Uma proposta de monitoramento hierárquico e em anel utilizando *heartbeat* para a biblioteca DeLIA

Cleverson Pereira da Silva[1], Gustavo Teixeira dos Santos[1], João Victor Silva Mota[1], Calebe de Paula Bianchini[12]

[1]Faculdade de Computação e Informática (FCI)
Universidade Presbiteriana Mackenzie – São Paulo, SP – Brasil

[2]Programa de Pós-Graduação em Computação Aplicada - PPGCA
Universidade Presbiteriana Mackenzie – São Paulo, SP – Brasil

{10391119, 10391083, 10416614}@mackenzista.com.br, calebe.bianchini@mackenzie.br

Abstract. *High-Performance Computing Systems are essential for scientific and industrial applications that require high processing capacity and high availability. To ensure application continuity even in adverse scenarios, the* ***DeLIA*** *library provides fault-tolerance mechanisms through techniques such as heartbeat and checkpoint/restart. However, its current centralized monitoring architecture introduces a single point of failure, thereby undermining the system's resilience. This paper proposes a new distributed architecture for* ***DeLIA****, structured in a ring topology with support for super-peers. The proposal distributes monitoring responsibilities among the nodes, eliminating the reliance on a single process. The adopted methodology includes system redesign, restructuring communication via heartbeat, and the definition of criteria for super-peer promotion.*

Resumo. *Sistemas de Computação de Alto Desempenho são essenciais para aplicações científicas e industriais que demandam elevada capacidade de processamento e alta disponibilidade. Para garantir a continuidade das aplicações mesmo em cenários adversos, a biblioteca* ***DeLIA*** *fornece mecanismos de tolerância a falhas por meio de técnicas como heartbeat e checkpoint/restart. No entanto, sua atual arquitetura de monitoramento centralizado introduz um ponto único de falha, comprometendo a resiliência do sistema. Este trabalho propõe uma nova arquitetura distribuída para a* ***DeLIA****, estruturada em uma topologia em anel com suporte a super-pares. A proposta distribui as responsabilidades de monitoramento entre os nós, eliminando a dependência de um único processo. A metodologia adotada inclui o redesenho do sistema, a reestruturação da comunicação via heartbeat e a definição de critérios para a promoção de super-pares.*

1. Introdução

Os sistemas de Computação de Alto Desempenho (do inglês HPC - *High-Performance Computing*) tornaram-se essenciais para o avanço de diversas áreas científicas e para o processamento de grandes volumes de dados, uma vez que oferecem potência e capacidade de processamento significativamente superior em comparação com servidores e computadores convencionais. No entanto, devido à alta criticidade dessas

aplicações, é imprescindível que os sistemas HPC apresentem alta disponibilidade e mecanismos robustos de tolerância a falhas, garantindo a continuidade das operações mesmo sob condições adversas [Netto et al. 2018].

Um dos maiores desafios enfrentados por esses sistemas é justamente assegurar a resiliência a falhas, ou seja, a capacidade de manter o funcionamento adequado mesmo diante de problemas inesperados, sejam eles decorrentes de falhas de *hardware*, *software* ou conectividade [Chetan et al. 2005]. Para mitigar esses riscos, são implementadas arquiteturas resilientes e técnicas avançadas de recuperação, como *checkpoint/restart* e replicação de dados, que asseguram que o desempenho e a confiabilidade do sistema permaneçam inalterados, mesmo quando ocorrem falhas. Esses mecanismos são cruciais para manter a integridade das operações e minimizar o impacto das interrupções em ambientes críticos.

Outro mecanismo fundamental empregado para garantir a tolerância a falhas em sistemas de Computação de Alto Desempenho é o *heartbeat* [Prakash et al. 2019]. Essa técnica envolve a troca constante de sinais entre componentes do sistema para monitorar sua disponibilidade e funcionamento correto. Caso um nó ou servidor falhe em responder a esses sinais em um intervalo de tempo definido, o sistema identifica automaticamente a falha e pode tomar medidas corretivas, como redirecionar tarefas para outros nós operacionais ou ativar procedimentos de recuperação. O uso de *heartbeat* é crucial para detectar rapidamente problemas e mitigar o impacto de falhas, garantindo que o sistema mantenha sua operação contínua e minimizando a interrupção de processos críticos. Essa abordagem complementa outras estratégias de resiliência, contribuindo para a robustez geral dos ambientes HPC.

