como Scala⁵, Java e Python. O streaming de dados funciona através de uma conexão constante a uma fonte de dados em que as informações são obtidas em tempo real e portanto constantemente atualizadas. A palavra inglesa *stream* significa *corrente* (como uma corrente elétrica ou a corrente de um rio). Isto significa que o mecanismo de entrega dos dados é um fluxo contínuo, virtualmente sem a necessidade de encerramento da conexão com a fonte de dados. Os dados são constantemente processados à medida que são recebidos. Uma aplicação típica de *streaming* de dados é a análise de tendências de opinião em redes sociais. Dado um determinado assunto, abre-se uma *stream* de dados vindos de redes sociais e monitora-se constantemente as postagens que possuem uma determinada palavra-chave relacionada ao assunto, muitas vezes fazendo-se análise do contexto em que as palavras-chave são mencionadas utilizando-se ferramentas analíticas. A

5 <https://www.scala-lang.org/>

combinação de *streaming* de dados com MapReduce é um workload encontrado em sistemas de Big Data.

3.4.2 Dados não-relacionais (NoSQL)

Bancos de dados NoSQL (não-SQL) são bancos de dados que não estão em conformidade com o modelo relacional. O termo NoSQL é comumente usado para identificar diversas tecnologias não-relacionais diferentes, e não necessariamente significa sempre a mesma abordagem para armazenamento de dados.

Em arquiteturas de Big Data é comum que o modelo de dados mude com o tempo. Não apenas dados são atualizados como também sua estrutura: novos campos são adicionados ou removidos, relacionamento entre diferentes campos mudam, coleções de dados ou tabelas precisam ser renomeados. A manutenção desse tipo de estrutura através de esquemas de dados relacionais é custosa, envolvendo *patches* e adaptações. Por este motivo, sistemas do tipo NoSQL tornaram-se uma constante em sistemas de Big Data, o que não invalida a utilização de sistemas relacionais típicos com SQL – trata-se apenas de uma aplicação diferente, com vantagens ou desvantagens de acordo com o cenário.

Dentro da área de bancos de dados NoSQL, sistemas de bancos de dados do tipo objeto são frequentemente utilizados. Sistemas de bancos de dados que suportam objetos possuem diversas vantagens em comparação a sistemas de bancos de dados relacionais:

- A transformação entre objetos de linguagens de programação (Java, C++, Scala, entre outras) e objetos do banco de dados (também chamado de mapeamento objeto-relacional, ou *object relational mapping* – ORM) é facilitada e muitas vezes trivial.
- A obtenção dos objetos (*querying*) é feita também com base em objetos e métodos, o que torna a obtenção de dados mais alinhada com o *workload*.

- Permite que informações sobre um objeto sejam armazenadas em um único documento, ao invés de diversas tabelas e relacionamentos entre suas colunas.
- Não há necessidade de normalização de dados.

3.4.3 Dados relacionais e Sistemas de arquivos

Além dos sistemas típicos de uma arquitetura de Big Data mencionados acima, sistemas de Big Data compreendem também bancos de dados relacionais e sistemas de arquivos.

Bancos de dados relacionais permitem que a arquitetura compreenda não apenas dados não-estruturados, como também dados estruturados que podem ser facilmente obtidos através de SQL e álgebra relacional. Os bancos de dados relacionais são incluídos aqui para uma maior abrangência dos tipos de dados suportados pela arquitetura.

Sistemas de arquivos são um caso de dados não-estruturados também existentes na era pré-Big Data e aumentam a abrangência da arquitetura de preservação aqui proposta.

Por tratarem-se de sistemas de armazenamento altamente conhecidos e utilizados, os bancos de dados relacionais e sistemas de arquivos não são discutidos em grande detalhe nesta tese, embora tenham feito parte da aplicação experimental e a arquitetura proposta contemple essas modalidades de armazenamento.

3.5 MÉTODO PARA CRIAÇÃO DE ARQUITETURAS

De acordo com o Rational Unified Process (KRUCHTEN, 2000), uma arquitetura de referência é, "em essência, um padrão arquitetural pré-definido, ou conjunto de

padrões, possivelmente parcial ou totalmente instanciado, projetado, e comprovado para uso em determinados contextos de negócios e técnicos, em conjunto com artefatos de suporte para permitir a sua utilização. Muitas vezes esses artefatos são resultantes de projetos anteriores”. Como dito em Viana e Sato (2014), a presente proposta de arquitetura está alinhada com este conceito, uma vez que é um conjunto de padrões para a criação de uma arquitetura concreta para preservação de longo prazo e recuperação de Big Data, podendo ser instanciada no mundo real através de uma arquitetura concreta ou protótipo, e contém artefatos de suporte que permitem a sua utilização.

O projeto da arquitetura utiliza como diretriz o trabalho de Angelov e Greefhorst (2012), que propõem um framework para o projeto de arquiteturas de referência, onde cada arquitetura compreende três dimensões:

- Objetivos da arquitetura
- Design e especificação da arquitetura
- Contexto do design e contexto pretendido de aplicação

Escolheu-se a proposta acima mencionada pois ela pode ser usada como uma ferramenta para o projeto de arquiteturas novas, e não apenas para a avaliação de arquiteturas existentes.

Segundo Angelov e Greefhorst (2012), uma arquitetura de referência pode ser projetada sob o escopo de uma *única organização* ou de *múltiplas organizações* que possuem alguma(s) característica(s) em comum. No segundo caso, tais organizações podem ser organizações *de software* que projetam ou aplicam a arquitetura de referência, *de usuários* do software que está baseado na arquitetura de referência ou *organizações independentes* que não são responsáveis por implementar arquiteturas de referência ou usam soluções que utilizam arquiteturas de referência.

Outra questão importante no *framework* é o momento em que a arquitetura é definida. Uma arquitetura de referência *preliminar* é aquela que é definida quando a tecnologia, as soluções de software ou algoritmos necessários para sua

implementação ainda não existem na prática no momento do seu design, enquanto no caso de uma arquitetura de referência *clássica*, a tecnologia, software e algoritmos necessários para a aplicação da arquitetura existem no momento de seu design e foram testados na prática (ANGELOV E GREEFHORST, 2012).

Ainda, o objetivo de uma arquitetura pode ser o de *padronização* de arquiteturas concretas, com foco em interoperabilidade, ou de *facilitação* do design de arquiteturas concretas, com foco no provimento de orientações e inspiração para o design de novos sistemas.

Dadas as dimensões e os parâmetros definidos acima, as arquiteturas de referência são separadas em cinco tipos:

- Tipo 1: arquiteturas clássicas, de padronização, servindo múltiplas organizações, com usuários e empresas de software envolvidos na definição da arquitetura.
- Tipo 2: arquiteturas clássicas, de padronização, servindo uma única organização, geralmente projetadas para servir como uma ferramenta de padronização para o design de um conjunto de soluções de software dentro da organização.
- Tipo 3: arquiteturas clássicas, de facilitação, servindo múltiplas organizações, projetadas por organizações independentes. A organização pode ser um ou mais centros de pesquisa ou ONGs.
- Tipo 4: arquiteturas de referência clássicas, de facilitação, servindo uma única organização. São semelhantes às arquiteturas do tipo 2, porém possuem o objetivo apenas de servir como uma ferramenta de orientação no design e implementação dos sistemas da organização.
- Tipo 5: arquiteturas preliminares, de facilitação, servindo múltiplas organizações. São projetadas para facilitar a criação de arquiteturas e sistemas que serão necessários no futuro. Por serem arquiteturas

preliminares, tipicamente um centro de pesquisa orientado a inovação é o elaborador do design da arquitetura, no entanto, organizações representando grupos de usuários e empresas de software devem ser envolvidas no processo de design. Tais arquiteturas são inovadoras em sua natureza (ANGELOV E GREEFHORST, 2012) e devem definir os *componentes* necessários para que a arquitetura seja implementada, *algoritmos* que podem ser usados para dar suporte à operação dos componentes e *protocolos* para demonstrar as interações entre os componentes.

Ainda em Angelov e Greefhorst (2012), uma arquitetura é chamada *congruente* "se seus objetivos são relevantes para o contexto da arquitetura de referência e seu design reflete adequadamente tanto o contexto quanto os objetivos da arquitetura, ou seja, é necessário haver congruência entre o contexto, objetivos e concepção de uma arquitetura de referência".

Os autores propuseram um método para a geração de uma arquitetura de referência e definiram as correlações entre as três dimensões. De acordo com o processo de verificação dessas correlações, a arquitetura é considerada congruente, ou o designer da arquitetura é levado a um passo anterior para redefinir os objetivos ou o contexto da arquitetura.

3.6 ARQUITETURAS RELACIONADAS EXISTENTES

De início foram encontradas diversas propostas de arquitetura de preservação de dados estruturados (BAKER, KEETON E MARTIN, 2005; NGUYEN, LAKE E HUBER, 2010; STRICOT, 2013; HAN, SUN E WANG, 2013; WALLACE E PORDESH, 2013). Duas das maiores deficiências de todas essas propostas são: 1) a abordagem de preservação compreende apenas a preservação física dos objetos preservados e, portanto, os objetos ainda estão sujeitos à obsolescência de formatos e softwares; e 2) o suporte à preservação é feito da maneira tradicional, englobando apenas dados estruturados.

A pesquisa de Factor *et al.* (2007) introduz os conceitos de preservação física e lógica e propõe uma arquitetura capaz de realizar preservação lógica de dados. Tal pesquisa apresenta uma inovação, a preservação lógica, o que garante uma menor chance de obsolescência dos dados preservados em relação às outras propostas, que compreendem apenas a preservação física dos dados.

Mesmo considerando a preservação lógica, grande parte das arquiteturas de preservação a longo prazo não possuem um formato padronizado de preservação, muitas vezes utilizando algum formato da área de arquivamento (como o AIP do OAIS), que geralmente não é suficiente para a preservação a longo prazo de dados. Rabinovici-Cohen *et al.* (2011) resolvem este problema ao introduzir o conceito de objeto de preservação no formato SIRF. Também Weiss (2014) propõe o formato OpenAXF que possui características de preservação em seus objetos AXF, focado no domínio da indústria cinematográfica.

Rabinovici-Cohen *et al.* (2013) propõem o armazenamento em nuvem como uma das formas de preservação suportadas pelo SIRF alguns anos após a proposta original do formato. O armazenamento em nuvem também é suportado por Weiss (2014). Em Nguyen e Lake (2011) também pode ser encontrada uma arquitetura de preservação a longo prazo na nuvem, apresentando o conceito de um LDPaaS (*Long-term Digital Preservation as a Service*, ou preservação digital de longo prazo como serviço). Nesta proposta os autores apontam que um sistema de preservação a longo prazo na nuvem “poderia oferecer elasticidade no provisionamento de capacidade computacional e de armazenamento”, de maneira que o consumidor do serviço não precisa se preocupar com o investimento em infraestrutura. Já Thibordeau e Zangaro (2013) definem o chamado “*Cloud Digital Archive Service*”, que é um serviço de armazenamento on-line baseado em nuvem usada para arquivamento, mas não focados em preservação a longo prazo.

No tocante a Big Data, alguns estudos (BEATH *et al.*, 2012; TALLON, 2013; MERRILL, 2011; ALNAFOOSI e STEINBACH, 2013; REICHMAN, 2011) abordam os aspectos econômicos de arquiteturas de Big Data e seu armazenamento. Há também propostas de arquiteturas de referência de Big Data (GEERDINK, 2013; DOSHI *et al.*, 2013).

Também encontrou-se arquiteturas de referência para a preservação a longo prazo de dados estruturados (HUHNLEIN *et al.*, 2009; BUTLER *et al.*, 2009; EDRM, 2014).

Lee *et al.* (2011) incluem a aprendizagem de máquina e busca de evidências relacionadas com as questões de litígio, usando o modelo de descoberta eletrônica de referência (EDRM, 2014) como base e, desta forma, apresentam propostas para a recuperação de dados em sistemas de preservação. Butler *et al.* (2009) utilizaram uma ontologia para representar os dados para o *E-Discovery*, criando assim um middleware semântico, capaz de abstrair fontes de dados heterogêneos, tais como bancos de dados, repositórios e diretórios LDAP.

3.7 COMPUTAÇÃO EM NUVEM, ESCALABILIDADE, EFICIÊNCIA E ELASTICIDADE

Nesta tese seguiu-se a definição de computação em nuvem da NIST (2011): “computação em nuvem é um modelo para permitir acesso ubíquo, conveniente e sob demanda via rede a um conjunto de recursos computacionais configuráveis (e.g. redes, servidores, armazenamento, aplicações e serviços) que podem ser rapidamente provisionados e disponibilizados com esforço mínimo de gerenciamento ou interação do provedor de serviço”.

A computação em nuvem foi estudada com o objetivo de ter um mecanismo na arquitetura de preservação para expandir a capacidade de processamento e de armazenamento dado um workload contínuo e com grande volume de dados. Em outras palavras, a computação em nuvem se mostra uma solução apropriada pois permite que os recursos usados na preservação possam ser escalados. Lehrig, Eikerling e Becker (2015) realizaram uma revisão sistemática da literatura das definições de escalabilidade, eficiência e elasticidade:

- **Capacidade:** o workload máximo que a camada da nuvem pode suportar de acordo com os objetivos do serviço.

- **Escalabilidade:** a habilidade da camada da nuvem de aumentar sua capacidade ao expandir a quantidade de serviços consumidos nas camadas mais baixas.
- **Elasticidade:** o grau em que a camada da nuvem autonomamente adapta a capacidade ao *workload* com o tempo.
- **Eficiência:** medida que relaciona a capacidade demandada com os serviços consumidos com o tempo.

Ainda, Herbst, Kounev e Reussner (2013) apontam que a escalabilidade é um pré-requisito para elasticidade, porém não considera aspectos temporais de quão rápido, quão frequente e com qual granularidade as ações de escalabilidade são realizadas. NIST (2011) aborda esse aspecto temporal ao definir rápida elasticidade como uma característica da computação em nuvem:

- **Rápida elasticidade:** permite que serviços na nuvem possam diminuir ou aumentar sua capacidade (*scale in* e *scale out*) dependendo da demanda do consumidor, às vezes automaticamente através de um sistema de gerenciamento de elasticidade.

Em um dado momento, como definido por Herbst, Kounev e Reussner (2013), um recurso de um sistema na nuvem pode estar em um dos três estados de elasticidade:

- Ótimo: a quantidade de recursos provisionados corresponde à demanda atual.
- Subprovisionado: a quantidade de recursos provisionados não é suficiente para suprir a demanda.
- Superprovisionado: a quantidade de recursos provisionados é maior do que a demanda por recursos.

Um sistema pode passar de um estado a outro em tempo de execução de acordo com as ações tomadas relativas à elasticidade. Particularmente, como mostrado na Figura 6, um sistema pode passar do estado superprovisionado para o estado ótimo através da desalocação de recursos inutilizados, podendo também passar do estado ótimo para o estado superprovisionado caso ocorra alocação de recursos que não serão utilizados. É possível também que um sistema em estado subprovisionado passe para o estado ótimo através da alocação de recursos que são necessários, e do estado ótimo para o estado subprovisionado caso recursos necessários sejam desalocados.

Figura 6. Diagrama de estados de elasticidade

Fonte: autor

Toeroe, Pawar e Khendek (2014) propõem um mecanismo de elasticidade, responsável por reagir a variações de workload, bem como monitores de workload para detectar flutuações de workload nos diversos componentes de um serviço. Tal mecanismo é essencial para um sistema que está recebendo um fluxo contínuo de dados, principalmente se há possibilidade de um volume crescente, pois permite que seja aplicada rápida elasticidade, como definida por NIST (2011), de maneira que não haja deterioração de desempenho ou mesmo falta de recursos de armazenamento ou computacionais em tempo de execução.

3.8 CONSIDERAÇÕES SOBRE OS TRABALHOS EXISTENTES APRESENTADOS

As arquiteturas preexistentes relacionadas com o tema desta tese foram de fundamental importância por abrirem caminho para que a proposta de arquitetura de preservação a longo prazo de Big Data pudesse ser elaborada. Através da extensa pesquisa e análise dos trabalhos existentes teve-se acesso aos principais desafios da área, bem como de suas soluções.

Por outro lado, dado o objetivo da tese em resolver o problema específico da preservação de Big Data e não apenas o arquivamento de Big Data ou a preservação de dados estruturados, surgem diversos desafios que exigem abordagens diferentes e, portanto, um trabalho focado em outros aspectos que não os das pesquisas anteriores. Nguyen e Lake (2011), por exemplo, sugerem que um sistema de preservação a longo prazo na nuvem “poderia oferecer elasticidade no provisionamento de capacidade computacional e de armazenamento”, sem no entanto implementar mecanismos que possibilitam tal elasticidade, desta maneira deixando o problema em aberto. As propostas de escalabilidade e elasticidade apresentadas anteriormente são capazes de propor certo nível de elasticidade no armazenamento, por exemplo, mas não são suficientes para garantir a elasticidade no processamento, obtenção e arquivamento dos dados, sendo necessária uma estratégia de elasticidade focada em preservação.

Particularmente nota-se que as principais pesquisas da área de preservação resolvem apenas o problema da preservação física, negligenciando os aspectos de preservação lógica. Mesmo as propostas que de alguma forma englobavam a preservação lógica pecavam, em sua maioria, por não possuírem um formato de metadados específico à preservação, tomando emprestados formatos da área de arquivamento, o que faz com que a preservação fique incompleta.

No caso de propostas que compreendem a preservação lógica e física, como no caso do OpenAXF (WEISS, 2014) e SIRF (RABINOVICI-COHEN *et al.*, 2013) bem como formatos específicos à preservação e não apenas ao arquivamento, em todos os casos não havia mecanismo, componente ou algoritmo que previsse a elasticidade aliada à preservação, e portanto as propostas tornam-se incompatíveis

com Big Data, já que o workload traria um grande desafio dos pontos de vista de gerenciamento de recursos, desempenho de busca e desempenho de arquivamento.

