Escalonamento Tolerante a Falhas na Recuperação de Aplicações em MPI

Idalmis Milián Sardina
Doutorado em Computação
Universidade Federal Fluminense

Motivação (Idéia)

Problemas:

- heterogeneidade: Arquiteturas de grades, conjunto de clusters conectados por redes. Ambientes tipicamente heterogêneos.
- ocorrência de falhas em recursos.

Necessidade:

- Alcançar alto desempenho nestes ambientes heterôgeneos
- Tolerar as falhas em recursos

Como (solução):

1. Heurísticas de escalonamento
2. Estratégias de tratamento de falhas

} → alg. de escalonamento tolerante a falhas

Problemas que surgem em estudos realizados:

- o desempenho das estratégias tolerantes a falha
- o uso de ambientes simulados

2

Motivação (Idéia)

Foco principal do trabalho : investir no estudo de políticas de escalonamento com mecanismos de tolerância a falhas voltadas para aplicações paralelas implementadas em MPI (ambientes reais).

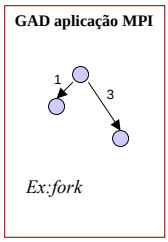
Para tal **objetiva-se**, a implementação de uma ferramenta que:

- considera as informações de escalonamento estático produzidas pela heurística, e
- oferece mecanismos de detecção automática de falhas, e recuperação eficiente da aplicação de acordo com dados produzidos pela mesma heurística.
- na fase inicial, viabiliza a estratégia tolerante a falhas em um ambiente real menor (*cluster*) e
- depois, estender a mesma a uma grade computacional, acoplando-se a uma topologia hierárquica.

3

Modelo heterogêneo do ambiente real

Aplicação paralela (GAD)

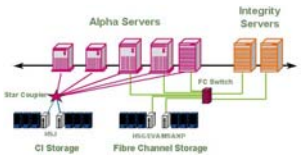


- *GAD (Grafo Acíclico Direcionado) G=(V, E, e, c)*
- *representado pelos arquivos de adjacências e de custos*
- *programa em MPI/LAM*
- *cada tarefa é um processo no MPI com Entrada, Execução e Saída*

4

Modelo heterogêneo do ambiente real

Arquitetura heterogênea (Cluster, Grid)



- *processadores heterogêneos*
- *fator de heterogeneidade (FH),*
- *latência*
- *custo de confiabilidade (RC)*
- *falha permanente de processador*
- *comunicação por troca de mensagens*

5

Estratégia de escalonamento estático tolerante a falhas

Escalonamento estático

- *list scheduling*

Tolerância a Falhas

- *replicação passiva (primaria/backup)*

Objetivos do algoritmo

- *minimizar o tempo total de execução (makespan)*
- *minimizar o custo de confiabilidade durante o processo de escalonamento considerando somente uma falha permanente de processador.*

*-> **Minimização de função multi-objetivo***

1. *makespan*
2. *custo de confiabilidade*

6

Algoritmo de escalonamento estático tolerante a falhas (3 etapas principais)

1) Ordenar tarefas usando critérios de prioridade: OrdenaTarefas();

- *ordenação crescente por deadline*
- *outros critérios de prioridade: caminho crítico (CP), blevel, tlevel e tlevel + blevel.*

2) Escalonar cópias primárias: EscalonaPrimarias();

- *list scheduling*
  - *a) obedecendo deadline.*
  - *b) melhor confiabilidade.*
  - *c) finalmente, aquele que minimiza o tempo de fim.*

3) Escalonar cópias backups: EscalonaBackups().

- *list scheduling (similar a 2) junto com a estratégia primaria/backup.*
- *escalona-se depois das primárias de acordo com certos critérios:*
  - a re-distribuição das mensagens.

7

Escalonamento estático tolerante a falhas

(procedimento para escalonar* backups*)

Procedimento *EscalonaBackups*()
saída: arq.de escalonamento (.sch)
arq. de distribuição de mensagens (.msg)

$f(v^B) := \infty$;
$rc := \infty$;
para cada tarefa $v \in L_{ord}$ **faça**
  VerificaPrimariaForte(v); /*C_1*/
  AchaCjtoProcs(v^B,F); /*C_2, C_4 e C_5*/
  para cada $p_i \in F$ **faça**
    CriaListaSemSobreposic(v, $Lnsup$); /*C_3*/
    EFT:=CalcEFT(v^B, p_i,$Lnsup$);
    AtualizaMinTempoProcRC(f,p,rc,EFT,p_i,$RC(p_i)$);
  fim para
  AlocaBackup(v^B, p)
  AtualizaMsgs(v^B) /*C_1, C_6 e C_7*/
fim para
fim.