Um exemplo prático de aplicação de múltiplas técnicas de tolerância a falhas, como *heartbeat*, *checkpoint*, e estratégias de recuperação, é o produto **DeLIA** [Santana et al. 2024]. Desenvolvido para ambientes de Computação de Alto Desempenho que executam aplicações em MPI, a **DeLIA** utiliza sinais *heartbeat* para monitorar continuamente o estado dos nós e detectar rapidamente quaisquer anomalias ou falhas nas aplicações. Além disso, o sistema implementa mecanismos de *checkpoint* para salvar periodicamente o estado das aplicações em execução, permitindo uma recuperação eficiente em caso de interrupções inesperadas.

O problema identificado na versão atual da **DeLIA** é a centralização do monitoramento baseado em *heartbeat*, no qual um único nó é responsável pela supervisão do sistema. Caso o processo responsável por essa tarefa enfrente falhas, o funcionamento do sistema como um todo é comprometido. Para superar essa limitação, este trabalho tem como objetivo propor e desenvolver uma arquitetura hierárquica de monitoramento estruturada em um modelo de anel. Nessa abordagem, um subconjunto de máquinas dentro do ecossistema **DeLIA** assumirá responsabilidades de monitoramento mútuo, utilizando o mecanismo de *heartbeat*. Esse modelo aprimorado oferece maior tolerância a falhas, assegurando que, mesmo na ocorrência de interrupções em máquinas supervisoras, as outras máquinas do anel serão responsáveis por ativar o sistema de recuperação da **DeLIA**, garantindo que as operações sejam retomadas sem interrupções significativas, preservando sua resiliência e continuidade operacional.

Este artigo condensa as principais contribuições desenvolvidas no trabalho [Silva et al. 2025a], o qual também foi resumido em [Silva et al. 2025b].

2. Biblioteca DeLIA

A **DeLIA** (*Dependability Library for Iterative Applications*) é uma biblioteca projetada para oferecer tolerância a falhas em aplicações paralelas iterativas que utilizam computação de alto desempenho. Ele permite a detecção de interrupções e a recuperação do estado global e local dos processos por meio de técnicas como *checkpoint/restart* e *rollback*. Ela foi desenvolvida para atender às necessidades de programas que sincronizam dados entre processos a cada iteração. Ela também combina flexibilidade e eficiência, permitindo configurar parâmetros como frequência de *checkpoints* e seleção de dados a serem salvos. A biblioteca também é capaz de lidar com cenários de falhas abruptas (*failstop*) e interrupções preemptivas, garantindo a continuidade das operações com impacto mínimo na performance [Santana et al. 2024].

Assim, o principal objetivo da **DeLIA** é abordar falhas do tipo *fail-stop*, caracterizadas por interrupções na execução de aplicações devido a falhas de *hardware* ou a problemas específicos de *software*, conforme discutido por [Herault and Robert 2015]. Essas falhas são particularmente críticas, pois resultam na interrupção abrupta da execução do sistema, comprometendo a integridade e a continuidade do processamento. Por meio da oferta de mecanismos de tolerância a falhas, a **DeLIA** proporciona maior resiliência às aplicações baseadas no modelo BSP (*Bulk Synchronous Parallel*). Esses mecanismos permitem a detecção precoce de falhas e a recuperação eficiente, garantindo que a continuidade das operações seja preservada e que o estado global ou local da aplicação seja mantido, minimizando perdas no processo.

Nesse raciocínio, a **DeLIA** considera o tratamento tanto de falhas de *software* quanto *hardware*, as quais podem resultar na interrupção ou parada de uma aplicação em execução. Assim, para mitigar o problema de reinício de execução de uma aplicação que sofreu algum tipo de falha, técnicas de *checkpoint/restart* são amplamente utilizadas como mecanismo de tolerância a falhas em aplicações paralelas e distribuídas, permitindo que o processo seja reiniciado a partir de um estado previamente salvo.

