Universidade Federal do Amazonas
Instituto de Computação
Programa de Pós-Graduação em Informática

Rayol de Mendonça Neto

Um Serviço de Diretórios Tolerante a Falhas e Intrusões

Manaus
Maio de 2016

Rayol de Mendonça Neto

Um Serviço de Diretórios Tolerante a Falhas e Intrusões

Trabalho apresentado ao Programa de Pós-Graduação em Informática do Instituto de Computação da Universidade Federal do Amazonas como requisito parcial para obtenção do grau de Mestre em Informática.

Orientador: Prof. Dr. Eduardo Luzeiro Feitosa

Manaus
Maio de 2016

Ficha Catalográfica

Ficha catalográfica elaborada automaticamente de acordo com os dados fornecidos pelo(a) autor(a).

Mendonca Neto, Rayol de
M539u Um serviço de diretórios tolerante a falhas e intrusões / Rayol de Mendonca Neto. 2016
75 f.: il. color; 31 cm.

Orientador: Eduardo Luzeiro Feitosa
Dissertação (Mestrado em Informática) - Universidade Federal do Amazonas.

1. Replicação de Máquinas de estado . 2. Diversidade. 3. Tolerância a Intrusão. 4. Serviços de diretório. 5. Protocolo LDAP. I. Feitosa, Eduardo Luzeiro II. Universidade Federal do Amazonas III. Título

FOLHA DE APROVAÇÃO

"Um Serviço de Diretórios Tolerante à Falhas e Intrusões"

RAYOL DE MENDONÇA NETO

Dissertação de Mestrado defendida e aprovada pela banca examinadora constituída pelos Professores:

Prof. Eduardo Luzeiro Feitosa - PRESIDENTE

Prof. Eduardo Freire Nakamura - MEMBRO INTERNO

Prof. José Luiz de Souza Pio - MEMBRO EXTERNO

Manaus, 17 de Maio de 2016

Aos meus pais e meus avós.

"Se vi mais longe foi por estar
de pé sobre ombros de gigantes."
(Isaac Newton)

Agradecimentos

Agradeço primeiramente à Deus.

Aos meus pais e minha avó que sempre estiveram comigo em todos os momentos, principalmente na vida escolar. Os ensinamentos deles sempre me levaram a um pensamento: “O sucesso começa no estudo”. Essa vitória não é exclusivamente minha, é nossa.

A todos os meu familiares, meu irmão Matheus, tios e primos, que me incentivaram e torceram por mim ao longo desses dois anos.

A minha noiva Luciana Mendes, minha companheira e maior incentivadora, que sempre me apoiou em todas as decisões que tomei e que em momento algum deixou de caminhar lado a lado comigo.

Ao Professor Eduardo Luzeiro Feitosa, meu mestre e mentor, que com toda sua sabedoria e serenidade soube me guiar com maestria para superar todos os desafios e assim alcançar os objetivos da pesquisa.

A todos os Professores do Mestrado que de alguma forma contribuíram para a minha formação.

Aos companheiros de pesquisa, Professor Aldri Santos, Professor Diego Kreutz e Thais Almeida que trabalharam junto comigo e tornaram o desenvolvimento da pesquisa mais prazeroso.

Aos colegas de mestrado Adria Menezes, Adriana Saraiva, Awdren Fontão, Bernardo Gatto, Caio Gregoratto, Carlos Ramos, Delano Campos, Isomar, Janaína Nascimento, Michel Yvano, Pablo e Thais Oliveira, aos colegas do laboratório ETSS, aos novos colegas que tive o prazer de conviver no final do mestrado e a tantos outros que deixei de citar, mas percorreram essa jornada junto comigo.

Aos funcionários do ICOMP, pela disponibilidade, simpatia e gentileza.

Resumo

Serviços de diretório são frequentemente usados para armazenar informações sensíveis (e.g., dados e credenciais de usuários) nos mais variados sistemas críticos, tais como serviços de autenticação e controle de acesso, servidores DNS, e-mail e infra-estrutura de chaves públicas. Embora possuam um certo grau de segurança para evitar ataques e falhas (e.g., protocolos de segurança e mecanismos próprios de replicação de dados), as atuais implementações de serviços de diretórios, baseadas no protocolo LDAP, não são de fato seguras, escaláveis e livres de intrusões. É neste contexto que esta dissertação apresenta a primeira arquitetura para serviços de diretórios, baseado no protocolo LDAP, capaz de tolerar falhas e intrusões. Para o desenvolvimento desta arquitetura é empregada a replicação de máquinas de estado, para tolerar falhas, e a diversidade de sistemas, em diferentes níveis, para tolerar intrusões. A viabilidade e aplicabilidade da solução proposta é demonstrada através de diversos experimentos. Como resultado, é possível atestar que o sistema é eficiente, com um desempenho no mínimo três vezes melhor que o estado da arte, bem como escalável e resiliente.
Palavras-Chave: Replicação de Máquinas de estado, Diversidade, Tolerância a Intrusão, Serviços de diretório, LDAP.

Abstract

Directory services are often used to store sensitive information (e.g., data and user credentials) in many critical systems such as access control and authentication services, DNS servers, e-mail and public key infrastructures. Although they provide a certain degree of security to prevent attacks and failures (e.g., security protocols and self replication mechanisms), the current directory services implementations, based on LDAP, they are not safe in fact, scalable and intrusions free. In this context, this work presents the first architecture for directory services based on the LDAP protocol, capable to tolerate failures and intrusions. For the development of this architecture is employed state machine replication to tolerate failure, and the diversity of systems at different levels to tolerate intrusions. The feasibility and applicability of the proposed solution is shown through several experiments. As a result, it is possible to attest that the system is efficient, with a performance at least three times better than the state of the art, as well as scalable and resilient.

Keywords: State Machine Replication, Diversity, Intrusion tolerance, Directory Services, LDAP.

Lista de Figuras

Lista de Tabelas

Nomenclatura

API	Application Programming Interface
BFT	Byzantine Fault Tolerant
CLBFT	Castro-Liskov Byzantine-Fault-Tolerant
DIT	Directory Information Tree
DN	Distinguished Name
JNDI	Jana Naming and Directory Interface
JVM	Java Virtual Machine
LDAP	Lightweight Directory Access Protocol
LDIF	LDAP Data Interchange Format
PBFT	Practical Byzantine Fault Tolerance
PKI	Public Key Infrastructure
RIT-LDAP	Resilient and Intrusion Tolerant - LDAP
SASL	Simple Authentication and Security Layer
SMR	State Machine Replication
SPI	Service Provider Interface
SSO	Single-Sign-On
TLS	Transport Layer Security

Sumário

Capítulo 1

Introdução

É notório que hoje em dia os usuários exigem cada vez mais dos serviços disponíveis em rede, em especial na Internet. Normalmente, o que buscam é acesso rápido, eficiente e simplificado aos dados e serviços. Um exemplo clássico são os mecanismos de *single-sign-on* (SSO), que permitem a um usuário utilizar diferentes serviços com um único processo de autenticação.

Uma abordagem legada, mas bastante utilizada para fornecer esses "requisitos" é o uso de serviços de diretórios, uma espécie de banco de dados otimizado para consultas. Em linhas gerais, diretórios são: (i) dinâmicos, uma vez que novos conteúdos podem ser adicionados sem parar o serviço; (ii) flexíveis, já que podem armazenar quase qualquer tipo de dados; (iii) e eficientes, pois realizam operações de busca mais rápido que banco de dados relacionais. Prova da versatilidade dos serviços de diretório está na sua vasta gama de aplicações, que incluem serviços de rede (e-mail, DNS e DHCP, por exemplo), mecanismos de controle de acesso [1], infraestruturas de chave pública (PKI - Public Key Infrastructure) [2], soluções de interoperabilidade entre sistemas [3] e acesso seguro e flexível a certificados em WS-Security (*Web Services Security*) [4].

Existem diferentes implementações de serviços de diretórios como eDirectory, Active Directory, NIS e StreeTalk [5], mas o LDAP (*Lightweight Directory Access Protocol*) [6] é o protocolo mais difundido e utilizado na implementação dos serviços de diretório e no suporte a interoperabilidade de sistemas. Entre as justificativas, pode-se enumerar ser de código aberto e amplamente suportado pela indústria, uma vez que permite acessar serviços de diretórios que estão em conformidade com o modelo de dados X.500[1].

Embora os serviços de diretório, especialmente os baseados no LDAP, forneçam acesso rápido e eficiente, garantir a disponibilidade do serviço e a segurança

[1] X.500 é uma série de padrões sobre serviços de diretórios, da *International Telecommunication Union* (ITU)

dos dados é essencial, já que podem conter informações privilegiadas como dados pessoais e empresariais. Em outras palavras, estes serviços precisam ser tolerantes a falhas e intrusões.

É neste cenário que esta dissertação se enquadra ao tentar fornecer uma solução para prover segurança e resiliência [2] em diretórios LDAP.

1.1 Motivação

Embora não seja a função principal, serviços de diretório possuem mecanismos de segurança a fim de evitar ataques e falhas. Por exemplo, diretórios baseado no LDAP fazem uso de alguns protocolos de segurança. De acordo com suas especificações, o LDAP [6] tem suporte aos protocolos SASL (*Simple Authentication and Security Layer*) e TLS (*Transport Layer Security*), com o intuito de proporcionar maior segurança tanto para o serviço de diretório quanto para a comunicação feita entre ele e os sistemas. O SASL fornece uma interface estruturada entre protocolos e mecanismos, permitindo assim que novos protocolos possam reutilizar mecanismos já existentes [8]. Já o objetivo do TLS é proporcionar a privacidade dos dados e a integridade da comunicação entre duas aplicações [9].

Além desses protocolos, a maioria das implementações LDAP possuem mecanismos próprios de replicação de dados. No caso específico da implementação LDAP mais conhecida e usada, o OpenLDAP, existem as replicações *Single-Master* e *Multi-Master*. Na primeira, apenas um servidor, chamado de mestre, pode efetuar operações de escrita e, por isso, case falhe nenhuma operação de escrita poderá ser feita. Na segunda, todos os servidores podem fazer leituras e escritas de dados, mas caso haja uma intrusão em qualquer servidor, todo o sistema será comprometido.

Percebendo essas falhas, pesquisadores como Wang et al [10], Hou et al [11] e Shoker e Bahsoun [12] vêm procurando soluções para aumentar a segurança e resiliência em diretórios LDAP. Para tanto, eles têm feito uso de protocolos BFT (*Byzantine Fault Tolerant*) para tolerar falhas Bizantinas[3]. Embora interessantes, estes trabalhos falham em dois aspectos. Primeiro, eles não tiram vantagem de múltiplas infraestruturas e nuvens (e.g., *data centers*) para aumentar a robustez e tolerar diferentes tipos de falhas. Pelo contrário, são implantados em uma única infraestrutura física (e.g., uma única máquina física ou múltiplos servidores num

[2] Resiliência é a persistência em evitar falhas frequentes ou graves, quando o sistema está diante de adversidades [7].

[3] Falhas Bizantinas são aquelas que permitem ao sistema se comportar de maneira aleatória (podendo transmitir respostas arbitrárias em tempos arbitrários). Em outras palavras, permite que um processo pare ou tome um passo errado, impedindo assim que o sistema funcione conforme a sua especificação [13, 14].

único *data center*), onde uma simples queda de energia pode impactar muitos usuários. O segundo, e pior, nenhuma dessas propostas é tolerante à intrusão.

Neste contexto, esta dissertação apresenta o primeiro serviço de diretórios LDAP tolerante a falhas e intrusões (resiliente), denominado RIT-LDAP (*Resilient and Intrusion Tolerant - LDAP*), que combina técnicas avançadas de sistemas distribuídos para garantir confiabilidade e segurança. Em outras palavras, a proposta do RIT-LDAP baseia-se: (i) no uso de um protocolo BFT, para garantir tolerância a falhas; (ii) diversidade de componentes, para garantir tolerância a intrusões; e (iii) uso de infraestrutura de nuvem, para garantir maior disponibilidade, fornecendo um esquema de diretório LDAP seguro e confiável.

1.2 Objetivos

O objetivo dessa dissertação é desenvolver um serviço de diretório LDAP resiliente, que utiliza técnicas de replicação e diversidade de sistemas para tolerar falhas bizantinas e intrusões.

Especificamente, pretende-se:

- Prover mecanismos que forneçam tolerância a intrusão, bem como alta disponibilidade ao ambiente desenvolvido;
- Adaptar um modelo funcional já existente com intuito de criar serviços de diretório resilientes, tolerantes a falhas e intrusões em infraestruturas de redes.
- Avaliar a viabilidade do diretório LDAP resiliente proposto em diferentes configurações.

1.3 Contribuições

Esta seção apresenta as principais contribuições deste trabalho:

1. O projeto e implementação de um serviço de diretórios resiliente de acordo com as especificação do protocolo LDAP versão 3.0;
2. Implementação de um serviço de diretórios tolerante a falhas e intrusões que mantém compatibilidade com sistemas existentes, ou seja, a implantação do sistema não requer nenhuma modificação no lado do cliente;
3. Análise da escalabilidade do sistema, assim como a comparação com o estado da arte. Além da avaliação do sistema mediante falhas nas réplicas;

4. Avaliação de desempenho do sistema e comportamento com diferentes configurações, tais como mudanças de sistemas operacionais e serviços de diretórios.

Além das contribuições enumeradas, podemos destacar também a publicação alcançada em uma conferência:

1. Mendonca-Neto, R., Barreto, B., Kreutz, D., Santos, A., and Feitosa, E. (2015). Fit-LDAP: Um serviço de diretório tolerante a falhas e intrusões. *Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais*, páginas 44 - 57. SBC.

1.4 Estrutura da Dissertação

Esta dissertação está organizada da seguinte forma: O Capítulo 2 apresenta os conceitos básicos com relação aos diretórios LDAP e alguns conceitos relacionados a replicação de máquinas de estado além de técnicas de tolerância a intrusão. Trabalhos relacionados sobre soluções LDAP usando replicação e protocolos BFT são brevemente descritos no Capítulo 3. O Capítulo 4 apresenta a arquitetura proposta e seu funcionamento. Os resultados dos experimentos são discutidos no Capítulo 5. Por fim, no Capítulo 6 são apresentadas as conclusões, dificuldades encontradas e trabalhos futuros.

Capítulo 2

Fundamentação Teórica

Este Capítulo apresenta os conceitos necessários para o entendimento desta dissertação. Primeiro, o protocolo LDAP é explicado, incluindo versões, estrutura, operações e principais implementações disponíveis. Em seguida, as técnicas de replicação de máquina de estados e de diversidade, utilizadas para garantir a tolerância a falhas e intrusões, são discutidas. Por fim, a biblioteca BFT-SMaRt, base da implementação proposta, é apresentada.

2.1 LDAP

Lightweight Directory Access Protocol (LDAP) é um protocolo simples e funcional para acessar serviços de diretórios baseados no padrão X.500. Especificado como um padrão pelo IETF (*Internet Engineering Task Force*) [6], o LDAP é um protocolo de comunicação que funciona sobre a pilha TCP/IP ou outro serviço de transporte orientado a conexão. Mais especificamente, define o transporte e o formato das mensagens usados por um cliente para acessar diretórios x.500, embora não defina o serviço de diretório propriamente dito.

O LDAP utiliza o modelo cliente-servidor (Figura 2.1). O cliente se conecta ao servidor e faz uma requisição. Ao receber essa requisição, o servidor a processa e responde ao cliente com a devida resposta ou um ponteiro indicando onde a informação solicitada pode ser obtida (normalmente um outro servidor LDAP). O LDAP aceita qualquer dado que o usuário deseje armazenar. Para tanto, faz uso de *schemas*, arquivos que definem a estrutura das entradas e dos atributos. Esta é uma importante característica dos serviços de diretórios.

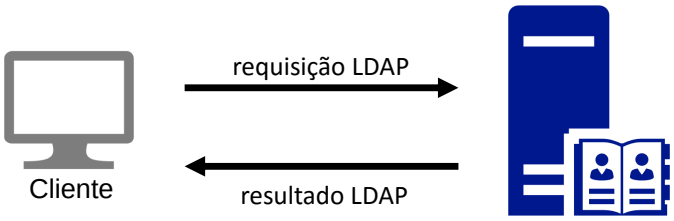


Figura 2.1: Comunicação LDAP. Adaptado de Couloris *et al* [13].

2.1.1 Versão

Atualmente, o LDAP está na terceira versão, definida na RFC4511 [6]. Desenvolvida no final dos anos 1990, a versão 3 acrescenta características como:

- Serviço de autenticação e segurança de dados fortes via SASL, a fim de oferecer uma maneira de adicionar mecanismos de autenticação ao protocolo;
- Autenticação de certificados e serviços de segurança de dados via TLS e SSL (*Secure Socket Layer*)[1];
- Internacionalização através do uso de Unicode[2];
- Referências e Continuações, para indicar a localização da entrada em outro serviço de diretório.
- Extensibilidade, uma vez que permite que os argumentos das operações LDAP podem ser estendidos, ou seja, podem ser inseridos a uma única mensagem LDAP.

LDAP também suporta criptografia de sessão para a privacidade. Todas a informações trocadas durante a sessão entre o cliente e o servidor podem ser encriptadas para evitar interpretação das mensagens.

2.1.2 Estrutura de Dados do LDAP

Diferentemente dos bancos de dados relacionais, que armazenam seus dados em colunas e linhas, o diretório é organizado em uma estrutura de árvore conhecida

[1]SSL gera uma sessão criptografada que proporciona uma maior privacidade para as informações do diretório.

[2]Unicode é um padrão internacional que possibilita com que todos os caracteres das linguagens escritas possam ser representados em computadores [15].

como DIT (*Directory Information Tree*), onde os nós são as entradas e o primeiro nó é conhecido como nó raiz (sufixo)[6]. Uma entrada LDAP é uma coleção de informações sobre uma entidade. Cada entrada consiste de três componentes principais: o nome distinto (DN - *Distinguished Name*), uma coleção de atributos e uma coleção de *object classes*.

A Figura 2.2 ilustra um exemplo de diretório LDAP.