A proposta encontrada em Rabinovici-Cohen *et al.* (2013) é a que mais se aproxima da presente tese, dado que apresenta algumas das principais características necessárias para uma arquitetura de preservação a longo prazo de Big Data:

- Suporte a preservação física
- Suporte a preservação lógica
- Independente de domínio
- Utilização de um formato de metadados específico à preservação (SIRF)
- Suporte a mídias heterogêneas, inclusive armazenamento em nuvem

Desta forma na proposta de Rabinovici-Cohen *et al.* (2013) encontrou-se um ponto de partida interessante, onde pode-se adicionar aspectos e mecanismos específicos a sistemas de Big Data para a geração de uma arquitetura de preservação a longo prazo de Big Data. Trata-se de uma das pesquisas principais sobre a qual se debruça a tese.

Toeroe, Pawar e Khendek (2014) abrem caminho para que nesta tese se pudesse estender o formato SIRF, dado o mecanismo de elasticidade proposto que reage a variações de *workload*, formado por monitores que detectam flutuações de *workload* nos diversos componentes e um atuador de elasticidade responsável por estender os recursos ou provisionar novos recursos de maneira autônoma. Através da junção das duas abordagens, uma de preservação (RABINOVICI-COHEN *et al.*, 2013) e outra de elasticidade (TOEROE, PAWAR e KHENDEK, 2014), com foco em preservação de Big Data, propõe-se nesta tese uma solução para o problema de preservação de um crescente volume de dados de natureza estruturada e não-estruturada. Por fim, para que a arquitetura definida pudesse ser reproduzida, utilizou-se o método de criação de arquiteturas de referência proposto por Angelov e Greefhorst (2012).

4 PROPOSTA DE ARQUITETURA DE PRESERVAÇÃO A LONGO PRAZO EM NUVEM COM ELASTICIDADE

Neste capítulo é apresentada a proposta de arquitetura de preservação a longo prazo de Big Data com elasticidade.

Dado o objetivo da tese, o da criação de uma arquitetura para a preservação a longo prazo de Big Data que servirá de base para futuras arquiteturas, estudou-se a criação de arquiteturas de referência. Embora a criação de uma arquitetura de referência esteja além do escopo da presente tese, a utilização de um método de criação de arquiteturas de referência garante que a proposta de arquitetura não tenha sido *ad hoc*, ou seja, que seguiu-se um método para a adequada elaboração de uma arquitetura focada em reuso. Entende-se que a proposta de arquitetura da presente tese possui diversas das características de uma arquitetura de referência, tais como:

- Foi criada de maneira suficientemente abstrata para permitir o uso de apenas alguns dos componentes propostos.
- Foi criada de maneira suficientemente abstrata para permitir o uso de tipos diferentes de componentes – por exemplo, os dispositivos de armazenamento podem ser magnéticos ou de estado sólido, ou em nuvem, ou uma combinação desses.
- Foi criada e documentada com foco no reuso.
- Tem como objetivo a sua reprodução para criação de arquiteturas concretas.
- Seguiu um método de criação de uma arquitetura de referência.

A proposta não trata de uma arquitetura de referência em si, por entender-se que a elaboração de uma arquitetura de referência é um trabalho mais extenso e que deve

ser desenvolvido por diversos pesquisadores durante vários anos, levando em consideração diversos projetos existentes que se relacionam com a tese. Desta forma, a proposta de uma arquitetura de referência fica fora do escopo e prazo da presente tese.

A arquitetura proposta é reprodutível e dá suporte à utilização de Big Data e sua preservação a longo prazo. Especificamente, uma das principais características da arquitetura é a elasticidade do armazenamento e da capacidade computacional. À medida que o fluxo de Big Data aumenta a arquitetura é capaz de aumentar seu armazenamento e o conjunto de servidores processando os dados. É utilizado o formato SIRF, que é uma especificação aceita pela indústria de preservação de dados.

4.1 CONCEPÇÃO DA PROPOSTA DE ARQUITETURA

A arquitetura proposta foi criada a partir da instanciação do método de criação de arquiteturas de referência proposto por Angelov e Greefhorst (2012). O método proposto aponta que deve-se gerar uma arquitetura definindo-se três pontos principais: o objetivo da arquitetura, o contexto de aplicação e os aspectos temporais da arquitetura. O objetivo da arquitetura cobre a chamada “dimensão de objetivos”, enquanto o contexto de aplicação e os aspectos temporais cobrem a “dimensão do contexto pretendido de aplicação”.

Em primeiro lugar, definiu-se o objetivo da arquitetura: servir como base para arquiteturas concretas com foco na preservação de longo prazo e recuperação de dados estruturados e não estruturados, capaz de alocar e desalocar recursos dinamicamente em tempo de execução. Além da definição do objetivo da arquitetura, foram identificados princípios importantes a que a arquitetura deve obedecer:

- A arquitetura proposta não deve ser específica a nenhum domínio.
- A arquitetura proposta deve dar suporte ao requisito da elasticidade.

- A arquitetura deve suportar bancos de dados relacionais (dados estruturados) e bases de dados NoSQL, bem como fluxos de dados em tempo real, ou *streaming* (dados não estruturados).
- A arquitetura deve ser independente de mídia e, portanto, tem de suportar qualquer tipo de dispositivo de armazenamento, incluindo a nuvem.

Em segundo lugar definiu-se o contexto de aplicação:

- A arquitetura deve servir várias organizações, em vez de uma única organização, uma vez que o objetivo da tese é fornecer uma arquitetura genérica que possa ser utilizada por profissionais que querem criar suas próprias arquiteturas de preservação a longo prazo para Big Data.
- Definiu-se as organizações de software como os principais interessados na proposta de arquitetura, e organizações de usuários como os atores secundários.

Em terceiro lugar, definiu-se os aspectos temporais que podem ter impacto sobre os objetivos e o design da arquitetura proposta:

- A arquitetura proposta é considerada como tendo características tanto de uma arquitetura preliminar quanto de uma arquitetura clássica.
- As tecnologias, soluções de software e algoritmos exigidos para a aplicação da arquitetura existiam separadamente no momento de sua concepção, no entanto, após extensiva busca por publicações relacionadas, não foram encontradas arquiteturas focadas na preservação a longo prazo de Big Data com elasticidade que tenha sido implementada ou proposta com os mesmos objetivos que propôs-se na presente tese.
- Existiam arquiteturas voltadas para a preservação a longo prazo de grandes quantidades de dados, mas nenhuma delas aborda especificamente a

preservação de dados de dados não estruturados e de *streaming* como o cerne de suas propostas.

Com base nos objetivos de arquitetura, no contexto e nos seus aspectos temporais, identificou-se a arquitetura como do tipo 5 (ANGELOV E GREEFHORST, 2012). Arquiteturas do tipo 5 são projetadas por pesquisadores para facilitar a criação de arquiteturas de sistemas que se tornarão necessárias no futuro.

Uma vez que se encontrou uma correspondência entre a arquitetura proposta e um dos tipos de arquitetura de referência proposto por Angelov e Greefhorst (2012), conclui-se que a arquitetura proposta é *congruente*.

4.2 VISÃO GERAL DA PROPOSTA DE ARQUITETURA

Na Figura 7 tem-se uma visão geral da arquitetura proposta, que foi projetada em camadas. Na camada inferior (camada de armazenamento) tem-se a mídia onde ficam armazenados os dados e os metadados. Nota-se que diversos tipos de mídia devem ser suportados, como mídia magnética (discos e fitas), discos de estado sólido (SSDs) e também armazenamento em nuvem. A camada de armazenamento é uma camada abstrata: pode conter qualquer tipo de mídia ou ainda uma rede ou arquitetura completa de armazenamento, desde que seja capaz de armazenar os dados vindos das camadas superiores.

Na camada superior (camada de dados) tem-se os sistemas de processamento e gerenciamento de Big Data. Trata-se dos dados que se deseja preservar. Os dados provenientes da camada de dados são processados, encapsulados em um formato de preservação e então gravados na camada de armazenamento. Deve-se ter suporte a diversos tipos de sistemas de gerenciamento de processamento de dados, tais como sistemas gerenciadores de bancos de dados relacionais e não-relacionais, sistemas de arquivos e sistemas de *streaming* (fluxo de dados em tempo real). Na camada de dados encontra-se também o *gateway de arquivamento*, cuja função é obter os dados dos sistemas de gerenciamento e processamento de dados de

maneira automática e enviá-los às camadas inferiores para preservação. O gateway de arquivamento é o componente responsável por fazer a ponte entre Big Data e a preservação, realizando o pré-processamento de dados estruturados e não-estruturados e criando um conjunto de metadados para o conteúdo vindo dos sistemas da camada de dados. A camada de dados é abstrata, ou seja, define-se que ela deve existir e que deve compreender ao menos um dos sistemas de dados aqui referidos, no entanto não há uma recomendação ou especificação concreta de quais sistemas a camada deve conter, a não ser no caso do gateway de arquivamento, que é explicado adiante.

Figura 7. Visão geral da arquitetura proposta

Fonte: autor

A camada de acesso recebe dados do gateway de arquivamento, da interface gráfica ou de serviços web através de uma interface (comandos HTTP GET, POST,

PUT e DELETE). De acordo com o que for recebido nesta camada, os dados são empacotados no formato SIRF e enviados para a camada de arquivamento (inclusão de novos dados), ou a camada de arquivamento é acessada diretamente (exclusão de dados ou reobtenção de dados). Nesta camada encontra-se o componente de empacotamento, responsável por receber metadados das demais camadas e encapsulá-los no formato SIRF.

A camada de arquivamento é onde ocorre a maior parte do processamento de dados do formato SIRF. Essa camada é responsável pela transformação entre o lógico e o físico, e vice-versa. Nesta camada estão presentes componentes para garantir que dados não serão perdidos na camada de armazenamento, que os formatos originais dos POs serão legíveis no futuro, bem como componentes responsáveis por obter POs da camada de armazenamento e devolvê-los as outras camadas ou receber POs de outras camadas e armazená-los em mídia.

No canto inferior direito encontra-se o mecanismo de elasticidade. O mecanismo de elasticidade tem por função receber informações sobre o *workload* nos diversos componentes da arquitetura (tais componentes possuem monitores de *workload* na forma de microsserviços), e, de acordo com uma política de elasticidade, instanciar novos recursos. O controlador de elasticidade tem como função orquestrar o recebimento das métricas, interpretar a política de elasticidade e requerer a alocação ou desalocação de recursos através do atuador de elasticidade. Para acelerar a alocação de novos recursos, o mecanismo de elasticidade conta também com um gerenciador de buffer, responsável por pré-alocar alguns recursos para uma demanda futura e deixá-los em modo de espera.

Mais detalhes sobre cada camada são dados nas seções que se seguem.

4.3 CAMADA DE DADOS

A camada de dados contém os dados originais que se tornarão objetos de preservação. É a fonte de Big Data na arquitetura. Nesta camada estão dados

considerados “quentes”, ou seja, de alta utilização e valor, portanto dados vindo de bancos de dados relacionais, processamento analítico e *data mining*, fluxos de dados em tempo real (*streaming*), dados não estruturados processados por workloads MapReduce, NoSQL, dados presentes em sistemas de arquivos, entre diversas outras possíveis fontes de dados. Os dados desta camada fazem parte das áreas (I), (II) e (III) da Figura 1 devido à sua alta utilização e seu alto valor.

O gateway de arquivamento é um componente responsável por obter automaticamente os dados vindos dos sistemas de dados mencionados anteriormente e invocar a camada de acesso para o devido arquivamento desses dados de acordo com políticas pré-definidas. Por tratarem-se de sistemas Big Data, o desempenho e resiliência do gateway de arquivamento são objetos de especial atenção desta tese, dado que o grande volume de dados pode gerar diversos gargalos de entrada e saída, rede, memória, processamento, entre outros.

Sobre o gateway de arquivamento recai a responsabilidade de varrer os componentes e sistemas Big Data da camada de dados, escolhendo o que será preservado e o que não será preservado. O gateway de arquivamento deve obedecer a alguns requisitos:

- O gateway de arquivamento deve se comportar de acordo com políticas pré-definidas para cada sistema de dados. Por exemplo, para um banco de dados relacional, pode-se definir que o gateway deverá preservar dados a cada número de transações; para sistemas de *streaming* em tempo real, a cada minuto, ou a cada hora, ou a cada dia, etc.
- Os sistemas de dados não devem precisar sofrer alterações em seu código-fonte ou necessitar de recompilação para que seus dados sejam obtidos pelo gateway de arquivamento.
- O gateway de arquivamento deve ser regido por um algoritmo ou servidor que esteja em constante execução, de modo que possa varrer periodicamente os sistemas da camada de dados. Pode ser, por exemplo, um *daemon* UNIX que obtém os dados dos sistemas de processamento de dados periodicamente

através de um servidor FTP e os encaminha para as camadas inferiores; ou ainda um serviço web em *loop* infinito que constantemente realiza uma varredura dos sistemas de arquivos da camada de dados, entre outras possíveis implementações.

A implementação do restante da camada de dados é independente da arquitetura aqui proposta e não há um padrão específico ou diretriz específica de como implementar essa camada. Isso dependerá de características do ambiente, como por exemplo o domínio de aplicação, a organização que está utilizando o sistema de preservação e o grupo de usuários.

4.4 CAMADA DE ACESSO

A camada de acesso é utilizada para que se possa pode-se criar, modificar e deletar instâncias de objetos existentes em um servidor através de uma interface bem definida e padronizada. Os comandos abstratos GET, POST, PUT e DELETE, presentes na figura 7, são responsáveis pela obtenção, atualização, criação e remoção de objetos.

Definiu-se a utilização da camada de acesso como camada de entrada de dados vindos de outras camadas. A camada deve fornecer integração com tecnologias, tais como:

- Interfaces gráficas web
- Serviços web
- Interfaces de linha de comando

A definição da camada de acesso foi inspirada pela tecnologia REST. REST (*Representational State Transfer*, ou Transferência de Estado Representacional) “é um estilo arquitetural que especifica algumas restrições, tais como a interface uniforme, que se aplicada a um serviço web induz propriedades desejáveis, como desempenho, escalabilidade e modificabilidade, que habilita serviços a trabalharem

melhor na web” (ORACLE, 2015). Em uma interface REST, todo o fluxo de dados se dá através do protocolo HTTP. Através de chamadas a um endereço (URI, ou *Uniform Resource Identifier*, composto pelo endereço web mais o endereço do recurso que se deseja acessar), pode-se localizar um objeto e então aplicar um dos métodos (GET, POST, PUT, DELETE) nesse objeto, atingindo o efeito desejado.

A Tabela 4 abaixo mostra a definição dos comandos abstratos GET, POST, PUT, DELETE e seus efeitos quando executados com um identificador de um objeto SIRF.

Tabela 4. Comandos da camada de acesso e operações SIRF

OPERAÇÃO → IDENTIFICADOR ↓	GET	POST	PUT	DELETE
/sirf/container	Listar contêineres SIRF	–	–	–
/sirf/container/{name}	Obter objeto mágico	–	Criar contêiner (+ objeto mágico + catálogo)	Deletar todos objetos de um contêiner
/sirf/container/{name}/catalog	Obter catálogo	–	Atualizar catálogo	–
/sirf/container/{name}/{po}	Obter metadados do objeto de preservação	Criar ou atualizar objeto de preservação	Atualizar informações de objeto de preservação	Deletar metadados de objeto de preservação
/sirf/container/{name}/{po}/data	Obter conteúdo do objeto de preservação (sem metadados)	–	–	Deletar conteúdo e metadados de objeto de preservação

Desta forma, por exemplo, um contêiner de nome “ContainerSIRF” pode ser criado em uma arquitetura sendo executada no endereço www.sirf.usp.br, na porta 8080, simplesmente invocando-se o método PUT com o endereço <http://www.sirf.usp.br:8080/sirf/container/ContainerSIRF>:

PUT <http://www.sirf.usp.br:8080/sirf/container/ContainerSIRF>

Ao receber a requisição PUT, a camada de acesso identifica que se está tentando criar um novo contêiner e que o nome deste novo contêiner será “ContainerSIRF”. Desta forma, cria-se um novo contêiner, um novo objeto mágico e um novo catálogo SIRF na camada de armazenamento.

Para obter o catálogo SIRF do contêiner recém-criado:

GET <http://www.sirf.usp.br:8080/sirf/container/ContainerSIRF/catalog>

Ao receber a requisição GET, a camada de acesso identifica que se está tentando obter o catálogo do contêiner “ContainerSIRF”. Desta forma, obtém-se o catálogo da camada de armazenamento e devolve-se o objeto para o cliente da requisição.

Outros componentes e camadas podem então acessar a camada de acesso para manipular objetos SIRF de maneira padronizada. Isto permite que todas as operações relevantes a um contêiner SIRF possam ser realizadas através de uma interface padronizada, bem conhecida e facilmente integrável com outras interfaces, tais como serviços web ou interfaces web.

O componente de empacotamento SIRF possui a responsabilidade de receber as informações sobre os objetos SIRF (POs, catálogo, objeto mágico, entre outros) vindas da camada de acesso e então empacotar essas informações em formato SIRF propriamente dito, tipicamente XML ou JSON.

Nota-se, no entanto, que embora a camada de acesso tenha sido inspirada em arquiteturas REST, ela é uma camada abstrata e deve dar suporte a futuras implementações de mecanismos de acesso e manipulação de objetos.

4.5 CAMADA DE ARQUIVAMENTO

Na camada de arquivamento ocorrem as ações relacionadas ao arquivamento, preservação e recuperação dos dados que foram preservados. Nesta camada os objetos de preservação que foram empacotados na camada de acesso são tratados e direcionados para a camada de armazenamento, e/ou os objetos de preservação presentes na camada de armazenamento são desempacotados e retornados ao cliente.