O *checkpoint/restart* do **DeLIA** é composto por três componentes principais: *checkpoint*, detecção de falhas e recuperação/reinício. O *checkpoint* é uma captura do estado completo de um processo em um ponto específico, permitindo que o processo seja reiniciado a partir desse ponto em caso de uma falha subsequente. A detecção de falha utiliza a técnica de *heartbeat*. Por fim, a recuperação/reinício, como resultado direto do método *checkpoint/restart*, adota um modelo *fail-stop*, onde um processo falho pode ser reiniciado a partir dos dados salvos em um *checkpoint*. Além disso, o *checkpoint/restart* oferece várias vantagens cruciais para a proteção contra falhas em sistemas paralelos: (i) possibilita que problemas computacionais que podem levar dias para serem executados em sistemas de HPC sejam salvos em *checkpoints* e reiniciados em caso de falhas, evitando a perda de progresso; (ii) permite o balanceamento de carga e a migração de aplicações para outros sistemas, permitindo que a computação seja retomada mesmo quando um nó de execução falha; e (iii) apresenta um custo de implementação mais baixo e menor consumo de energia elétrica em comparação com

a redundância de *hardware* [Egwutuoha et al. 2013].

Para a técnica de *heartbeat*, a **DeLIA** utiliza a estratégia de enviar mensagens – conhecidas como "*estou vivo*" – para um dispositivo central, confirmando que os processos ativos estão funcionando corretamente [Prakash et al. 2019]. Esse envio ocorre em intervalos fixos e serve como um indicador de operação saudável do nó. Caso uma mensagem de *heartbeat* chegue atrasada ou não seja recebida dentro do tempo esperado, o nó remetente pode ser considerado como potencialmente inativo.

3. Metodologia

A estrutura de anel em sistemas distribuídos é uma topologia em que os nós são organizados de forma que cada nó tenha exatamente dois vizinhos: um anterior e um posterior, formando um anel fechado. Nessa configuração, as mensagens e os dados podem ser encaminhados de um nó para o próximo em uma direção definida (ou, em alguns casos, em ambas as direções), garantindo que as informações circulem por todo o sistema. Essa estrutura oferece simplicidade na comunicação e evita a sobrecarga de conexão centralizada [van Steen and Tanenbaum 2024] [Bosilca et al. 2017] [Di Francia Rosso and Francesquini 2022].

Tendo em vista os princípios dessa topologia, houve uma adequação da biblioteca **DeLIA**, que inicialmente opera com um modelo de um servidor central de monitoramento de *heartbeat* utilizando *sockets*, para uma estrutura baseada em uma rede de super-pares. A proposta, apresentada neste artigo, descreve a alteração do modelo centralizado por um modelo distribuído, promovendo alguns nós do ambiente a super-pares, como pode ser observado na Figura 1.

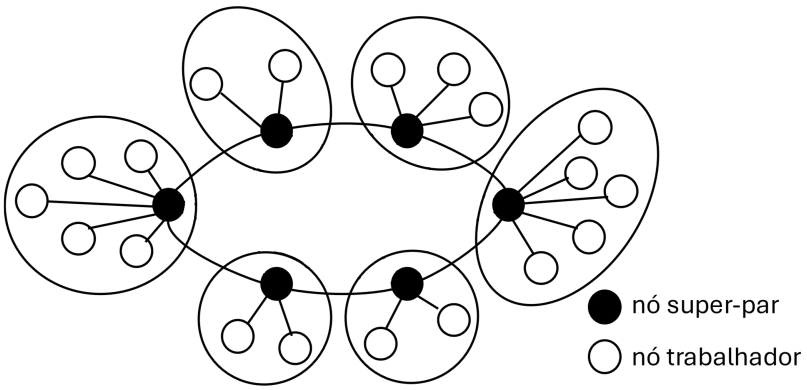

Figura 1. Proposta de topologia hierárquica e anel para o monitoramento *heartbeat*, adaptado de [van Steen and Tanenbaum 2024][Bosilca et al. 2017]