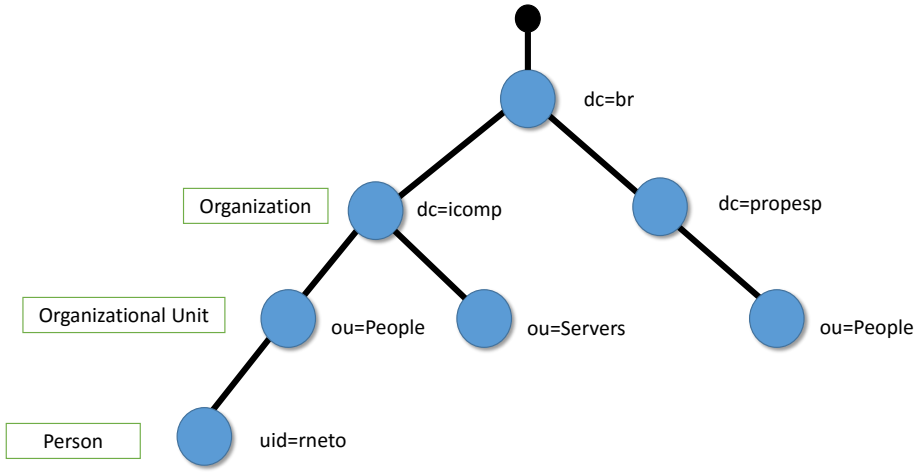


Figura 2.2: Estrutura do LDAP.

Um nome distinto (DN) tanto identifica de forma única uma entrada como também descreve sua posição na DIT.

Por exemplo, na Figura 2.2, o DN $uid = rneto, ou = People, dc = icomp, dc = br$ representa uma entrada que está imediatamente subordinada a entrada $ou = People, dc = icomp, dc = br$, que esta é imediatamente subordinada a entrada $dc = br$.

Uma entrada é composta de um ou mais atributos. Cada atributo possui: um tipo de atributo; zero ou mais opções de atributos; e um conjunto de valores que são de fato o dado do atributo. Os tipos de atributo são elementos do *schema*, que especificam como um atributo deve ser tratado tanto pelo cliente quanto pelo servidor LDAP. Todos os tipos de atributo devem possuir um OID (*object identifier*). Também devem possuir uma sintaxe, que especifica o tipo de dado que pode ser armazenado, além de indicar se um atributo pode ter múltiplos valores na mesma entrada.

Object Classes são elementos do *schema* que especificam coleções de tipos de atributos que podem estar relacionados a uma entrada. Cada entrada tem um *object class* estrutural, que indica que tipo de objeto uma entrada representa (e.g., pode ser informações sobre uma pessoa, um grupo, um dispositivo, um

serviço, etc.).

Além disso, a DIT pode determinar como os dados são particionados entre múltiplos serviços de diretórios, onde cada partição existente é conhecida como *host* e cada *host* é um subárvore (Figura 2.3).

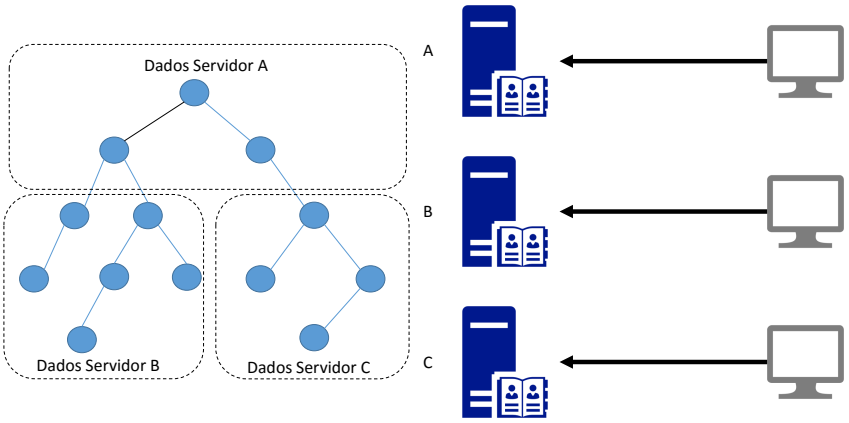


Figura 2.3: DIT distribuída. Adaptado de Howes *et al* [5].

Em um serviço de diretório, os *hosts* podem possuir diferentes porções da árvore de diretório, ou seja, cada *host* somente acessa uma determinada parte da DIT, não tendo acesso a ela por completo. A Figura 2.3 ilustra como os dados são disponibilizados entre os *hosts* existentes. No caso da Figura, as entradas de usuário são separadas das entradas de impressoras. Existem diversas razões para se implementar uma árvore de diretórios distribuída em múltiplos *hosts*[16], entre as que mais se destacam estão:

- **Desempenho** - Um determinada seção da DIT pode ser mais utilizada que outras. Alocando-se essa seção em um *host* separado, permite-se que os usuários acessem os dados dessa subárvore mais rapidamente.
- **Localidade Geográfica** - Os serviços de diretórios podem ser globais, tendo clientes em localidades diferentes. Com isso, é interessante que determinados *hosts* sejam alocados em uma região mais próxima de onde estão todos os seus usuários, para poder evitar demora no acesso (e.g., Uma multinacional alocar um *host* no Brasil para acesso dos clientes brasileiros).
- **Limites Administrativos** - Quando serviços de diretórios são muito extensos, é mais apropriado delegar o controle administrativo de um *host* do

diretório a um determinado grupo administrativo. Com isso, os administradores dessa subárvore podem aplicar quaisquer configurações administrativas (e.g., replicações e *backups*) sem interferir no serviço de diretório como um todo.

2.1.3 Schemas

Os *Schemas* de um diretório são um conjunto de definições e restrições sobre a estrutura da DIT, incluindo as maneiras como as entradas podem ser chamadas, a informação que pode ser armazenada em uma entrada, os atributos usados para representar essa informação e sua organização hierárquica para facilitar a pesquisa e a recuperação da informação. Além disso, define as maneiras como os valores dos atributos podem ser combinados com outros atributos e suas regras de correspondência [17]. Em outras palavras, *Schemas* é modo padrão para apresentar a estrutura dos objetos que podem ser armazenados dentro de diretórios.

Em geral, *Schemas* LDAP são compactos e utilizam-se basicamente de duas diretivas: *Attribute Type* e *Object Class*. A primeira define um atributo, incluindo ainda os nomes dos atributos que poderá conter. A segunda especifica o nome da classe do objeto, os atributos obrigatórios e opcionais, e seu tipo de objeto.

Para resumir, *Schemas* LDAP são usados para definir atributos, classes de objetos e várias outras regras que determinam a estrutura da árvore de informações do diretório.

2.1.4 LDIF

Arquivos LDIF (*LDAP Data Interchange Format*), definidos na RFC 2849 [18], são arquivos de texto que servem para descrever as informações de configuração e os conteúdos de um diretório LDAP [18]. Contudo, não servem somente para representar o conteúdo de um diretório. Servem também para representar certas operações no LDAP, como adições, alterações e exclusões, além de ser o único meio de entrada de dados em um diretório LDAP e a única forma possível de importar e exportar dados LDAP [19]. Os dados contidos no arquivo LDIF devem obedecer as regras do *schema* do seu diretório LDAP [16]. Um dos principais usos do LDIF é quando um diretório LDAP é carregado pela primeira vez ou quando muitas entradas precisam ser alteradas ao mesmo tempo. Em outras palavras, o LDIF permite uma fácil manipulação de quantidades maciças de dados.

2.1.5 Operações LDAP

O LDAP possui nove operações divididas em três categorias principais [6]:

1. **Operações de interrogatório (*search* e *compare*)**: aquelas que permitem recuperar dados do diretório. ***Search*** permite ao cliente encontrar entradas no diretório enquanto ***Compare*** permite ao cliente testar se uma entrada contém ou não um atributo específico.

2. **Operações de atualização (*add*, *delete*, *rename* e *modify*)**: são responsáveis por atualizar as informações já contidas no diretório. ***Add*** adiciona uma entrada nova ou um atributo, ***Delete*** permite excluir uma entrada no serviço de diretório, ***Rename*** renomeia um atributo de uma entrada ou a entrada em si, e ***Modify*** altera os valores de uma entrada/atributo.

3. **Operações de autenticação e controle (*bind*, *unbind*, e *abandon*)**: são responsáveis pela autenticação, encerrar as sessões e abandonar uma operação anterior. ***Bind*** é usado para autenticar o cliente, ***Unbind*** é usado pelo servidor para devoluções de quaisquer informações de autenticação que encerram qualquer operação e desconectam o cliente, e ***Abandon*** permite ao cliente abandonar a operação e o servidor LDAP não processa mais a operação que foi previamente solicitada pelo cliente.

2.1.6 Implementações de diretórios LDAP

Os diretórios LDAP possuem diversas implementações, cada uma com uma característica específica. Dentre as que se destacam mais, estão: (i) OpenLDAP, por ser a mais utilizada e difundida; (ii) 389-Directory, uma derivação do OpenLDAP mantida pela *Red Hat*; e (iii) OpenDJ, por ser uma implementação em Java e com isso, ser independente de sistemas.

OpenDJ

OpenDJ é um serviço de diretório oriundo de um projeto anterior, chamado OpenDS, e tem as mesmas raízes do *Oracle Unified Directory*, uma vez que foi herdado da *Sun Microsystems*.

O OpenDJ está disponível em módulos, que incluem o serviço de diretório OpenDJ, ferramentas do usuário e um SDK (*Software Development Kit*) LDAP. Um dos pontos que mais se destacam do OpenDJ é que todos os módulos do sistema são baseados em JAVA.

As principais características do OpenDJ são: (i) compatível com o LDAPv3; (ii) desenvolvido em plataforma java, provendo assim alto desempenho, alta disponibilidade e segurança dos dados; e (iii) possuir fácil instalação, além de ser simples e rápido.

OpenLDAP

OpenLDAP foi desenvolvido inicialmente pela Universidade de Michigan e é o servidor LDAP mais difundido entre as implementações *open-source* existentes. Está disponível em muitas distribuições do Linux e Unix. O OpenLDAP é composto basicamente por dois elementos: (i) SLAPD (*Stand Alone LDAP Daemon*) - Servidor; (ii) SLURPD (*Stand Alone LDAP Update Replication Daemon*) - Bibliotecas diversas de implementação do protocolo LDAP. Além disso, também são disponibilizadas ferramentas, utilitários diversos e clientes de exemplo.

Entre as características que mais se destacam estão:

- Suporte a IPv4 e IPv6;
- Autenticação (Cyrus Sasl-Kerberos V, GSSAPI, Digest-MD5);
- Segurança no transporte SSL e TLS;
- Controle de acessos;
- Capacidade de atender a múltiplos bancos de dados simultaneamente;
- Alta performance em múltiplas chamadas e replicação de base.

389-Directory

O 389 Directory é um produto da Red Hat, que teve sua origem junto com o OpenLDAP a partir do projeto *slapd* desenvolvido pela Universidade de Michigan. Utiliza o banco de dados Berkley para armazenar seus dados.

Principais características:

- Desempenho: É capaz de processar milhares de operações por segundos;
- Confiabilidade: Serviço de diretório estável, com mecanismos de tolerância (e.g, replicação *Multi-Master*)a falhas e mecanismos de *backup*;
- Escalabilidade: De fácil configuração para expansão do serviço de diretório.

2.2 Técnicas de Tolerância a Falhas e Intrusões

2.2.1 Replicação de Máquinas de Estado

De acordo com o Dicionário Oxford [20], o termo réplica se refere a reprodução idêntica. No contexto da computação, replicação refere-se a vários serviços executando, em computadores distintos, as mesmas tarefas com o mesmo propósito.

As replicações são utilizadas para diversos objetivos. Por exemplo, em sistemas distribuídos seu propósito é ser tolerante à falhas. Já em bancos de dados sua utilização é principalmente por razões de desempenho. De modo geral, replicação é muito usada para melhorar a robustez de sistemas no que se refere a disponibilidade e segurança.

Dentre os diversos tipos de soluções de replicação, destacam-se:

- **Replicação Passiva**, também conhecida como backup primário, é uma forma de replicação no qual os usuários enviam suas requisições a um servidor e o mesmo transmite a atualização para os servidores de backup [21]. Porém, caso haja uma falha na réplica primária, o sistema sofre de um elevado custo de reconfiguração, apresentando assim um atraso significativo na operação realizada pelo cliente;

- ***Quorum System*** explora a ideia de intercessão entre leitura w e escrita r para garantir a consistência e a disponibilidade de um dado replicado. Para garantir a consistência, a soma de $w + r$ tem que ser maior do que o número de réplicas de dados de um objeto específico [22]. Por exemplo, para um sistema de armazenamento distribuído de leitura/escrita de 3 processos, escrever um valor em qualquer 2 processos (maioria) garante que cada lida depois por quaisquer 2 processos irá retornar um valor consistente. Contudo, no *quorum system*, uma escolha de votação errada por parte do algoritmo pode levar a conflitos de escrita, prejudicando assim o usuário com uma resposta incorreta;

- **Replicação Ativa**, também chamada de abordagem de máquinas de estado, tem como conceito principal que todas as réplicas recebem e processam na mesma ordem e sequência os pedidos dos clientes [21]. A consistência é garantida assumindo que quando as réplicas recebem a mesma entrada na mesma ordem, elas produzirão o mesmo resultado. Os clientes não tratam o sistema como um conjunto de réplicas, mas sim como um sistema só que é responsável por propagar a mensagem do cliente para todas as réplicas. E isto pode ser feito através de um *broadcast* atômico.

Como a proposta dos serviços de diretórios é apresentar eficiência na consulta, a replicação ativa é a mais indicada, sem mencionar que é o método mais usado na implementação de sistemas tolerantes a falhas, tanto por parada [23] quanto bizantinas [24]. Basicamente, a replicação de máquina de estados exige que todas as réplicas de um serviço comecem com um estado inicial e executem todas as instruções (pedidos dos clientes) em ordem, provendo assim o mesmo resultado [22].

Pode-se afirmar que para haver uma replicação de máquinas de estado, três propriedades devem ser seguidas: (1) todas as réplicas são iniciadas a partir do

mesmo estado; (2) são executados o mesmo conjunto de requisições e na mesma ordem; (3) e todas as réplicas tem que chegar ao mesmo estado final, o que define o determinismo das réplicas.

A Figura 2.4 apresenta as 5 etapas para a realização da Replicação de Máquinas de Estado.

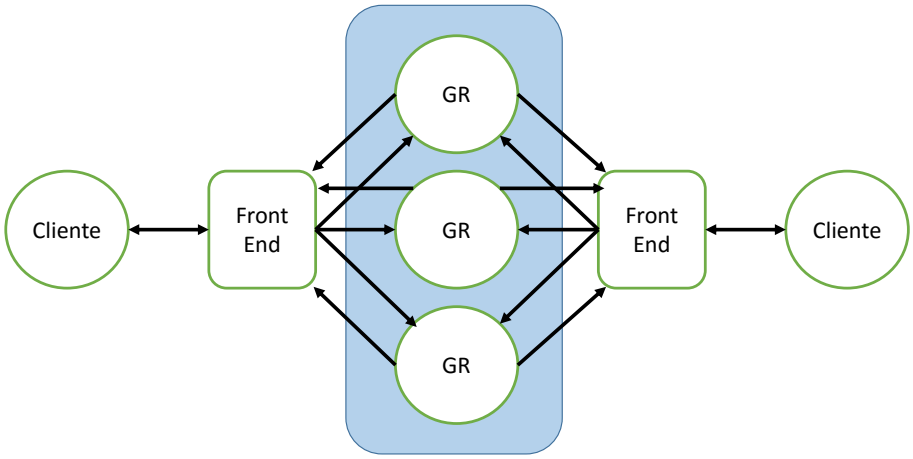


Figura 2.4: Replicação de Máquinas de Estado. Adaptado de Couloris *et al* [13].

1. **Requisição** - O *Front End* anexa um identificador único na requisição do cliente e transmite a mensagem para o grupo de gerenciadores (GR) das réplicas, utilizando um algoritmo de difusão atômica.

2. **Coordenação** - A ordem das mensagens é dada pela propriedade de ordenamento total do algoritmo de difusão atômica.

3. **Execução** - Todas as réplicas executam a requisição na ordem em que são recebidas.

4. **Acordo** - Não é necessária a fase de acordo, pois todas as réplicas processam a mesma requisição na mesma ordem. Como as réplicas são deterministas, todas elas produzem os mesmos resultados.

5. **Resposta** - Todas as réplicas enviam o resultado da requisição de volta ao *Front-end*. O número de respostas que o *Front-end* espera depende do número de falhas que o sistema tolera. Assim que o sistema recebe o número mínimo de respostas para realizar o consenso, as outras respostas são descartadas. Por fim, o cliente aguarda apenas a resposta vinda do *Front-end*.

2.2.2 Diversidade

Protocolos BFT não são suficientes para alcançar a tolerância a intrusão em um sistema, pois são implementados apenas para tolerar algumas falhas, entre elas as bizantinas. Com base nessa assertiva um método para diminuir a probabilidade de vulnerabilidade comuns nas réplicas de sistemas tolerantes à intrusões é a utilização da diversidade de sistemas operacionais e outros componentes de software.

Diversidade é usado como um importante mecanismo para melhorar a solidez dos sistemas criados com o objetivo de serem seguros e confiáveis [25, 26]. Em linha gerais, o princípio básico da diversidade de sistemas é evitar falhas comuns (e.g., erros de software ou vulnerabilidades). Tomando com base sistemas operacionais, a diversidade pode ajudar no controle de falhas comuns em seus diversos componentes. Na verdade, os sistemas operacionais *off-the-shelf*, de famílias diferentes, têm apenas algumas vulnerabilidades similares [26, 27].

No caso do RIT-LDAP a diversidade pode ajudar em diversas camadas da arquitetura. Nas réplicas SMR (*State Machine Replication*), tendo como base a camada de aplicação, a diversidade de software (e.g., Serviços de diretórios, Versões da JVM (*Java Virutal Machine*)) contribui para que um serviço inteiro não seja afetado por uma vulnerabilidade de software que todas as réplicas possuem, ou seja, o principal objetivo com essa diversidade é aumentar a probabilidade de criar sistemas cujas vulnerabilidades (se existirem) são completamente independentes. As réplicas SMR também podem ser implementadas com sistemas operacionais diferentes, com isso algumas vulnerabilidades referentes ao *kernel* são tratadas. Essas vulnerabilidades são as que afetam os protocolos de rede, cuja implementação é dependente do sistema operacional, os sistemas de arquivos, processos e gerenciamento de tarefas, bibliotecas e vulnerabilidades derivadas da arquitetura do processador. Portanto criando a diversidade de sistemas operacionais, aumenta-se a robustez do sistema e impede que um atacante possa explorar a mesma vulnerabilidade nas diversas réplicas SMR [26, 28].