A camada de arquivamento é composta dos seguintes componentes:

- Componente de arquivamento: esse componente se conecta à camada de armazenamento para a gravação de dados e à camada de acesso para a obtenção de dados. Os dados recebidos devem estar previamente empacotados em formato SIRF e seguir as políticas definidas de preservação, sendo então escritas em alguma mídia de armazenamento.
- Componente de preservação física: este componente é responsável por garantir a preservação física dos dados. Não há diretrizes para a maneira como a preservação é realizada, portanto trata-se de um componente abstrato. Para atingir o objetivo da preservação física esse componente pode realizar outras ações, como a deduplicação e migração entre as mídias.
- Componente de preservação lógica: o componente de preservação lógica é responsável pela preservação lógica dos dados e pela gestão de metadados. Neste componente devem existir sistemas (máquinas virtuais, emuladores, software, serviços web, entre outros) que são capazes de ler POs que estão armazenados na camada de armazenamento, desta forma garantindo a preservação lógica do conteúdo preservado.
- Componente de reobtenção: este componente é responsável por realizar buscas pelos POs na camada de armazenamento, acessando metadados

presentes no catálogo e recuperando o conteúdo dos POs e/ou seus metadados da camada de armazenamento e retornando-os para a camada de apresentação.

4.6 CAMADA DE ARMAZENAMENTO

Na camada de armazenamento estão presentes os dados e metadados do contêiner SIRF, ou seja, é onde a mídia física se encontra. Dado que a camada de armazenamento é uma camada abstrata, pode-se ter diversos tipos diferentes de mídia, tais como:

- Discos magnéticos
- Fitas magnéticas
- Armazenamento em nuvem
- Discos de estado sólido
- Outros
- Uma combinação de dois ou mais tipos de mídia diferentes

As requisições para inserção de novos POs e recuperação de dados e metadados são em algum dado momento traduzidas em requisições de entrada e saída (I/O) para os componentes de armazenamento desta camada.

4.7 COMPONENTES DE APRESENTAÇÃO E SERVIÇOS WEB

Além das camadas de dados, de acesso, de arquivamento e de armazenamento, na maior parte dos casos os clientes conectam à camada de acesso através de alguma interface, como:

- Interface gráfica web
- Aplicação gráfica

- Interface de linha de comando (por exemplo, o comando cURL)
- Serviços web

Um componente de apresentação permite que um usuário abstraia a complexidade de um contêiner SIRF e suas operações, desta maneira tornando a experiência de preservação mais amigável ao usuário e portanto facilitando o acesso a leigos. Além disso, um serviço web pode conectar-se à camada de acesso de maneira padronizada e, desta forma facilita-se a integração da arquitetura que utiliza SIRF com outros sistemas.

4.8 GERENCIAMENTO DE ELASTICIDADE

Visando o gerenciamento da alocação de recursos em uma arquitetura de preservação de dados, e portanto também o custo associado à preservação, definiu-se que é necessário um componente de gerenciamento de elasticidade. O componente de gerenciamento de elasticidade tem as responsabilidades de monitorar os diversos elementos da arquitetura, dar suporte a políticas de elasticidade, gerenciar um buffer de recursos e atuar no sistema através da alocação ou desalocação de recursos de acordo com o *workload*.

Lehrig, Eikerling e Becker (2015a) dizem que “na computação em nuvem, engenheiros de software projetam sistemas com recursos virtualmente ilimitados pelos quais os provedores de computação em nuvem cobram de acordo com o uso”. Desta forma, com o provisionamento de recursos sendo considerado flexível, é ideal que os consumidores sejam capazes de aumentar ou diminuir os recursos dinamicamente, tendo única e exclusivamente disponíveis (e portanto cobrados) os recursos que estão sendo utilizados em um dado momento.

Toeroe, Pawar e Khendek (2014) propõem um mecanismo de elasticidade, responsável por reagir a variações de *workload*. Utiliza-se monitores de *workloads* cuja responsabilidade é captar métricas dos diversos componentes que podem ser elásticos. As métricas são enviadas frequentemente para um controlador de

elasticidade que decide se ajustes são necessários e aplica as mudanças na arquitetura através de elasticidade. Essa arquitetura de elasticidade inspirou o gerenciamento de elasticidade da tese, como é descrito nas seções que se seguem.

4.8.1 Política de elasticidade

Um elemento central do mecanismo de elasticidade é a política de elasticidade, cuja função é conter configurações de elasticidade, limites de métricas de *workload* e ações de elasticidade para os diversos possíveis estados de elasticidade e recursos.

Após receber as métricas dos diversos monitores de *workload*, o controlador de elasticidade consulta as regras da política de elasticidade para definir se é o momento de elastecer a arquitetura.

Através da política de elasticidade definem-se concretamente os estados ótimo, subprovisionado e superprovisionado. O gerenciador de buffer e o controlador de elasticidade levarão em consideração esses limites para alocar ou desalocar recursos. O estado ótimo pode ser não apenas um estado em que ocorra desperdício zero, mas sim um intervalo de recursos adicionais necessários para acelerar uma futura alocação pelo atuador de elasticidade. Ou seja, o estado ótimo pode não significar necessariamente zero recursos inutilizados, mas sim um ou mais recursos ativos no buffer à espera de uma nova alocação. Também na política de elasticidade define-se o limite de capacidade desejável dos recursos, de maneira que novas alocações de recursos se iniciem antes que os recursos atuais estejam sendo utilizados à exaustão de sua capacidade. Neste caso, o mecanismo de elasticidade pode elastecer o armazenamento, por exemplo, quando a utilização chegar a 80% ou 90% ao invés de 100% (o que poderia causar indisponibilidade ou falhas).

A política de elasticidade tem impacto direto no custo de preservação. Simões e Kamienski (2014) apontam que métricas e limites devem ser cuidadosamente

escolhidos de modo a obter o efeito duplo de prover níveis de desempenho adequados ao mesmo tempo que o custo seja mantido tão baixo quanto possível.

4.8.2 Mecanismo de elasticidade e seus componentes

Como mostrado na Figura 7, a proposta de arquitetura da tese engloba elementos da preservação de dados e também um mecanismo de elasticidade, cuja responsabilidade é obter métricas dos diversos componentes e determinar se o sistema é capaz de suportar o *workload* atual. Essas métricas são obtidas através de diversos monitores de *workload* distribuídos entre os vários componentes da arquitetura. De posse das métricas de *workload*, o mecanismo de elasticidade é capaz de atuar sob o sistema, alocando ou desalocando recursos de acordo com uma política de elasticidade e configurando a arquitetura para que possa acomodar e integrar os novos recursos em tempo de execução. Mais detalhes sobre os elementos do mecanismo de elasticidade são dados a seguir.

4.8.2.1 Monitor de *workload*

Um monitor de *workload* deve ser capaz de detectar flutuações no *workload* nas instâncias do serviço monitorado (TOEROE, PAWAR e KHENDEK, 2014). Trata-se de um software dedicado ao monitoramento, tendo uma interface exclusiva para o envio de métricas para outros componentes da arquitetura. Em uma arquitetura RESTful, pode ser implementado como um microsserviço.

No caso de componentes de armazenamento, por exemplo, um monitor de *workload* tem como responsabilidade captar métricas sobre o armazenamento, como por exemplo, a quantidade de espaço livre e a porcentagem de espaço livre.

O funcionamento de um monitor de *workload* se dá como na Figura 8. O monitor de *workload* deve atuar como um serviço ou *daemon* que fica aguardando requisições

vindas do mecanismo de elasticidade. Quando uma requisição é recebida, o monitor do *workload* deve interagir com o sistema operacional e/ou diversos componentes monitorados para fazer a leitura das métricas. No caso de componentes que realizam algum tipo de processamento, como o componente de empacotamento e o gateway de arquivamento, o monitor de *workload* deve captar métricas sobre CPU e memória RAM, de maneira a garantir que não haja significativa queda no desempenho sem que um novo recurso seja provisionado pelo mecanismo de elasticidade para balancear o *workload*. Já no caso de componentes que realizam muitas requisições de entrada e saída, como é o caso do componente de arquivamento e do gateway de arquivamento, o monitor de *workload* é responsável por garantir que a largura de banda não esteja se aproximando do limite, de forma que o mecanismo de elasticidade possa provisionar novos recursos para distribuir as requisições de entrada e saída.

Figura 8. Fluxo do monitor de *workload*

Fonte: autor

Uma vez que as métricas são lidas no componente, o monitor de *workload* deve empacotar as métricas para que as mesmas possam ser retornadas ao cliente. No caso de uma API RESTful, por exemplo, as métricas podem ser empacotadas em XML ou JSON para serem devolvidas como parte da resposta HTTP retornada ao cliente.

Nota-se, no entanto, que os detalhes da implementação do monitor de *workload* vão depender do modelo do software da arquitetura de preservação. Como a premissa de sistemas de preservação é que os dados possam ser lidos décadas após a sua concepção, detalhes sobre a implementação física dos monitores de *workload* não são fornecidos neste capítulo. Vale ressaltar que, como os monitores de *workload*

dependem da mídia sendo utilizada, os monitores devem ser implementados de acordo com a infraestrutura de armazenamento disponível na época. Isto significa que numa arquitetura de preservação que esteja ativa durante décadas, a cada inserção de um novo tipo de armazenamento será necessário também implementar o monitor de *workload* correspondente a essa nova mídia.

4.8.2.2 Controlador de elasticidade

O controlador de elasticidade é a peça central do mecanismo de elasticidade, e é responsável por orquestrar os diversos componentes relacionados ao gerenciamento de elasticidade para elastecer a arquitetura. O controlador de elasticidade recebe as métricas de *workload* dos diversos monitores de *workload* da arquitetura e é responsável por definir, de acordo com a política de elasticidade, se a arquitetura está subprovisionada, superprovisionada ou em estado ótimo, e então criar requisições para a alocação ou desalocação de recursos para acomodar o *workload* com o menor custo possível.

A Figura 9 mostra o fluxo de atividades do controlador de elasticidade quando é necessário elastecer a arquitetura através do provisionamento de um novo recurso. Primeiramente, o controlador recebe métricas dos monitores de *workload* e verifica a política de elasticidade, desta maneira detectando se há ou não necessidade de elastecer a arquitetura. Caso não haja necessidade, o controlador de elasticidade não toma nenhuma ação.

Caso haja necessidade de provisionar um novo recurso, o controlador de elasticidade deve primeiramente avaliar se há recursos alocados no buffer que podem ser utilizados. Isto é feito através de uma chamada para o gerenciador de buffer. Caso o recurso já esteja alocado, este recurso é selecionado. Caso um recurso não esteja previamente disponível no gerenciador de buffer, o controlador de elasticidade requer um novo recurso através de uma chamada ao atuador de elasticidade e então seleciona o novo recurso para ser configurado.

Uma vez que o novo recurso é selecionado, independentemente de ter vindo do buffer ou de uma nova alocação, o recurso deve ser configurado para fazer parte da arquitetura de preservação, ou seja, deve tornar-se visível como um contêiner de armazenamento, gateway de arquivamento, componente de empacotamento ou componente de arquivamento, como apontado na Figura 7. No caso de contêineres de armazenamento em arquiteturas SIRF, isto significa adicionar o contêiner ao arquivo de configuração.

Uma vez que o novo recurso é configurado, o monitor de *workload* também deve ser instalado e iniciado no mesmo servidor. Finalmente, os dados de preservação devem ser redistribuídos de acordo com a política de distribuição caso necessário.

Figura 9. Fluxo do controlador de elasticidade para provisionamento de recursos

Fonte: autor

A Figura 10 mostra o fluxo do controlador de elasticidade quando um recurso deve ser removido por não ser mais necessário. A remoção de um recurso, do ponto de vista da computação em nuvem, significa diminuição do custo para manter a arquitetura funcionando. Primeiramente, o controlador recebe métricas dos monitores de *workload* e verifica a política de elasticidade, desta maneira detectando se há ou não necessidade de remover algum recurso através da funcionalidade de elasticidade. Caso não haja necessidade, o controlador de elasticidade não toma nenhuma ação.

Caso haja necessidade de remover o recurso, o controlador de elasticidade deve primeiramente verificar se há necessidade de redistribuir os dados existentes, e redistribuí-los caso necessário. Este passo se aplica exclusivamente para o caso em que o recurso a ser removido é um contêiner de armazenamento. Uma vez que os dados são redistribuídos, o recurso deve ser desconfigurado, de forma a não mais fazer parte da arquitetura de preservação, ou seja, deve tornar-se invisível como um contêiner de armazenamento, gateway de arquivamento, componente de empacotamento ou componente de arquivamento (como apontado na Figura 3).

Figura 10. Fluxo do controlador de elasticidade para remoção de recursos

Fonte: autor

É importante ressaltar que um recurso que está desconfigurado não faz parte da arquitetura de preservação, o que não significa que o recurso não esteja alocado na nuvem e pronto para ser (re)configurado em um momento futuro. Neste sentido, o controlador de elasticidade avalia se há recursos suficientes alocados no buffer para uma futura adição de um novo recurso. Isto é feito através de uma chamada para o gerenciador de buffer. Caso o buffer já possua recursos suficientes para uma futura

alocação, o recurso é removido completamente, ou seja, é desalocado da nuvem e portanto deixa de ser cobrado. Caso um recurso extra não esteja previamente disponível no gerenciador de buffer, o controlador de elasticidade adiciona o recurso ao buffer.

4.8.2.3 Atuador de elasticidade

O atuador de elasticidade tem como função alocar ou desalocar novos recursos de acordo com as decisões tomadas pelo mecanismo de elasticidade (decisões estas baseadas nas métricas de *workload* recebidas pelo controlador de elasticidade).

As tarefas do atuador de elasticidade são:

- Determinar os componentes internos (softwares instalados no sistema operacional) do recurso que deve ser alocado.
- Provisionar um novo servidor (geralmente uma máquina virtual).
- Instalar os softwares necessários no novo recurso.
- Configurar as conexões de rede dos novos recursos:
 - Interface de rede
 - Nomes de domínio
 - Sincronização com os relógios dos outros servidores da arquitetura
 - Firewall
 - Serviços do sistema operacional (e.g. ssh, servidores HTTP para comunicação com outros componentes da arquitetura, serviços de infraestrutura como código)

Após a alocação ou desalocação de recursos pelo atuador de elasticidade, o controlador de elasticidade é responsável por configurar o novo recurso à arquitetura de preservação (configuração do recurso como parte do armazenamento, distribuição de dados, etc.).

4.8.2.4 Gerenciador de buffer

O gerenciador de buffer é responsável por garantir uma margem de recursos adicionais para elasticidade. Toeroe, Pawar e Khendek (2014) apontam que, para acelerar futuros ajustes, o gerenciador de buffer reserva nós adicionais.

O gerenciador de buffer deve alocar os nós, sem no entanto configurar a arquitetura de preservação para utilizar esses recursos. Nenhum dado deve ser preservado no novo recurso enquanto ele não for configurado como parte da arquitetura de preservação, mesmo que o recurso esteja disponível e online.

Nota-se que, dada a utilização do gerenciador de buffer o chamado “estado ótimo” torna-se mais do que simplesmente um número de recursos, mas um intervalo de número de recursos que pode ser configurado de acordo com a política de elasticidade.

O gerenciador de buffer também pode ser capaz de prever a demanda futura e antecipar a alocação de novos recursos como parte do buffer. Rosa *et al.* (2014) definem um mecanismo de predição de *workload* que auxilia o gerenciamento de buffer. O objetivo do mecanismo é combinar provisionamento com a demanda no futuro próximo. Para alcançar o objetivo, utiliza-se informações históricas de demanda como entrada para gerar a previsão de demanda no futuro próximo. Sabendo de antemão como a demanda será no futuro permite o gerenciamento preemptivo de recursos. Embora seja impossível saber com precisão como a demanda irá variar, a análise de regressão e técnicas de aprendizado de máquina (*machine learning*) combinados podem produzir boas aproximações.

É também responsabilidade do gerenciador de buffer alocar novos recursos quando um limite mínimo for atingido, conforme definido na política de elasticidade. Por exemplo, dada uma nova configuração de um recurso na arquitetura, o buffer deixa de ter este recurso disponível em *standby* (já que o recurso passa a estar online). Neste caso o gerenciador de buffer deve contactar o atuador de elasticidade para que um novo recurso seja criado e disponibilizado como parte do buffer.

5 IMPLEMENTAÇÃO E RESULTADOS

Neste capítulo é apresentado o protótipo da arquitetura descrita no capítulo 4, utilizado para a parte experimental, bem como os detalhes de sua implementação em linguagem Java. O código-fonte do protótipo está disponível atualmente como um projeto de código aberto em <http://github.com/opensirf>.

5.1 INTRODUÇÃO AO OPENSIRF

Dado o formato SIRQ e o design de arquitetura descrito anteriormente, iniciou-se a implementação de um protótipo da arquitetura. Parte dessa implementação (que compreende apenas o formato SIRQ, sem levar em conta a elasticidade e Big Data) tornou-se a implementação de referência do formato SIRQ e está disponível como um software de código fonte aberto. Praticantes da área de preservação a longo prazo podem criar seus próprios sistemas SIRQ e instanciá-los usando a proposta de arquitetura como base. O protótipo recebeu o nome de OpenSIRQ, foi implementado em linguagem Java e está disponível publicamente no repositório GitHub em <https://github.com/opensirf>.

O OpenSIRQ engloba um componente principal (opensirf-core), que contém as classes de modelo com todos os elementos que pertencem ao formato SIRQ, e um componente REST (opensirf-server), que define uma API RESTful para manipular o catálogo SIRQ e seus objetos associados. Devido à API RESTful, o OpenSIRQ é compatível e pode ser facilmente integrado com interfaces de usuário e serviços web.