8

Critérios para escalonamento de backups

C_1 *seja* (v_j->v_i), verifica se primária v_i é ou não cópia primária forte, se a falha ocorre antes de terminar e recebe todas as mensagens.

C_2 *backups* e primárias escalonadas em procs diferentes

C_3 sobreposição de *backups* escalonadas no mesmo proc. se primarias foram escalonadas em processadores diferentes.

C_4 *backups* e primárias escalonadas antes de seus *deadlines*. A *backup* escalonada depois da sua primária. A *backup* começa depois do terminar a primária mais o tempo de detecção da falha.

C_5 seja (v_j->v_i), verifica se v_j^B e v_i^P não podem ser escalonadas no mesmo processador

9

Critérios para redistribuição de mensagens

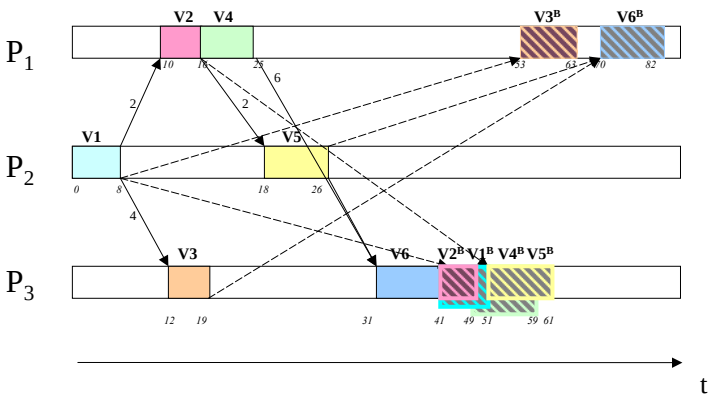
C_6 (v_j->v_i), verifica se v_i^B não precisa enviar msg para v_j^B se:

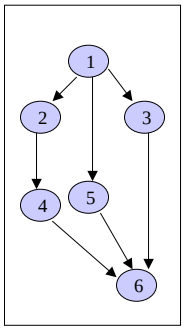
C_1 (v_j->v_i), verifica se v_i^B não precisa enviar msg para v_j^P se:

A ferramenta MPI proposta recebe como entrada a informação obtida pelo algoritmo de escalonamento tolerante a falhas

10

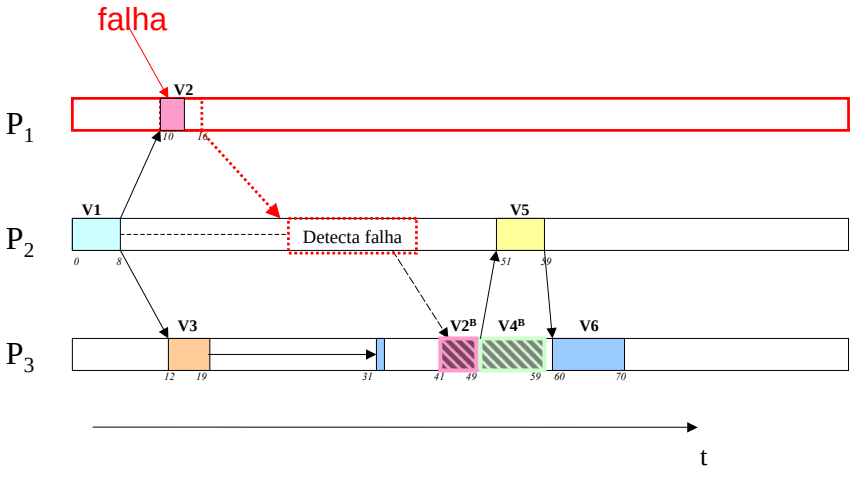
Exemplo do resultado do escalonamento (GAD de 6 tarefas)