Da mesma forma, a diversidade pode ser utilizada em outras abordagens, tais como sistemas de gerenciamento de banco de dados, que segundo Gashi [29] também pode ser muito eficaz para evitar falhas e erros comuns. Consequentemente, diversos sistemas operacionais, bancos de dados e outras ferramentas podem dificultar um ataque que explora uma vulnerabilidade específica nesses sistemas, porque pode estar presente apenas em algumas instâncias do sistema, mas não na maioria deles.

2.3 Implementações para Replicação de Máquinas de Estado

A replicação de Máquinas de Estado vem sendo estudada ao longo dos tempos. Alguns pesquisadores apresentaram diferentes implementações para resolução desse problema.

Castro e Liskov [24] apresentaram a primeira implementação de máquinas de estado viável, o PBFT (*Practical Byzantine Fault Tolerance*), onde funciona em ambientes asssíncronos como a internet e incorpora mecanismos de otimização.

Kotla et al. [30] apresentaram o Zyzzyva, um protocolo que utiliza especulação para reduzir o custo e simplificar a replicação de máquina de estado de replicação BFT, com base em três sub-protocolos: acordo, *view change* e *checkpoint*. O protocolo de acordo ordena as requisições para a execução das réplicas. O protocolo *view change* coordena a eleição de uma nova réplica primária quando a mesma está faltosa ou o sistema está executando lentamente. O protocolo *checkpoint* limita o estado que deve ser armazenado por réplicas e reduz o custo do protocolo *view change*.

Clement et al. [31] apresentaram uma nova biblioteca a fim de lidar com BFT que visa a simplicidade e atinge uma alternativa viável para tolerância a falhas por parada para uma gama de serviços de *cluster*. A robustez e o desempenho da biblioteca proposta é comparável com as bibliotecas do estado-da-arte, proporcionando uma robustez adicional.

Bessani *et al* [32] apresentaram o BFT-SMaRt, uma biblioteca para concretização de máquinas de estados replicadas tolerantes a falhas bizantinas, baseada no paradigma cliente-servidor. Devido sua robustez na implementação de mecanismo tolerante a falhas, a biblioteca permite ao usuário focar sua atenção na construção das funcionalidades de sua aplicação, sabendo que o mecanismo de tolerância a falhas bizantinas é da responsabilidade da biblioteca. Nos testes de resultado, ela apresentou maior eficiência que o PBFT e o Upright, maior escalabilidade de clientes, além de realizar operações mais rápido que os outros 2 protocolos.

2.4 BFT-SMaRt

Com base nos trabalhos citados anteriormente, como esta dissertação tem por objetivo apresentar um serviço de diretório tolerante a falhas e intrusões com desempenho viável, a biblioteca BFT-SMaRt foi escolhida por apresentar melhor resultados de acordo com experimentos previamente realizados por Bessani *et al* [32] em comparação as outras bibliotecas disponíveis.

A criação da biblioteca BFT-SMaRt foi baseada nos seguintes princípios:

- **Implementação em Java.** A linguagem de programação Java foi selecionada por questões de segurança, portabilidade, facilidade de programação e manutenção de código. Uma atenção especial foi dada ao desafio de criar uma biblioteca BFT que tivesse um desempenho altamente significativo, uma vez que Java é considerada uma linguagem bem menos eficiente que C (que é utilizado para implementar outros algoritmos BFT, como por exemplo o PBFT [24]).
- **Modularidade.** Existem algoritmos de máquinas de estados replicadas BFT de alto desempenho implementados como uma estrutura única (tal como o PBFT), que integram os mecanismos para viabilizar o algoritmo. O BFT-SMaRt foi desenvolvido para haver a separação de módulos, ou seja, não haver uma estrutura única.
- **Não adicionar complexidade.** O BFT-SMaRt foi desenvolvido para evitar otimizações que implicam em complexidade, que são normalmente usadas em outros algoritmos BFT e, ainda assim, mantém um bom nível de desempenho. Bessani *et al* [32], autores da BFT-SMaRt, comprovam o desempenho da biblioteca.

2.4.1 Arquitetura da biblioteca BFT-SMaRt

A Figura 2.5 ilustra a arquitetura do BFT-SMaRt. São apresentados os seguintes módulos:

1. ***Communication System.*** Módulo responsável pela comunicação entre réplicas e clientes e réplicas entre si. A comunicação entre cliente e réplica é feita utilizando a biblioteca Netty e a comunicação entre as réplicas é feita utilizando-se a API de sockets disponíveis no JAVA.
2. **Estado de Transferência.** Módulo responsável pela transferência de estado entre as réplicas, caso uma réplica se recupere de uma falha ou se na atrase na execução.
3. ***Byzantine Paxos Consensus.*** Módulo que implementa o algoritmo *paxos at war*, responsável pela ordenação das mensagens a executar.
4. ***Total Order Multicast.*** A primitiva de difusão de ordem total deriva da ordenação das mensagens por parte do consenso tolerante a falhas bizantinas, e permite que as mensagens tenham uma ordem global partilhada por todas as máquinas do sistema.

5. ***Reconfiguração.*** Módulo que permite a adição e remoção de réplicas no sistema. Um cliente com permissão para reconfigurar o sistema (TTP [3]) pode adicionar ou remover as réplicas. Isto faz com que, embora o número de réplicas no sistema tenha que obedecer a regra $n \geq 3f + 1$, os números identificadores das réplicas não têm que ser necessariamente sequenciais.

6. ***Checkpoints.*** Módulo que possui o mecanismo de checkpointing do estado das réplicas. O mesmo permite que as réplicas reconheçam a evolução do seu estado e permite a ordenação dessa evolução, utilizado pela transferência de estado.

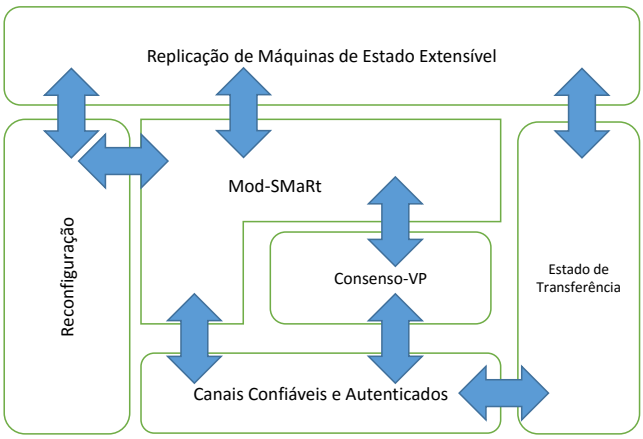


Figura 2.5: Módulos BFT-SMaRt. Adaptado de Bessani *et al* [32].

Funcionamento do BFT-SMaRt

De acordo com Brito [33], o funcionamento do BFT-Smart ocorre da seguinte forma:

1. **Chegada da mensagem às réplicas.** As mensagens enviadas pelos clientes são recebidas pelas *threads* existentes numa *thread pool* criada para atender pedidos dos clientes. O líder das réplicas vai então preparar um lote de mensagens para ordenar e propor um consenso para decidir essa ordem. Existe, no entanto, um mecanismo que permite a uma mensagem ser executada apenas localmente e não ser incluída nos lotes de mensagens, o que torna o seu processamento bastante mais rápido. Isto poderá ser utilizado

[3] Trusted Third Party

se o pedido não causar alterações no estado do sistema, por exemplo, uma operação de leitura local. Neste caso, o cliente vai esperar $2f + 1$ respostas iguais. Se não as receber, significa que existe um problema de consistência do estado do sistema no momento em que este pedido foi processado. Assim, o cliente irá reenviar o pedido, que será obrigatoriamente incluído no lote de mensagens, de modo a ser possível obter um número de respostas suficiente de modo a resolver o problema de consistência.

2. **Consenso e Difusão com Ordem Total.** O líder envia o lote de mensagens que preparou às outras réplicas. As réplicas executam um *round* do consenso para determinar a ordem das mensagens. Um *round* executa uma série de passos de modo a avaliar uma aceitação forte da proposta. Se cada réplica receber um número suficiente de mensagens a indicar a aceitação forte então a proposta foi aceita, o *round* do consenso acaba para ser iniciado um novo *round* com um novo líder, que será a próxima réplica na sequência de IDs existentes. O acordo acerca da ordem das mensagens faz com que exista uma primitiva de difusão com ordem total [34] entre as réplicas, já que todas as réplicas do BFT-SMaRt vão executar o mesmo conjunto de mensagens, pela mesma ordem.

3. **Processamento de mensagens.** Enquanto o algoritmo de difusão com ordem total está sendo executado, as réplicas do BFT-SMaRt continuam a receber mensagens enviadas pelos clientes. Essas mensagens são armazenadas em filas (uma para cada cliente) e é dessas filas que o líder retira mensagens para preparar o lote de mensagens. Para as mensagens recebidas das outras réplicas, existe uma *thread*, chamada de *Message Processor Thread*, que trata dessas mensagens. Esta *thread* descarta mensagens que já são irrelevantes para a execução (por exemplo, uma mensagem de um consenso já decidido que, por alguma razão, demorou tempo demais a chegar) e armazena mensagens relevantes a consensos que ainda não foram executados.

4. **Entrega da mensagem à aplicação.** Quando um *round* do consenso chega ao fim e as réplicas chegam a uma decisão, a mesma é colocada numa fila para valores decididos. Cabe então a uma *thread*, chamada *Delivery Thread*, retirar o valor decidido da fila, recolher todas as mensagens existentes no lote de mensagens (removendo-as das filas de mensagens dos respectivos clientes) e entregar as mensagens ordenadas à aplicação. É também nesta altura que o estado da réplica é armazenado, e, se necessário, é criado um *checkpoint* do estado da réplica.

2.4.2 Serviços Replicados Utilizando a Biblioteca BFT-SMaRt

Como avaliação de viabilidade, alguns autores [35, 36] utilizaram a biblioteca BFT-SMaRt para concretização da replicação de máquinas de estado e provaram que seu uso é prático.

O vigia dos vigias: um serviço RADIUS resiliente

Malichevskyy et al. [35] propuseram o primeiro serviço RADIUS tolerante a falhas bizantinas. Em linhas gerais, o trabalho propôs uma arquitetura com modificações no serviço RADIUS, capaz de tolerar falhas, através do uso do BFT-SMaRt, e intrusões, utilizando mecanismos de autenticação forte.

Na implementação do RADIUS resiliente, além do serviço de replicação oferecido pela BFT-SMaRt, foi utilizado o componente seguro do Typhon [37] como *back-end*. O sistema foi projetado para tolerar quaisquer falhas bizantinas no serviço de autenticação e algumas falhas bizantinas em componentes como o NAS[4] e o *gateway*. Estes componentes podem ser substituídos a qualquer hora, uma vez que não contém estado. Os experimentos mostraram que Typhon é mais lento que FreeRadius[5]. Isso acontece porque o Typhon é um sistema replicado e precisa de mais trocas de mensagens, levando em conta que utiliza o protocolo BFT.

OpenID

Cunha [36] propôs uma solução de replicação de OpenID para resolver os problemas de confiabilidade através da biblioteca BFT-SMaRt. Na arquitetura replicada do OpenID, um cliente usa um serviço Web (*relying party*) que o redireciona para seu respectivo provedor de OpenID (*gateway* IdP). As credenciais do usuário são usadas no processo de identificação, feito pelas réplicas de serviços IdP.

Os resultados apresentados mostram que os protótipos implementados podem tolerar diferentes tipos de falhas e ataques, tais como travamentos, comportamento arbitrário de alguns componentes e ataques de negação de serviço. Além disso, os resultados de taxa de transferência e latência indicam que os protótipos implementados são capazes de atender a demanda de ambientes reais com centenas de usuários.

[4]NAS (*Network Access Service*) é um elemento da arquitetura do RADIUS (tipamente um *switch*, ponto de acesso sem fio ou roteador) que recebe requisições de acesso a rede e as encaminha para o(s) servidor(es) de autenticação e autorização.

[5]Servidor RADIUS mais amplamente utilizado e de código livre.

Capítulo 3

Trabalhos Relacionados

Este Capítulo discute as soluções existentes para aumentar a disponibilidade e segurança de serviços de diretório LDAP. Primeiro, os métodos de replicação oriundos do protocolo LDAP são apresentados. Em seguida, as soluções que utilizam protocolos BFT em serviços de diretórios são explicadas. É importante notar que estas soluções podem ser utilizadas como *baseline* da solução proposta nesta dissertação. Por fim, uma discussão sobre todas as soluções apresentadas é realizada.

3.1 Replicação de Serviços de Diretório

Em sua definição, o protocolo LDAP implementa dois métodos de replicação para aumentar a disponibilidade do serviço de diretórios: a replicação *Single-Master* e a replicação *Multi-Master*.

De acordo com Timothy [5], na replicação *Single-Master* todas as réplicas possuem cópias somente de leitura das entradas e somente uma réplica do servidor (mestre) possui a responsabilidade de escrita. A Figura 3.1 ilustra a replicação *Single-Master*. Assim, somente o servidor mestre possui a permissão para realizar escritas. Qualquer outro servidor pode realizar as operações de busca, comparação e *bind* solicitadas por um cliente (1), mas quando o cliente envia uma requisição de atualização ao servidor LDAP (2), essa requisição é encaminhada somente para a réplica mestre.

Desta forma, após realizar (efetivar) a requisição, o mestre atualiza todas as outras réplicas. No entanto a replicação *Single-Master* apresenta uma desvantagem que faz com que não seja tão utilizada atualmente. Quando a réplica mestre cai por algum motivo (e.g., falhas de *software*, queda de energia), todas as operações de escrita ficam suspensas fazendo com o que os clientes fiquem impossibilitados de efetuá-las.

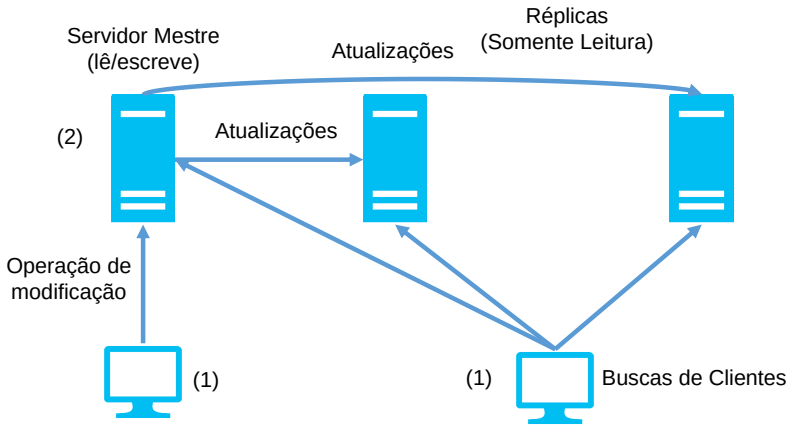


Figura 3.1: Replicação *Single-Master*. Adaptado de Howes *et al* [5].

Diferente da abordagem de replicação *Single-Master*, na *Multi-Master* todos os servidores são capazes de realizar atualizações no diretório. A Figura 3.2 ilustra a replicação *Multi-Master*, onde um cliente submete uma operação de escrita para qualquer réplica disponível e a atualização é propagada para todas as réplicas como forma de manter a consistência dos dados no sistema (todas as réplicas).

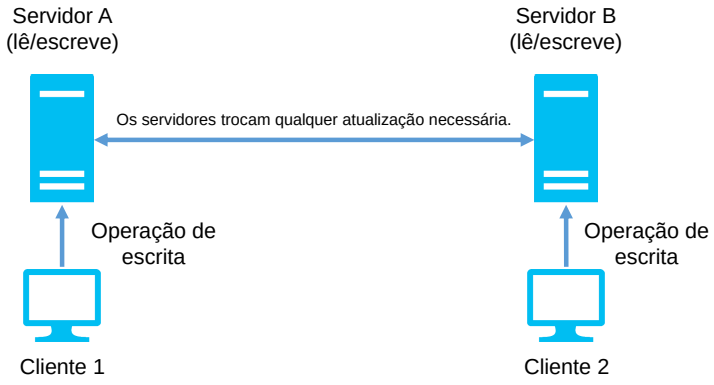


Figura 3.2: Replicação *Multi-Master*. Adaptado de Howes *et al* [5].

Além disso, na replicação *Multi-Master* os dados do serviço de diretório estão disponíveis em mais de um servidor ao mesmo tempo para serem lidos/escritos. Portanto, todas as réplicas possuem os mesmos dados. Logicamente isto acarreta

uma taxa de transferência um pouco maior do que na replicação *Single-Master*, porém evita problemas quando o servidor mestre fica *offline*, levando assim a suspensão de todas as operações de escrita.

3.1.1 Replicação de Diretório baseada em Filtros

Cui [38] propôs um esquema de replicação *Multi-Master* baseado em *templates* - modelos de filtro de consultas LDAP - onde somente entradas que combinam com os *templates* são replicadas. A ideia é que dado um conjunto consultas LDAP, *templates* sejam criados para generalizar diferentes subconjuntos dessas consultas. Os *templates* gerados são mantidos em *cache* e toda nova consulta é comparada a eles visando reduzir o tempo das consultas.

De acordo com o autor, em comparação com a replicação padrão do LDAP, a replicação baseada em *templates* obteve resultados 60% superiores nas consultas. Mesmo que o uso de *templates* tenha apresentado bons resultados, o autor afirma que não existem formas de realizar um balanceamento de cargas no modelo replicação *Multi-Master*. Assim, é provável que o desempenho da solução proposta seja afetado em ambientes com grande volumes de consultas.

Kumar [39] propôs um novo modelo de replicação de diretórios LDAP baseado em filtros. A ideia central é replicar somente entradas que correspondem a um determinado filtro, ao contrário do modelo padrão, onde todas as sub-árvores da DIT são replicadas. O modelo consiste em uma réplica armazenar entradas e metadados correspondentes a um ou mais filtros LDAP. Os filtros replicados são consultas generalizadas de usuários, que correspondem às regiões semânticas demonstrando localidade de referência.

O desempenho do modelo proposto é avaliado e comparado com uma implementação convencional LDAP em um diretório corporativo real. Uma taxa de acerto (porcentagem de consultas completamente respondida pela réplica) de 0.5 é relatada para o modelo proposto ao replicar menos de 10% do diretório da empresa para uma consulta típica que solicita uma entrada (nome do empregado combinando com uma identificação única). O tráfego de atualização também é consideravelmente menor do que réplicas LDAP normais. O autor afirma que o uso de filtros na replicação parcial pode ser usado para melhorar significativamente o desempenho de aplicações baseadas em diretórios.