Os dois componentes do OpenSIRQ, portanto, são:

- OpenSIRQ Core (opensirf-core): contém as classes de modelo para o contêiner SIRQ, objeto mágico, catálogo, PO e todas as categorias descritas

pelo formato SIRF. As classes que fazem parte do catálogo contêm anotações JAXB para que possam ser facilmente transformados de/para XML e JSON.

- OpenSIRF Server (opensirf-server): camada RESTful usando JAX-RS (implementação Jersey), suportando a transformação de objetos OpenSIRF Core para XML e JSON quando os métodos HTTP são chamados. Atualmente o OpenSIRF Server suporta OpenStack Swift e sistemas de arquivos como contêineres de armazenamento subjacentes.

5.2 OPENSIRF CORE

O componente OpenSIRF Core contém as classes de modelo (*Java beans*) para o contêiner SIRF, objeto mágico, catálogo, PO e todas as categorias. Em outras palavras, todos os elementos contidos na especificação SIRF estão contidos no OpenSIRF Core com exatamente os mesmos nomes e atributos. Classes que representam categorias que são persistentes no catálogo contêm anotações JAXB para transformar de/para JSON e XML. Por exemplo:

```
@XmlAccessorType(XmlAccessType.FIELD)
@XmlRootElement
public class SIRFCatalog
```

Na Figura 11 pode-se ver as classes do OpenSIRF Core que servem como modelos (*Java beans*) para os elementos do padrão SIRF.

Figura 11. Classes do OpenSIRF Core usadas para modelar os elementos SIRF

Fonte: autor

5.3 OPENSIRF SERVER

O componente OpenSIRF Server define uma API para manipular elementos SIRF. Uma vez que o OpenSIRF Server é implantado através de um servidor de aplicativos web (por exemplo, Apache Tomcat ou WebSphere Application Server), os usuários podem fazer chamadas para a API definida na Tabela 4.

Por exemplo, um contêiner SIRF pode ser criado usando uma instância OpenSIRF Server executando no IP \$OPENSIRF_IP e porta \$OPENSIRF_PORT simplesmente invocando-se o método PUT usando a URI /sirf/container/{nomeDoContêiner}. Isso pode ser feito usando-se uma aplicação simples capaz de chamar métodos HTTP, como por exemplo o cURL:

```

$ curl -i -X PUT http://$OPENSIRF_IP:$OPENSIRF_PORT/opensirf-
jaxrs/sirf/container/Example
HTTP/1.1 201 Created
Server: Apache-Coyote/1.1
  
```



```
Location: http://127.0.0.1:8080/opensirf-jaxrs/sirf/container/Example
Content-Length: 0
Date: Wed, 26 Nov 2014 13:28:51 GMT
```

Para recuperar o objeto mágico do contêiner SIRQ recém-criado, o método GET pode ser utilizado:

```
$ curl -i -X GET http://$OPENSIRF_IP:$OPENSIRF_PORT/opensirf-
jaxrs/sirf/container/Example
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Type: application/xml
Content-Length: 202
Date: Wed, 26 Nov 2014 13:32:00 GMT
<?xml version="1.0" encoding="UTF-8" standalone="yes"?
><magicObject><sirfLevel>1</sirfLevel><containerSpecification>0.15</containerSpeci
fication><sirfCatalogId>catalog.json</sirfCatalogId></magicObject>
```

A descrição de todos os comandos da API RESTful é dada no apêndice e um resumo pode ser encontrado na Tabela 4. Detalhes sobre como essas requisições são realizadas no servidor são apresentados na seção a seguir.

5.4 UTILIZAÇÃO DOS ELEMENTOS DO OPENSIRF CORE PELO OPENSIRF SERVER

Nesta seção é explicado mais detalhadamente o que acontece quando uma das várias solicitações HTTP é enviada ao OpenSIRF Server e como este instancia objetos do Core para manipular o conteúdo do catálogo SIRQ e POs. Uma visão esquemática do OpenSIRF pode ser vista na Figura 12. A camada de armazenamento fica na parte inferior da arquitetura, e é constituída por discos rígidos, discos de estado sólido, OpenStack Swift ou uma combinação destes. Acima da camada de armazenamento está a camada REST, que é onde fica o OpenSIRF Server e é a única camada que acessa o conteúdo da camada de armazenamento. A camada REST aceita solicitações HTTP e lida com esses pedidos normalmente criando novos objetos OpenSIRF Core, quando pedidos PUT e POST são enviados, e gravando-os nos dispositivos de armazenamento; ou recuperando objetos

OpenSIRF Core no caso de requisições GET e devolvendo-os para o cliente. Além disso, quando o conteúdo de um PO é recuperado, um fluxo de octetos (*octet stream*) é aberto para o cliente, similar ao que acontece quando um download é solicitado em um navegador web. Na parte superior tem-se a camada de dados, onde um fluxo de dados do tipo Big Data é absorvido constantemente, processado e encaminhado para o gateway de arquivamento, que então gera as requisições para o OpenSIRF Server.

Nota-se na parte inferior da Figura que o mecanismo de elasticidade foi implementado, bem como monitores de *workload* para os diversos contêineres de armazenamento.

Figura 12. Visão esquemática dos componentes REST e Core na arquitetura do protótipo

Fonte: autor

Elementos do OpenSIRF Core são transmitidos na camada REST entre os diversos componentes e o cliente (interface gráfica ou serviços web). Os elementos do OpenSIRF Core existem como objetos Java apenas enquanto uma solicitação está sendo processada. Depois que um pedido é processado os objetos devem ser devolvidos para o cliente ou persistidos nos dispositivos de armazenamento. Objetos

Java normalmente não são persistentes em armazenamento ou retornados em um serviço web ou GUI, e portanto estes objetos devem primeiro ser convertidos para um formato que seja de fácil leitura e fácil transporte através de uma conexão HTTP. No protótipo utiliza-se o framework JAXB para isso. As classes do OpenSIRF Core são anotadas com *annotations* JAXB e isto permite a conversão relativamente fácil entre objetos Java de/para XML e JSON. Assim, quando um objeto Core é gravado na camada de armazenamento, ele é primeiro convertido para JSON e, em seguida, gravado como um arquivo de texto com extensão .json. O processo de conversão de um objeto Java simples para JSON é chamado de *marshalling*.

A Figura 13 ilustra o fluxo de objetos Java durante requisições de inserção (POST/PUT) ou recuperação (GET). Em ambos os casos, o contêiner de armazenamento é identificado assim que a requisição é recebida. Então o driver para o contêiner de armazenamento (de destino, no caso de POST/PUT e de origem no caso de GET) é instanciado. Os metadados então são convertidos de objetos Java para JSON e escritos no contêiner de destino (POST/PUT), ou recuperados da mídia, convertidos para objetos Java e então retornados para o cliente em formato JSON ou XML.

Quando um objeto Core é trazido da camada de armazenamento ele é recuperado inicialmente em formato JSON e em seguida convertido a um objeto Java comum (esse processo é chamado de *unmarshalling*). Quando um objeto Core é devolvido para o cliente, ele também é codificado em XML ou JSON.

O *marshalling* de um objeto que será devolvido ao cliente da requisição é feito automaticamente no OpenSIRF Server usando o plugin MOXy Jersey. Em outras palavras, no código-fonte, os métodos que manipulam as solicitações HTTP retornam objetos OpenSIRF Core simples e o MOXy Jersey fica a cargo de realizar o *marshalling* automaticamente desse objeto usando JAXB e retorná-lo como XML ou JSON. Ou seja, o *marshalling* não precisa ser feito de forma programática.

Figura 13. Fluxo de objetos Java durante requisições

Fonte: autor

Algumas operações simples podem acontecer sem o uso de objetos OpenSIRF Core, geralmente porque só necessitam de um acesso simples ao sistema de arquivos para serem cumpridas. Por exemplo, quando um contêiner inteiro é removido (`DELETE /sirf/container/{name}`), quando os contêineres são listados (`GET /sirf/container`), ou quando o conteúdo de um PO é recuperado (`GET /sirf/container/{containername}/{po}/data`).

5.5 IMPLEMENTAÇÃO DE MÚLTIPLOS CONTÊINERES DE ARMAZENAMENTO

Como apontado anteriormente, um contêiner SIRF pode estar associado a diversos contêineres de armazenamento, e isso acontece ainda mais frequentemente quando o mecanismo de elasticidade decide alocar um novo recurso de armazenamento de acordo com a política de elasticidade.

A implementação do suporte a múltiplos contêineres se deu através da utilização do *Strategy* (estratégia) *design pattern* (GAMMA et al., 1994). A função deste design pattern é permitir que o comportamento de um dado algoritmo possa ser definido em tempo de execução. Neste caso, implementou-se uma estratégia para permitir que a mídia pudesse ser escolhida durante as operações de entrada e saída. Ou seja, através da utilização de uma estratégia é possível ter uma política de elasticidade que suporta diferentes mídias, e a decisão sobre o tipo de mídia que será usado é feita em tempo de execução pelo componente de arquivamento.

O código Java possui uma interface **IStorageContainerStrategy** que define as operações que uma estratégia deve ser capaz de realizar. Essas operações são operações abstratas sobre um contêiner SIRF, como criação de um contêiner, recuperação de objeto mágico, submissão de PO, entre outras.

```
public interface IStorageContainerStrategy extends Closeable {
    public ContainerConfiguration getConfig();
    public void setConfig(ContainerConfiguration c);
    public MagicObject retrieveMagicObject(String containerName);
    public void createContainer(String containerName);
    public void pushPreservationObject(String poID, byte[] b, String containerName);
    public void pushProvenanceInformation(String authorName, String containerName);
    public void pushCatalog(SIRFCatalog catalog, String containerName);
    public void deleteContainer(String containerName);
    public void deletePreservationObject(String poName, String containerName);
    public SIRFCatalog getCatalog(String containerName);
    public StreamingOutput getPreservationObjectStreamingOutput(String poUUID, String containerName);
    public InputStream getPreservationObjectInputStream(String poUUID, String containerName);
}
```

Para que diversas mídias pudessem ser suportadas, definiu-se também uma interface **ISirfDriver**. A interface define operações de baixo nível que devem ser realizadas na camada de armazenamento e, por esse motivo, deve-se implementar um driver para cada tipo de mídia suportada. No experimento descrito nesta tese, dois drivers foram implementados: **FileSystemDriver**, que implementa as operações em sistemas de arquivos, e **SwiftDriver**, que implementa as operações em um contêiner do OpenStack Swift.

```
public interface ISirfDriver {
    public void createContainerAndMagicObject(String containerName);
    public MagicObject containerMetadata(String containerName);
    public InputStream getFileInputStream(String container, String filename) throws
IOException;
    public void uploadObjectFromString(String containerName, String fileName, String
content);
    public void uploadObjectFromByteArray(String containerName, String fileName,
byte[] b);
    public void deleteContainer(String containerName);
    public void deleteObject(String containerName, String objectName);
}
```

O método `createContainerAndMagicObject()` cria o contêiner SIRF na mídia de armazenamento escolhida de acordo com a política de distribuição. A criação do contêiner SIRF passa pela criação do contêiner físico no armazenamento (contêiner Swift no driver do OpenStack Swift ou diretório no driver de sistemas de arquivos) mais o objeto mágico.

O método `containerMetadata()` recupera o objeto mágico da mídia. No caso do OpenStack Swift, esses metadados estão no próprio contêiner Swift. No caso do driver de sistemas de arquivos, os metadados se encontram em um arquivo chamado `magic.json`. A classe `MagicObject` do OpenSIRF Core é utilizada para recuperar esses metadados da mídia, e posteriormente o objeto do tipo `MagicObject` passa pelo processo de *unmarshalling* para ser devolvido como XML ou JSON para o cliente.

O método `getFileInputStream()` abre uma stream de dados do contêiner de armazenamento para o cliente HTTP fazendo a requisição. Este método é utilizado para recuperar POs na camada de armazenamento.

O método `uploadObjectFromString()` transforma uma `String` (recebida como parâmetro) em um array de bytes e invoca o método `uploadObjectFromByteArray()`.

O método `uploadObjectFromByteArray()` faz o upload de um array de dados para o contêiner de armazenamento. Na prática, é o método que de fato escreve um PO para a camada de armazenamento.

O método `deleteContainer()` é responsável por remover o contêiner S3 de um contêiner de armazenamento.

O método `deleteObject()` remove um PO do contêiner de armazenamento.

A estratégia de múltiplos contêineres decide em tempo de execução qual será a mídia utilizada para escrever dados. A estratégia, então, instancia um driver para o contêiner escolhido de acordo com a mídia do contêiner, e chama a implementação de baixo nível do contêiner.

Na Figura 14 pode-se ver um fluxo que descreve como a estratégia decide qual driver será utilizado em tempo de execução para uma operação de inserção. Primeiramente a requisição para inserção de novos dados é recebida. De acordo com a política de distribuição, um dos contêineres de armazenamento é eleito como o contêiner alvo, o destino onde os dados serão escritos. A estratégia então, sabendo qual é o contêiner alvo, instancia um driver para o tipo de mídia (por exemplo, `SwiftDriver` se o contêiner é do tipo OpenStack Swift ou `FilesystemDriver` se o contêiner é do tipo sistema de arquivos). A informação então é escrita para o contêiner através do método `uploadObjectFromByteArray()`, que é invocado pelo método `pushPreservationObject()` da estratégia.

Figura 14. Fluxo da instanciação do driver do contêiner a ser utilizado

Fonte: autor

A decisão sobre qual dos contêineres de armazenamento será utilizado para escrever dados (ou seja, submeter POs e/ou adicionar novos metadados) depende do algoritmo de distribuição escolhido. O algoritmo de distribuição é definido em um arquivo de configuração do OpenSIRF. Duas opções de algoritmos de distribuição foram implementadas:

- **Serial:** sempre escolhe o contêiner de armazenamento que estiver mais cheio para operações de escrita, de modo que um novo contêiner apenas será utilizado quando o contêiner atual esgotar sua capacidade.
- **Igualmente livre:** sempre escolhe o contêiner de armazenamento que tiver mais espaço livre, de modo que os contêineres tendem a ter a mesma capacidade livre ao longo do tempo.

Já para a leitura de dados existentes, a estratégia utiliza um índice que encontra-se localizado junto ao catálogo. O índice contém relações entre os POs e os contêineres de armazenamento que os contêm. Desta forma, quando um PO é requerido pelo componente de reobtenção, a estratégia busca no índice a localização do PO em algum dos contêineres de armazenamento, e então utiliza o driver correspondente para obter o conteúdo do PO.

5.6 MAPEAMENTO PARA O OPENSTACK SWIFT

O OpenStack Swift é um sistema de armazenamento de dados do tipo objeto (*object storage*). Utilizando-se o OpenStack Swift pode-se fazer uso de uma camada de comandos RESTful para realizar operações em objetos (arquivos), tais como PUT, POST, DELETE, entre outras, que permitem que se adicione, remova e altere objetos remotamente. O OpenStack Swift utiliza o OpenStack Identity como ferramenta de autenticação e segurança, para garantir que objetos apenas possam ser manipulados por usuários autorizados. É necessário apresentar credenciais ao OpenStack Identity para obter um *token* (conjunto de caracteres que confere uma determinada autenticação). De posse do correto *token* do OpenStack Identity, o OpenStack Swift considera o usuário autenticado e então é possível manipular objetos em contêineres do OpenStack Swift.

O OpenSIRF Server usa a biblioteca Apache jclouds para se conectar ao OpenStack Swift e realizar o mapeamento entre objetos SIRF e objetos Swift.

O mapeamento entre objetos SIRF e objetos OpenStack Swift se dá como a seguir:

- Um contêiner SIRF é mapeado para um contêiner Swift.
- O objeto mágico do SIRF é mapeado para os metadados do contêiner Swift.
- O catálogo SIRF é mapeado para um objeto Swift chamado "catalog.json" na raiz do contêiner Swift.
- POs são mapeados para objetos Swift.

Por exemplo, suponha que se esteja utilizando o OpenSIRF Server para criar um contêiner chamado `Example`. Em seguida, submeteu-se um PO cujo nome é "EdgarAllanPoe-Alone" (portanto, o identificador lógico é `Example-EdgarAllanPoe-Alone` e o identificador de versão é `Example-EdgarAllanPoe-Alone-1.0`) e seu conteúdo é o poema "Alone", de Edgar Allan Poe.

