# Gerenciamento Hierárquico de Falhas para Aplicações MPI

**Lucas Baptista de Moraes**[1], **Fernanda Gonçalves de Oliveira Passos**[1]

[1]Escola de Engenharia – Universidade Federal Fluminense (UFF)
Niterói – RJ – Brasil

{lucasbaptista,fernanda}@midiacom.uff.br

***Abstract.*** *One of the biggest challenges in large scale scenarios are the failures, since the Message Passing Interface (MPI) doesn't have, originally, support to fault tolerance. Therefore, libraries with such support, as ULFM, become necessary. In this paper, an application management system, called AMS Easygrid, was taken as a reference in order to implement its communicator hierarchy using updated mechanisms of detection and repair of failures. Preliminary results show that the proposed fault treatment is effective with low overheads.*

***Resumo.*** *Um grande desafio para cenários de larga escala são as falhas, uma vez que a biblioteca de troca de mensagens MPI não possui, nativamente, um suporte de tolerância a falhas. Por isso, é necessário o uso de bibliotecas com tal suporte, como a ULFM. Neste trabalho, é proposto um protótipo simplificado de um sistema gerenciador de aplicações MPI, chamado EasyGrid SGA, com uma nova implementação do mecanismo de sua tolerância a falhas. Resultados preliminares mostram sua eficácia na detecção e reparo de falhas e suas baixas sobrecargas no tempo de execução da aplicação.*

## 1. Introdução

O MPI — *Message Passage Interface* — é um dos padrões de programação paralela mais popular do mundo e é, de fato, a biblioteca de troca de mensagens mais utilizada em aplicações científicas e industriais, permitindo sua execução eficiente em infraestruturas de *clusters*, grades e nuvens de computadores [Passos and Rebello 2016].

Gerenciar falhas de processos [Tanenbaum and Van Steen 2007], como de queda ou de *software*, em um sistema distribuído é um assunto amplamente discutido na comunidade científica [Ivaki et al. 2018], uma vez que uma falha pode comprometer toda a execução de uma ou mais aplicações no sistema. O MPI não possui nativamente tal função, porém existem implementações que estendem essa funcionalidade, como a biblioteca *User Level Failure Mitigation* (ULFM) [Bland et al. 2013], que nos apresenta soluções de detecção e recuperação de falhas para que os processos da aplicação prossigam sua execução mesmo em situações adversas.

O EasyGrid SGA [Boeres and Rebello 2004] é um sistema de gerenciamento de aplicações MPI em ambientes computacionais heterogêneos e compartilhados. O objetivo deste trabalho[1] é propor um protótipo funcional de uma versão atualizada deste SGA, para que aplicações MPI possam ser executadas com novas e mais robustas extensões do MPI para detecção e tratamento de falhas tornando o *middleware* capaz de oferecer soluções de tolerância a falhas mais eficientes.

[1]Este trabalho é apoiado pelo PIBIC-CNPq 2019-2020, UFF.

## 2. Protótipo de Tolerância a Falhas para o EasyGrid SGA

O *middleware* EasyGrid SGA realiza o gerenciamento das aplicações através de uma hierarquia de três níveis. O Gerenciador Global (GG) é único e é responsável por realizar o gerenciamento dos *sites*. O Gerenciador de *Sites* (GS) faz o gerenciamento da execução das suas respectivas máquinas. O Gerenciador de Máquina (GM) está no nível mais baixo e possui o papel de escalonar, criar e executar os processos MPI a ele destinados. O EasyGrid é também implementado em MPI e seu mecanismo de tolerância a falhas defasado, baseado numa versão funcional porém depreciada do MPI, a LAM/MPI [da Silva 2010]. Deste modo, a comunicação entre os gerenciadores do EasyGrid pode ser atualizada para uma das mais recentes implementações do OpenMPI junto a sua extensão de tolerância a falhas, a ULFM. A biblioteca ULFM permite a detecção e tratamento de falhas em intracomunicadores, tornando viável a criação da hierarquia através de grupos de processos, conforme mostra a Figura 1, o que não era possível na versão antiga do *middleware*, uma vez que os mecanismos de tolerância a falhas existentes com LAM/MPI exigiam que cada processo possuísse seu comunicador individual.