Embora os trabalhos supracitados trabalhem com replicação de serviços diretórios, ambos só replicam os dados mais utilizados para melhorar o desempenho e não buscam replicar os dados para aumentar a segurança nos serviços de diretório.

3.2 Diretórios Tolerantes a Falhas Bizantinas

Wang et al. [10] foram os primeiros a propor um serviço de diretórios tolerante a falhas. Denominado **bftLDAP**, a solução é baseada em cinco princípios:

- Consistência: Os dados não faltosos em todos os servidores LDAP são consistentes.
- Tolerância a falhas bizantinas: Um sistema tolerante a falhas bizantinas pode prover um serviço correto, mesmo que alguma réplica esteja com um componente comprometido.
- Segurança: O sistema O LDAP deve comportar-se como uma implementação centralizada que executa operações atomicamente uma de cada vez.
- Vivacidade: Os clientes recebem as respostas de suas requisições.
- Desempenho prático: O sistema foi desenvolvido não somente para ser seguro e confiável, mas também para ser prático.

O serviço implementado consiste de $3f + 1$ réplicas, onde f é o número de réplicas que podem falhar. Além disso, no **bftLDAP** as chaves de sessão, determinadas por meio de pares de chaves públicas e privadas, são utilizadas para autenticar as comunicações entre clientes e servidores em vez de uma infraestrutura de chave pública, comprometendo assim a a segurança do serviço de diretório.

Para o desenvolvimento da solução, os autores utilizaram a biblioteca CLBFT (*Castro-Liskov Byzantine-Fault-Tolerant*) para fornecer tolerância a falhas bizantinas e o OpenLDAP como serviço de diretório. Foram testadas quatro tipos de operações LDAP (*ldapsearch*, *ldapadd*, *ldapdelete* e *ldapmodrdn*). O **bftLDAP** foi comparado a um OpenLDAP sem replicação, com e sem SSL. A solução apresentou um desempenho 42.29% mais lento que o LDAP simples, mas, no entanto, é 29.53% mais rápido que o LDAP com SASL. O tempo das operações *ldapadd*, *ldapdelete* e *ldapmodrdn* no **bftLDAP** são respectivamente 18.09%, 8.87% e 4.23% mais lentas do que no LDAP com SASL. Em linhas gerais, o serviço só tem um desempenho melhor do que o LDAP com SSL.

Hou et al. [11] propuseram uma melhoria no trabalho de Wang et al. [10], criando um serviço de diretórios tolerante a falhas, escalável e hierárquico - o **HBFTLDAP**. Com isso, segundo os autores, o sistema pode fornecer tolerância a falhas bizantina ao adicionar outro servidor LDAP ao sistema, e tolerar falhas bizantinas quando é hierárquico.

Esta implementação do serviço de diretório possui um sexto princípio, além dos cinco presentes no **bftLDAP**, hierarquia. Segundo os autores, este princípio pode prover tolerância a falhas bizantinas quando um outro servidor LDAP é adicionado no sistema de forma hierárquica. Entretanto, os resultados apresentados usando o **HBFTLDAP** são os mesmos do **bftLDAP**.

Shoker et al. [12] apresentaram um serviço de diretório replicado utilizando a implementação OpenLDAP, chamado **BFT-LDAP**. Os autores usaram quatro protocolos diferentes para prover a tolerância a falhas bizantinas no OpenLDAP: PBFT (*Practical Byzantine Fault Tolerance*), Zyzzyva, Chain e Quorum. Com o objetivo de integrar os protocolos de tolerância a falhas bizantinas com o OpenLDAP, foi criada uma arquitetura que assegura que o código do OpenLDAP seja mantido intacto. A arquitetura é composta basicamente de cinco componentes: (i) aplicação do cliente; (ii) proxy BFT do cliente; (iii) proxy BFT do servidor; (iv) unidade de acesso ao LDAP; e (v) o OpenLDAP.

O funcionamento é o seguinte: quando o cliente envia uma requisição LDAP, o proxy BFT do cliente recebe esta solicitação e a envia para o proxy BFT do servidor. Ao receber a solicitação, o proxy BFT do servidor invoca operações baseadas na aplicação através da unidade de acesso ao LDAP e chama a API do OpenLDAP para executar as operações localmente. Por fim, o OpenLDAP executa as operações.

Nos experimentos, os autores consideraram somente operações de buscas, por serem responsáveis por 98% das operações realizadas em serviços de diretórios. Além disso, somente os protocolos Zyzzyva e PBFT foram utilizados nos experimentos, por serem os mais difundidos segundo os autores. Como resultado, a taxa de transferência do OpenLDAP sem replicação domina os protocolos PBFT e Zyzzyva com poucos clientes, mas quando o quantidade de cliente é superior a 10, a taxa de transferência do **BFT-LDAP** com PBFT ou Zyzzyva torna-se muito próximo a taxa da versão padrão do OpenLDAP. O OpenLDAP sem replicação BFT realiza consultas apenas 5% melhor do que com Zyzzyva e 10% melhor do que com PBFT com tamanhos de mensagens de 1byte. Com mensagens maiores de 1b, as curvas de rendimento tornam-se ainda mais equivalente se a replicação BFT é usada ou não. O OpenLDAP sem replicação falha com mais de 120 clientes simultâneos, porém as versões com Zyzzyva e PBFT podem ser expandidas para até 200 clientes simultâneos.

Viera et al. [40] apresentaram uma ideia de como aumentar a confiança de serviços de diretórios genéricos utilizando replicação de máquinas de estado BFT. Os autores propuseram uma arquitetura para serviços de diretórios tolerante a falhas bizantinas utilizando componentes COTS (*Commercial Off-The-Shelf*) e

mantendo a transparência em relação aos aplicativos dos clientes. Além disso, adicionaram uma camada externa ao serviço de diretório, a fim de manter intacta sua implementação (compatibilidade). Para implementar BFT de maneira transparentes, os autores utilizaram JNDI (*Java Naming and Directory Interface*) para implementar um *driver* específico de SPI (*Service Provider Interface*) que atua como ponte entre a API (*Application Programming Interface*) JNDI e biblioteca de replicação BFT com o cliente. Embora os autores tenham implementado um protótipo da arquitetura utilizando somente o OpenLDAP como serviço de diretório, eles não apresentaram nenhum resultado, limitando-se somente a dizer que o desempenho do OpenLDAP com e sem tolerância a falhas bizantinas possui um *overhead* mínimo.

Por fim, como prova de conceito, a ideia conceitual do RIT-LDAP foi implementada (Neto *et al.* [41]) empregando a biblioteca BFT-SMaRt para replicação de máquinas de estado. Entretanto, foram feitas somente operações de busca, realizadas de forma ordenada e serializada. Para validação, foi adicionada a diversidade de apenas dois (2) serviços diretórios (OpenLDAP e OpenDJ). Nos experimentos realizados, os resultados obtidos alcançaram 1.400 buscas por segundo.

3.3 Discussão

A Tabela 3.1 resume as principais características das soluções existentes, bem como da solução proposta.

Nos trabalhos apresentados, as replicações nativas dos OpenLDAP (*Single-Master* e *Multi-Master*) necessitam de $f + 1$ réplicas para tolerar falhas por parada. Enquanto todas as outras soluções toleram apenas falhas por parada[1] e parcialmente arbitrárias, por isso utilizam $3f+1$ réplicas para tolerar f falhas sem comprometer a operação do serviço. Contudo, o RIT-LDAP suporta falhas por parada ao utilizar a abordagem de replicação de máquina de estados e suporta falhas arbitrárias, pois possui a capacidade de implementar a diversidade de sistemas.

No que diz respeito aos protocolo para replicação, ambas as soluções do OpenLDAP (*Single-Master* e *Multi-Master*) utilizam o protocolo Syncrepl, um mecanismo que permite o servidor LDAP manter uma cópia de um pedaço da sua DIT, mas esta abordagem pode ter desvantagens quando o padrão de uso envolve mudanças individuais em vários objetos.

[1]Falhas em que um servidor que está em funcionamento correto para e não volta a um estágio de funcionamento, a não ser que seja reiniciado (e.g., *crash*) [13, 14].

Tabela 3.1: Comparação dos trabalhos relacionados e abordagem proposta

Solução	Tipos de Falhas Toleradas	Protocolo de replicação	Serviço LDAP	# réplicas	Tolerância a Intrusão
OpenLDAP *SingleMaster*	Por parada (*)	Syncrepl	OpenLDAP	$f+1$	não
OpenLDAP *MultiMaster*	Por parada	Syncrepl	OpenLDAP	$f+1$	não
COTS (**)	Por Parada e parcialmente arbitrária	Biblioteca BFT	OpenLDAP	$3f+1$	não
bftLDAP	Por parada e parcialmente arbitrárias	CLBFT	OpenLDAP (2.1.22)	$3f+1$	não
HBFTLDAP	Por parada e parcialmente arbitrárias	CLBFT	OpenLDAP (2.1.22)	$3f+1$	não
BFTLDAP	Por parada e parcialmente arbitrárias	PBFT, Zyzzyva	OpenLDAP (2.4)	$3f+1$	não
RIT-LDAP	Por parada e arbitrárias	BFT-SMaRt	Múltiplos	$3f+1$	sim
(*) A parada do servidor Master interrompe a operação de escrita.					
(**) Apenas uma arquitetura funcional.					

Os trabalhos que utilizam replicação de máquinas de estado implementam os protocolos PBFT e Zyzyyva (BFTLDAP), CLBFT (bftLDAP e HBFTLDAP). Já o RIT-LDAP se beneficia dos mecanismos oferecido pela biblioteca BFT-SMaRt para permite a implementação do sistema em ambientes *multi-data center* e *multi-cloud*. Além disso, Bessani e Alchiere [32] demonstram que a biblioteca BFT-SMaRt possui um *throughput* maior, além de suportar um número maior de clientes, chegando a ser 10 vezes mais que o PBFT.

No critério Diversidade, todas as soluções apresentadas na Tabela 3.1, com excessão ao RIT-LDAP e COTS, utilizam OpenLDAP como servidor LDAP. A única solução projetada e implementada para suportar múltiplas implementações (e.g., OpenLDAP, OpenDJ e 389-Directory) de serviços de diretórios LDAP é o RIT-LDAP. Vale lembrar que COTS é somente um modelo arquitetural ainda não testado para diversos serviços de diretórios.

No critério Tolerância a Intrusão, as soluções **bftLDAP**, **HBFTLDAP** e **BFTLDAP** utilizam apenas o OpenLDAP e suportam apenas falhas parcialmente arbitrárias. Assim, tomando como exemplo um bug de implementação no serviço LDAP, nenhuma dessas soluções é capaz de efetivamente suportar esse tipo de falha. A única solução projetada para tolerar intrusões é o RIT-LDAP.

Resumidamente, os principais diferenciais do RIT-LDAP, quando comparado às soluções existentes, são:

1. Maior disponibilidade, resistindo a diferentes tipos de falhas físicas e lógicas

(e.g., bugs de implementação e ataques paralelos);

2. Resistência a ataques de exaustão de recursos, uma vez que as réplicas podem ser instanciadas em diferentes máquinas físicas e/ou domínios administrativos;

3. Capacidade de tirar vantagem de ambientes *multi-data center* e *multi-cloud*, usufruindo dos respectivos mecanismos de defesa dessas infraestruturas, como mecanismos de mitigação de ataques de negação de serviços [42, 43].

As técnicas de replicação baseadas em filtro, discutidas neste Capítulo, não foram apresentadas na Tabela pois somente replicam as entradas mais utilizadas com o intuito de obter uma consulta mais eficiente. Nenhuma delas tem o propósito de aumentar a segurança e a disponibilidade.

Capítulo 4

RIT-LDAP

Este Capítulo apresenta o modelo funcional do RIT-LDAP, bem como cenários de implantação. Além disso, os modelos e suposições para o funcionamento do sistema são elencados, bem como é feita uma explicação da implementação do RIT-LDAP. Em seguida, os componentes do sistema são detalhados um a um. Por fim, a comunicação do sistema é apresentada e discutida.

4.1 Modelo Funcional

Recentemente, o projeto SecFuNet elaborou um modelo funcional [22, 44], componentes para criação de sistemas e técnicas essenciais para o desenvolvimento e implementação de provedores de identidade confiáveis mais seguros para infraestruturas de redes do futuro. Além disso, o projeto SecFuNet desenvolveu dois sistemas resiliente (R-OpenID [45] e R-Radius [35]) que utilizavam a biblioteca BFT-SMaRt para prover resiliência.

O serviço de diretório LDAP proposto nesta dissertação (RIT-LDAP) é completamente baseado no referido modelo funcional no que diz respeito tanto aos componentes do sistema quanto na arquitetura funcional, permitindo projetar e implementar serviços de diretórios tolerante a falhas e intrusões. Em linhas gerais, o RIT-LDAP é composto por quatro elementos principais: (a) cliente; (b) *gateway* LDAP; (c) as réplicas SMR e (d) o serviço de diretório. A Figura 4.1 ilustra o modelo funcional do RIT-LDAP.

O cliente representa um usuário ou serviço que tenta acessar alguma informação do diretório LDAP. Em termos práticos, o cliente é, por exemplo, uma aplicação de autenticação.

O *gateway* LDAP possui duas funções principais. A primeira é ser responsável por manter a compatibilidade com os clientes existentes. Em outras palavras, é visto com um servidor LDAP convencional pelos clientes, ou seja, não é neces-

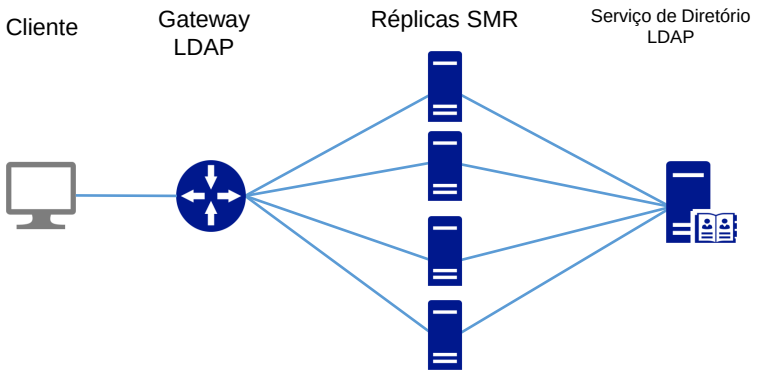


Figura 4.1: Modelo Funcional do LDAP Resiliente.

sário qualquer tipo de modificação nos clientes. A segunda função é mascarar o protocolo de replicação BFT-SMaRt e outros mecanismos utilizados para tolerar falhas arbitrárias e intrusões em diretórios LDAP.

Ao receber a requisição do cliente, o *gateway* simplesmente encapsula os dados no protocolo BFT e envia para as $3f + 1$ réplicas do sistema, onde f é o número de réplicas faltosas que o sistema tolera. Similarmente, a resposta das réplicas é enviada ao *gateway* e em seguida aos respectivos clientes. Em outras palavras, o *gateway* atua de forma análoga a um *gateway* de rede, repassando dados de um lado para outro sem a necessidade dos processos comunicantes conhecerem os diferentes segmentos de rede. Em termos práticos, o *gateway* pode também agregar funcionalidades adicionais como um *firewall*, limitando as atividades maliciosas dos clientes. Além disso, o *gateway* pode ser implementado de várias formas (e.g. *cluster* de *gateways*) impedindo que um atacante possa, assim, derrubar todos os *gateway* disponíveis.

O papel das réplicas SMR é processar as requisições dos clientes de forma determinista e ordenada. Por isso existe a necessidade de protocolos de replicação de máquinas de estado, no caso o BFT-SMaRt. As réplicas podem conectar-se a um ou mais servidores LDAP. Caso a conexão com um servidor falhe, a réplica SMR utiliza automaticamente a conexão com o próximo servidor LDAP da lista. Tanto a réplica quanto o serviço de diretórios LDAP podem ser instanciados na mesma máquina por questões de desempenho e segurança. No caso de ambos os componentes estarem na mesma máquina, não é necessário a utilização de canais cifrados (e.g., LDAPS) entre a réplica e o servidor LDAP, por exemplo.

Finalmente, o serviço de diretório LDAP pode ser parte integrante do domínio local ou ainda um serviço prestado por terceiros, i.e., localizado geograficamente em outro domínio. Essa flexibilidade aumenta a robustez do RIT-LDAP, uma

vez que múltiplas infraestruturas físicas podem ser utilizadas para aumentar a disponibilidade do sistema e a resistência contra diferentes tipos de ataques. Múltiplos domínios e infraestruturas físicas são um importante aliado contra ameaças avançadas, para aumentar a disponibilidade do sistema, para evitar problemas de *vendor lock-in* em provedores de nuvem, entre outros [43, 44, 46].

4.2 Cenários de Implantação

A arquitetura proposta foi elaborada para permitir que *gateways*, réplicas e serviços de diretório LDAP possam ser instanciados de várias formas. As Figuras 4.2 e 4.3 ilustram os dois típicos cenários para implantação do RIT-LDAP.

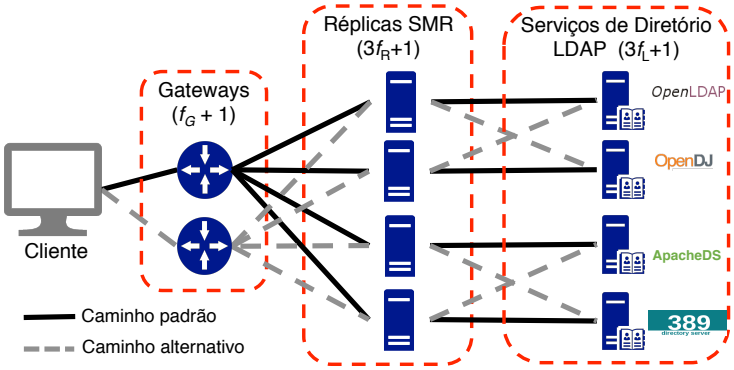


Figura 4.2: Todos os componentes separados

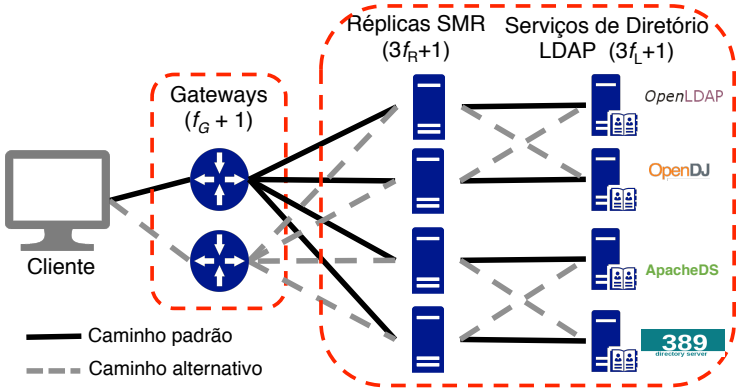


Figura 4.3: Réplicas e Serviços LDAP juntos

No cenário ilustrado na Figura 4.2, todos os componentes podem ser implantados separadamente, ou seja, em: (i) diferentes máquinas físicas em um único domínio; (ii) múltiplas máquinas físicas em diferentes domínios; (iii) múltiplos

data centers, ou ainda (iv) múltiplos provedores de nuvem. Vale ressaltar que a implantação separada dos componentes fora de único domínio - casos (ii), (iii) e (iv) - contribui para garantir altos níveis de disponibilidade e fornecer mecanismos extras de proteção contra diferentes tipos de ataques de exaustão de recursos, DDoS em grande escala, e assim por diante [43, 44, 46].