Dado um sistema Swift sendo executado em `$SWIFT_IP:$SWIFT_PORT`, usando `$SWIFT_ACCOUNT` como o nome da conta (usuário). Também supõe-se que foi solicitado um *token* de autorização do OpenStack Identity e este foi armazenado em `$AUTH_TOKEN`. Ao usar-se a API RESTful do OpenStack Swift para recuperar informações sobre o contêiner Swift, tem-se o seguinte resultado (nota-se que esta API é diferente da API OpenSIRF Server):

Lista de todos os contêineres SIRF na instância OpenStack Swift (método GET sem argumentos na URI):

```
$ curl -X GET -i -H "X-Auth-Token:$AUTH_TOKEN" http://$SWIFT_IP:
$SWIFT_PORT/v1/$SWIFT_ACCOUNT
```

```
HTTP/1.1 200 OK
Content-Length: 8
X-Account-Object-Count: 2
X-Account-Storage-Policy-Policy-0-Bytes-Used: 2094
X-Timestamp: 1406584442.12182
X-Account-Storage-Policy-Policy-0-Object-Count: 2
X-Account-Bytes-Used: 2094
X-Account-Container-Count: 1
Content-Type: text/plain; charset=utf-8
Accept-Ranges: bytes
X-Trans-Id: txe6c1aad3a822464ab3440-00547cadbe
Date: Mon, 01 Dec 2014 18:04:46 GMT
```

Example

Recupera-se o objeto mágico do contêiner `Example` recuperando-se os metadados do contêiner Swift (método HEAD):

```
$ curl -X HEAD -i -H "X-Auth-Token:$AUTH_TOKEN" http://$SWIFT_IP:
$SWIFT_PORT/v1/$SWIFT_ACCOUNT/Example

HTTP/1.1 204 No Content
Content-Length: 0
X-Container-Object-Count: 3
X-Container-Meta-Sirfcatalogid: catalog.json
Accept-Ranges: bytes
X-Storage-Policy: Policy-0
X-Container-Meta-Sirflevel: 1
X-Container-Meta-Containerspecification: 0.15
X-Container-Bytes-Used: 3786
X-Timestamp: 1417008580.94069
Content-Type: text/plain; charset=utf-8
X-Trans-Id: tx6540442f141f42e68e69c-00547cadfb
Date: Mon, 01 Dec 2014 18:05:47 GMT
```

O conteúdo do contêiner Example, ou seja, a lista de objetos – o catálogo e dois POs (proveniência e o PO que foi submetido anteriormente) é obtido através do método GET com o nome do contêiner na URI:

```
$ curl -X GET -i -H "X-Auth-Token:$AUTH_TOKEN" http://$SWIFT_IP:
$SWIFT_PORT/v1/$SWIFT_ACCOUNT/Example

HTTP/1.1 200 OK
Content-Length: 64
X-Container-Object-Count: 3
X-Container-Meta-Sirfcatalogid: catalog.json
Accept-Ranges: bytes
X-Storage-Policy: Policy-0
X-Container-Meta-Sirflevel: 1
X-Container-Meta-Containerspecification: 0.15
X-Container-Bytes-Used: 3786
X-Timestamp: 1417008580.94069
Content-Type: text/plain; charset=utf-8
X-Trans-Id: tx5f29c30e90164fe9b65c5-00547cae1b
Date: Mon, 01 Dec 2014 18:06:19 GMT

Example-EdgarAllanPoe-Alone-1.0
catalog.json
provenance.po.json
```

O conteúdo do PO é obtido através do método GET com o nome do contêiner e o identificador único do PO na URI:

```
$ curl -X GET -i -H "X-Auth-Token:$AUTH_TOKEN" http://$SWIFT_IP:
$SWIFT_PORT/v1/$SWIFT_ACCOUNT/Example/Example-EdgarAllanPoe-Alone-1.0
```

```
HTTP/1.1 200 OK
Content-Length: 736
Accept-Ranges: bytes
Last-Modified: Mon, 01 Dec 2014 18:01:17 GMT
Etag: a393d7c293d9820dac7656853da640cc
X-Timestamp: 1417456876.21899
Content-Type: application/unknown
X-Trans-Id: tx4280dc796f534ff4a0f34-00547caeb1
Date: Mon, 01 Dec 2014 18:08:49 GMT
```

```
From childhood's hour I have not been
As others were; I have not seen
As others saw; I could not bring
My passions from a common spring.
From the same source I have not taken
My sorrow; I could not awaken
My heart to joy at the same tone;
And all I loved, I loved alone.
Then- in my childhood, in the dawn
Of a most stormy life- was drawn
From every depth of good and ill
The mystery which binds me still:
From the torrent, or the fountain,
From the red cliff of the mountain,
From the sun that round me rolled
In its autumn tint of gold,
From the lightning in the sky
As it passed me flying by,
From the thunder and the storm,
And the cloud that took the form
(When the rest of Heaven was blue)
Of a demon in my view.
```

```
Edgar Allan Poe
```

5.7 MAPEAMENTO PARA SISTEMAS DE ARQUIVOS

O mapeamento para sistemas de arquivos se dá de maneira similar ao do mapeamento para o OpenStack Swift, com exceção do objeto mágico:

- Um contêiner SIRF é mapeado para um diretório no sistema de arquivos.
- O catálogo SIRF é mapeado para um arquivo chamado "catalog.json" na raiz do diretório.
- POs são mapeados para arquivos.
- O objeto mágico do SIRF é mapeado para um arquivo chamado "magic.json".

No exemplo mostrado acima, para um sistema de arquivos do sistema operacional Linux, com um contêiner de armazenamento chamado "lv1" e contêiner SIRF chamado "Example", tem-se:

```
# ls -lh lv1/Example
total 16K
-rw-r-----. 1 root root 1.6K Dec 28 14:13 catalog.json
-rw-r--r--. 1 root root 736 Dec 28 16:06 Example-EdgarAllanPoe-Alone-1.0
-rw-r-----. 1 root root 79 Dec 28 14:13 magic.json
-rw-r-----. 1 root root 54 Dec 28 14:13 provenance.po.json
```

Onde o comando `ls -lh` lista com detalhes o conteúdo do diretório (no caso `lv1/Example`). Nota-se que o diretório possui quatro arquivos: um para o objeto mágico (`magic.json`), um para o catálogo (`catalog.json`) e dois POs, o de proveniência (adicionado automaticamente ao criar-se um novo contêiner) e o PO adicionado através da requisição para o OpenSIRF Server.

5.8 IMPLEMENTAÇÃO DO MECANISMO DE ELASTICIDADE

O mecanismo de elasticidade foi implementado através da utilização das tecnologias Chef e Vagrant.

O Vagrant é um software de código aberto que permite que máquinas virtuais sejam provisionadas rapidamente através de um único arquivo de configuração. O arquivo de configuração possui informações sobre a imagem do sistema operacional que deve ser utilizada, quantidade de CPUs e memória RAM, entre outros atributos. No

caso deste experimento, criou-se diversas imagens do sistema operacional Linux com componentes do experimento pré-instalados, como o OpenStack Identity, OpenStack Swift, monitores de *workload*, Apache Tomcat (servidor web) e OpenSIRF Server. Desta forma, através de um único comando Vagrant, passando como parâmetro um arquivo de configuração especificando a imagem e os atributos da máquina virtual, tem-se um componente da arquitetura instanciado e pronto para ser configurado.

O Chef é uma ferramenta de gestão de configuração utilizada para manter elementos de um servidor ou arquitetura através do modelo “infraestrutura como código” (*infrastructure as code*). Com o Chef, define-se um ou mais arquivos chamados *cookbooks*, onde ações de configuração são representadas como código e podem então ser executadas em um dado servidor. Por exemplo, a instalação de um novo software pode ser feita através de três linhas de código:

```
yum_package 'java-1.8.0-openjdk-devel.x86_64'
  action :install
end
```

Ou um serviço do sistema operacional pode ser iniciado de maneira semelhante:

```
tomcat_service '8' do
  action :start
end
```

Desta forma, diversas ações de configuração da infraestrutura podem ser tratadas como código dentro de um cookbook. É necessário um servidor central (Chef Server) que contém todos os cookbooks necessários para a arquitetura e também tem todos os servidores (nós) da arquitetura registrados. Para que uma mudança de configuração aconteça, é necessário requerer que o servidor Chef aplique uma ou mais ações de um cookbook em um ou mais servidores (nós). Um cookbook é composto por uma ou mais *receitas*. Receitas de cookbooks são executadas separadamente por serem tipicamente fases ou operações diferentes de um conjunto de ações que precisam ser tomadas em um servidor.

A Figura 15 mostra um diagrama esquemático de como a combinação de Chef e Vagrant foi utilizada para implementar a elasticidade nesta tese. No canto inferior direito, o mecanismo de elasticidade requisita uma ou mais máquinas virtuais para o Vagrant, dado que novos recursos precisam ser alocados dada a necessidade de elastecer a arquitetura. Cada uma dessas máquinas virtuais V_1 , V_2 , ... V_n possui o cliente Chef instalado e portanto podem ser enxergadas pelo servidor Chef.

Os cookbooks e configurações estão armazenados como software de código aberto no repositório GitHub (parte central inferior da figura). A ferramenta *knife*, que é parte do Chef, é responsável por compilar os cookbooks e configurações presentes no repositório e atualizar o servidor Chef com as versões mais recentes.

Uma vez que os novos servidores foram requeridos pelo mecanismo de elasticidade e instanciados pelo Vagrant, o mecanismo de elasticidade então entra em contato com o servidor Chef e pede que um ou mais cookbooks sejam executados no novo nó. Esses cookbooks tem como responsabilidade configurar a conectividade do novo nó com o restante da arquitetura, configurar o novo nó como parte do armazenamento do OpenSIRF Server e redistribuir os dados caso necessário.

Figura 15. Esquema da elasticidade

Fonte: autor

O processo completo para elasticidade com inserção de um novo contêiner de armazenamento pode ser visto na Figura 16. Quando uma nova requisição do tipo PUT ou POST é enviada via HTTP para o OpenSIRF Server, primeiramente o servidor decide o contêiner no qual o PO será escrito de acordo com a política de distribuição e faz o mapeamento do novo PO para o contêiner alvo. Por exemplo, mapeia o conteúdo do PO para um objeto Swift ou um arquivo em um sistema de arquivos. O monitor de *workload* automaticamente envia métricas relativas à capacidade de armazenamento para o controlador de elasticidade a cada requisição.

De posse das novas métricas de *workload* e da política de elasticidade, o controlador de elasticidade decide se há a necessidade de provisionar novos recursos de armazenamento. Caso haja a necessidade, primeiramente o gerenciador de buffer é acessado para que um recurso já provisionado possa ser configurado caso exista. Caso não exista um recurso provisionado, o atuador de elasticidade instancia um novo servidor com um contêiner de armazenamento.

Uma vez que o novo servidor com o contêiner de armazenamento é provisionado, o atuador de elasticidade invoca o comando knife, que acessa o servidor Chef e faz uma requisição para que uma receita Chef seja executada no cliente Chef presente no novo servidor. O servidor Chef invoca o cliente no novo contêiner, que executa a receita de configuração e na prática torna o novo contêiner de armazenamento visível para a arquitetura. O controlador de elasticidade então redistribui os dados caso necessário de acordo com a política de elasticidade.

Um processo semelhante ocorre quando o mecanismo de elasticidade é utilizado para desalocar recursos que estão subutilizados, como mostra a Figura 17. Quando uma nova requisição do tipo DELETE é enviada via HTTP para o OpenSIRF Server, primeiramente o servidor acessa o índice para determinar em qual contêiner o PO se encontra. O elemento é então deletado através do driver do contêiner de armazenamento respectivo (por exemplo, OpenStack Swift ou sistema de arquivos). O monitor de *workload* automaticamente envia métricas relativas à capacidade de armazenamento para o controlador de elasticidade a cada requisição.

Figura 16. Processo de elasticidade para inserção de um novo contêiner de armazenamento

De posse das novas métricas de workload e da política de elasticidade, o controlador de elasticidade decide se há recursos subutilizados que podem ser liberados. Caso haja subutilização, primeiramente o controlador de elasticidade redistribui os dados caso necessário de acordo com a política de elasticidade. O comando knife é então invocado, que faz uma requisição para que uma receita Chef seja executada no servidor a ser desconfigurado. O servidor Chef invoca o cliente no recurso a ser desconfigurado, que executa a receita para desconfigurar o recurso, o que na prática torna o recurso invisível para a arquitetura.

Após a desconfiguração do recurso, o gerenciador de buffer é acessado para que o recurso seja adicionado ao buffer para uma futura reinserção à arquitetura ao invés de ser completamente removido. Caso o buffer já possua recursos suficientes como definido pela política de elasticidade, o recurso é desalocado completamente, sendo removido da nuvem e portanto não mais sendo cobrado.

Figura 17. Processo de elasticidade para remoção de um contêiner de armazenamento

Fonte: autor

5.9 IMPLEMENTAÇÃO DO GATEWAY DE ARQUIVAMENTO E CAMADA DE DADOS

Nesta seção são apresentados o gateway de arquivamento e a camada de dados. Esses são os elementos da arquitetura responsáveis por obter Big Data de diferentes fontes de dados e criar as requisições que são enviadas para a camada REST.

5.9.1 Gateway de Arquivamento – Jenkins

O gateway de arquivamento foi implementado através do sistema Jenkins, que é uma ferramenta de automação de scripts shell. Primeiramente, instanciou-se uma máquina virtual executando um sistema Jenkins. Depois, definiu-se um conjunto de trabalhos (*jobs*) automatizados cuja função é buscar os dados dos outros servidores da camada de dados. O mecanismo de obtenção de dados é do tipo *pull*, ou seja, os sistemas de Big Data não *enviam* dados para o gateway de arquivamento, mas é o gateway de arquivamento que tem a responsabilidade de *requisitar* os dados dos diversos servidores da camada de dados.

Como o foco da elasticidade foi apenas nos sistemas de preservação (SIRF), os sistemas da camada de dados foram implementados sem elasticidade. Desta maneira, sabia-se de antemão quantos e quais sistemas estavam executando *streaming*, bancos de dados relacionais, bancos de dados não-relacionais e sistemas de arquivos. Os sistemas foram então registrados no gateway de arquivamento através do sistema Jenkins. Uma vez registrados todos os sistemas, o gateway de arquivamento pôde realizar a obtenção de dados periodicamente através da funcionalidade de agendamento de tarefas.

Utilizou-se no gateway de arquivamento o comando *scp*, ou *secure copy*, que permite copiar arquivos de maneira segura entre um servidor e um cliente, para buscar arquivos do Apache Spark processados do Twitter4s, arquivos dos servidores

com os sistemas de arquivos, backups do DB2 e coleções do mongoDB que foram exportadas utilizando-se o comando `mongoexport`, nativo da ferramenta mongoDB. A Figura 18 mostra um esquema da comunicação do gateway de arquivamento com outros componentes da arquitetura.

Figura 18. Interações do gateway de arquivamento com outros sistemas

Fonte: autor

5.9.2 Dados não estruturados e *Streaming* – Apache Spark e Twitter4s

Para o componente de dados não estruturados e *streaming* de dados, utilizou-se o Apache Spark como mecanismo de processamento de Big Data, e o Twitter4s, que é uma API em linguagem Scala capaz de obter dados através de duas APIs do Twitter: a API REST e a API de streaming.

O Apache Spark é capaz de realizar funções típicas de sistemas de Big Data, como `map`, `reduce`, `filter` e `zip`. É também um mecanismo que suporta a utilização de clusters, desta forma paralelizando o processamento em diversas máquinas

autônomas. O Spark foi utilizado para processar e refinar os dados não-estruturados vindos do Twitter4s. Utilizou-se o Apache Spark em detrimento do Apache Hadoop pois o Apache Spark possui suporte a streaming de dados em tempo real.

O Twitter4s é uma API em linguagem Scala para o Twitter, suportando as APIs REST e de streaming. Configurou-se máquinas virtuais executando Scala com a API Twitter4s, continuamente recebendo dados do Twitter de acordo com palavras-chave e *hashtags* escolhidas. Uma vez que os dados eram recebidos, eram também armazenados em uma localização conhecida no sistema de arquivos. Esses dados são por natureza não-estruturados, já que contêm textos enviados por diversos usuários do Twitter em suas contas.

As palavras-chave e *hashtags* utilizadas para o experimento foram escolhidas de acordo com a frequência de utilização e os acontecimentos mais recentes na época do experimento, de maneira a maximizar o volume de dados recebidos pelo Twitter4s. Considerou-se que o conteúdo específico vindo do Twitter não tinha relevância, contanto que se pudesse captar um volume adequado de dados para a realização do experimento. Alguns exemplos de *hashtags* utilizadas durante o experimento foram:

- #Election2016** (eleições americanas 2016)
- #Brexit** (saída do Reino Unido da União Europeia)
- #Trump** (eleição de Donald Trump como presidente dos EUA)
- #Oscars** (cerimônia do Oscar)
- #theforceawakens** (lançamento de novo filme)
- #WorldEnvironmentDay** (dia mundial do meio ambiente)
- #OneLoveManchester** (atentado em Manchester)
- #RIP**
- #news**
- #NowPlaying**
- #netflix**
- #cats**
- #newprofilepic**

Uma vez que mais e mais dados foram adicionados pelo Twitter4s à localização conhecida no sistema de arquivos, o Spark passou então a analisar esses novos arquivos aplicando operações típicas de Big Data, como map, reduce, filter e zip.

Os dados analisados pelo Spark possuíam dois destinos: parte dos dados mais brutos, semelhantes aos textos não-estruturados originais, ficaram armazenados no sistema de arquivos do servidor executando Spark. Dados que foram analisados pelo Spark e que sofreram modificações por funções como map, reduce, filter e zip foram então armazenados em formato JSON no banco de dados não-relacional mongoDB. Os dados existentes no mongoDB foram então preservados pelo gateway de arquivamento.

Figura 19. Envio de informações dos sistemas de Big Data para o gateway de arquivamento

Fonte: autor

5.9.3 Dados estruturados não-relacionais – mongoDB

Foi criada uma instância do banco de dados não-relacional mongoDB para o experimento. Os dados provenientes das APIs de streaming e REST do Twitter que foram analisados pelo Apache Spark foram armazenados no mongoDB através de coleções (*collections*).

Para cada conjunto de palavras-chave utilizado para coletar os dados do Twitter, uma nova coleção no mongoDB foi criada para armazenar o resultado da análise dos dados feita pelo Spark. Dentro de cada collection foi criado um documento (em formato JSON) para cada data em que o streaming de dados estava rodando. Por exemplo, a coleção **netflix** possui um documento JSON para a data de 19/07/2017 com os seguintes campos:

```
{
  "_id": ObjectId("596fb21106a65e1152fbf667"),
  "timestamp": "20170719",
  "sampleSize": "4646",
  "numUsers": "4308",
  "topContributor": "TV__Newz",
  "mostCommonWord": "Marvel",
  "top3RelatedTags": [ "#ilovethebbc", "#thepunisher", "#sdcc2017" ]
}
```

O campo `_id` é automaticamente criado pelo mongoDB quando um novo registro é inserido. Os demais campos foram obtidos através de processamento com o Apache Spark (funções de mapeamento e redução):

- O campo `topContributor` representa o usuário que postou mais *tweets*.
- O campo `mostCommonWord` representa a palavra mais mencionada (não-trivial) nos *tweets*.
- O campo `top3RelatedTags` é um vetor com as 3 principais hashtags mencionadas nos *tweets*.
- Os demais campos são metadados da amostra
 - `timestamp` = data da amostra

- `sampleSize` = tamanho da amostra em número de *tweets*
- `numUsers` = número de usuários que enviaram postagens na amostra

5.9.4 Sistemas de arquivos

Parte dos dados provenientes das APIs de streaming e REST do Twitter foram armazenados em seu formato original (texto puro) em sistemas de arquivos para exercitar o gateway de arquivamento com *workloads* diferentes.

