Relatório Técnico de Projeto Final de Programação

Raft Consensus Plugin:
Módulo de Consenso Distribuído para GrADyS-SIM NextGen

Laércio Lucchesi (Matrícula: 2312775)
Orientador: Professor Markus Endler

Novembro 2025

1 Introdução e Contexto do Projeto

Este relatório descreve o projeto *Raft Consensus Plugin* — um sistema de consenso distribuído tolerante a falhas baseado no algoritmo Raft, desenvolvido como um módulo para o framework de simulação *GrADyS-SIM NextGen* (https://project-gradys.github.io/gradys-sim-nextgen/). O projeto está inserido no contexto de *Sistemas Distribuídos e Redes*, com ênfase em algoritmos de consenso distribuído e tolerância a falhas em ambientes dinâmicos.

O *GrADyS-SIM NextGen* é um framework de simulação de redes projetado para representar cenários nos quais múltiplos nós coexistem em um ambiente controlado e trocam mensagens entre si. O framework permite a criação de simulações complexas de sistemas distribuídos e está sendo expandido com uma coleção de *plugins* que visam simplificar e acelerar o desenvolvimento de protocolos distribuídos. Essa arquitetura modular torna o *GrADyS-SIM NextGen* uma plataforma de experimentação acessível e extensível para pesquisadores e estudantes do LAC (Laboratory for Advanced Collaboration) e também de outras instituições interessadas fora da DI/PUC.

O *Raft Consensus Plugin* foi desenvolvido como um desses componentes integrados ao framework, fornecendo capacidades de consenso distribuído para protocolos e aplicações que necessitam de coordenação e acordo entre múltiplos nós em ambientes simulados. A implementação fundamenta-se no algoritmo Raft proposto por Ongaro e Ousterhout (2014) (https://dl.acm.org/doi/10.5555/2643634.2643666), mas introduz extensões voltadas a ambientes dinâmicos e cenários de falhas massivas. Diferentemente da implementação clássica do Raft, que se concentra em replicação de logs, o *Raft Plugin* opera sobre valores, permitindo que múltiplas variáveis sejam objeto de consenso simultaneamente — abordagem particularmente adequada a sistemas de coordenação distribuída, como formações de veículos aéreos não tripulados (VANTs), onde posição, velocidade e estado precisam ser consensados de forma contínua e confiável.

O sistema oferece duas modalidades de operação: *Classic*, que reproduz o comportamento padrão do algoritmo Raft, e *Fault-Tolerant*, que adiciona inovações como descoberta dinâmica de nós ativos e cálculo adaptativo de maiorias. Esta última modalidade permite recuperação automática após falhas massivas de nós, ajustando-se dinamicamente ao tamanho efetivo do cluster. Além disso, o plugin incorpora um sistema de detecção de falhas baseado em heartbeats, que possibilita a identificação automática de nós desconectados e a reorganização dinâmica da rede.

O plugin apresenta uma limitação conhecida que, por sua vez, configura um problema de pesquisa em aberto: a detecção de nós ativos falha em convergir para a estabilidade quando o alcance de comunicação é insuficiente para cobrir todo o enxame, resultando em ciclos repetidos de eleição e instabilidade generalizada.

O código-fonte completo do projeto, sob licença MIT, juntamente com documentação técnica, exemplos de uso e referência de API, está disponível publicamente no repositório https://github.com/Project-GrADyS/gradys-sim-nextgen. A implementação foi desenvolvida em Python 3.11+, seguindo boas práticas de engenharia de software.

2 Objetivos Gerais

O desenvolvimento do *Raft Consensus Plugin* tem como propósito central disponibilizar uma implementação pronta, testada e extensível do algoritmo Raft, integrada ao *GrADyS-SIM NextGen*, para apoiar

a pesquisa e o desenvolvimento de protocolos distribuídos. O software se enquadra na área de *Redes e Sistemas Distribuídos*, especificamente no domínio de *algoritmos de consenso distribuído* e mecanismos de tolerância a falhas.