Embora permitida, a implantação de componentes no mesmo local não deve ser feita sem critérios. O motivo é simples, caso o sistema apresente uma vulnerabilidade, todos os componentes existentes seriam potencialmente atingidos. O *gateway*, por exemplo, deve ser separado dos demais elementos da arquitetura. Isso porque ele é o responsável por mascarar o protocolo BFT-SMaRt, logo é o único componente que possui a informação de quem são as réplicas SMR.

Além disso, o *gateway* possui chaves públicas e privadas que são trocadas com as réplicas, para garantir a segurança da comunicação. Em termos práticos, se o *gateway* for implantado: (1) em cada réplica, o cliente saberá quem são todas as réplicas e será o elo mais fraco do sistema; (2) em apenas uma réplica (conhecida agora como réplica mestre), a carga de comunicação já existente entre as réplicas (oriunda da BFT-SMaRt) será aumentada e essa máquina torna-se um potencial ponto de falha do sistema. Em ambos os casos, o risco de intrusão é aumentado consideravelmente.

O cenário mais adequado para implantar componentes de forma conjunta, é mostrado no cenário da Figura 4.3, onde réplicas SMR e serviços de diretório LDAP são implantados na mesma máquina. Esse cenário de implantação pode ser colocado em prática em todos os casos - (i), (ii), (iii) e (iv) -, reduzindo o número de máquinas necessárias para operar o sistema.

4.3 Suposições

Em primeiro lugar, é assumido um modelo de sincronia parcial [47], ou seja, o sistema pode se comportar de forma assíncrona por algum tempo, até que se torne síncrono, i.e., com limites de tempo de processamento e comunicação. Este é um requisito essencial para garantir o término dos protocolos de consenso, um alicerce fundamental para a implementação de replicação de máquinas de estado.

É assumido que os *gateways* conseguem comunicar-se com todas as réplicas SMR através do protocolo BFT-SMaRt. Ao receber um pacote de um cliente, o *gateway* encapsula os dados no protocolo BFT e enviada para as réplicas. O gateway aguarda a resposta correta (igual) de $2f_R+1$ réplicas antes de responder ao cliente.

Com relação ao modelo de falhas, são assumidas falhas arbitrárias de até f_G *gateways* e falhas arbitrárias nas réplicas SMR (f_R) e serviços LDAP (f_L). Não é assumido nada em relação aos clientes, ou seja, o sistema suporta um número

ilimitado de clientes.

Por último, em relação ao modelo de ameaça, assume-se que um atacante pode: (i) comprometer simultaneamente até f_G *gateways*, f_R réplicas, e f_L diretórios LDAP, respectivamente; (ii) interceptar e injetar mensagens em até f_G *gateways*, f_R réplicas, e f_L diretórios LDAP; e (iii) alterar mensagens em trânsito. O RIT-LDAP considera um modelo de ameaça onde o atacante pode ter controle dos canais de comunicação entre os elementos do sistema.

No entanto, é assumido que o atacante não é capaz de quebrar quaisquer mecanismos criptográficos sem obter as chaves criptográficas adequadas. Um atacante pode ainda ter o controle da rede por algum tempo, mas não pode controlar toda a rede durante todo o tempo. Isto pode ser suposto já que todos os elementos do sistema, incluindo réplicas, podem estar em execução em diferentes infraestruturas físicas e, consequentemente, é altamente improvável que um atacante obtenha controle sobre os canais de comunicação ou recursos em lugares distintos.

4.4 Implementação

Na implementação do RIT-LDAP foi utilizada a biblioteca UnboundID LDAP [48] (versão 0.9.8), que suporta a versão 3 do protocolo LDAP, e a biblioteca BFT-SMaRt [49] para replicação de máquina de estados. Esta biblioteca fornece um conjunto de módulos e protocolos de comunicação para assegurar canais de comunicação seguros e eficientes entre as réplicas.

A Figura 4.4 ilustra a implementação do RIT-LDAP. Os clientes comunicam-se com os *gateways* através de TCP/IP, assim como ocorre com servidores LDAP tradicionais. Para fins de testes, foi implementada uma aplicação cliente utilizando a biblioteca UnboundID LDAP. Esta aplicação realiza operações (e.g., consultas) no diretório LDAP.

4.4.1 Componentes do RIT-LDAP

Cliente LDAP

Como previamente apresentado em detalhes no modelo conceitual, o cliente é um componente genérico que pode representar diversos sistemas, tais como, um programa rodando em uma rede privada, um aplicativo de acesso a e-mail, entre outros. O único requisito necessário é que siga as especificações padrões do protocolo LDAP versão 3.0.

LDAP *Gateway*

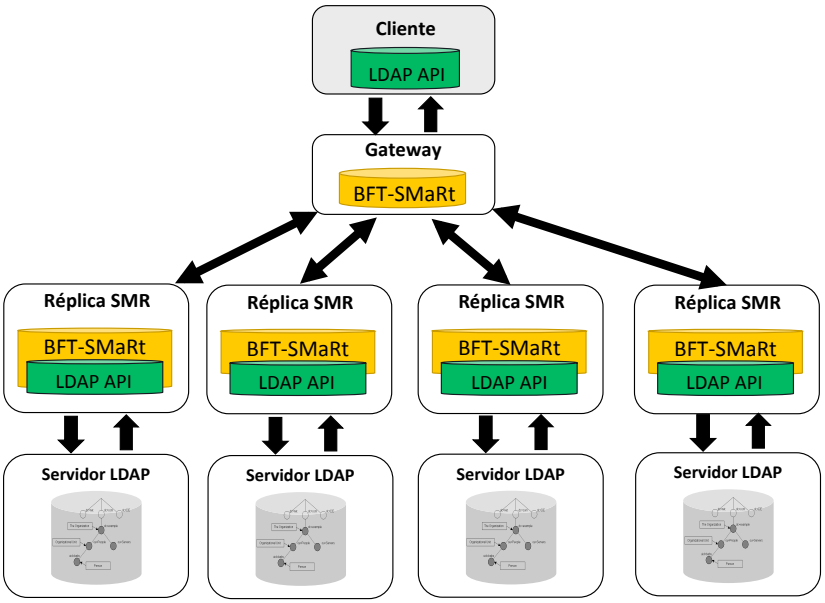


Figura 4.4: Elementos de implementação do diretório LDAP replicado.

O LDAP *gateway* é o novo componente introduzido no sistema, com o propósito de manter a compatibilidade de sistemas previamente desenvolvidos. Para este fim, é necessário que o *gateway* aceite conexões TCP/IP dos clientes LDAP e encapsule os pacotes recebidos com o protocolo BFT utilizado. Isto é necessário para poder encaminhar as requisições para todas as réplicas LDAP. Da mesma forma, os pacotes recebidos pelo *gateway*, enviados pelas réplicas, são desencapsulados e devolvidos para o cliente pela conexão previamente estabelecida. Portanto, *gateway* é visto pelo cliente como um serviço LDAP normal.

Em outras palavras, o cliente não sabe que está se conectando com o serviço LDAP resiliente. Ainda mais, como o *gateway* somente repassa as mensagem entre os clientes e as réplicas (de forma bi-direcional), ou seja, é um componente bem simples, que pode ser facilmente replicado e seguro.

LDAP Réplicas

As réplicas do sistemas implementam a versão 3.0 do protocolo LDAP, sendo usado a biblioteca UnboundID LDAP [48] (versão 0.9.8). A biblioteca BFT-SMaRt [49] é utilizada para proporcionar a tolerância à falha por parada e tolerância à falhas bizantinas. Nossa implementação suporta apenas o protocolo mais atual do LDAP, a versão 3.0.

4.4.2 Comunicação

O *gateway* recebe as conexões dos clientes (via TCP, na porta 389) e simplesmente as repassa a interface BFT proxy do BFT-SMaRt. O BFT proxy envia, de forma ativa, uma cópia das requisições dos clientes para cada réplicas SMR. De maneira similar, respostas vindas das réplicas são enviadas aos clientes através dos canais TCP/IP previamente estabelecidos.

Uma réplica SMR mantém um arcabouço de conexões com um ou mais serviços de diretório LDAP. Ao receber a requisição do cliente, encaminhada pelo *gateway*, a réplica encaminha a requisição para o(s) respectivo(s) servidor(es) LDAP. O servidor LDAP pode ser qualquer implementação LDAP disponível, desde que suporte a versão 3 do protocolo.

Após executar a operação requisitada, a réplica SMR encaminha a resposta do serviço LDAP ao *gateway*. Ao receber $2f_R + 1$ respostas idênticas, o *gateway* envia a resposta ao respectivo cliente. Caso não hajam $2f_R+1$ respostas idênticas, o *proxy* BFT reenvia a requisição original às réplicas SMR.

Capítulo 5

Experimentos e Resultados

Este Capítulo apresenta a avaliação da solução proposta. O objetivo é medir o desempenho do RIT-LDAP durante o funcionamento em diferentes ambientes de computação, especialmente durante o emprego da diversidade. Além disso, também são fornecidos resultados referentes a escalabilidade da solução, o que permite estimar até quantos usuários concorrentes o RIT-LDAP consegue dar vazão em comparação com um serviço de diretórios não replicado. Por fim, é apresentado o funcionamento mediante parada das réplicas SMR, o que permite aumentar o entendimento do comportando do RIT-LDAP quando uma falha acontece em um réplica, sendo ela arbitrária ou não.

5.1 Ambiente de Experimentação

Para a avaliação do RIT-LDAP foi utilizado um ambiente composto por seis máquinas virtuais (VMs). A Tabela 5.1 apresenta todas as máquinas virtuais utilizadas.

Tabela 5.1: Ambiente utilizado para os testes

Tipo	**# VMs**	**# vCPUs**	**Memória**	**Disco**	**Rede**
Clientes	1	8	8	15GB	Gigabit
Gateway	1	4	12	15GB	Gigabit
Replicas	4	4	4	20GB	Gigabit

Resumidamente, para executar os clientes foi utilizada uma única máquina virtual com 8 processadores virtuais (vCPUs) e 8GB de memória RAM. Uma máquina virtual foi instanciada para executar o *gateway*, com 4 vCPUs e 12GB

de memória RAM. E as réplicas foram criadas com 4 vCPUs e 8GB de memória, sendo uma réplica e um serviço de diretório por VM. A Figura 4.3, na seção 4.2, ilustra melhor o cenário de experimentação envolvendo as VMs.

5.1.1 Diversidade no Ambiente

Para atender a diversidade, os experimentos utilizaram três implementações de servidores LDAP: OpenLDAP (versão 2.4.31), OpenDJ xpress (versão 2.5.0) e 389-Directory (versão 1.3.4.5). Vale ressaltar que uma quarta implementação (ApacheDS) foi testada, mas mostrou-se incompatível com o protótipo. Na seção 6.2 este problema será melhor explicado.

No quesito sistemas operacionais, foram utilizados quatro sistemas operacionais distintos: Ubuntu (versão 14.04.1 LTS), Debian GNU/Linux 8, CentOS Linux release 7.2.1511 (Core) e FreeBSD 10.2-STABLE. Por comodidade, as máquinas empregadas para representar o cliente e o *gateway* LDAP utilizam Ubuntu 14.04.1.

5.1.2 Cenários de Experimentação

A fim de observar e comparar o comportamento do RIT-LDAP e suas possíveis limitações foram criadas três configurações de execução distintas:

- **Cenário 1**: Todas as réplicas SMR utilizam o mesmo sistema operacional, mas com diversidade de serviços de diretório. Foram testadas as seguintes combinações - Ubuntu (OpenLDAP e OpenDJ), Debian (OpenLDAP e OpenDJ), CentOS (389-Directory e OpenDJ) e FreeBSD (OpenDJ);
- **Cenário 2** : Foi introduzida diversidade parcial nos sistemas operacionais e nos serviços de diretório. Assim, foram testadas duas Réplicas SMR com Ubuntu (uma utilizando OpenDJ e outra OpenLDAP) e duas réplicas SMR com CentOS (uma com 389-directory e outra com OpenDJ);
- **Cenário 3** : Foi aplicada a diversidade total, com cada réplica SMR criada em um sistema operacional e um serviço de diretório diferentes. A combinação gerada é a seguinte - Réplica 01 (Ubuntu e OpenLDAP), Réplica 02 (Debian e OpenDJ), Réplica 03 (FreeBSD e OpenDJ) e Réplica 04 (CentOS e 389-Directory).

É preciso destacar que cada uma das quatro instâncias de serviços de diretório foi povoada com **50.000** entradas para cada tamanho de mensagem existente (e.g., 50.000 entradas para 1B, 50.000 para 32B, 50.000 entradas para 64B, até 512B). Devido a existência de nós dessas entradas, cada servidor LDAP tem, após

ser totalmente povoado, **357.716** entradas. Para os testes com as operações de escrita (e.g., *add*, *delete* e *modify*), o serviço de diretório foi povoado com 359 entradas, que correspondem apenas os nós de entrada.

Ainda sobre a experimentação, para avaliar as operações de busca foram utilizados 10, 25 e 50 clientes simultâneos, executados na VM clientes. Este número de clientes foi definido conforme o nível de saturação do sistema. Foram realizados experimentos com 75 clientes, porém percebeu-se que acima desse número o sistema apresenta queda de rendimento. Além disso, percebeu-se uma diferença de desempenho com 10, 25 e 50 clientes, mas o mesmo não acontece acima de 60.

Cada cliente foi configurado para realizar 1.000 operações de busca com tamanhos de mensagem variando de 1B a 512B. Vale ressaltar que o trabalho de Wang *et al.* [50] afirma que o tamanho de mensagem mais comum em diretórios LDAP é de 488B, mas tendo em vista que outros trabalhos fazem uso de tamanhos variados, optou-se pela mesma abordagem nesta dissertação. Para cada configuração de cliente, foram realizadas 60 execuções de busca.

Para as operações de escrita, os serviços de diretórios foram iniciados somente com nós de entrada para que fosse possível realizar, respectivamente, as operações de adição (*ldapadd*), modificação (*ldapmodrdn*) e deleção (*ldapdelete*). Para avaliar essas operações, foram efetuadas 1.000 operações de cada por cliente, com o número máximo de 10 clientes simultâneos, pois houve saturação do sistema com mais clientes. As entradas também variaram entre 1B e 512B. Para cada operação foram realizadas 30 rodadas de execuções. Esse número inferior deve-se ao fato das três operações serem responsáveis por apenas 1% de todas as operações nos serviços de diretórios [12]. Para fins estatísticos, todas as médias foram apresentadas com um intervalo de confiança de 95% [51].

5.2 Experimentos e resultados

Esta seção descreve os resultados do RIT-LDAP ao ser submetido a avaliação de desempenho nos três cenários.

5.2.1 Cenário 1

Os resultados do Cenário 1 compreendem todas as operações (*search*, *add*, *delete* e *modify*). Embora autores como Shoker e Bahsoun [12] e Wang *et al.* [10] deixem bem claro que a principal operação realizada por diretórios LDAP é a busca, por questões de validação, também foram realizados testes com outras operações envolvendo o uso de escrita. Neste cenário, cada serviço de diretório foi avaliado individualmente para perceber como é o seu funcionamento/desempenho

nos diversos sistemas operacionais e como a biblioteca BFT-SMaRt se comporta quando implementada nesses diretórios LDAP.

Operações de Busca

As Figuras 5.1, 5.2, 5.3, 5.4, 5.5, 5.6 e 5.7 apresentam os resultados das operações de busca no Cenário 1. O OpenDJ foi o único serviço de diretório capaz de ser implementado em todos os sistemas operacionais. O motivo dessa capacidade é explicado pelo fato de ser desenvolvido em JAVA. Sendo assim, sua instalação é possível para qualquer sistema operacional que tenha suporte a linguagem Java. Vale ressaltar que todos os experimentos foram feitos com Java versão 1.7.

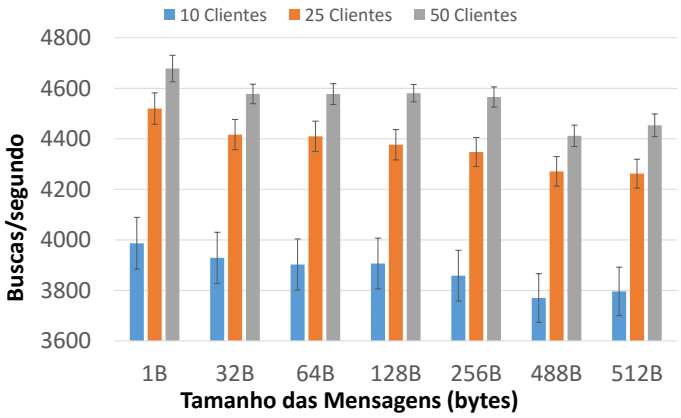


Figura 5.1: Vazão do RIT-LDAP no **Cenário 1**, usando FreeBSD e OpenDJ.

No FreeBSD, Figura 5.1, os resultados apontam que no pior caso o sistema alcançou 3.510 buscas por segundo, para 10 clientes, com mensagens de 512B, enquanto no melhor caso chegou a 4.720 buscas por segundo, com 50 clientes, para mensagens de 1B.

Quando executando no Ubuntu e no Debian (Figuras 5.2 e 5.3, respectivamente), os resultados do OpenDJ apresentaram pouca variação, com melhor resultado no Ubuntu. No melhor caso, com o Debian, o RIT-LDAP alcançou 4.678 buscas por segundo, com 50 clientes e mensagens de 1B, enquanto o Ubuntu chegou a 4.824. Já no no pior caso, o Ubuntu apresentou 3.766 buscas por segundo com 10 clientes e mensagens de 512B, enquanto o Debian apresentou 3.769 buscas por segundo.