À medida que dados foram recebidos pelas API REST e de streaming do Twitter, parte dos dados foram armazenados (aleatoriamente) no sistema de arquivos. O sistema de arquivos utilizado era conhecido pelo gateway de arquivamento, que então enviou requisições do tipo *pull* para o sistema de arquivos.

5.9.5 Dados estruturados relacionais – HammerDB/TPC-C com DB2

Para a realização do experimento com dados estruturados, utilizou-se um banco de dados relacional com um *workload* típico utilizado pela indústria. O HammerDB foi selecionado como ferramenta de *benchmarking* e testes de carga. A ferramenta permite medir o desempenho e realizar testes de carga, provendo também uma interface gráfica que permite selecionar o tipo de *workload* desejado. Selecionou-se o *workload* TPC-C, que é uma especificação do TPC (um consórcio da indústria de bancos de dados relacionais) e que está disponível desde 1992, recebendo atualizações frequentes desde então. O TPC-C é considerado um dos padrões para testes de bancos de dados relacionais, especialmente para *workloads* do tipo OLTP (processamento online de transações, ou *Online Transaction Processing*), simulando um ambiente relacional de transações de um fornecedor de peças, com diversos armazéns e regiões de venda. A maior parte das transações são simulações de novos pedidos que chegam a este fornecedor ou pagamentos recebidos. O TPC-C foi utilizado para que se pudesse ter dados transacionais, e não apenas analíticos.

Utilizou-se o DB2 como gerenciador de banco de dados. Primeiramente criou-se uma máquina virtual rodando DB2, onde então foi instalado o HammerDB. Utilizou-se a interface gráfica de configuração que permite a seleção do *workload* a ser utilizado, a construção do esquema relacional do banco de dados, o carregamento dos dados e a execução.

Uma vez que o TPC-C foi selecionado, o esquema dos dados foi construído e então os dados foram carregados. A partir desse momento tinha-se um banco de dados disponível e pronto para receber novas transações que simulam uma carga típica de bancos de dados em sistemas reais.

Figura 20. Fluxo dos testes com banco de dados relacional

Fonte: autor

A ferramenta HammerDB também possui a funcionalidade de execução do *workload*. Através de cliques na interface gráfica do HammerDB, iniciou-se a execução do *workload* TPC-C no banco de dados criado no DB2.

5.10 DADOS DO EXPERIMENTO

O experimento foi executado em um servidor Dell PowerEdge T30 com processador Intel Xeon E3-1225 v5 (Quad-core) em sistema operacional Linux 3.10 (distribuição CentOS 7) e 32 GB de RAM. O VirtualBox foi utilizado como aplicação de virtualização, e em uma camada acima o Vagrant como gerenciador das máquinas virtuais utilizadas para o experimento.

Foram realizados testes visando verificar o funcionamento do mecanismo de elasticidade em diferentes estados de elasticidade, como subprovisionado, ótimo e superprovisionado, bem como a mudança de um estado para outro. O suporte a contêineres heterogêneos também foi um dos objetivos dos testes, com diferentes combinações de tipos de mídia sendo cobertas no experimento. As quatro operações possíveis sobre objetos de preservação foram executadas de maneira a exercitar o código da arquitetura: inserção, remoção, atualização e recuperação dos objetos de preservação arquivados. Também foram utilizados sistemas de Big Data tipicamente utilizados pela indústria e academia no momento da elaboração do trabalho, como sistemas de bancos de dados relacionais e não-relacionais, *streaming* de dados e ferramentas de análise com mapeamento e redução. Além disso, coletou-se estatísticas sobre as políticas de elasticidade e distribuição, desta maneira verificando que os testes das diferentes combinações de mídias e diferentes estados de elasticidade foram executados em diversos cenários que uma arquitetura instanciada em um domínio específico de utilização. Ressalta-se, contudo, que não foram realizados testes com o objetivo de analisar desempenho, pois considerou-se que tais testes estavam fora do escopo do trabalho, cujo foco principal era a funcionalidade de preservação em nuvem com elasticidade.

Foram realizados testes em cenários com diferentes combinações de tipos de mídia: disco rígido, SSD, Swift, disco rígido e SSD, e disco rígido, SSD e Swift. Em cada cenário tipicamente havia mais de uma instalação ou instância de cada tipo de mídia (dois ou mais discos rígidos, dois ou mais SSDs e assim por diante). A Figura 21 mostra o número de requisições de alocação, desalocação e aumento de capacidade (*scale-up*). O aumento de capacidade significa a expansão de um

contêiner de armazenamento existente, como uma partição de disco rígido ou um contêiner do OpenStack Swift. Através da execução desses cenários, foi possível identificar que a arquitetura proposta dá suporte a operações de elasticidade com combinações diferentes de mídia.

Figura 21. Número de cenários do experimento por mídia e operação de elasticidade

Outro fator levado em conta na execução do experimento foi a política de distribuição de dados. Como explicado anteriormente, quando um novo objeto de preservação é submetido à arquitetura, ou quando ocorre uma operação de redistribuição de dados, a política de distribuição dita onde os dados serão armazenados. A arquitetura implementada tinha duas políticas: serial, em que os dados sempre são armazenados no contêiner mais cheio; e contêineres igualmente livres, em que o dispositivo de armazenamento é escolhido de maneira que todos os dispositivos tendem a ter a mesma quantidade de espaço livre. A Figura 22 mostra a porcentagem de operações em cenários cuja política de distribuição foi serial e cenários cuja política de distribuição foi de contêineres igualmente livres.

Figura 22. Porcentagem de operações do experimento por política de distribuição

Operações por política de distribuição

Serial=36,3% Igualmente livres=63,7%

■ Igualmente livres ■ Serial

Fonte: autor

A Figura 23 mostra as políticas de elasticidade testadas. Foram definidas três políticas: a de menor custo, a intermediária e a de maior confiabilidade, que também era a de maior custo. Neste caso considera-se custo o número de recursos extra que são alocados e disponibilizados sem utilização. Como mencionado antes na tese, os conceitos de subprovisionado, superprovisionado e estado ótimo dependem na prática desses limites definidos na política de elasticidade. Portanto, pode-se dizer que a política de menor custo é a que tem o menor limite de subprovisionamento e o maior limite de superprovisionamento. A política de menor custo utilizou os limites de 90% e 10%, a intermediária 80% e 40%, e a política de maior confiabilidade (e também maior custo) 70% e 30%. Através da execução dos cenários com diferentes políticas de elasticidade, foi possível identificar que a arquitetura proposta suporta diferentes limites para a elasticidade, o que traz flexibilidade ao modelo e permite que arquiteturas baseadas na proposta possam ser configuradas de acordo com a confiabilidade e custo planejados.

Figura 23. Políticas de elasticidades e seus limites

Fonte: autor

A Figura 24 apresenta o número de operações sobre objetos de preservação no experimento de acordo com o tipo de mídia de armazenamento do contêiner. O experimento focou mais nos cenários multicontêiner por exercitarem uma porção maior do código e também porque a tendência em sistemas de preservação a longo prazo é de ter mídias heterogêneas dada a evolução dos tipos de mídia ao longo das décadas. Também focou-se mais nas operações de recuperação e atualização de objetos de preservação por serem as operações mais comuns em sistemas de preservação a longo prazo. As operações testadas ajudaram a exercitar o código da arquitetura e, desta forma, serviram como uma forma de testar o comportamento de um sistema de preservação a longo prazo de Big Data diante de uma demanda crescente e paralela.

Figura 24. Operações sobre POs de acordo com o tipo de contêiner

Fonte: autor

Na Figura 25 tem-se o número de cenários executados de acordo com a tecnologia utilizada na camada de dados. Foram executados três cenários com HammerDB sobre DB2 com benchmark TPC-C. O TPC-C é um cenário cuja carga geralmente é grande, de dezenas ou centenas de Gigabytes, e que executa de várias horas até dois dias.

Foram também executados 13 cenários de Apache Spark/Twitter, cada cenário relacionado a uma hashtag. Os dados coletados e analisados foram armazenados na instância mongoDB e portanto executou-se também 13 cenários de NoSQL com mongoDB.

Finalmente, 25 cenários foram testados cujos dados originaram-se de sistemas de arquivos. Tratam-se de backups das informações coletadas nos cenários de Apache Spark/Twitter, que foram arquivados como objetos de preservação como arquivos comprimidos.

Figura 25. Cenários testados pelo tipo de dados

5.11 REPRODUÇÃO DA ARQUITETURA

Como forma de facilitar a reprodução da arquitetura, além da presente tese trabalhou-se também no desenvolvimento de um tutorial para implementação do OpenSIRF Core e OpenSIRF Server. Todo o código-fonte do OpenSIRF, inclusive os cookbooks do Chef, está disponibilizado como um projeto de código aberto sob licença MIT.

A documentação do formato SIRF está em domínio público (SNIA, 2014), sendo um padrão da indústria definido pela SNIA. Portanto, os elementos do formato estão documentados e facilmente acessíveis. Juntamente com o formato foi disponibilizado também um esquema XML que permite que um catálogo em formato SIRF, inclusive um catálogo gerados com o OpenSIRF, possa ser validado como um catálogo válido.

As máquinas virtuais do Vagrant também estão disponíveis em domínio público em <https://app.vagrantup.com/opensirf/>. Isto significa que praticantes da área de armazenamento de longo prazo e preservação de dados podem facilmente baixar

máquinas virtuais pré-configuradas e portanto instanciar uma arquitetura SIRF em alguns minutos.

6 DISCUSSÃO

Dada a hipótese central da tese, da utilização de elasticidade para fornecer um crescimento automático de uma arquitetura de preservação a longo prazo de Big Data, a parte experimental atingiu resultados satisfatórios. Criou-se uma arquitetura em nuvem de preservação a longo prazo de Big Data com gerenciamento de elasticidade, desta forma permitindo que se faça a preservação de dados estruturados e não-estruturados através da utilização de tecnologias abertas capazes de adaptar sua configuração de acordo com o workload. Sistemas de dados de diversos tipos (streaming, relacionais, não-relacionais e sistemas de arquivos) tiveram seus dados, que eram uma combinação de dados estruturados e não-estruturados, obtidos pelo gateway de arquivamento, que então utilizou uma API REST para fazer a preservação dos objetos em formato SIRF. Os objetos foram preservados em uma camada de armazenamento com nuvem e discos magnéticos. Através do crescimento do volume de dados ao longo do tempo, as mídias de armazenamento muitas vezes atingiram o limite de suas capacidades, o que fez com que a arquitetura provisionasse mais capacidade nas mídias através da expansão de mídias existentes (scale-up) ou do provisionamento de novas mídias (scale-out), ou seja, a camada de armazenamento foi alterada através de um mecanismo de elasticidade. A capacidade das mídias de armazenamento foi monitorada pelos monitores de workload, cujas informações foram captadas por um controlador de elasticidade responsável pelo provisionamento de novos recursos. O controlador de elasticidade decidiu como realizar a elasticidade dos recursos através de uma política de elasticidade. O padrão SIRF foi resultado de uma iniciativa que iniciou-se após obter os resultados de uma pesquisa sobre requisitos de arquivamento (SNIA, 2007). O SIRF foi votado e adotado como padrão por um consórcio sem fins lucrativos de fabricantes de armazenamento e, portanto, é um padrão validado pela indústria. Além disso, o projeto OpenSIRF foi concebido como uma arquitetura de referência do padrão SIRF, validado pelo grupo de trabalho responsável pela criação do formato. Desta forma, o OpenSIRF também foi validado pela indústria através da SNIA.

Outro aspecto importante da tese é a reprodutibilidade. Um dos objetivos da tese foi que pesquisadores e praticantes do domínio da preservação a longo prazo pudessem se basear na proposta de arquitetura para criar seus próprios sistemas e arquiteturas. Na tabela abaixo listou-se algumas características que são pertinentes para a reprodutibilidade da arquitetura, desta forma realizando tais objetivos da tese.

Tabela 6. Características de reprodutibilidade

Características da arquitetura	Impacto para a reprodutibilidade
Código-fonte aberto	Desenvolvedores do domínio de preservação de dados, bem como outros pesquisadores, podem reutilizar o código-fonte da arquitetura ou criar uma nova arquitetura baseada no código-fonte preexistente.
Máquinas virtuais	Desenvolvedores e praticantes do domínio de preservação podem rapidamente instanciar seus próprios ambientes de preservação de Big Data a longo prazo utilizando o OpenSIRF.
Infraestrutura como código	Permite que características da infraestrutura sejam configuráveis e portanto adaptadas, tornando a arquitetura mais flexível e portanto reprodutível.
Documentação	A documentação do OpenSIRF, bem como a tese, que menciona como um método de criação de arquiteturas de referência foi utilizado para criar a arquitetura proposta, permitem que o leitor trace os passos necessários para criar e rodar a arquitetura de preservação.
Anexo da tese	O anexo da tese possui exemplos reais de documentos em formato SIRF que são esperados em um sistema em conformidade com o padrão.
Especificação SIRF aberta	Por ser aberta e texto puro, a especificação SIRF é autocontida e de fácil leitura e entendimento tanto por humanos quanto por máquinas. Através da referência aos elementos da especificação pode-se compreender os metadados necessários para realizar a preservação.

7 CONCLUSÕES

Neste trabalho foi criada uma arquitetura de preservação a longo prazo de Big Data com elasticidade. Arquiteturas preexistentes possuíam características ou foco em preservação de longo prazo de dados. Entretanto, dada a tendência de sistemas de análise de dados de suportar *workloads* contínuos e com grande volume e variação de dados, surgiu o desafio de aumentar a capacidade de armazenamento em tempo de execução enquanto ainda é suportada a análise de dados estruturados e não-estruturados (Big Data).

O capítulo 1 da presente tese foi dedicado à contextualização do problema e à enunciação clara da hipótese e dos objetivos da tese, tendo como base e justificativa outras pesquisas da área. A hipótese apresentada, da utilização de elasticidade em em uma arquitetura de preservação a longo prazo de Big Data, de maneira a ter um sistema capaz de preservar os dados e crescer ao longo do tempo de maneira automática, foi proposta, implementada e testada durante a pesquisa. Os resultados apresentados mostram que a hipótese é válida.

No capítulo 2 foi apresentado o estado da arte, onde as pesquisas relacionadas mais recentes das áreas de preservação de dados, armazenamento e preservação em nuvem foram apresentadas. Em sequência, no capítulo 3, foi feita uma revisão da literatura contendo especificamente as pesquisas e propostas sobre as quais a tese se debruçou, portanto a base fundamental, o ponto de partida da pesquisa.

No capítulo 4 foi apresentada a proposta do trabalho, uma arquitetura cuja base está diretamente relacionada ao formato S3, que também foi desenvolvido pelo autor da pesquisa e que se tornou um padrão da indústria de armazenamento, desta forma contribuindo para um maior respaldo do conteúdo da pesquisa. No capítulo 5 detalhou-se como se deu a aplicação experimental, documentando-se também o código-fonte e as interfaces de aplicação, desta maneira tornado o modelo proposto reproduzível.

Por fim, no capítulo 6 discutiu-se os resultados da pesquisa e comparou-se os resultados obtidos com o estado da arte imediatamente anterior (e concorrente) à pesquisa. Através da discussão apontou-se que a hipótese e os objetivos da tese foram satisfeitos, tendo-se então uma arquitetura reprodutível de preservação a longo prazo de Big Data com gerenciamento de elasticidade, além de outras contribuições que são explicadas a seguir.

7.1 Contribuições

Os seguintes itens são considerados contribuições deste trabalho:

- A proposta de arquitetura em si, que permite a utilização de elasticidade para adaptação automática de *workloads* de preservação a longo prazo de dados, proposta esta que suporta Big Data.
- A implementação de uma arquitetura do padrão SIRF, disponibilizada como um projeto de código livre.
- A documentação do código-fonte e da API da arquitetura.
- A tese em si, que possui informações que permitem que a arquitetura seja reproduzida por praticantes do domínio de preservação a longo prazo de Big Data, bem como outros pesquisadores da academia.
- As máquinas virtuais pré-configuradas que permitem a instanciação de um sistema de preservação sem a necessidade de compilação e implementação.
- A revisão do estado da arte e comparação da tese com outras propostas, que permite avaliar onde as pesquisas se encontram atualmente na área de preservação a longo prazo e quais são os desafios futuros.
- A publicação em anais de um congresso (VIANA e SATO, 2014).

7.2 Trabalhos futuros

A presente tese teve um escopo delimitado pelos objetivos descritos no capítulo 1. É possível aprimorar o modelo de arquitetura aqui proposto. Identificou-se alguns pontos a serem explorados em trabalhos futuros, como:

- Análise de desempenho durante a preservação: este item esteve fora do escopo da tese mas é importante para sistemas em produção. Pode-se avaliar diversos aspectos de desempenho na arquitetura, inclusive o *overhead* do gateway de arquivamento e dos monitores de *workload*.
- Alta disponibilidade: a alta disponibilidade é um requisito de diversas arquiteturas de armazenamento e é citada em pesquisas que compreendem o estado da arte desta tese. Embora não tenha sido abordado o assunto da alta disponibilidade, considera-se que a utilização da elasticidade como um mecanismo de obtenção de alta disponibilidade para preservação é um ângulo de pesquisa que pode ser explorado.
- Spanning de objetos de preservação: dentro do contexto desta tese, embora haja suporte para a utilização de diversos contêineres e mídias diferentes para a preservação, não contemplou-se a utilização de um mecanismo de *spanning*, que é utilizado em outras propostas para que um único objeto de preservação possa ser armazenado de forma fragmentada em diversos contêineres diferentes. Como não é possível prever o tamanho de um objeto de preservação, podendo ser de poucos Kilobytes até centenas de Gigabytes ou, no futuro, Terabytes, faz-se de interesse a utilização de uma técnica de *spanning* como trabalho futuro.