O *GrADyS-SIM NextGen*, em constante expansão por meio de novos *plugins*, foi concebido para oferecer a pesquisadores e estudantes uma infraestrutura modular que reúna funcionalidades comuns em protocolos distribuídos. Nesse contexto, o *Raft Consensus Plugin* representa uma contribuição específica voltada a prover capacidades de consenso generalizado — um dos pilares de coordenação em sistemas distribuídos. Ao oferecer esse componente pronto e bem documentado, o projeto reduz o esforço de implementação, permitindo que pesquisadores concentrem-se nos aspectos inovadores de seus protocolos sem necessidade de reimplementar o mecanismo de consenso.

Entre as principais características que tornam o *Raft Consensus Plugin* valioso, destacam-se:

- a **implementação completa do algoritmo Raft**, com extensões para tolerância a falhas simples e massivas;
- a **flexibilidade de configuração**, por meio de dois modos de operação (*Classic* e *Fault-Tolerant*);
- a **integração transparente** com o *GrADyS-SIM NextGen*, seguindo padrões de arquitetura bem definidos;
- a **detecção automática de falhas**, baseada em heartbeats, que permite adaptação dinâmica da rede a condições adversas;
- e a **abstração da complexidade interna** do algoritmo, por meio de uma API de alto nível que expõe apenas métodos essenciais, como propor valores, consultar valores consensados e verificar o estado de liderança.

O público-alvo do plugin inclui pesquisadores e desenvolvedores da área de sistemas distribuídos interessados em validar algoritmos de consenso ou integrar mecanismos de coordenação em protocolos simulados. O projeto segue uma filosofia de código aberto e documentação extensiva, garantindo transparência, reprodutibilidade e potencial de reutilização por membros do LAC e por colaboradores externos.

Além de seu valor prático, o plugin também fomenta pesquisa sobre estabilidade e convergência de algoritmos de consenso em redes parcialmente conectadas. Um dos objetivos futuros do projeto é justamente abordar o *problema de pesquisa em aberto* identificado — a falta de convergência da detecção de nós ativos em cenários com alcance de comunicação limitado.

Por fim, o *Raft Consensus Plugin* reflete o compromisso do LAC com o desenvolvimento de ferramentas modulares, bem documentadas e acessíveis, que apoiem tanto o ensino quanto a investigação científica em sistemas distribuídos.

3 Requisitos Funcionais e Não Funcionais

3.1 Requisitos Funcionais

Esta seção enumera e descreve os requisitos funcionais mais relevantes do *Raft Consensus Plugin*.

1. **Gerenciamento de Estados Raft** O sistema deve implementar os três estados fundamentais do algoritmo Raft: FOLLOWER, CANDIDATE e LEADER. Cada nó deve poder transicionar entre esses estados conforme as regras do protocolo Raft, gerenciando adequadamente o termo atual, o líder identificado e o estado de voto.

2. **Eleição de Líder com Timeout Aleatório** O sistema deve realizar eleições de líder utilizando timeouts aleatórios dentro de um intervalo configurável (min, max). Quando um nó não recebe heartbeats do líder dentro do timeout, deve iniciar uma eleição, incrementando o termo, votando em si mesmo e solicitando votos dos demais nós. Um nó deve se tornar líder quando obtiver a maioria dos votos dos nós ativos.

3. **Gestão de Variáveis de Consenso** O sistema deve permitir a definição e gerenciamento de múltiplas variáveis de consenso, cada uma com um tipo específico (int, str, float, bool, dict, etc.). O líder deve poder propor novos valores para essas variáveis, que serão replicados para os seguidores e comprometidos quando a maioria dos nós ativos confirmar.

4. **Detecção de Falhas Baseada em Heartbeat** O sistema deve detectar falhas de nós através de monitoramento de heartbeats, mantendo contadores de falhas consecutivas e sucessos consecutivos para cada nó. Quando o número de falhas consecutivas exceder um threshold configurável, o nó deve ser marcado como falho. Quando o número de sucessos consecutivos exceder um threshold de recuperação, o nó deve ser marcado como recuperado.