A Figura 5.4 mostra o desempenho do OpenDJ no CentOS, o que obteve melhor resultado entre a combinação serviço de diretório e sistema operacional. Percebe-se que a medida que se aumenta o número de clientes, o RIT-LDAP se comporta melhor. Essa diferença pode ser atribuída diretamente ao aumento da

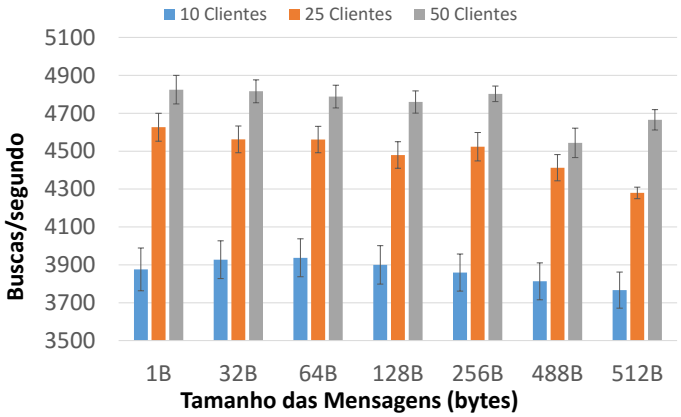


Figura 5.2: Vazão do RIT-LDAP no **Cenário 1**, usando Ubuntu e OpenDJ.

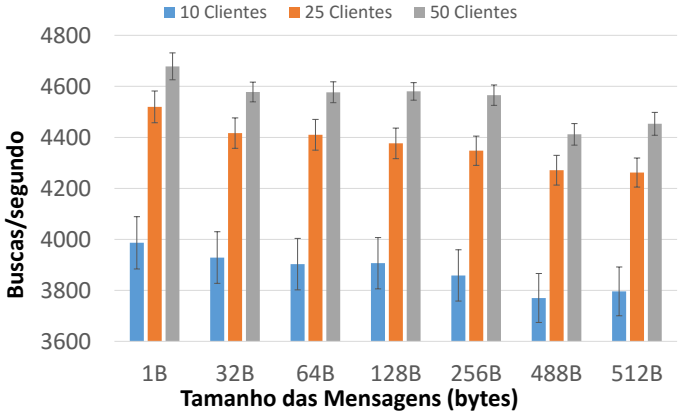


Figura 5.3: Vazão do RIT-LDAP no **Cenário 1**, usando Debian e OpenDJ.

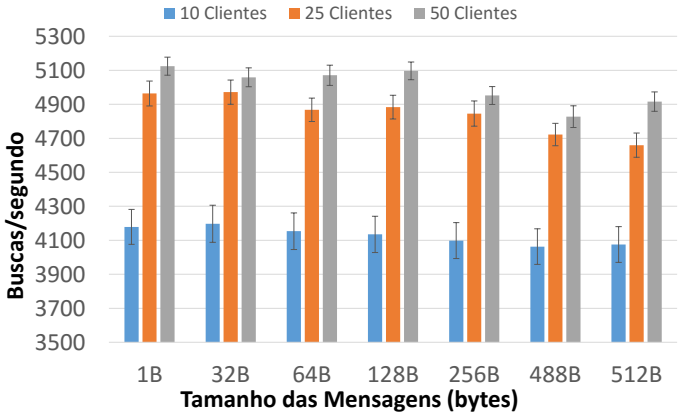


Figura 5.4: Vazão do RIT-LDAP no **Cenário 1**, usando CentOS e OpenDJ.

carga de trabalho, uma vez que o BFT-SMaRt consegue gerenciar melhor um número maior de clientes. O pior caso alcançou 4.060 buscas por segundo com mensagens 488B e 10 clientes, enquanto o melhor caso alcançou 5.120 buscas por segundo com 50 clientes e mensagens de 1B. Esses valores são explicados pela influência direta do tamanho da mensagem e do número de clientes sobre o desempenho.

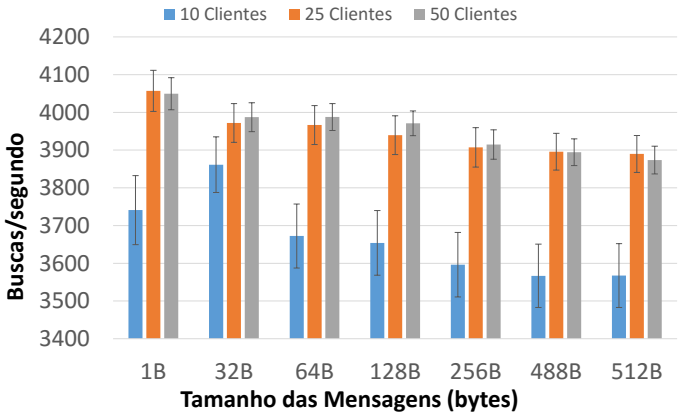


Figura 5.5: Vazão do RIT-LDAP no **Cenário 1**, usando Ubuntu e OpenLDAP.

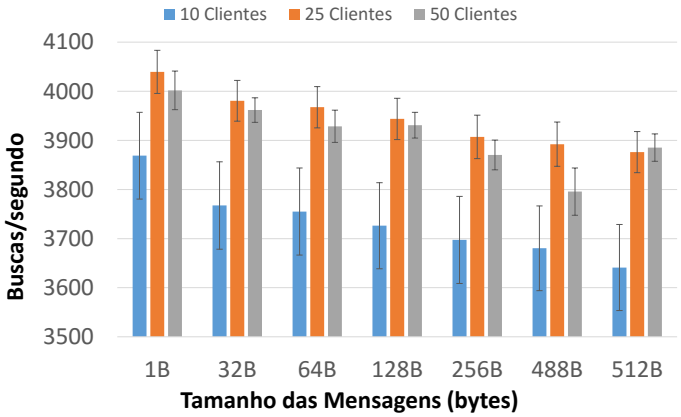


Figura 5.6: Vazão do RIT-LDAP no **Cenário 1**, usando Debian e OpenLDAP.

O OpenLDAP foi instanciado no Ubuntu e no Debian (Figuras 5.5 e 5.6, respectivamente), pois é um serviço nativo destas distribuições. Os resultados apresentados nos experimentos foram semelhantes, fato explicado pelo Ubuntu ser baseado no Debian. No melhor caso, ambos obtiverem valores um pouco superiores 4.000 buscas por segundo com 25 clientes e mensagens de 1B. No pior

caso, ambos obtiverem valores em torno de 3.600 buscas por segundo com 10 clientes e mensagens de 512B.

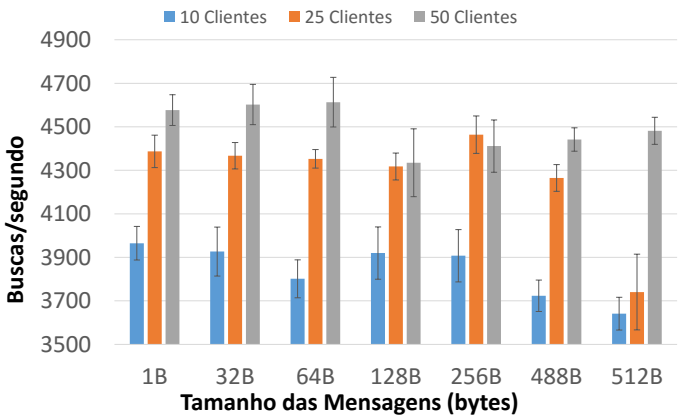


Figura 5.7: Vazão do RIT-LDAP no **Cenário 1**, usando CentOS e 389-Direcotry.

Por fim, os testes com o 389-Directory (Figura 5.7) foram realizados somente no CentOS, pois é o serviço de diretório nativo desta distribuição. Os resultados apresentaram uma média de consultas superior ao OpenLDAP, porém em comparação com o OpenDJ, sua média é inferior em todos os resultados, com exceção para os experimentos realizados com 10 clientes no FreeBSD com OpenDJ. Além disso, os experimentos apresentaram que o 389-Directory não é estável, tendo em vista que quando o OpenDJ foi instanciado no CentOS, os resultados apresentaram coerência.

Em linhas gerais, o OpenDJ, quando comparado aos outros diretórios LDAP, comportou-se melhor em todos os sistemas operacionais. Além disso, houve variação de desempenho de acordo com o número de clientes. Por exemplo, quando instanciado com 50 clientes, o serviço de diretório obteve um desempenho superior ao quando instanciando com 10 e 25 clientes. Tal fato se deve a capacidade de vazão do OpenDJ.

O OpenLDAP, obteve um desempenho similar tanto no Ubuntu quanto no Debian, porém, diferentemente do OpenDJ, o desempenho do OpenLDAP é similar quando instanciado com 25 e 50 clientes, mas bem inferior com 10 clientes. Isso também é explicado pela capacidade de vazão do serviço de diretório. Por fim, o 389-Directory foi somente instanciado no CentOS e apresentou um comportamento instável, justificado pelo fato de não ter sido feita nenhuma configuração para melhora do sistema.

Operações de Escrita

O desempenho do RIT-LDAP também foi avaliado nas operações de escrita. A primeira a ser realizada é a operação de adição (*ldapadd*), onde cada cliente faz a solicitação de adição para o *gateway*, repassando todos os atributos necessários para criação de um entrada nova. Os atributos enviados são: *ObjectClass*, cn (*common name*), sn (*surname*) e *description*. A segunda operação é a de modificação (*ldapmodrdn*), onde as entradas adicionadas pela requisição *ldapadd* são modificadas em 1B no atributo *description*. Por fim, na operação de deleção (*ldapdelete*), todas as entradas dos serviços de diretório LDAP, previamente adicionadas e modificadas, são apagadas.

Um detalhe a ser observado é que as operações de escrita são processadas de forma diferente pela biblioteca BFT-SMaRt. Durante a execução de uma operação de escrita, os clientes enviam seus pedidos para todas as réplicas, em lotes, e esperam por suas respostas. O ordenamento total das mensagens é realizado através de uma sequência de instâncias de consenso entre as réplicas.

OpenDJ

A Figura 5.8 ilustra o desempenho do RIT-LDAP com OpenDJ como serviço de diretório sobre os quatro sistemas operacionais testados.

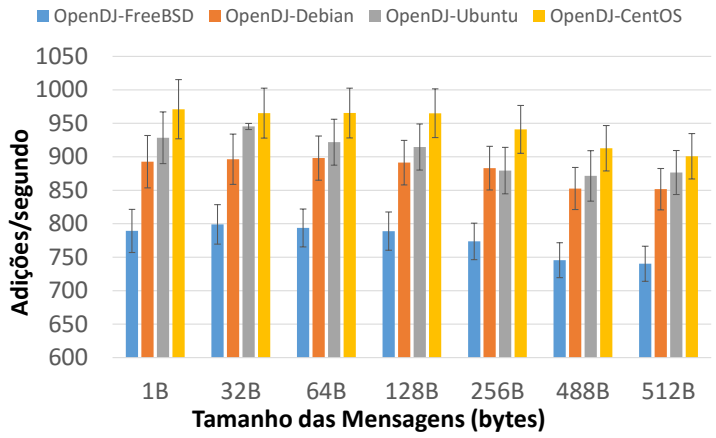
Como pode-se perceber, no **OpenDJ**, o desempenho dos sistemas operacionais nas operações de escrita se comportaram igualmente as operações de busca, onde o mais eficiente foi o CentOS e o FreeBSD foi quem apresentou o pior desempenho.

Na operação de adição, Figura 5.8(a), observa-se o menor resultado com 740 adições por segundo, no FreeBSD, e o melhor resultado com 970 adições por segundo no CentOS. Já na operação de modificação, Figura 5.8(b), no melhor caso (CentOS) chegou-se a 1.300 modificações por segundo com mensagens de 1B, enquanto no pior caso as operações chegaram a 1.030 modificações por segundo com o Freebsd e mensagens de 488B. Por fim, na operação de deleção, Figura 5.8(c), as operações variaram entre 820 deleções por segundo (FreeBSD) a 1.080 deleções por segundo (CentOS).

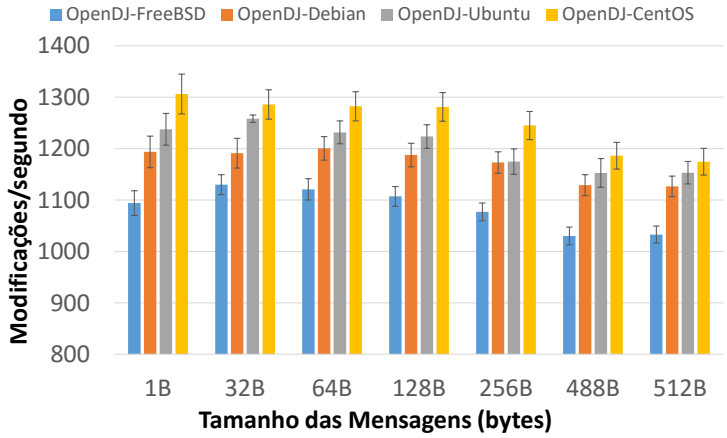
O sistema no FreeBSD possui um desempenho bem inferior aos outros, pois o sistema operacional é mais fechado e não permite ajustes de forma simplificada. Para obter um melhor desempenho, é necessário ajustar configurações.

389-Directory

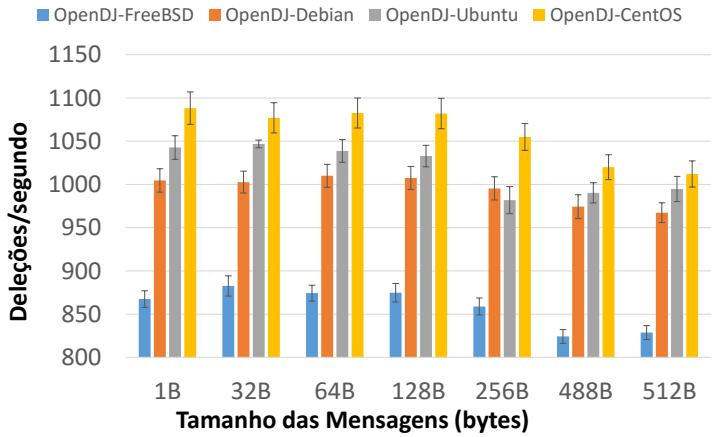
Avaliando os resultados do **389-Directory** sobre o sistema operacional CentOS (Figura 5.9), resultados obtidos em duas ocasiões diferentes, percebe-se que



(a) Adição



(b) Modificação



(c) Deleção

Figura 5.8: Operações de adição, modificação e deleção com OpenDJ.

as operações apresentam um desempenho instável, não mantendo uma média constante. Nas Figuras 5.9(a), 5.9(c) e 5.9(e) os experimentos foram realizados em horários onde os servidores poderiam estar em uso por outros pesquisadores e, por isso, assumiu-se tal fato para explicar os resultados instáveis apresentados. Contudo, para fins de verificação, foram realizados novamente os experimentos em horários onde os servidores estariam com disposição exclusiva (Figuras 5.9(b), 5.9(d) e 5.9(f)) e percebeu-se que o sistema ainda se comporta de forma instável.

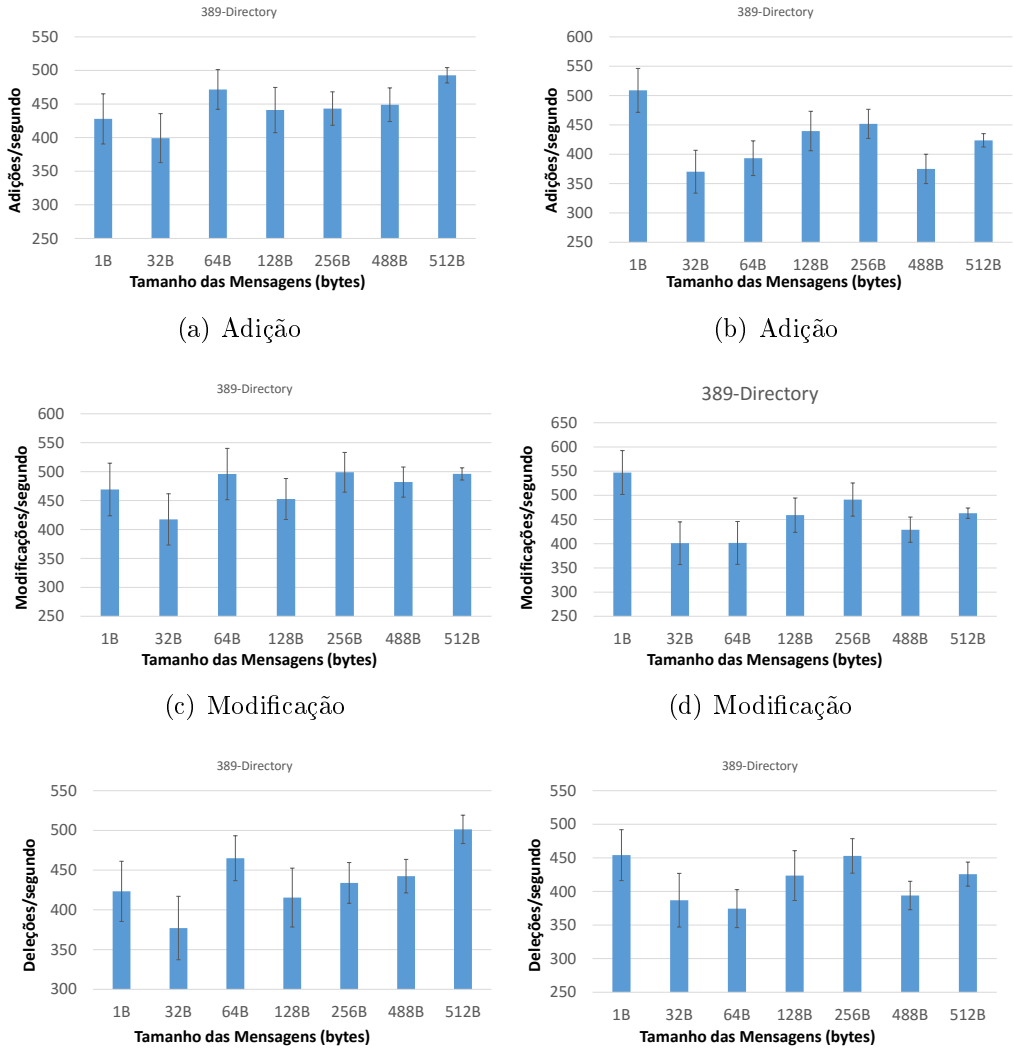
Analisando os dois experimentos, pode-se concluir que uma explicação plausível para o desempenho ruim é a questão da falta de ajuste no sistema operacional em questão. Vale lembrar que o 389-Directory só pode ser instanciado no CentOS.
OpenLDAP

Nos testes com o **OpenLDAP**, todas as operações apresentam um desempenho inferior aos testes com os demais serviços de diretório (OpenDJ e 389-Directory). O desempenho nas operações de adição, Figura 5.10(a), apresenta, no pior caso, um valor de 264 adições por segundo (Debian) com mensagens de 488B e 290 adições por segundo (Ubuntu) com mensagens de 512B. No melhor caso chega a 287 adições por segundo (Debian) para mensagens de 64B e 312 adições por segundo (Ubuntu) para mensagens de 32B. Já na operação de modificação, Figura 5.10(b), no melhor caso (Ubuntu) chegou-se a 363 modificações por segundo com mensagens de 32B, enquanto no pior caso as operações chegaram a 304 modificações por segundo (Debian) para mensagens de 488B. Por fim, na operação de deleção, Figura 5.10(c), as operações alcançaram uma média de 247 deleções por segundo (Debian) com mensagens de 32B e 294 deleções por segundo (Ubuntu) com mensagens de 64B.