REFERÊNCIAS

ALBONICO, M. MOTTU, J-M. SUNYÉ, G. "Monitoring-Based Testing of Elastic Cloud Computing Applications". ICPE '16 – International Conference on Performance Engineering . Março de 2016. pp. 3-6.

ALNAFOOSI, A.B. STEINBACH, T. "An Integrated Framework for Evaluating Big-Data Storage Solutions – IDA Case Study". *Science and Information Conference (SAI)* 2013. pp. 947-956. 2013

ANDRADE, N.N.G.d. MARTIN, A. MONTELEONE, S. "'All the better to see you with, my dear' Facial recognition and privacy in online social networks". *Security & Privacy*, IEEE. Volume 11 , Issue 3. 2013. pp. 21 - 28

ANGELOV, S. GREEFHORST, P.G. en D. "A framework for analysis and design of software reference architectures", *Information and Software Technology*. Issue 54. 2012, pp. 417-431.

BAKER, M. KEETON, K. MARTIN, S. and "Why traditional storage systems don't help us save stuff forever". Technical Report 2005-120, HP Laboratories Palo Alto, Junho de 2005.

BAKER, M. SHAH, M. ROSENTHAL, D. ROUSSOPOULOS, M. MANIATIS, P. GIULI, T.J. BUNGALÉ, P. "A fresh look at the reliability of long-term digital storage". Proc. European Systems Conference (EuroSys). Abril de 2006.

BASS, L. CLEMENTS, P. KAZMAN, R. *Software Architecture in Practice*, second ed., Addison-Wesley Professional, 2003.

BEATH, C. BECERRA-FERNANDEZ, I. ROSS, J. SHORT, J. "Finding Value in the Information Explosion" *Sloan Management Rev.*, vol. 53, no. 4, 2012, pp. 18-20.

BEERNAERT, L. MATOS, M. VILAÇA, R. OLIVEIRA, R. "Automatic Elasticity in OpenStack". *SDMCMM'12*, 3 e 4 de dezembro de 2012, Montreal, Quebec, Canadá.

BEATH, C. *et al.* "Finding Value in the Information Explosion" *Sloan Management Rev.*, vol. 53, no. 4. 2012, pp. 18-20.

BIER, C. KREMPEL, E. "Common privacy patterns in video surveillance and smart energy". *Computing and Convergence Technology (ICCCT)*, 2012 7th International Conference on. 2012, pp. 610 – 615.

BRUNS, A. WELLER, K. "Twitter as a first draft of the present: and the challenges of preserving it for the future". *WebSci '16 Proceedings of the 8th ACM Conference on Web Science*. 2016, pp. 183-189.

BUTLER, M. REYNOLDS, D. DICKINSON, I. MCBRIDE, B. GROSVENOR, D. SEABORNE, A. "Semantic Middleware for E-Discovery," *icsc*, IEEE International Conference on Semantic Computing. 2009, pp.275-280.

CLUA, O. FELDGEM, M. "Using a research project as classroom support: the case of digital preservation of degraded historic manuscripts at the University of Buenos Aires School of Engineering". *World Engineering Education Conference (EDUNINE)*, IEEE. 2017.

CUZZOCREA, A. BERTINO, E. "Further Theoretical Contributions to a Privacy Preserving Distributed OLAP Framework". *Computer Software and Applications Conference (COMPSAC)*, 2013 IEEE 37th Annual. Publicação em 2013. pp. 234 - 239

DA SILVA JR., L. P. BORGES, M.M. "Fundamental concepts in digital preservation". *TEEM '16 Proceedings of the Fourth International Conference on Technological Ecosystems for Enhancing Multiculturality*. 2016, pp. 285-290.

DOSHI, K.A. ZHONG, T. LU, Z. TANG, Xi. LOU, T. DENG, G. " Blending SQL and NewSQL Approaches: Reference Architectures for Enterprise Big Data Challenges".

Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC), 2013 International Conference on. Publicação em 2013. pp. 163-170.

DUHIGG, C. "How Companies Learn Your Secrets". The New York Times. 16/02/2012.

ENSURE – Enabling kNowledge Sustainability Usability and Recovery for Economic Value. <http://ensure-fp7-plone.fe.up.pt/site/>. Acessado em 20/02/2014.

ESTADOS UNIDOS DA AMÉRICA. **Lei**. Public Law 104-191, 110 Stat. 1936. Publicada em 1996. HEALTH INSURANCE PORTABILITY AND ACCOUNTABILITY ACT. <http://aspe.hhs.gov/admnsimp/pl104191.htm>, acessado em 28/03/2014.

ESTADOS UNIDOS DA AMÉRICA. **Lei**. Public Law 107-204, 116 Stat. 745. Publicada em 2002. SARBANES-OXLEY ACT. <http://www.gpo.gov/fdsys/pkg/PLAW-107publ204/html/PLAW-107publ204.htm>, acessado em 28/03/2014.

ESTADOS UNIDOS DA AMÉRICA. **Lei**. Public Law 91-596, 84 Stat. 1590. Publicada em 1970. Amendada em 2004. OCCUPATIONAL SAFETY AND HEALTH ACT. https://www.osha.gov/pls/oshaweb/owadisp.show_document?p_table=oshact&p_id=2743, acessado em 28/03/2014.

(A) FACTOR, M. NAOR, D. RABINOVICI-COHEN, S. "Preservation DataStores Architecture for Preservation Aware Storage". Mass Storage Systems and Technologies, 2007, 24th IEEE Conference on. 2007. pp. 3-15.

(B) FACTOR, M. NAOR, D. RABINOVICI-COHEN, S. RAMATI, L. RESHEF, P. SATRAN, J. and "The Need for Preservation Aware Storage - A Position Paper". *ACM SIGOPS Operating Systems Review, Special Issue on File and Storage Systems*, Volume 41, Issue 1 (January 2007), pp. 19-23.

FUJITA, Y. NAONO, K. HANAI, T. "Proposal and Evaluation of Metadata Management Method for eDiscovery," srii, pp.778-786, 2012 Annual SRII Global Conference, 2012

GAMMA, E. HELM, R. JOHNSON, R. VLISSIDES, J. *Design Patterns: Elements of Reusable Object-Oriented Software*. Pearson Education, 1994. ISBN 0321700694.

GEERDINK, B. "A reference architecture for big data solutions - Introducing a model to perform predictive analytics using big data technology". Internet Technology and Secured Transactions (ICITST), 2013 8th International Conference for. 9-12 de dezembro de 2013, pp. 71-76.

GHARAIBEH, A. *et al.* "DedupT: Deduplication for Tape Systems". IBM Research Report, 2012

HAN, Z. SUN, Y. WANG, Y. "Audit recommendation for privacy protection in Personal Health Record systems". Computer Supported Cooperative Work in Design (CSCWD), 2013 IEEE 17th International Conference on. 2013. pp. 533 - 538

HARKINS, M. *Managing Risk and Information Security: Protect to Enable*. Apress Media, 2013.

HENNESSY, S.D. LAUER, G.D. ZUNIC, N. GERBER, B. NELSON, A.C. "Data-centric security Integrating data privacy and data security". IBM Journal of Research and Development. Volume 53 , Issue 2. 2009. pp. 2 1 - 2 12.

HERBST, N.R. KOUNEV, S. REUSSNER, R. "Elasticity in Cloud Computing : What It Is and What It Is Not". 10th International Conference on Autonomic Computing (ICAC '13). 2013, pp. 23–28.

HIRANO, M. OGAWA, H. "A Log-structured Block Preservation and Restoration System for Proactive Forensic Data Collection in the Cloud". Availability, Reliability and Security (ARES), 2016 11th International Conference on. 2016.

HUHNLEIN, D. KORTE, U. LANGER, L. WIESMAIER, A. "A Comprehensive Reference Architecture for Trustworthy Long-Term Archiving of Sensitive Data". New

Technologies, Mobility and Security (NTMS), 2009 3rd International Conference on. 2009, pp. 1-5.

ISO 14721:2003, Blue Book. Issue 1. CCSDS, 650.0- B-1: Reference Model for an Open Archival, Information System (OAIS), 2002.

ISO 8601:2004, Data elements and interchange formats -- Information interchange -- Representation of dates and times. http://www.iso.org/iso/catalogue_detail?csnumber=40874, acessado em 07/01/2015.

KHAN, M. RAHMAN, A.U. AWAN, M.D. ALAM, S.M. "Normalizing Digital News-Stories for Preservation". Digital Information Management (ICDIM), 2016 Eleventh International Conference on. 2016.

KRUCHTEN, P. *The Rational Unified Process: An Introduction*, 2e. Addison-Wesley, 2000.

LAGOS, N. VION-DURY, J.Y. "Digital Preservation Based on Contextualized Dependencies". DocEng '16 Proceedings of the 2016 ACM Symposium on Document Engineering. 2016, pp. 35-44.

LEE, T.R. GOO, B.M. KIM, H. SHIN, S.U. "Efficient e-Discovery Process Utilizing Combination Method of Machine Learning Algorithms," cis, pp.1109-1113, 2011 Seventh International Conference on Computational Intelligence and Security, 2011.

(A) LEHRIG, S. EIKERLING, H. BECKER, S. "The CloudScale Method for Software Scalability, Elasticity, and Efficiency Engineering: a Tutorial". ICPE'15, 31 de janeiro a 4 de fevereiro de 2015, Austin, Texas, EUA.

LEHRIG, S. EIKERLING, H. BECKER, S. "Scalability, Elasticity, and Efficiency in Cloud Computing: a Systematic Literature Review of Definitions and Metrics". Proceedings of the 11th International ACM SIGSOFT Conference on Quality of Software Architectures (QoSA '15), Montreal, QC, Canada, 4 a 7 de maio de 2015.

LIBRARY OF CONGRESS. "Understanding PREMIS". Library of Congress Network Development and MARC Standards Office. <https://www.loc.gov/standards/premis/understanding-premis.pdf>, acessado em 05/04/2016.

McGOVERN, N. "Data Rescue. Observations from an Archivist". ACM SIGCAS Computers and Society Homepage archive. Volume 47 Issue 2. Junho de 2017, pp. 19-26.

MERRILL, D.R. "Four Principles for Reducing Total Cost of Ownership". Hitachi Data Systems, 2011.

MOORE, R.W. MARCIANO, R. "Building preservation environments". In JCDL '05: Proceedings of the 5th ACM/IEEE-CS joint conference on Digital libraries. 2005.

NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY (NIST). "Final Version of NIST Cloud Computing Definition Published". Publicado em 25 de outubro de 2011. <http://www.nist.gov/itl/csd/cloud-102511.cfm>, acessado em 03/05/2016.

NGUYEN, Q. LAKE, A. HUBER, M. "Content Server System Architecture for Providing Differentiated Levels of Service in a Digital Preservation Cloud". Cloud Computing (CLOUD), 2011 IEEE International Conference on. Washington, DC. 2011, pp. 557-564.

NGUYEN, Q. LAKE, A. HUBER. M. "Evolvable and Scalable System of Content Servers for a Large Digital Preservation Archives". Systems Conference, 2010 4th Annual IEEE. San Diego, CA. 2010, pp. 306-310.

OPENSTACK. "Swift's Documentation". <http://docs.openstack.org/developer/swift/>, acessado em 18/02/2015.

ORACLE. "What Are RESTful Web Services?". The Java EE 6 Tutorial. <http://docs.oracle.com/javaee/6/tutorial/doc/gijqy.html>, acessado em 12/01/2015.

PEASE, D. RABINOVICI-COHEN, S. "Long-term Retention of Big Data". SNIA SDC Analytics and Big Data Summit, 2012 (slides).

RABINOVICI-COHEN, S. BAKER, M. CUMMINGS, R. FINEBERG, S. MARBERG, J. "Towards SIRF: Self-contained Information Retention Format". SYSTOR '11, May 30 - Jun 01 2011, Haifa, Israel. 2011, pp. 15 1-15 10.

RABINOVICI-COHEN, S. MARBERG, J. NAGIN, K. PEASE, D. "PDS Cloud: Long Term Digital Preservation in the Cloud". Cloud Engineering (IC2E), 2013 IEEE International Conference on. Redwood City, CA. 2013, pp. 38 – 45.

REICHMAN, A. "File storage costs less in the cloud than in-house", Forrester, 25 de agosto de 2011.

ROSA, B. FREDERICO, V. BITTENCOURT, L. PEREIRA, M. HISATOMI, K. "An Experimental Tool for Elasticity Management Through Prediction Mechanisms". IEEE/ACM 7th International Conference on Utility and Cloud Computing , 2014.

RICHTHAMMER, C. NETTER, M. RIESNER, M. PERNUL, G. "Taxonomy for Social Network Data Types from the Viewpoint of Privacy and User Control". Availability, Reliability and Security (ARES), 2013 Eighth International Conference on. 2013, pp. 141 – 150.

SHANKAR, K. "Future proofing the digital society". ACM SIGCAS Computers and Society Homepage archive. Volume 46 Issue 1. Março de 2016, pp. 54-57.

SIMÕES, R. KAMIENSKI, C. "Elasticity Management in Private and Hybrid Clouds ". IEEE International Conference on Cloud Computing . Anchorage, AK, EUA. Julho de 2014, pp. 793-800.

SNIA (Storage Network Industry Association). "Self-contained Information Retention Format (SIRF)". <http://www.snia.org/SIRF>, acessado em 15/12/2014.

SNIA (Storage Network Industry Association). "100 Year Archive Requirements Survey". Publicado em 2007.

https://www.snia.org/sites/default/orig/100YrATF_Archive-Requirements-Survey_20070619.pdf, acessado em 20/07/2016.

STORER, M.W. GREENAN, K.M. MILLER, E.L. "Long-term threats to secure archives". In Proceedings of the 2nd International Workshop on Storage Security and Survivability (StorageSS 2006), Alexandria, VA. Outubro de 2006. pp. 9–16.

STRICOT, M. "Digital Art Preservation: Practical answers to theoretical questions". Digital Heritage International Congress (DigitalHeritage), 2013 (Volume 2). Marseille. 2013, pp. 169-172.

TALLON, P.P. "Corporate Governance of Big Data: Perspectives on Value, Risk, and Cost". IEEE Computer Volume: 46 , Issue: 6, 2013. pp. 32-38.

THIBORDEAU, C. ZANGARO, S. "Big Data Meets Big Cloud Archives" (apresentação). <https://www.brighttalk.com/webcast/499/66961>, Apresentado em 03/13/2013.

TOEROE, M. PAWAR, N. KHENDEK, F. "Managing Application Level Elasticity and Availability". 10th International Conference on Network and Service Management (CNSM) and Workshop. 17 a 21 de novembro de 2014, pp. 348-351 .

VIANA, P. SATO, L. "A proposal for a Reference Architecture for long-term archiving, preservation and retrieval of Big Data". The 13th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (IEEE TrustCom-14). Pequim, China, 24-26 de setembro, 2014 .

WALLACE, C. PORDESCH, U. BRANDNER, R. "Long-term archive service requirements" Request For Comments – RFC 4810, March 2007.

WEBER, K. OTTO, B. ÖSTERLE, H. "One Size Does Not Fit All: A Contingency Approach to Data Governance". ACM J. Data and Information Quality, vol. 1, no. 1, 2009, pp. 1-27.

WEILL, P. WOERNER, S.L. RUBIN, H.A. "Managing the IT Portfolio (Update Circa 2008): It's All about What's New". MIT Center for Information Systems Research (CISR), vol. 8, no. 2B, 2008, pp. 1-4.

WEISS, M. "AXF—Archive eXchange Format: Interchange and Interoperability for Operational Storage and Long-Term Preservation". SMPTE Motion Imaging Journal, Janeiro/Fevereiro de 2014.

YAN, W. YAO, J. CAO, Q. XIE, C. JIANG, H. "ROS: A Rack-based Optical Storage System with Inline Accessibility for Long-Term Data Preservation". EuroSys '17 Proceedings of the Twelfth European Conference on Computer Systems. 2017, pp. 161-174.

ZHANG, Y. LIN, X. ZHANG, W. WANG, J. LIN, Q. "Effectively indexing the uncertain space". IEEE Transactions on Knowledge and Data Engineering, vol. 22, no. 9, pp. 1247-1261, Setembro de 2010.

APÊNDICE A – INTERFACE REST

A seguir, é detalhado o que acontece quando uma determinada requisição é feita à interface REST. É mostrada a assinatura do método no código-fonte que executa as ações necessárias, as ações que ocorrem na camada de armazenamento, os objetos do OpenSIRF Core que são instanciados e suas conversões para JSON e XML.

Os cabeçalhos dos métodos mostrados a seguir contêm:

- As anotações do JAX-RS que definem o tipo de método HTTP que é chamado (@GET, @PUT, etc.), a URI que invocará o método (@Path) e o formato de entrada ou saída do método HTTP (@Produces ou @Consumes).
- A visibilidade dos métodos (todos são públicos).
- O tipo de retorno do método (Response, caso seja uma resposta HTTP, ou algum objeto do OpenSIRF Core).
- O nome do método conforme implementado.
- Os parâmetros de entrada do método, contendo também anotações JAX-RS que indicam se o parâmetro é embutido na URI (@PathParam) ou vem via formulário HTML (@FormDataParam).
- As exceções Java que podem ser lançadas caso um erro ocorra (cláusula throws e seus argumentos).