5. **Descoberta de Nós Ativos (Modo Fault-Tolerant)** No modo fault-tolerant, o sistema deve realizar uma fase de descoberta antes das eleições, enviando mensagens de descoberta (DiscoveryHeartbeat) para todos os nós conhecidos e coletando respostas. Com base no número de nós ativos descobertos, o sistema deve calcular corretamente o threshold de maioria para a eleição, permitindo recuperação mesmo após falhas massivas.

6. **Modos de Operação Dual** O sistema deve suportar dois modos de operação: Classic e Fault-Tolerant. No modo Classic, o sistema usa o tamanho total do cluster para cálculos de maioria, sem fase de descoberta. No modo Fault-Tolerant, o sistema usa contagem dinâmica de nós ativos e realiza descoberta antes das eleições.

7. **Sistema de Mensagens Raft** O sistema deve implementar todos os tipos de mensagens Raft necessários: RequestVote, RequestVoteResponse, AppendEntries, AppendEntriesResponse, DiscoveryHeartbeat e DiscoveryHeartbeatResponse. Todas as mensagens devem suportar serialização e deserialização em JSON, incluindo campos obrigatórios como term, sender_id e informações específicas de cada tipo de mensagem.

8. **Configuração Flexível via Builder Pattern** O sistema deve fornecer uma interface de configuração usando o padrão Builder, permitindo configurar timeouts de eleição, intervalos de heartbeat, variáveis de consenso, modo de operação, detecção de falhas e opções de logging. A configuração deve validar parâmetros e fornecer valores padrão apropriados.

9. **Integração com o GrADyS-SIM** A integração com a plataforma *GrADyS-SIM* deve ocorrer de modo transparente, garantindo compatibilidade total e operação automática, sem necessidade de ações explícitas por parte do usuário.

10. **Controle de Estado de Simulação de Nós** O sistema deve permitir controlar manualmente o estado ativo/inativo de nós na simulação, permitindo simular falhas e recuperações para fins de teste. Quando um nó é marcado como inativo, ele não deve enviar ou receber mensagens, mas os timers internos continuam funcionando normalmente.

11. **Compartilhamento de Informação de Nós Ativos** O líder deve compartilhar informações sobre nós ativos com os seguidores através das mensagens AppendEntries, incluindo tanto a contagem de nós ativos quanto a lista completa de IDs de nós ativos. Os seguidores devem armazenar essas informações e usá-las para cálculos de maioria quando a informação estiver fresca.

12. **Processamento de Mensagens e Timers** O sistema deve processar mensagens recebidas de outros nós, extraindo automaticamente o sender_id quando possível, e encaminhar para o processamento interno do protocolo Raft. O sistema deve também processar timers expirados, incluindo election_timeout, heartbeat e discovery_timeout.

13. **Validação de Variáveis de Consenso** O sistema deve validar que valores propostos para variáveis de consenso correspondem aos tipos configurados. Valores inválidos devem resultar em exceções apropriadas. O sistema deve também evitar propor valores que já estão comprometidos para evitar operações desnecessárias.

14. **Cálculo Dinâmico de Maioria** O sistema deve calcular thresholds de maioria dinamicamente baseado no número de nós ativos, não no número total de nós conhecidos (no modo fault-tolerant). O cálculo deve considerar informações recebidas do líder quando disponíveis, ou usar descoberta local quando necessário.

3.2 Requisitos Não Funcionais

Esta seção enumera e descreve os requisitos não funcionais do *Raft Consensus Plugin*, relacionados à qualidade, desempenho e outras características não funcionais do sistema.

1. **Modularidade** O sistema deve ser organizado em módulos claramente separados por responsabilidade: gerenciamento de estado Raft, processamento de mensagens, configuração e detecção de falhas. Cada módulo deve ter interfaces bem definidas e baixo acoplamento com outros módulos.

2. **Extensibilidade** O sistema deve ser extensível para suportar novos tipos de mensagens e novas estratégias de detecção de falhas sem modificar o código core. A arquitetura deve permitir adicionar funcionalidades através de padrões como Factory e Strategy.

3. **Testabilidade** O código deve ser testável através de testes unitários, permitindo injeção de dependências e mock de callbacks. Componentes críticos como eleição de líder e detecção de falhas devem poder ser testados isoladamente.