Em todos os experimentos realizados, o intervalo de confiança mostra que o Ubuntu instanciado com o OpenLDAP apresentou um desempenho ligeiramente superior ao Debian também instanciado com o OpenLDAP. Além disso, o OpenLDAP no Ubuntu não varia de desempenho a medida que o tamanho da mensagem cresce.

Discussão

No que diz respeito ao resultados das operações, excluindo a operação de busca que é a mais eficiente devido a natureza do diretório LDAP, é possível afirmar que o processo de adição com o RIT-LDAP, dentre todas as operações de escrita, é a menos eficiente, pois o cliente tem que passar todas as informações para criar uma entrada no serviço de diretório. Na operação de modificação, o serviço de diretório LDAP faz buscas pelo identificador da entrada (nome do usuário e base DN), não pelo atributo a ser alterado, e por isso essa é a operação mais eficiente. Por fim, na operação de deleção, onde todas as entradas dos serviços de

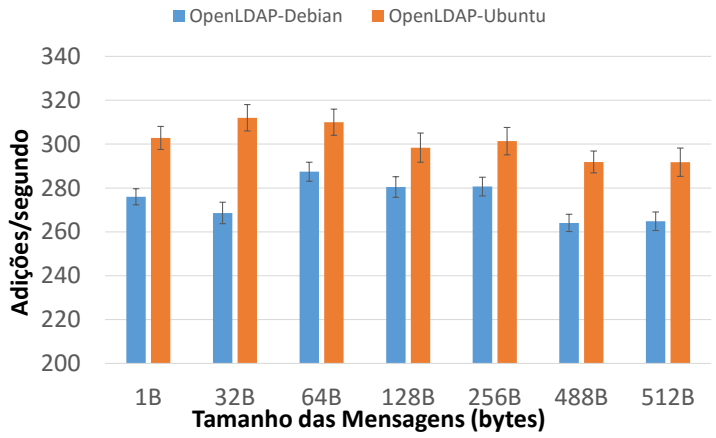


(a) Adição (b) Adição

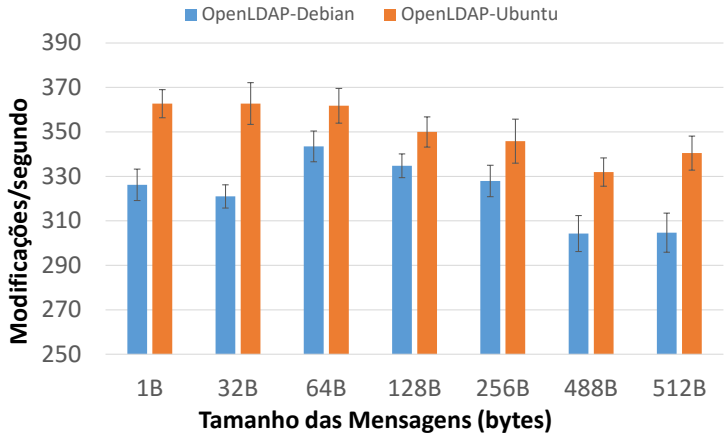
(c) Modificação (d) Modificação

(e) Deleção (f) Deleção

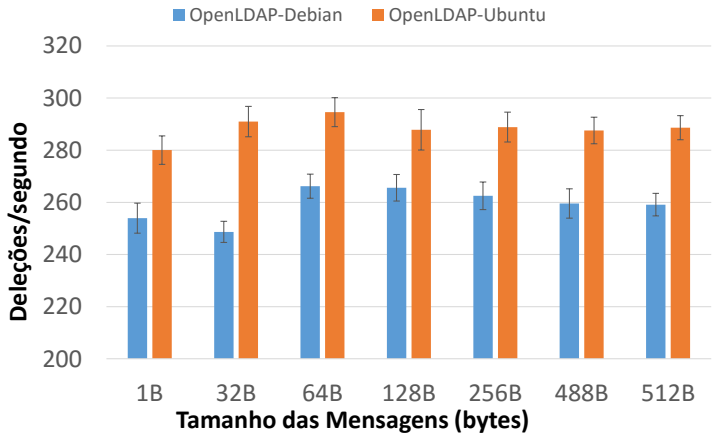
Figura 5.9: Operações de adição, modificação e deleção com 389-Directory no CentOS.



(a) Adição



(b) Modificação



(c) Deleção

Figura 5.10: Operações de adição, modificação e deleção com OpenLDAP.

diretório LDAP são apagadas, percebe-se um resultado médio entre os resultados das operações de adição e modificação. Isto ocorre porque é necessário passar como parâmetro somente o identificador da entrada. Porém, o serviço de diretório necessita realizar toda a operação de deleção para completar a requisição do cliente.

No que diz respeito aos serviço de diretórios, conforme mostram as Figuras 5.8, 5.9 e 5.10, o desempenho do OpenDJ é muito superior ao do OpenLDAP e ao do 389-Directory em todas as operações de escrita. O mesmo pode ser dito em relação as operações de busca.

Em relação aos sistemas operacionais, os experimentos realizados nos sistemas operacionais e levando em consideração que os mesmos foram instalados usando a versão padrão (sem melhorias ou ajustes), o FreeBSD, descendente do BSD, apresentou o pior desempenho, enquanto o Debian e Ubuntu, ambos da família Debian, apresentaram desempenho similar. Porém o desempenho do Ubuntu foi ligeiramente inferior ao do Debian, fato explicado devido ao Ubuntu ser baseado no Debian. Por fim, o CentOS, da família *Red Hat Linux*, apresentou o melhor desempenho dentre os sistemas operacionais testados.

No que diz respeito à biblioteca BFT-SMaRt, os experimentos com operações de escrita (Figuras 5.8, 5.10 e 5.9) apresentaram variação de desempenho muito inferior em comparação com as operações de busca (Figuras 5.1 a 5.7). Esta variação pode ser explicada pela segurança trazida pela BFT-SMaRt. Ao suportar operações de escrita totalmente ordenadas e serializadas nas réplicas, a BFT-SMaRt mantêm a consistência forte dos dados, mas obriga o *gateway* a esperar por todos os lotes de respostas de todas as réplicas SMR para, somente então, realizar a função de consenso. Assim, qualquer atraso, inconsistência ou erro em um dos serviços LDAP pode ocasionar uma demora na tomada de decisão ou até mesmo o reenvio de todas as solicitações para as réplicas.

Por fim, cabe esclarecer que não foi possível validar os testes das operações de escrita com diversidade de serviços de diretório. Nos testes realizados, notou-se que existe uma grande diferença de desempenho entre eles, induzindo a biblioteca BFT-SMaRT a falhar ao realizar a difusão das mensagens, pois como cada réplica (sistema operacional e serviço de diretório) trabalha em tempos diferentes, o *gateway* ou recebe mensagens muito rapidamente ou espera longamente por informações, o que gera *timeouts* e acarreta falhas.

5.2.2 Cenário 2

Como mencionado no início deste Capítulo, os resultados do Cenário 2 compreendem somente a operação de *search*. A Figura 5.11 apresenta os resultados do **Cenário 2** para operações de busca. Vale lembra que neste cenário tem-se duas réplicas com Ubuntu, sendo uma com OpenDJ e outra com OpenLDAP; e duas

réplicas com CentOS, sendo uma com 389-Directory e outra com OpenDJ.

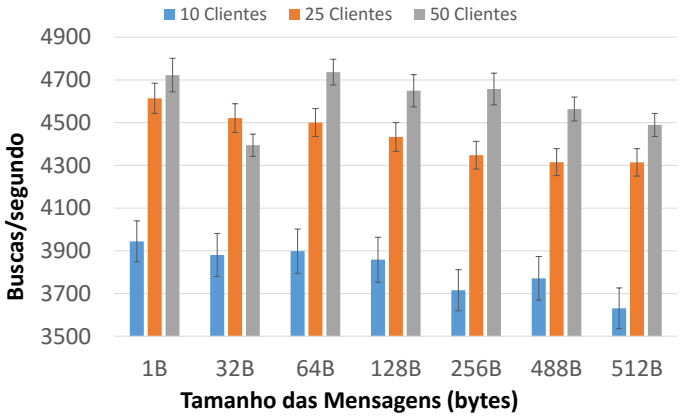


Figura 5.11: Vazão do RIT-LDAP no **Cenário 2**.

Observa-se que o desempenho médio do RIT-LDAP é superior à 4000 buscas por segundo. Comparado ao trabalho de Shoker e Bahsoun [12], que alcança 1.200 buscas por segundo com mensagens de 0B usando 20 clientes, o RIT-LDAP é quase 4 vezes melhor, chegando, no melhor caso, a 4.736 buscas por segundo para 50 clientes e mensagens de 64B. Esse desempenho deve-se a eficiência do BFT-SMaRt nas operações de busca (*ldapsearch*) de forma não ordenada e não serializada.

Também percebe-se na Figura 5.11 que existem diferenças no desempenho das buscas em relação a Figura 5.7, melhor caso no cenário 1. Isto ocorre porque os serviços de diretório utilizados (OpenLDAP e 389-Directory) possuírem um desempenho inferior ao OpenDJ. Além disso, a diversidade de sistemas operacionais, incluída neste cenário, impacta diretamente no desempenho.

O intervalo de confiança mostra que quando o sistema está executando com 25 e 50 clientes, ele possui um desempenho superior a quando executando com 10 clientes. Porém, pode-se observar que com mensagens de 1B e 32B, 25 e 50 clientes apresentam desempenho similares, mas a partir de 64B, o cenário com 50 clientes possui desempenho melhor do que com 25 clientes.

5.2.3 Cenário 3

Assim como o Cenário 2, os resultados do Cenário 3 compreendem somente a operação de *search*. A Figura 5.12 mostra os resultados do sistema no **cenário 3**. Vale lembra que neste cenário tem-se quatro réplicas diferentes, sendo a primeira composta por Ubuntu e OpenLDAP, a segunda por Debian e OpenDJ, a terceira por FreeBSD e OpenDJ e a quarta por CentOS e 389-Directory.

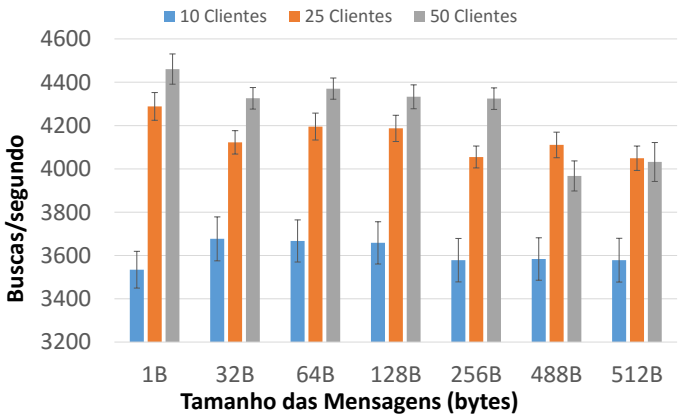


Figura 5.12: Vazão do RIT-LDAP no **Cenário 3**.

Os resultados apresentados evidenciam que o RIT-LDAP ainda apresenta um desempenho muito superior ao do trabalho de Shoker e Bahsoun [12], comportando-se, no pior caso, com 3.534 buscas por segundo para mensagens de tamanho de 1B e 10 clientes simultâneos, e no melhor caso com 4.460 buscas por segundo para mensagens de tamanho de 1B e 50 clientes simultâneos. Assim, no pior caso, o RIT-LDAP apresenta um resultado quase 3 vezes maior que o BFTLDAP (Shoker e Bahsoun) e bem próximo de 4 vezes maior no melhor caso. Analisando o intervalo de confiança, percebe-se que com mensagens de até 256B, 50 clientes possuem um desempenho melhor do que com 25 clientes, porém com mensagens de 488B e 512B, 25 e 50 clientes possuem desempenho similar.

Contudo, quando comparado ao cenário anterior (Figura 5.11), o desempenho torna-se inferior para todos os tamanhos de mensagem e número de clientes. O principal motivo para essa queda de desempenho é a maior diversidade apresentada nessa configuração. Neste cenário, os clientes recebem respostas de suas solicitações somente quando todos os serviços LDAP enviem suas respostas e o *gateway* obtenha o consenso. Como as implementações são distintas, o OpenDJ é baseado em Java enquanto o OpenLDAP e o 389-Directory são binários executáveis, não é possível verificar ou realizar otimizações. Além disso, na mesma linha, todos os sistemas operacionais foram instalados de forma padrão, sem melhorias e otimizações que pudesse permitir melhores desempenhos.

5.2.4 Escalabilidade de Clientes

A Figura 5.13 apresenta o número máximo de clientes que o RIT-LDAP consegue suportar.

Para esse teste, foi utilizado o **cenário 3** comparando o RIT-LDAP com o

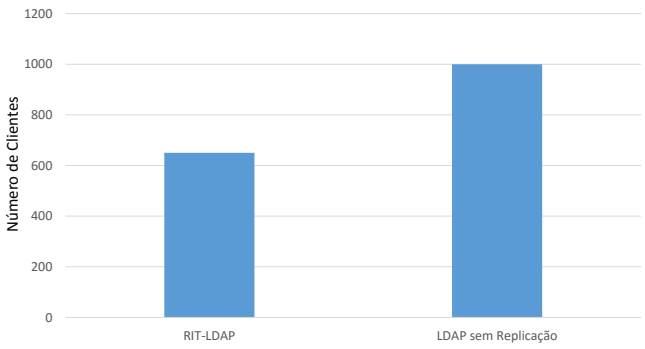


Figura 5.13: Escalabilidade RIT-LDAP x LDAP sem Replicação.

OpenLDAP sem replicação. Enquanto a versão sem replicação do OpenLDAP consegue suportar até 1.000 clientes, o RIT-LDAP suporta 650 clientes. Isto ocorre principalmente devido ao número que conexões que o sistema abre, ou seja, enquanto a comunicação do OpenLDAP sem replicação somente abre uma conexão cliente/servidor, o RIT-LDAP abre a conexão do cliente com o *gateway* e o *gateway* abre uma conexão para cada réplica SMR, fazendo assim com que o sistema atinja o gargalo da rede mais rapidamente. Além disso, após o RIT-LDAP alcançar seu limite de escalabilidade, os clientes que estavam sendo atendidos, continuaram recebendo as requisições normalmente, somente conexões de novos clientes que estavam sendo recusadas.

Entretanto, é preciso considerar que o OpenLDAP sem replicação não possui nenhum tipo de segurança, enquanto o RIT-LDAP é tolerante a falhas e intrusões conforme apresentado na seção 3.3. Vale ressaltar que Shoker e Bahsoun [12] apresentaram em seu trabalho as limitações do BFTLDAP em comparação com o OpenLDAP sem replicação, e constataram que o sistema proposto suporta somente até 200 clientes, muito inferior ao obtido pelo RIT-LDAP.

5.2.5 Comportamento mediante Paradas

A Figura 5.14 apresenta o comportamento do RIT-LDAP durante 311 segundos, mediante falhas, recuperações e reconfigurações das réplicas SMR em comparação com uma execução normal.

Neste experimento, foram instanciados 50 clientes, onde cada cliente realizou 30.000 operações de busca com mensagens de 1B. O **Cenário 3** foi utilizado para os testes, ou seja, foi utilizada a diversidade total do sistema. O RIT-LDAP é iniciado somente com 3 réplicas. No início das execuções, as conexões começam a ser feitas, deixando uma média de 4.000 buscas por segundo, superior às 3.200 realizadas pelo RIT-LDAP iniciando com 4 réplicas.

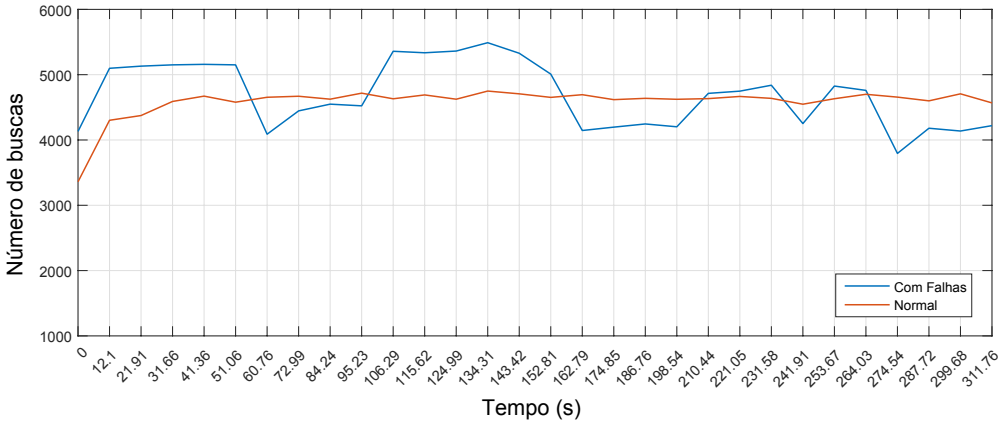


Figura 5.14: Comportamento RIT-LDAP mediante eventos nas Réplicas SMR.