Após os cabeçalhos dos métodos, são detalhadas as ações que se passam no OpenSIRF Server e como os objetos do OpenSIRF Core são instanciados para cumprir os objetivos do método em questão. Logo em seguida, nas caixas cinzas, é mostrado um exemplo de utilização do método através de chamadas HTTP a um servidor usando o IP \$OPENSIRF_IP e porta \$OPENSIRF_PORT. Tais exemplos são reais e foram extraídos na prática, e seu conteúdo é fiel à saída mostrada pelo comando quando foi executado.

1 Criando um contêiner

```
@PUT
@Path("container/{containername}")
public Response createContainer(@PathParam("containername") String containerName)
    throws IOException, URISyntaxException
```

Quando um pedido de criação de um novo contêiner é recebido, o OpenSIRF Server irá:

- Criar o contêiner no dispositivo de armazenamento subjacente. Por exemplo, um diretório em um sistema de arquivos ou um contêiner Swift no OpenStack Swift.
- Instanciar um objeto `SIRFContainer` e um `MagicObject` associados ao contêiner.
- Criar um objeto `ProvenanceInformation` com as informações padrão de proveniência e o *timestamp* atual.
- Criar um catálogo vazio (objeto do tipo `SIRFCatalog`).
- Adicionar o objeto `ProvenanceInformation` ao catálogo como um PO.
- Fazer o *marshalling* dos objetos para o formato JSON.
- Escrever os documentos JSON para o dispositivo de armazenamento subjacente.
- Retornar HTTP 201 (CREATED), se todo o fluxo acima for bem sucedido, ou outro código em caso de erro.

```
$ curl -i -X PUT http://$OPENSIRF_IP:$OPENSIRF_PORT/opensirf-
jaxrs/sirf/container/Example

HTTP/1.1 201 Created
Server: Apache-Coyote/1.1
Location: http://127.0.0.1:8080/opensirf-jaxrs/sirf/container/Example
Content-Length: 0
Date: Fri, 28 Nov 2014 17:48:46 GMT
```

2 Recuperando o objeto mágico de um contêiner

```
@GET
```

```
@Path("container/{containername}")
@Produces({ MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML })
public MagicObject getMagicObject(@PathParam("containername") String containerName)
```

A obtenção de um objeto mágico depende do contêiner a ser utilizado. No caso da implementação para o OpenStack Swift, o objeto mágico fica nos metadados do contêiner Swift. O processo se dá como a seguir:

- O objeto mágico é obtido (no caso do Swift, os metadados do contêiner Swift são obtidos; em alguns sistemas de arquivos o objeto mágico pode estar salvo como um arquivo).
- O objeto mágico é convertido para formato JSON ou XML.
- O objeto mágico é retornado.

```
$ curl -i -X GET http://$OPENSIRF_IP:$OPENSIRF_PORT/opensirf-
jaxrs/sirf/container/Example

HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Type: application/xml
Content-Length: 202
Date: Fri, 28 Nov 2014 17:49:09 GMT

<?xml version="1.0" encoding="UTF-8" standalone="yes"?
><magicObject><sirfLevel>1</sirfLevel><containerSpecification>0.15</containerSpecif
ication><sirfCatalogId>catalog.json</sirfCatalogId></magicObject>
```

```
$ curl -i -H "Accept:application/json" -X GET http://$OPENSIRF_IP:
$OPENSIRF_PORT/opensirf-jaxrs/sirf/container/Example

HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Type: application/json
Content-Length: 80
Date: Fri, 28 Nov 2014 17:49:26 GMT

{"sirfLevel":"1","containerSpecification":"0.15","sirfCatalogId":"catalog.json"}
```

3 Submetendo um PO ao contêiner

```
@POST
@Consumes(MediaType.MULTIPART_FORM_DATA)
@Path("container/{containername}/{po}")
public Response submitPO(@PathParam("containername") String container,
    @PathParam("po") String poName, @FormDataParam("objectName") String
    objectName, @FormDataParam("inputstream") InputStream inputStream)
    throws IOException, URISyntaxException
```

Um PO é submetido ao OpenSIRF Server usando um formulário do tipo *multipart*. Atualmente este formulário contém como parâmetros de entrada apenas o nome do PO e um fluxo de entrada para o conteúdo binário. Para versões futuras haverá suporte ao recebimento tanto do conteúdo binário quanto do PO completo em formato XML ou JSON, de modo que é possível criar uma instância de um PO em Java no lado do cliente usando OpenSIRF Core e, em seguida, apresentar a versão pós-*marshalling* do objeto para o OpenSIRF Server.

Quando um pedido de submissão é enviado via POST, as seguintes ações ocorrem:

- O fluxo de entrada que contém o conteúdo do arquivo é obtido e guardado temporariamente em um array de bytes.
- O catálogo SIRF é recuperado do contêiner de armazenamento em formato JSON e é feito o *unmarshalling* para um objeto SIRFCatalog.
- Um novo objeto PreservationObjectInformation é instanciado usando o nome do PO enviado no formulário como parte da versão e identificador lógico.
- O *checksum* SHA-1 dos bytes é calculado, e as informações de fixidez são adicionadas ao PO.
- O PO é adicionado ao objeto SIRFCatalog.
- É feito o *marshalling* do catálogo para o formato JSON e ele é então escrito na camada de armazenamento.
- O arquivo com o conteúdo do PO é escrito na camada de armazenamento.

```
$ curl -i -X POST -F inputstream=@Alone.txt -F objectName="EdgarAllanPoe-Alone"
http://$OPENSIRF_IP:$OPENSIRF_PORT/opensirf-
jaxrs/sirf/container/Example/EdgarAllanPoe-Alone

HTTP/1.1 100 Continue

HTTP/1.1 201 Created
Server: Apache-Coyote/1.1
Location: http://127.0.0.1:8080/opensirf-
jaxrs/sirf/container/Example/EdgarAllanPoe-Alone
Content-Length: 0
Date: Fri, 28 Nov 2014 17:49:44 GMT
```

4 Recuperando o catálogo ou metadados de um PO

```
@GET
@Path("container/{containername}/catalog")
@Produces({ MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML })
public SIRFCatalog getCatalog(@PathParam("containername") String containerName)
    throws IOException

@GET
@Path("container/{containername}/{po}")
@Produces({ MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML })
public PreservationObjectInformation getPOMetadata(@PathParam("containername")
    String containerName, @PathParam("po") String poUUID)
    throws IOException
```

Se o OpenSIRF Server recebe uma solicitação para recuperar o catálogo ou metadados de um PO (via GET) as seguintes ações ocorrem:

- O catálogo SIRF é recuperado do contêiner de armazenamento em formato JSON e é feito o *unmarshalling* a um objeto `SIRFCatalog`.
- O objeto (`SIRFCatalog` ou `PreservationObjectInformation`) é retornado ao cliente via HTTP, no formato especificado no cabeçalho `HTTP Accept`.

- Se o cliente usou o cabeçalho "Accept:application/json", o plugin MOXy Jersey irá empacotar o objeto Core para JSON antes de retornar para o cliente.
- Se a opção "Accept:application/xml" é usada, um documento XML será devolvido.
- O formato XML é o padrão e será escolhido caso o cabeçalho Accept não seja especificado.

Abaixo um exemplo de um catálogo sendo obtido em formato JSON.

```
$ curl -i -X GET -H "Accept:application/json" http://$OPENSIRF_IP:
$OPENSIRF_PORT/opensirf-jaxrs/sirf/container/Example/catalog

HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Type: application/json
Content-Length: 2734
Date: Fri, 28 Nov 2014 17:50:55 GMT

{"catalogId":"catalog.json","containerInformation":{"containerAuditLog":
[],"containerIdentifier":
{"identifierType":"containerIdentifier","identifierLocale":"en","identifierValue":"
Example"},"containerProvenance":{"objectIdentifier":{"objectName":
[{"identifierType":"name","identifierLocale":"en","identifierValue":"provenance.po.
json"}],"objectVersionIdentifier":
{"identifierType":"versionIdentifier","identifierLocale":"en","identifierValue":"Ex
ample-provenance.po.json-1.0"},"objectLogicalIdentifier":
{"identifierType":"logicalIdentifier","identifierLocale":"en","identifierValue":"Ex
ample-provenance.po.json"},"objectParentIdentifier":
{"identifierType":"parentIdentifier","identifierLocale":"en","identifierValue":"nul
l"}},"objectProvenanceObjectId":"provenance.po.json"},"containerSpecification":
{"containerSpecificationIdentifier":"SIRF-
0.15","containerSpecificationSirfLevel":"1","containerSpecificationVersion":"0.15"}
,"containerState":
{"containerStateType":"ready","containerStateValue":"true"}}, "objectsSet":
{"objectInformation":[{"objectIdentifiers":[{"objectName":
[{"identifierType":"name","identifierLocale":"en","identifierValue":"EdgarAllanPoe-
Alone"}],"objectVersionIdentifier":
{"identifierType":"versionIdentifier","identifierLocale":"en","identifierValue":"Ex
ample-EdgarAllanPoe-Alone-1.0"},"objectLogicalIdentifier":
{"identifierType":"logicalIdentifier","identifierLocale":"en","identifierValue":"Ex
ample-EdgarAllanPoe-Alone"},"objectParentIdentifier":
```

```
{
  "identifierType": "parentIdentifier",
  "identifierLocale": "en",
  "identifierValue": "null"
}, {
  "objectRelatedObjects": [],
  "objectExtension": [],
  "objectAuditLog": [],
  "creationTimestamp": "20141128154944",
  "modificationTimestamp": "20141128154944",
  "lastAccessTimestamp": "20141128154944",
  "packagingFormat": {
    "packagingFormatName": "none",
    "objectFixity": {
      "lastCheckDate": "20141128154944",
      "digestInformation": [
        {
          "digestOriginator": "ObjectApi",
          "digestAlgorithm": "SHA-1",
          "digestValue": "E0C3782686641E7D36185C1DE8581E59C2FC4D60"
        }
      ]
    }
  },
  "objectIdentifiers": [
    {
      "objectName": {
        "identifierType": "name",
        "identifierLocale": "en",
        "identifierValue": "provenance.po.json"
      },
      "objectVersionIdentifier": {
        "identifierType": "versionIdentifier",
        "identifierLocale": "en",
        "identifierValue": "Example-provenance.po.json-1.0",
        "objectLogicalIdentifier": {
          "identifierType": "logicalIdentifier",
          "identifierLocale": "en",
          "identifierValue": "Example-provenance.po.json",
          "objectParentIdentifier": {
            "identifierType": "parentIdentifier",
            "identifierLocale": "en",
            "identifierValue": "null"
          }
        }
      }
    }
  ],
  "objectRelatedObjects": [],
  "objectExtension": [],
  "objectAuditLog": [],
  "creationTimestamp": "20141128154845",
  "modificationTimestamp": "20141128154845",
  "lastAccessTimestamp": "20141128154845",
  "packagingFormat": {
    "packagingFormatName": "none"
  }
}
}
```

O PO apresentado no exemplo anterior terá o identificador de versão "Example-EdgarAllanPoe-Alone-1.0", que é a concatenação do nome do contêiner, o nome do PO e a versão do PO.

O formato SIRF especifica que o identificador de versão deve ser universalmente único. Portanto, a recuperação de metadados é feita adicionando-se o identificador de versão do PO como um parâmetro embutido na URI. Abaixo um exemplo de recuperação de um PO em formato XML:

```
$ curl -i -X GET http://$OPENSIRF_IP:$OPENSIRF_PORT/opensirf-jaxrs/sirf/container/Example/Example-EdgarAllanPoe-Alone-1.0
```

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Type: application/xml
Content-Length: 1415
Date: Fri, 28 Nov 2014 17:53:31 GMT
```

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<preservationObjectInformation>
  <objectIdentifiers>
    <objectName>
      <identifierType>name
```

```

</identifierType><identifierLocale>en</identifierLocale><identifierValue>EdgarAllan
Poe-
Alone</identifierValue></objectName><objectVersionIdentifier><identifierType>versio
nIdentifier</identifierType><identifierLocale>en</identifierLocale><identifierValue
>Example-EdgarAllanPoe-Alone-
1.0</identifierValue></objectVersionIdentifier><objectLogicalIdentifier><identifier
Type>logicalIdentifier</identifierType><identifierLocale>en</identifierLocale><iden
tifierValue>Example-EdgarAllanPoe-
Alone</identifierValue></objectLogicalIdentifier><objectParentIdentifier><identifie
rType>parentIdentifier</identifierType><identifierLocale>en</identifierLocale><iden
tifierValue>null</identifierValue></objectParentIdentifier></objectIdentifiers><cre
ationTimestamp>20141128154944</creationTimestamp><modificationTimestamp>20141128154
944</modificationTimestamp><lastAccessTimestamp>20141128154944</lastAccessTimestamp
><packagingFormat><packagingFormatName>none</packagingFormatName></packagingFormat>
<objectFixity><lastCheckDate>20141128154944</lastCheckDate><digestInformation><dige
stOriginator>ObjectApi</digestOriginator><digestAlgorithm>SHA-
1</digestAlgorithm><digestValue>E0C3782686641E7D36185C1DE8581E59C2FC4D60</digestVal
ue></digestInformation></objectFixity></preservationObjectInformation>

```

5 Recuperando o conteúdo de um PO

```

@GET
@Produces(MediaType.APPLICATION_OCTET_STREAM)
@Path("container/{containername}/{po}/data")
public Response getPreservationObjectData(@PathParam("containername")
    String containerName, @PathParam("po") String poName)
    throws IOException

```

Dado que um PO é um arquivo comum com metadados adicionados ao catálogo SIRF, para acessar o conteúdo real de um PO, o OpenSIRF Server retornará um fluxo de octetos, semelhante ao que ocorre durante o download de um navegador:

```

$ curl -i -X GET http://$OPENSIRF_IP:$OPENSIRF_PORT/opensirf-
jaxrs/sirf/container/Example/Example-EdgarAllanPoe-Alone-1.0/data

HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
content-disposition: attachment; filename = Example-EdgarAllanPoe-Alone-1.0
Content-Type: application/octet-stream

```



```

Content-Length: 736
Date: Fri, 28 Nov 2014 17:57:42 GMT

From childhood's hour I have not been
As others were; I have not seen
As others saw; I could not bring
My passions from a common spring.
From the same source I have not taken
My sorrow; I could not awaken
My heart to joy at the same tone;
And all I loved, I loved alone.
Then- in my childhood, in the dawn
Of a most stormy life- was drawn
From every depth of good and ill
The mystery which binds me still:
From the torrent, or the fountain,
From the red cliff of the mountain,
From the sun that round me rolled
In its autumn tint of gold,
From the lightning in the sky
As it passed me flying by,
From the thunder and the storm,
And the cloud that took the form
(When the rest of Heaven was blue)
Of a demon in my view.

Edgar Allan Poe

```

6 Listando contêineres

```

@GET
@Path("container")
@Produces({ MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML })
public HashSet<Container> listContainers() throws IOException

```

Esta operação lista todos os contêineres na camada de armazenamento. Por exemplo, todos os diretórios no sistema de arquivos ou todos os contêineres Swift numa instância do OpenStack Swift.

```
$ curl -i -X GET http://$OPENSIRF_IP:$OPENSIRF_PORT/opensirf-jaxrs/sirf/container

HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Type: application/xml
Content-Length: 167
Date: Fri, 28 Nov 2014 18:01:27 GMT

<?xml version="1.0" encoding="UTF-8" standalone="yes"?
><containers><container><name>Example</name></container><container><name>Example2</
name></container></containers>
```

7 Removendo metadados de um PO, conteúdo e metadados de um PO ou um contêiner SIRF

```
@DELETE
@Path("container/{containername}/{po}")
public Response deletePO(@PathParam("containername") String containerName,
    @PathParam("po") String poName) throws IOException, URISyntaxException

@DELETE
@Path("container/{containername}/{po}/data")
public Response deletePOAndMetadata(@PathParam("containername") String
    containerName, @PathParam("po") String poName)
    throws IOException, URISyntaxException

@DELETE
@Path("container/{containername}")
public Response deleteContainer(@PathParam("containername") String containerName)
    throws IOException, URISyntaxException
```

Quando um PO é removido (sem que seus metadados sejam também removidos) ou quando todo um contêiner é removido o OpenSIRF Server realiza as ações diretamente sobre a camada de armazenamento, sem a necessidade de manipulação do catálogo ou quaisquer outros objetos SIRF. Nesses casos a remoção é feita de acordo com o dispositivo de armazenamento onde o PO ou o contêiner se encontram. No caso do OpenStack Swift, por exemplo, um PO é

removido ao deletar-se um objeto em um contêiner Swift; um contêiner SIRF é removido ao deletar-se o contêiner Swift associado o contêiner SIRF.

Já quando um PO é removido juntamente com seus metadados, é necessário que o catálogo seja obtido da camada de armazenamento, os metadados do PO removidos e então o catálogo é novamente escrito.

```
$ curl -i -X DELETE http://$OPENSIRF_IP:$OPENSIRF_PORT/opensirf-jaxrs/sirf/container/Example/Example-EdgarAllanPoe-Alone-1.0
```

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Length: 0
Date: Fri, 28 Nov 2014 18:08:04 GMT
```

```
$ curl -i -X DELETE http://$OPENSIRF_IP:$OPENSIRF_PORT/opensirf-jaxrs/sirf/container/Example/Example-EdgarAllanPoe-Alone-1.0/data
```

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Length: 0
Date: Fri, 28 Nov 2014 18:08:07 GMT
```

```
$ curl -i -X DELETE http://$OPENSIRF_IP:$OPENSIRF_PORT/opensirf-jaxrs/sirf/container/Example
```

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Length: 0
Date: Fri, 28 Nov 2014 18:08:13 GMT
```