4. **Compatibilidade** O sistema deve ser compatível com Python 3.11 ou superior e integrar-se com GradySim 0.7.3 ou superior. A API pública deve manter compatibilidade retroativa dentro de versões principais.

5. **Usabilidade da API** A API pública deve ser intuitiva e fácil de usar, fornecendo uma interface de alto nível (RaftConsensus) que esconde a complexidade interna. O padrão Builder deve ser usado para configuração, permitindo código legível e fluente. Métodos devem ter nomes descritivos e documentação clara.

6. **Documentação** Todas as classes e métodos públicos devem ter docstrings completos descrevendo propósito, parâmetros, retornos e possíveis exceções. Exemplos de uso devem ser fornecidos para funcionalidades principais.

7. **Escalabilidade** O sistema deve funcionar eficientemente com clusters de tamanhos variados, desde pequenos clusters (5-10 nós) até grandes clusters (1000+ nós). O uso de memória deve crescer linearmente com o número de nós, sem overhead quadrático.

8. **Robustez** O sistema deve lidar graciosamente com erros de rede, nós falhando, mensagens corrompidas e condições de corrida. Exceções devem ser tratadas apropriadamente sem causar falhas catastróficas. O sistema deve continuar operando mesmo quando alguns nós falham.

9. **Logging Configurável** O sistema deve fornecer logging configurável, permitindo habilitar/desabilitar logs e configurar níveis (DEBUG, INFO, WARNING, ERROR).

10. **Validação de Configuração** O sistema deve validar configurações fornecidas pelo usuário, detectando parâmetros inválidos (como min_timeout $\geq$ max_timeout) e fornecendo mensagens de erro descritivas. Validação deve ocorrer antes da inicialização do sistema.

4 Descrição da Arquitetura do Software

O *Raft Consensus Plugin* foi projetado seguindo princípios de *modularidade*, *baixo acoplamento* e *alta coesão*. A arquitetura é organizada em camadas bem definidas, permitindo que cada componente tenha uma responsabilidade única e clara. O sistema utiliza dois padrões de design principais: *Facade*, para simplificar a interface pública e *Builder*, para configuração flexível.

A arquitetura do sistema pode ser visualizada através do diagrama de componentes apresentado na Figura 1, que ilustra as principais camadas e suas interações. Os componentes da arquitetura são:

- *RaftConsensus:* Public facade for consensus operations.
- *RaftNode:* Core Raft algorithm implementation.
- *GradySim:* Simulation framework integration.
- *RaftConfig:* Configuration and settings management.
- *HeartbeatDetector:* Internal failure detection (Fault-Tolerant mode only).

Para mais detalhes sobre a arquitetura, consulte a documentação disponível online no repositório do projeto. Mais especificamente https://project-gradys.github.io/gradys-sim-nextgen/Guides/3_raft_consensus/#architecture-overview.

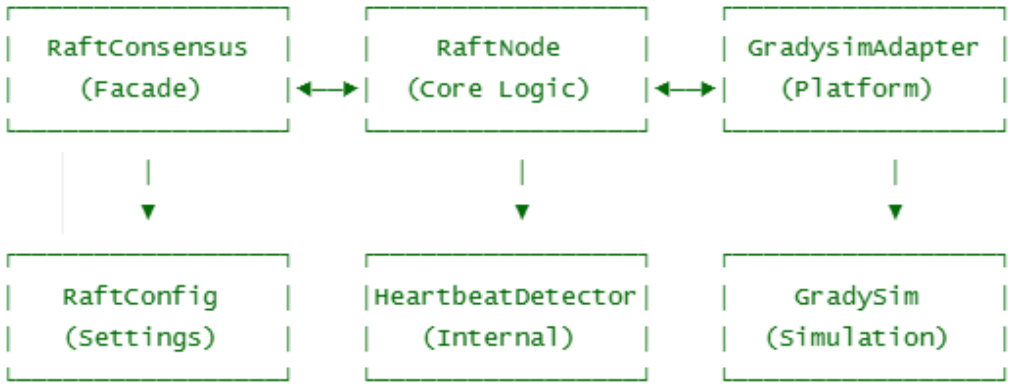

Figura 1: Visão geral da arquitetura do sistema.