Do segundo 12.1 ao segundo 51.06, quando as conexões já estão estabelecidas, as buscas ficam em torno de 5.150 buscas. Este aumento é explicado pelo simples fato do *gateway* receber somente resposta de três réplicas, tornando o consenso mais eficiente. No segundo 51.06, a réplica 04 é iniciada e no segundo 60.76, quando todas as conexões estão estabelecidas entre as 4 réplicas, observa-se uma queda para 4.000 buscas por segundo. Do segundo 72.99 ao segundo 95.23, as operações ficam em torno de 4.500 buscas por segundo, semelhante ao RIT-LDAP normal. Isto ocorre pois com 4 réplicas há uma troca maior de mensagens com o *gateway*.

Nos segundos 106.29 ao 143.42, quando a réplica 01 é derrubada, as operações chegam próximo a 5.500 buscas por segundo. Isto se deve ao fato da réplica 01 ser implementada com OpenLDAP, o diretório LDAP mais lento dentre os utilizados. Então, as mensagens chegam mais rápido ao *gateway* para o consenso. No segundo 152.81, a réplica 01 é novamente iniciada e assim o sistema volta a utilizar 4 réplicas. Já no segundo 210.44, a réplica 02, que utiliza o OpenDJ como serviço de diretório, é desligada, acarretando assim uma variação mínima de desempenho em relação as rodadas normais.

Por fim, na segundo 264.03, a réplica 02 é novamente iniciada e mais uma vez o sistema sofre queda no desempenho. Os segundos seguintes apresentam o sistema recuperando o desempenho até ficar próximo a execução normal.

Em linhas gerais, o desempenho do RIT-LDAP é influenciado diretamente pelo número de réplicas em funcionamento. No experimento realizado, quando há somente 3 réplicas operando, o sistema possui um desempenho melhor, pois o *gateway* recebe apenas 3 respostas para fazer o consenso e não precisa descartar a resposta mais lenta. Porém, se o sistema estiver com as 4 réplicas operando normalmente, a velocidade diminui tendo em vista que o *gateway* recebe uma

resposta a mais, mesmo que esta mensagem seja descartada no final.

5.2.6 Tolerância a Falhas Bizantinas

Por padrão, diretórios LDAP não toleram falhas arbitrárias (e.g., bugs, configurações erradas, atraso de mensagens por causa de um atacante). Visto que tais falhas podem ter diferentes tipos de impacto no funcionamento do sistema, como recusar um autenticação de um usuário ou responder um busca com um resultado inválido, alguma solução precisa ser dada. Por empregar a biblioteca BFT-SMaRt, a solução proposta no RIT-LDAP tolera f_R falhas nas réplicas LDAP, uma vez que a biblioteca implementa o algoritmo Paxos at war [52], responsável pelo consenso bizantino. Assim, com a utilização desses mecanismos, pode-se afirmar que o RIT-LDAP é tolerante falhas bizantinas.

5.2.7 Capacidade e dimensionamento do sistema

No trabalho de Kreutz *et al.* [45], são apresentados dados de um ambiente universitário real, com aproximadamente 11.500 usuários, com as seguintes características:

1. O número máximo de autenticações, na hora de pico do uso, atinge 143.907 autenticações, ou seja, média de 39,97 autenticações por segundo;

2. O maior pico de autenticações, somando o pior segundo dos dois sistemas de autenticação existentes no ambiente (Active Directory e OpenLDAP), atinge-se 118 autenticações por segundo.

Entretanto, em mais de 99.99% do tempo, o número de autenticações por segundo fica abaixo de 100. Cabe ressaltar ainda que esses números representam as demandas de autenticação de praticamente todos os sistemas da instituição, incluindo login em computadores de laboratórios ou de uso pessoal, logins de acesso à rede (e.g. autenticações 802.1x), autenticações em sistemas *Web online*, verificações de usuários nos servidores SMTP, entre outros.

A partir desses números pode-se inferir que se o RIT-LDAP fosse empregado no ambiente universitário descrito como *back-end* para o serviço de autenticação, ele seria capaz de suportar a demanda de ambientes com algumas dezenas de milhares de usuários. Além disso, considerando o pior caso, o RIT-LDAP é capaz de atender aos dois serviços de diretório LDAP utilizados naquele ambiente, sem a necessidade de qualquer modificação na infraestrutura já existente.

Capítulo 6

Conclusão

Este Capítulo finaliza esta dissertação de mestrado e apresenta: (i) as considerações finais do trabalho; (ii) as principais contribuições do trabalho desenvolvido; e (iii) as possibilidades de trabalhos futuros.

6.1 Considerações Finais

Neste trabalho foi proposto uma arquitetura para desenvolvimento serviços de diretórios LDAP resiliente, capaz de tolerar falhas arbitrárias e intrusões, mantendo a compatibilidade com infraestruturas LDAP já existentes. A arquitetura proposta utilizou técnicas de tolerância a falhas e intrusões, como replicação de máquinas de estado e diversidade de sistemas aplicada aos serviços de diretórios e sistemas operacionais.

Os conceitos necessários para compreender cada componente e o funcionamento da proposta foram apresentados e explicados e um protótipo funcional foi projetado e implementado como prova de conceito. Nos mais diversos experimentos realizados com o protótipo, foi possível observar a viabilidade de implantação de diretórios LDAP resilientes.

Ainda utilizando o protótipo desenvolvido, demonstrou-se que o sistema é capaz de mascarar até f_R falhas nas réplicas SMR. Além disso, o *gateway* LDAP foi implementado a partir do modelo funcional da SecFuNet, para realizar comunicação entre os clientes e as réplicas SMR, mascarando assim o protocolo BFT e permitindo que sistemas já desenvolvidos e em funcionamento possam fazer uso do RIT-LDAP sem alterações para o cliente.

Ainda mais, foi avaliado e discutido a escalabilidade do sistema em relação a um LDAP sem replicação. Embora o *gateway* LDAP precise fazer 4 conexões, uma com cada réplica SMR, o sistema foi capaz de suportar até 650 clientes simultâneos, menos do que o LDAP normal, porém com propriedades para tolerar

falhas e intrusões

Também discutiu-se o comportamento do RIT-LDAP mediante paradas nas réplicas SMR e percebeu-se que o sistema atende ao propósito da replicação de máquinas de estado. Cada vez que uma réplica SMR parava, o sistema continuou em pleno funcionamento, bem como aumentava seu desempenho, uma das características da biblioteca empregada neste sistema.

Por fim, em comparação com outros trabalhos foi possível estimar a capacidade de dimensionamento do sistema. Os resultados obtidos nos experimentos demonstraram que o sistema é capaz de atender a demanda de um ambiente real, atendendo a necessidades de ambientes de produção como, por exemplo, uma universidade com mais de 136K usuário

6.2 Dificuldades Encontradas

As principais dificuldades encontradas ao longo do desenvolvimento desta dissertação foram:

- Adaptação do ambiente - Como cada sistema operacional possui características diferentes, era necessário adaptar a configuração dos serviços de diretórios. Por exemplo, no FreeBSD precisou-se fazer adaptações nos *scritps*, como variáveis de ambiente e comandos do terminal. No CentOS, o *firewall* padrão precisou ser desligado para realização dos experimentos. Com relação ao Ubuntu e o Debian, nada precisou ser feito.

- LDAP - Em relação aos serviços de diretório, cada um tinha suas especificidades. O CentOS faz o *Bind* de modo diferente. Por exemplo, o usuário precisava informar somente o *cn* (*Common Name*) enquanto os outros serviços de diretório precisavam informar o *DN* completo. O OpenDJ precisou ter desabilitadas certas configurações para realização das operações de escrita. O ApacheDS não foi capaz de fazer parte dos experimentos, pois, embora atenda as especificações do LDAP v3.0, o mesmo não realiza operações quando instanciado com vários clientes.

Levando em consideração esses fatores, foi necessário um grande estudo para o entendimento e adaptação dos sistemas operacionais e serviços de diretórios para executar com o RIT-LDAP.

6.3 Trabalhos Futuros

Embora esta dissertação apresente resultados relevantes, ela também abre caminho para novas pesquisas. Os trabalhos futuros estão elencados a seguir:

- Aplicação de outras técnicas que aumentem ainda mais a resiliência do sistema - Uma solução é aplicar o rejuvenescimento das réplicas. Assim, em tempos arbitrários, uma réplica aleatória é desligada e uma outra réplica é iniciada com um sistema operacional e serviço de diretório diferente, aumentando ainda mais a robustez e resiliência do sistema.

Referências Bibliográficas

[1] W. Zhou and C. Meinel, "Implement role based access control with attribute certificates," in *In Proceedings of the 6th International Conference on Advanced Communication Technology (ICACT)*, vol. 1, pp. 536–540, 2004.

[2] V. Koutsonikola and A. Vakali, "Ldap: framework, practices, and trends," *IEEE Internet Computing*, vol. 8, no. 5, pp. 66–72, 2004.

[3] M. Flechl and L. Field, "Grid interoperability: joining grid information systems," *Journal of Physics: Conference Series*, vol. 119, no. 6, p. 7, 2008.

[4] S. Lim, J. Choi, and K. Zeilenga, "Secure and flexible certificate access in ws-security through ldap component matching," in *Proceedings of the 2004 Workshop on Secure Web Service (SWS)*, pp. 87–96, 2004.

[5] T. Howes, M. Smith, and G. Good, *Understanding and Deploying LDAP Directory Services*. Addison-Wesley, 2 ed., 2003.

[6] J. Sermersheim, "RFC 4511: Lightweight directory access protocol (LDAP): The protocol," *Internet Engineering Task Force*, 2006.

[7] J.-C. Laprie, "From dependability to resilience," in *Proceedings of the 38th IEEE/IFIP International Conference On Dependable Systems and Networks (DSN)*, p. 2, 2008.

[8] J. Myers, "RFC 4422: Simple authentication and security layer (SASL)," *Internet Engineering Task Force*, 1997.

[9] T. Dierks, "RFC 5246: The transport layer security (TLS)," *Internet Engineering Task Force*, 2008.

[10] X. Wang, H. Hou, and Y. Zhuang, "Secure byzantine fault tolerant LDAP system," in *Proceedings of the First International Multi-Symposiums on Computer and Computational Sciences (IMSCCS)*, pp. 34–39, 2006.

[11] H. Hou, X. Wang, and M. Wu, "Hierarchical byzantine fault tolerant secure ldap," in *Proceeding of the IEEE International Conference on Systems, Man and Cybernetics Society (SMC)*, vol. 5, pp. 3844–3849, 2006.

[12] A. Shoker and J.-P. Bahsoun, "Towards byzantine resilient directories," in *Proceeding of the IEEE International Symposium on Network Computing and Applications (NCA)*, pp. 52–60, 2012.

[13] G. Coulouris, J. Dollimore, and T. Kindberg, *Distributed systems: Concepts and Design*. Pearson Education, 2005.

[14] A. Tanenbaum and M. Van Steen, *Distributed systems: Principles and Paradigms*. Prentice Hall Englewood Cliffs, 2 ed., 2002.

[15] C. Unicode Staff, *The Unicode Standard: Worldwide Character Encoding*. Addison-Wesley, 1 ed., 1991.

[16] G. Carter, *LDAP system administration*. O'Reilly Media, 1 ed., 2003.

[17] K. Zeilenga, "RFC 5412: Lightweight directory access protocol (LDAP): Directory information models," *Internet Engineering Task Force*, 2006.

[18] G. Good, "RFC 2849: The ldap data interchange format (LDIF) - technical specification," *Internet Engineering Task Force*, 2000.

[19] C. Trigo, *OpenLDAP: Uma Abordagem Integrada*. Novatec Editora, 1 ed., 2007.

[20] J. Pearsall and P. Hanks, *The new Oxford dictionary of English*. Clarendon Press, 1998.

[21] M. Wiesmann, F. Pedone, A. Schiper, B. Kemme, and G. Alonso, "Understanding replication in databases and distributed systems," in *Proceeding of the 20th International Conference on Distributed Computing Systems (ICDCS)*, pp. 464–474, 2000.

[22] D. Kreutz, H. Niedermayer, E. Feitosa, J. Fraga, and O. Malichevskyy, "Architecture components for resilient networks," 2013.

[23] F. Schneider, "Distributed systems," ch. Replication Management Using the State-machine Approach, pp. 169–197, Addison-Wesley, 2 ed., 1993.

[24] M. Castro and B. Liskov, "Practical byzantine fault tolerance," in *Proceedings of the Third Symposium on Operating Systems Design and Implementation (OSDI)*, pp. 173–186, 1999.

[25] S. Neti, A. Somayaji, and M. Locasto, "Software diversity: Security, entropy and game theory.," in *Proceedings of the 7th USENIX Workshop on Hot Topics in Security - HotSec*, p. 7, USENIX, 2012.

[26] M. Garcia, A. Bessani, I. Gashi, N. Neves, and R. Obelheiro, "Analysis of operating system diversity for intrusion tolerance," *Software: Practice and Experience*, vol. 44, no. 6, pp. 735–770, 2014.

[27] M. Garcia, A. Bessani, I. Gashi, N. Neves, and R. Obelheiro, "Os diversity for intrusion tolerance: Myth or reality?," in *Proceedings of the IEEE/IFIP 41st International Conference on Dependable Systems and Networks (DSN)*, pp. 383–394, 2011.

[28] V. Cogo, *Diversity in automatic cloud computing resource selection.* Tese de Doutorado, Faculdade de Ciências, Universidade de Lisboa, 2012.

[29] I. Gashi, P. Popov, and L. Strigini, "Fault tolerance via diversity for off-the-shelf products: A study with sql database servers," *IEEE Transactions on Dependable and Secure Computing,*, vol. 4, no. 4, pp. 280–294, 2007.

[30] R. Kotla, L. Alvisi, M. Dahlin, A. Clement, and E. Wong, "Zyzzyva: speculative byzantine fault tolerance," in *Proceedings of the Special Interest Group on Operating Systems (SIGOPS)*, vol. 41, pp. 45–58, 2007.

[31] A. Clement, M. Kapritsos, S. Lee, Y. Wang, L. Alvisi, M. Dahlin, and T. Riche, "Upright cluster services," in *Proceedings of the Special Interest Group on Operating Systems (SIGOPS), 22nd symposium on Operating systems principles*, pp. 277–290, 2009.

[32] A. Bessani, J. Sousa, and E. Alchieri, "State machine replication for the masses with BFT-SMaRt," in *Proceedings of the IEEE/IFIP 44th International Conference on Dependable Systems and Networks (DSN)*, pp. 355–362, 2014.

[33] B. Brito, "Evolução da biblioteca para replicação tolerante a faltas bizantinas BFT-SMaRt," Dissertação de Mestrado, Faculdade de Ciências, Universidade de Lisboa, 2011.

[34] V. Hadzilacos and S. Toueg, "A modular approach to fault-tolerant broadcasts and related problems," tech. rep., Cornell University, 1994.

[35] O. Malichevskyy, D. Kreutz, M. Pasin, and A. Bessani, "O vigia dos vigias: um serviço RADIUS resiliente," in *Proceedings of the IV INForum Simpósio de Informática*, p. 12, 2012.

[36] H. Cunha, "An architecture to resilient and highly available identity providers based on openid standard," Dissertação de Mestrado, Instituto de Computação, Universidade Federal do Amazonas, 2014.

[37] J. Sousa, A. Bessani, and P. Sousa, "Typhon: um serviço de autenticação e autorização tolerante a intrusões," in *Proceedings of the II INForum Simpósio de Informática*, pp. 649–660, 2010.

[38] T. Cui, "LDAP directory template model on multi-master replication," in *Proceedings of the IEEE Asia-Pacific Services Computing Conference (APSCC)*, pp. 479–484, 2010.

[39] A. Kumar, "Efficient filter based replication for ldap directories," in *Proceedings of the IEEE 23rd International Conference on Data Engineering (ICDE)*, pp. 1373–1375, 2007.

[40] F. Vieira, P. Sousa, and A. N. Bessani, "Transparent byzantine fault-tolerant directory service using cots components," in *In Proceedings of the 39th IEEE International Conference on Dependable Systems and Networks (DSN)*, p. 2, 2009. Fast Abstract.

[41] R. Mendonca-Neto, B. Barreto, D. Kreutz, A. Santos, and E. Feitosa, "Fit-ldap: Um serviço de diretório tolerante a falhas e intrusões," in *Proceedings of the XV Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais (SBSEG)*, pp. 44–47, 2015.

[42] M. Prince, "The DDoS that almost broke the internet," 2013. http://goo.gl/oeDrMY.

[43] D. Kreutz and E. Feitosa, "Identity providers-as-a-service built as cloud-of-clouds: challenges and opportunities," in *Proceedings of the Federated Conference on Computer Science and Information Systems (FedCSIS)*, vol. 3, pp. 101–108, 2014. Position Papers.

[44] D. Kreutz, O. Malichevskyy, E. Feitosa, K. Barbosa, and H. Cunha, "System design artifacts for resilient identification and authentication infrastructures," in *Proceedings of the 18th International Conference on Networking and Services (ICNS)*, pp. 41–47, 2014.

[45] D. Kreutz, A. Bessani, E. Feitosa, and H. Cunha, "Towards secure and dependable authentication and authorization infrastructures," in *Proceedings of the IEEE 20th Pacific Rim International Symposium on Dependable Computing (PRDC)*, pp. 43–52, 2014.

[46] D. Kreutz, E. Feitosa, H. Cunha, H. Niedermayer, and H. Kinkelin, "Increasing the resilience and trustworthiness of openid identity providers for future networks and services," in *Proceedings of the 9th IEEE International Conference on Availability, Reliability and Security (ARES)*, pp. 317–324, 2014.

[47] C. Dwork, N. A. Lynch, and L. Stockmeyer, "Consensus in the presence of partial synchrony," *Journal of the ACM (JACM)*, vol. 35, no. 2, pp. 288–322, 1988.

[48] UnboundID, "UnboundID LDAP SDK for Java," 2015. https://www.ldap.com/unboundid-ldap-sdk-for-java.

[49] BFT-SMaRt, "Bft-smart," 2015. http://bft-smart.github.io/library/.

[50] X. Wang, H. Schulzrinne, D. Kandlur, and D. Verma, "Measurement and analysis of LDAP performance," *IEEE/ACM Transactions on Networking (TON)*, 2008.

[51] G. E. Box *et al.*, *Statistics for experiments: an introduction to design, data analysis, and model building*. John Wiley and Sons, 1978.

[52] P. Zielinski and P. Zieliński, "Paxos at war," in *In Proceedings of the 2001 Winter Simulation Conference*, p. 30, 2004.