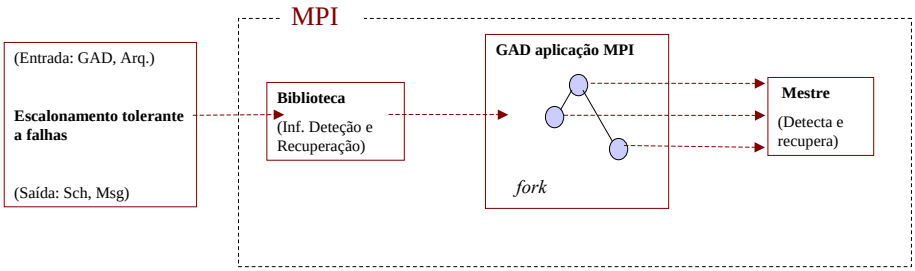
11

Exemplo de re-execução da aplicação (em caso de P1 falhar)



12

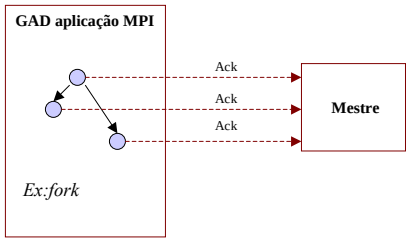
Estrutura da Ferramenta em MPI proposta (1 etapa)



2 etapas
1-testar e viabilizar a estratégia tolerante a falhas em um ambiente real menor
2-estender a uma grade computacional

13

Execução da aplicação com ferramenta e sem falha



Aplicação:

1-compilar com a biblioteca e os arquivos do mecanismo e o Mestre.

2-nenhum código é alterado

3- pronta para executar com capacidade de recupar-se em caso de falha

14

Implementação da ferramenta em MPI

Possíveis mecanismos de detecção

Internos *(do MPI)*

- *Handler para comunicação ponto a ponto (error handlers).*

Externos *(implementação externa ao MPI)*

- *Heartbeat*
- *NWS*

Híbridos

- *Combina vantagens de internos com externos*

15

Mecanismo de tratamento de falha

Detecção e Recuperação de falha

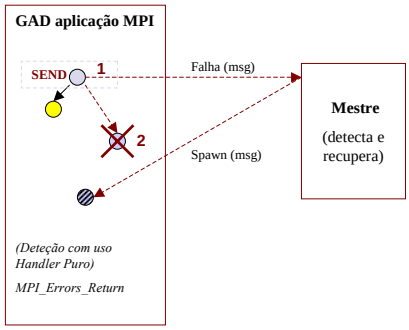
1. Mestre: monitora e processa
2. Biblioteca: redefine as funções que a aplicação MPI pode usar
(em conjunto fazem o tratamento de falhas)

- *detecção usando mecanismo interno (Handler)*
- *funções interceptam o erro na execução retornando um código de ocorrência de falha*
- *tratamento de Handlers, comunicação ao Mestre*
- *tratamento da falha*
- *criação dinâmica das backups*
- *re-encaminhamento das mensagens segundo a informação obtida no escalonamento tolerante a falhas*
- *continuidade da execução da aplicação*

16

Mecanismo de detecção proposto (1 caso)

1) o proc. 1 envia (MPI_Send) msg ao proc. 2

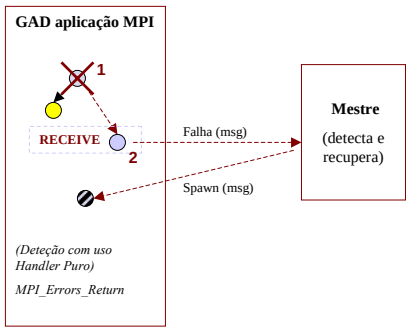


-função MPI_Send detecta se 2 falhou (biblioteca)
-comunica ao Mestre que cria backup e re-envia mensagem

17

Mecanismo de detecção proposto (2 caso)

2) o proc. 2 recebe (MPI_Recv) msg do proc. 1

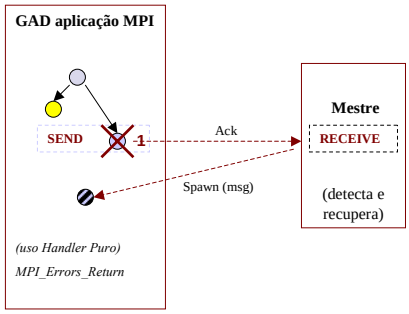


-função MPI_Recv detecta se 1 falhou (biblioteca)
-comunica ao Mestre que cria backup e re-envia mensagem