Inicialmente, definiu-se a seleção dos processos que são promovidos a super-pares de forma dinâmica com base em uma função logarítmica, conforme Equação (1), onde x representa o número total de nós. Essa estratégia permite promover, de maneira dinâmica, um número de nós ao papel de super-pares.

$$f(x) = \log_2(x) \tag{1}$$

A Listagem 1 apresenta um trecho do código[1] desenvolvido onde está a escolha

[1]Maiores detalhes do código estão em `https://gitlab.com/calebepb/delia`

dos pares e dos super-pares, bem como do grupo hierárquico, o qual é monitorado por um super-par – conforme ilustrado na Figura 1.

Na versão original da **DeLIA**, apenas o processo de *rank* 0 era designado como servidor, enquanto todos os demais operavam exclusivamente como clientes, sendo monitorados por esse único nó, conforme explica [Santana et al. 2024].

```
1     // ...
2     double num_servers = (double)(floor(log2(comm_sz)) < 1 ? 1 : (
          double)floor(log2(comm_sz)));
3     fUtil->setNumServers(num_servers);
4     // ...
5     if (id < num_servers) { // super-par
6         // ...
7         // Incluindo o servidor "a direita" como um dos clientes
8         clients[contador] = (id+1) % (int)num_servers;
9         // ...
10    } else { // par "normal"
11        // ...
12    }
13    // ...
```

Listagem 1. Função `DeLIA_DefineServers` Algoritimo para eleição dos processos super-pares na DeLIA

Nesta nova configuração, todo processo cujo *rank* satisfaz $r \leq \lfloor \log_2 x \rfloor$, em que x é o número total de processos, também é promovido a super-par. Os processos promovidos passam a utilizar duas *threads* distintas: (i) uma dedicada à execução do monitoramento dos pares em um grupo, e (ii) outra dedicada à execução de tratamento do monitoramento por um super-par vizinho, garantindo também a topologia em anel entre super-pares.

Originalmente no **DeLIA**, o nó *servidor* seguia o padrão de *bind* de um *socket* na porta padrão 8080, e com só um processo tendo o comportamento centralizado de monitoramento, enquanto os demais processos apenas se comunicavam com aquele processo centralizado, ficando assim com uma porta efêmera [Kerrisk 2010].

Na versão proposta neste trabalho, os nós super-pares executam *bind* em portas calculadas dinamicamente pela soma da porta base 8080 com o identificador de *rank* r, conforme apresenta a Equação (2).

$$P_{\text{bind}}^{\text{super-par}}(r) \;=\; 8080 + r, \tag{2}$$

Por exemplo, o super-par de *rank* 1 utiliza a porta 8081, o de *rank* 2 utiliza a porta 8082 e assim por diante. Essa estratégia assegura que cada super-par utiliza uma porta exclusiva, eliminando conflitos de endereçamento e favorecendo a escalabilidade da aplicação.

O *bind* em uma porta para o tratamento das mensagens de monitoramento é definido pela Equação (3), em que $N_{\text{super-pares}}$ corresponde ao número de super-pares. O objetivo é garantir que todos os processos recebam os *broadcasts* dos super-pares em uma porta fixa; para isso, utilizam-se as opções `SO_REUSEADDR` e `SO_REUSEPORT`, que permitem a vários processos compartilhar o mesmo par IP/porta [Kerrisk 2010].

$$P_{\text{bind}}^{\text{par}} = 8080 + N_{\text{super-pares}} + 1 \quad (3)$$

Considere um exemplo de quatro processos com *ranks* de 0 a 3, sendo os *rank* 0 e 1 promovidos para super-pares. Aplicando a Equação (2), esses super-pares associam seus *sockets* às portas 8080 e 8081, respectivamente. Como $N_{\text{super-pares}} = 2$, a Equação (3) resulta em $P_{\text{bind}}^{\text{par}} = 8080 + 2 + 1 = 8083$; assim, quando os processos atuarem como clientes, o `bind()` será feito na porta 8083 com as opções `SO_REUSEADDR` e `SO_REUSEPORT`.