5 Descrição ou Modelo Funcional do Software

Essa informação está detalhada em diferentes seções do manual do *GrADyS-SIM NextGen*. Seguem as seções mais relevantes, com descrições conceituais, exemplos práticos e detalhes da API.

Guia do Plugin

Quick Start
https://project-gradys.github.io/gradys-sim-nextgen/Guides/3_raft_consensus/#quick-start

Advanced Usage with Failure Detection
https://project-gradys.github.io/gradys-sim-nextgen/Guides/3_raft_consensus/#advanced-usage-with-failure-detection

Dispatcher Integration
https://project-gradys.github.io/gradys-sim-nextgen/Guides/3_raft_consensus/#dispatcher-integration

Configuring the Plugin
https://project-gradys.github.io/gradys-sim-nextgen/Guides/3_raft_consensus/#configuring-the-plugin

Working with the Consensus Instance
https://project-gradys.github.io/gradys-sim-nextgen/Guides/3_raft_consensus/#working-with-the-consensus-instance

Running the Showcase
https://project-gradys.github.io/gradys-sim-nextgen/Guides/3_raft_consensus/#running-the-showcase

Referência da API (Aspectos Operacionais)

Além da documentação conceitual apresentada no Guia do Plugin, a descrição funcional do sistema é complementada pela referência da API, que detalha as classes, métodos e parâmetros responsáveis pela implementação das funcionalidades descritas. As principais interfaces estão documentadas nos seguintes links:

Pacote Raft:
https://project-gradys.github.io/gradys-sim-nextgen/Modules/Protocol/plugins/raft/#raft

Failure Config:
https://project-gradys.github.io/gradys-sim-nextgen/Modules/Protocol/plugins/raft/#failureconfig

Raft Config:
https://project-gradys.github.io/gradys-sim-nextgen/Modules/Protocol/plugins/raft/#raftconfig

Raft Consensus Plugin:
https://project-gradys.github.io/gradys-sim-nextgen/Modules/Protocol/plugins/raft/#raftconsensusplugin

Raft Mode:
https://project-gradys.github.io/gradys-sim-nextgen/Modules/Protocol/plugins/raft/#raftmode

Raft State:
https://project-gradys.github.io/gradys-sim-nextgen/Modules/Protocol/plugins/raft/#raftstate

6 Código e Testes

O projeto *"Raft Consensus Plugin"* está integrado ao framework *"GrADyS-SIM NextGen"*, composto tanto pelo código-fonte quanto pela documentação técnica do sistema. Desenvolvido em Python 3.11+, o módulo segue as boas práticas de programação adotadas no *GrADyS-SIM NextGen*, e seu código, testes automatizados, showcase e documentação oficial encontram-se incorporados ao projeto principal, conforme os links a seguir.

- **Código e Estrutura do Projeto:**
  O código-fonte está disponível publicamente no repositório oficial do *GrADyS-SIM NextGen*:
  https://github.com/Project-GrADyS/gradys-sim-nextgen/tree/main/gradysim/protocol/plugin/raft

  A estrutura de diretórios do projeto organiza os principais componentes da seguinte forma:

`..raft/raft_consensus.py`	– camada de fachada do consenso;
`..raft/raft_node.py`	– lógica central do algoritmo;
`..raft/raft_config.py`	– parâmetros e inicialização;
`..raft/heartbeat_detector.py`	– detecção de falhas;
`..raft/utils/`	– utilitários e funções auxiliares.

- **Testes Automatizados:**
  https://github.com/Project-GrADyS/gradys-sim-nextgen/blob/main/tests/test_raft_plugin.py
- **Demonstração:**
  https://github.com/Project-GrADyS/gradys-sim-nextgen/tree/main/showcases/raft

7 Manual e Referências

O manual completo do *Raft Consensus Plugin* faz parte da documentação oficial do projeto *GrADyS-SIM NextGen* e está disponível no site institucional, incluindo guias práticos, instruções de configuração, referência da API e exemplos de uso integrados ao simulador. O acesso pode ser feito por meio dos seguintes links principais.