18

Mecanismo de detecção proposto (3 caso)

3) o proc. Mestre recebe (MPI_Recv) msg do proc. 1 (folha)



-função MPI_Recv detecta se 1 falhou (biblioteca)
-Mestre cria backup e re-envia mensagem

19

Ambiente de testes

***cluster de computadores do Grid Sinergia**:*
8 processadores Pentium IV 2.6 GHz com 512 Mb de RAM, executando Linux Fedora Core2, Globus toolkit 2.4 e MPI LAM 7.0.6, interconectados por uma rede Gigabit Ethernet.

***aplicação alvo :** Eliminação Gausseana (GAD)*

***uma falha permanente de processador**: script disparado enquanto a aplicação ainda executada (comando killall).*

modo exclusivo

***cenários de execução da aplicação**:*
-sem falha
-com falha

20

Análise de Desempenho

medições de:
- tempo de execução
- número de mensagens

variando:
- o número de tarefas

estuda-se o comportamento de parâmetros como:
- delay e
- overheads
(ambos introduzidos pelo mecanismo de detecção e recuperação durante a execução da aplicação em caso de falha)

21

Resultados experimentais preliminares

Tabela 1. Comparação de tempo de execução (s) e número de mensagens para a aplicação Gauss nos cenários Sem Falha e Com Falha sobre a arquitetura com 8 processadores

n **(N. de tarefas)**	**Cenários**	**Tempo de Execução**	**N. de Mensagens**
3 (5)	Sem Falha	4,26	279
	Com Falha	4,27	344
4 (9)	Sem Falha	7,52	559
	Com Falha	8,51	678
10 (54)	Sem Falha	55,68	935
	Com Falha	57,41	1124
13 (90)	Sem Falha	94,01	1407
	Com Falha	94,30	1682
17 (152)	Sem Falha	146,78	2132
	Com Falha	148,44	2537
18 (170)	Sem Falha	174,07	2132
	Com Falha	180,52	2537

22

Resultados experimentais preliminares

Tabela 2. Comparação do Delay (s), Overhead de mensagens e Overhead de tolerância a falhas para a aplicação Gauss no cenário Com Falha sobre a arquitetura com 8 processadores

n (N. de Tarefas)	Delay (s)	Overhead de Mensagens	Overhead Total
3 (5)	0,01	30,71%	0,23%
4 (9)	0,99	46,52%	13,16%
10 (54)	1,73	44,79%	3,10%
13 (90)	0,29	32,30%	0,3%
17 (152)	1,66	29,25%	1,13%
18 (170)	6,45	28,71%	3,7%

23

Discussões dos resultados

Em resultados preliminares obtidos observa-se:

- *ao ocorrer uma falha de processador, existe um custo computacional adicional referente a detecção da falha e re-encaminhamento de novas mensagens para ativar as cópias backups*
- *o tempo de execução quando ocorre falhas não são tão maiores quando considerando a execução sem falhas.*

Tópicos a serem abordados

- *fatores que influenciam no desempenho da aplicação:*
  - *número de processos criados,*
  - *número de tarefas escalonadas por processador e*
  - *número máximo de mensagens que a memória usada pelo processador pode armazenar.*

24

Conclusões e Trabalhos futuros

O objetivo foi uma ferramenta no MPI capaz recuperar falhas utilizando uma técnica de escalonamento estático tolerante a falhas. A ferramenta implementada nesta primeira etapa tem um escopo menor voltada para funcionar dentro de um cluster computadores.

.

Futuro

- *Continuar estudos analisando escalabilidade, diferentes momentos de falha, comparação com makespan, cálculo do tempo de detecção etc.*
- *Uma segunda etapa será:*
  - *grade computacional com a adaptação ao SGA do Projeto EasyGrid da UFF*
  - *incluir também a implementação de algoritmos de escalonamento dinâmico com critérios de tolerância a falhas para replicação passiva,*
  - *analisar a vantagem de utilizar outros tipos de heurísticas de escalonamento.*
  - *considerar a ocorrência de múltiplas falhas assim como de outros tipos de falhas.*

25