4. Resultados e discussão

Após o desenvolvimento da solução proposta, foram realizados experimentos e as saídas foram instrumentadas para que fosse observado o comportamento do sistema. Foram utilizados também os testes disponíveis na biblioteca **DeLIA** para garantir que seu comportamento ainda continuava estável – aplicação de testes regressivos. Além disso, foi avaliado se o funcionamento do mecanismo de promoção se comportava como esperado. As instrumentações e as mensagens também foram capturadas pelo CI (*Continuous Integration*) da ferramenta do repositório.

O cenário básico de teste preexistente da biblioteca na **DeLIA** utiliza uma aplicação com quatro processos (*ranks* 0, 1, 2 e 3). Os *ranks* 0 e 1 foram promovidos a super-par, enquanto os *ranks* 2 e 3 permaneceram somente como par. Observou-se ainda que os super-pares se monitoravam mutuamente, formando o anel proposto: o processo 0 atua como cliente para o super-par 1, e o processo 1 desempenha o papel de cliente do super-par 0. Essa configuração apresenta, de forma concisa, o funcionamento correto da arquitetura hierárquica em anel.

A validação do mecanismo de detecção de falha em nós na rede de super-pares também utiliza esse mesmo cenário de teste preexistente. Este nó de *rank* 1 é submetido a um evento de falha (usando o comando `KILL`). Conforme observado, o super-par de *rank* 0 detecta a ausência de respostas de *heartbeat* provenientes do nó 1 monitorado. Em seguida, ele emite um sinal de *trigger* por *broadcast* para a porta 8083 – conforme a Equação (3). Os demais nós, por sua vez, receberam a mensagem de *trigger*.

A Figura 2 apresenta três testes com um total de 10 processos para avaliar a detecção de falhas via *heartbeat*. No primeiro, simulou-se a falha do nó 1 para verificar se os demais identificavam sua ausência. No segundo, todos os nós foram encerrados simultaneamente, testando a reação do sistema diante da perda total de comunicação. No terceiro, o nó 0 recebeu um sinal `SIGTERM`, avaliando se ele conseguia informar sua saída aos demais antes de encerrar. Esses testes permitiram analisar a robustez e a propagação de falhas em um ambiente distribuído.

5. Conclusão

Este trabalho tratou o problema do ponto único de falha no módulo de monitoramento da **DeLIA**, cuja arquitetura centralizada compromete a resiliência de uma aplicação. O objetivo proposto foi alcançado, projetando e desenvolvendo uma topologia hierárquica e em anel, com promoção dinâmica de super-pares, conforme

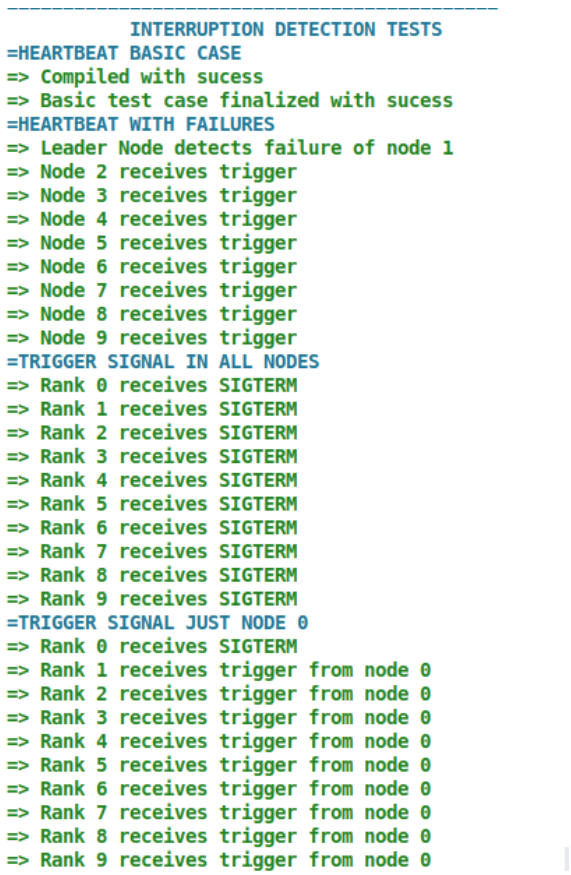

Figura 2. Realizando teste com 10 processos

descreve a Equação (1), de modo a distribuir a responsabilidade de monitoramento e a notificação de falhas.