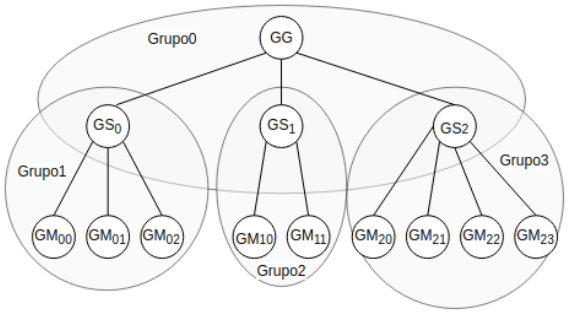


**Figura 1. Grupos de comunicação no gerenciamento de falhas.**

O algoritmo proposto, que está dividido entre os três tipos de gerenciadores, é descrito da seguinte forma: 1) GM: Recebe do seu GS as suas tarefas e as insere em uma fila. As tarefas são criadas dinamicamente e, assim que terminam, enviam uma mensagem de finalização para o GS. Caso ocorra algum erro no comunicador, o GM não faltoso receberá novas tarefas. 2) GS: Recebe a quantidade de tarefas do GG, que serão distribuídas entre suas máquinas, e recebe também do GG uma tabela que faz o mapeamento da tarefa para a máquina que será escalonada, status de finalização, entre outros atributos. Caso não ocorram falhas, o *site* enviará os identificadores das tarefas concluídas para o GG. No caso de falhas, o GS irá utilizar a função `MPIX_Comm_shrink`, que cria um novo comunicador, eliminando o processo faltoso. Além disso, o *site* envia a quantidade de tarefas não concluídas no processo morto para que o GG realize a divisão de novas tarefas entre os *sites*. 3) GG: Responsável de distribuir tarefas entre os *sites*. Além de auxiliar o gerenciador de *site* no tratamento das falhas, ele também é responsável por sinalizar a finalização do programa quando detectar o fim de todas as tarefas através dos GS.

## 3. Experimentos e Resultados

Com o intuito de avaliar o desempenho do gerenciamento hierárquico de falhas proposto, três casos de teste foram executados em uma máquina do *cluster* do laboratório MídiaCom com processadores Intel Core i7-860 e 32 GB de memória principal. Foram utilizados os seguintes gerenciadores: 1 GG, 2 GS e 4 GM (2 para cada *site*). No total, são computadas

24 tarefas da aplicação, 6 para cada GM, onde as tarefas são executadas 4 por vez. Serão abordados os seguintes casos ao longo de 50 repetições de execução realizadas: 1) tempo de computação sem falhas; 2) tempo de computação com falha de uma máquina em um site; 3) tempo de computação com falha de uma máquina em cada um dos sites.

A Tabela 1 apresenta os tempos obtidos: total (média $\pm$ intervalo de confiança de $95\%$) da execução da aplicação; e de detecção e reparo das falhas, todos em segundos. Na última linha, temos a sobrecarga que os procedimentos de detecção e reparo agregam no tempo total da execução. Em todos os testes as falhas foram detectadas e reparadas.

**Tabela 1. Tabela de tempo de execução nos três cenários**

| | Caso 1 | Caso 2 | Caso 3 |
|---|---|---|---|
| Tempo Total (s) | $9,86\ (\pm 0,12)$ | $12,11\ (\pm 0,30)$ | $15,83\ (\pm 0,42)$ |
| Tempo de Detecção e Reparo (s) | - | $0,37\ (\pm 0,15)$ | $0,41\ (\pm 0,09)$ |
| Sobrecarga de Detecção e Reparo | - | 3,05% | 2,6% |

## 4. Conclusões e Trabalhos Futuros

Este trabalho apresenta um protótipo simplificado do EasyGrid SGA com uma nova implementação do mecanismo de tolerância a falhas utilizando bibliotecas MPI atuais juntamente a ULFM. Através dela é possível analisar o desempenho de técnicas de detecção e recuperação de falhas de modo a futuramente ser incorporado ao *middleware* original.

Os resultados preliminares mostram que, em casos de falha, o programa terá um tempo total de execução maior que o caso normal, o que é esperado uma vez que essa solução trabalha com a reexecução de tarefas. No entanto, eles também mostram uma sobrecarga no tempo total de execução de até $3\%$ comparado à execução sem falhas. Este valor é aceitável uma vez que o desejável nesses cenários é de $2\%$ a $4\%$ [da Silva 2010].

São trabalhos futuros a implementação do gerenciamento de falhas de um GS e do GG, análise dos tempos em cenários de maior escala e possíveis variações de parâmetros de ajuste do ULFM para garantir acurácia do acerto das falhas.

## Referências


Bland, W. et al. (2013). Post-failure recovery of MPI communication capability: Design and rationale. *Int. J. of High Performance Computing Applications*, 27:244 – 254.

Boeres, C. and Rebello, V. E. F. (2004). Easygrid: Towards a framework for the automatic grid enabling of legacy MPI applications. *Concurrency and Computation: Practice and Experience*, 16(5):425–432.

da Silva, J. A. (2010). *Tolerância a Falhas para Aplicações Autônomas em Grades Computacionais*. PhD thesis, Universidade Federal Fluminense.

Ivaki, N., Laranjeiro, N., and Araujo, F. (2018). A survey on reliable distributed communication. *Journal of Systems and Software*, 137:713 – 732.

Passos, F. G. O. and Rebello, V. E. F. (2016). An autonomic parallel strategy for the projection of ecological niche models in heterogeneous computational environments. In *European Conference on Parallel Processing*, pages 363–375. Springer.

Tanenbaum, A. and Van Steen, M. (2007). *Sistemas distribuídos: princípios e paradigmas*. Pearson.