- **Plugin Guide:** https://project-gradys.github.io/gradys-sim-nextgen/Guides/3_raft_consensus/
- **API Reference:** https://project-gradys.github.io/gradys-sim-nextgen/Modules/Protocol/plugins/raft/

8 Casos de Uso - Positivo e Negativo

Cenário Positivo 1: Recuperação Automática de Falhas Massivas

Objetivo: Coordenar um enxame de 40 drones que precisam concordar sobre uma variável crítica (ex: "sequência").
Descrição: Um pesquisador configura o plugin Raft no *GradySim* usando o modo **Fault-Tolerant**. Após a eleição inicial, o Líder começa a propor valores (‘propose_value‘). Subitamente, devido a uma falha simulada, 80% dos nós (32 de 40), incluindo o Líder original, caem. O Raft Consensus Plugin entra em fase de Descoberta Ativa de Nós e identifica que apenas 8 nós permanecem ativos. O cálculo dinâmico de maioria é reajustado (de 21 para 5), e um novo Líder é eleito rapidamente entre os 8 nós sobreviventes, permitindo que o consenso sobre o valor "sequência"seja retomado sem intervenção manual.

Cenário Positivo 2: Otimização em Cluster Estático

Objetivo: Utilizar o consenso em um ambiente estável e totalmente conectado, priorizando o desempenho e o comportamento padrão Raft.
Descrição: Um desenvolvedor está validando um protocolo em um ambiente de simulação onde os 10 nós são estáticos e nunca falham. Para reduzir a sobrecarga e garantir o comportamento padrão (compatível com Raft clássico), ele configura o plugin para o modo **Classic** (‘RaftMode.CLASSIC‘). Como o modo Classic desabilita a fase de Descoberta Ativa de Nós e o cálculo de maioria é fixo (6 de 10), as eleições ocorrem diretamente após o *timeout*, resultando em uma latência de eleição e comunicação ligeiramente menor (melhor *Performance*) do que no modo Tolerante a Falhas.

Cenário Negativo 1: Uso Inesperado (Não Replicação de Logs)

Objetivo: Expor a limitação conhecida do RaftFT de não ser um sistema de Log Replication tradicional.
Descrição: Um estudante de pós-graduação, acostumado com implementações Raft que replicam logs de comandos, utiliza o Raft Consensus Plugin para registrar uma **sequência histórica de 100 eventos** em vez de apenas o estado atual. Ele chama ‘propose_value("log_entry", "evento": X)‘ 100 vezes. O sistema alcança consenso, mas quando ele tenta recuperar o histórico de eventos, descobre que ‘get_committed_value("log_entry")‘ retorna apenas o 100º (último) valor confirmado. O plugin **não manteve um log replicado** dos 99 eventos anteriores, pois o RaftFT foca em **variáveis de consenso** (o valor final) e não na replicação de logs.

Cenário Negativo 2: Problema de Pesquisa em Alcance Limitado

Objetivo: Ilustrar o problema de pesquisa aberto na documentação do Raft Consensus Plugin, que ocorre em ambientes onde o requisito de conectividade global não é atendido.
Descrição: Um pesquisador testa o modo **Fault-Tolerant** em uma simulação onde os nós estão distribuídos em uma grande área, e o alcance de comunicação é limitado, criando múltiplos **clusters desconectados**. O sistema inicia, mas o processo de Descoberta Ativa de Nós falha. **Nós de borda** (aqueles alcançáveis por mais de um cluster) recebem solicitações de eleição conflitantes, levando a cálculos incorretos de maioria dinâmica. O sistema entra em um ciclo vicioso de eleições repetidas, sem conseguir eleger um líder estável (‘No Leader Elected‘) ou confirmar valores (‘Values Not Committing‘), ilustrando o **problema conhecido**. As **soluções sugeridas** são aumentar o alcance, usar o modo Classic ou reprojetar a topologia de rede.