Os resultados indicam que o risco de indisponibilidade foi reduzido pela eliminação do nó supervisor central. Ademais, o tráfego de *heartbeat* e o pico de uso de CPU deixaram de se concentrar em um único processo, passando a distribuir-se entre os super-pares, sem incremento significativo de *overhead* global.

Por fim, alguns trabalhos futuros podem ser enumerados:

- **Validação em ambientes maiores**: experimentar essa nova arquitetura em ambientes de *clusters* maiores;
- **Estratégias avançadas de monitoramento de super-pares**: comparar o mecanismo atual de topologia em anel com abordagens baseadas em *gossip*, investigando latência de detecção, *overhead* de tráfego e tolerância a partições de rede.

Agradecimentos

Os autores agradecem o apoio da MackCloud[2]; e do projeto SPRACE – Processo nº 2018/25225-9, Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP).

[2]Laboratório Multidisciplinar de Computação Científica e Nuvem, em `https://mackcloud.mackenzie.br`

Este trabalho foi financiado em parte pelo Fundo Mackenzie de Pesquisa e Inovação (MackPesquisa) – Projetos nº 231009 e 251005.

Referências

Bosilca, G., Bouteiller, A., Guermouche, A., Hérault, T., Robert, Y., Sens, P., and Dongarra, J. (2017). A Failure Detector for HPC Platforms. Research Report RR-9024, INRIA.

Chetan, S., Ranganathan, A., and Campbell, R. (2005). Towards fault tolerance pervasive computing. *IEEE Technology and Society Magazine*, 24(1):38–44.

Di Francia Rosso, P. H. and Francesquini, E. (2022). Ocftl: An mpi implementation-independent fault tolerance library for task-based applications. In Gitler, I., Barrios Hernández, C. J., and Meneses, E., editors, *High Performance Computing*, pages 131–147, Cham. Springer International Publishing.

Egwutuoha, I. P., Levy, D., Selic, B., and Chen, S. (2013). A survey of fault tolerance mechanisms and checkpoint/restart implementations for high performance computing systems. *The Journal of Supercomputing*, 65:1302–1326.

Herault, T. and Robert, Y. (2015). *Fault-Tolerance Techniques for High-Performance Computing*. Springer Publishing Company, Incorporated, 1st edition.

Kerrisk, M. (2010). *The Linux programming interface: a Linux and UNIX system programming handbook*. No Starch Press.

Netto, M. A. S., Calheiros, R. N., Rodrigues, E. R., Cunha, R. L. F., and Buyya, R. (2018). Hpc cloud for scientific and business applications: Taxonomy, vision, and research challenges. *ACM Computing Surveys*, 51(1):1–29.

Prakash, S., Vyas, V., and Bhola, A. (2019). Proactive fault tolerance using heartbeat strategy for fault detection. *no*, 1:4927–4932.

Santana, C., Araújo, R. C., Sardina, I. M., Ítalo A.S. Assis, Barros, T., Bianchini, C. P., de S. Oliveira, A. D., de Araújo, J. M., Chauris, H., Tadonki, C., and de Souza, S. X. (2024). Delia: A dependability library for iterative applications applied to parallel geophysical problems. *Computers & Geosciences*, 191:105662.

Silva, C., Santos, G., Mota, J., and Bianchini, C. (2025a). Uma proposta de monitoramento hierárquico utilizando heartbeat em sistemas de computação de alto desempenho. In *Anais da XVI Escola Regional de Alto Desempenho de São Paulo*, pages 13–16, Porto Alegre, RS, Brasil. SBC.

Silva, C. P. d., Santos, G. T. d., and Mota, J. V. S. (2025b). Uma proposta de monitoramento hierárquico utilizando heartbeat em sistemas de computação de alto desempenho.

van Steen, M. and Tanenbaum, A. S. (2024). *Distributed Systems*. Maarten van Steen, 4th edition.