

Tolerância a Falhas

Computação Distribuída

- falhas em Sistemas Distribuídos
 - crash
 - omissão
 - retardo em respostas
 - respostas erradas
 - respostas arbitrárias
- simplificadaamente:
 - falhas bizantinas
 - falhas fail-stop

Tolerância a Falhas em Computação Distribuída

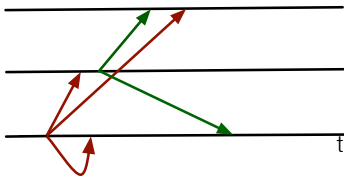
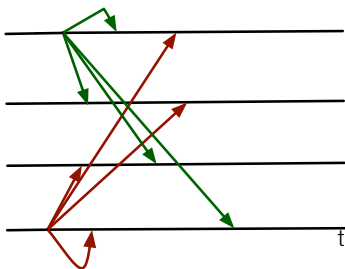
- preocupação específica com processos e processadores que interrompem execução devido a falha
 - relação com saída da máquina do pool de disponibilidade
 - saída elegante X saída sem aviso
- técnicas
 - replicação
 - checkpointing e restauração
 - relação com migração



- cópias de processos ou dados evitam perdas em caso de falhas
- replicação de processos: ativa e passiva
 - replicação ativa: simetria entre cópias
 - replicação passiva: cópia primária e cópias secundárias

- caso simples em que processos não se comunicam
- problema nesse caso são falhas maliciosas
- construção de reputação
- redundância: envio de mesmo bloco de dados para diferentes trabalhadores
- tolerância por replicação complicada em aplicações com muita comunicação

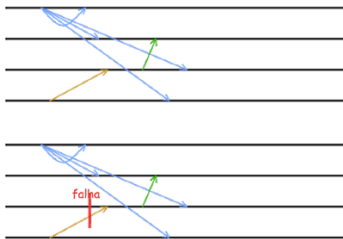
problemas de ordenação



- uso de protocolos de broadcast
- broadcasts com garantias
 - ordem total
 - ordem causal

Replicação

- estado do processo pode não ser exatamente igual ao de suas réplicas



- cópias de processos causam processamento extra...

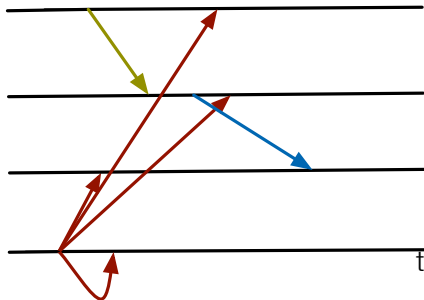
Checkpointing

- armazenamento do estado do processo de tal forma que seja possível reiniciá-lo desse ponto
- processo:
 - dados globais
 - código
 - pilha de execução ??
 - arquivos abertos ??
 - mensagens ??



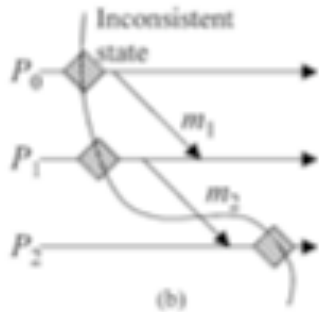
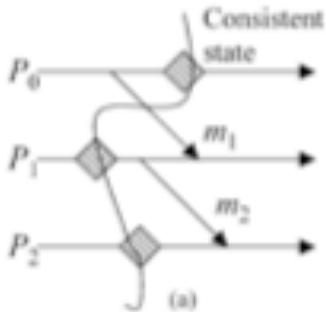
- dump de memória só é útil em situações de muita homogeneidade
- tradução para formatos de nível mais alto exige conhecimento do que está armazenado
 - relação com problema de migração

Checkpoints Distribuídos



- problema: obter conjuntos consistentes de checkpoints dos processos individuais

Cortes Consistentes e Inconsistentes



- problema de gerenciar estado global

- checkpoints globais consistentes
 - checkpoint coordenado
 - todos os processos têm que voltar ao último ponto de cp para a aplicação se recuperar de uma falha
- checkpoints individuais + logging de mensagens
 - assume que estado de processo depende apenas de msgs trocadas
 - recupera processo e reenvia todas as msgs
 - log de mensagens: gargalo de armazenamento (confiável)
mensagens pre-checkpoints viram lixo coletável


```
int function () {
    int lastFunctionCalled = -1;
    int localVar = 0;
    ckp_push_data(&lastFunctionCalled, sizeof(int));
    ckp_push_data(&localVar, sizeof(int));
    if ( ckpRecovering == 1 ) {
        ckp_get_data(&lastFunctionCalled, sizeof(int));
        ckp_get_data(&localVar, sizeof(int));
        if( lastFunctionCalled == 0 )
            goto ckp0;
    }
    // Do computations (...)
ckp0:
    lastFunctionCalled = 0;
    functionA ( ) ;
    // Do computations (...)
    ckp_npop_data(2);
    return localVar;
}
```

Original Code Modified Code

- erros podem ser tratados
 - `MPI_Comm_set_errhandler`
- por default `MPI_ERRORS_ARE_FATAL`
- tratadores definidos por comunicador
- para isolar efeitos, necessidade de criar comunicadores específicos

- implementações genéricas não têm suporte a TF
- tratamento padrão de erros: finalização de todos os processos
- propostas
 - checkpointing na aplicação
 - uso de intercomunicadores e redefinição de tratamento de erro
 - *wrappers* de funções MPI com extensões

- antecipação da falha
 - sistemas de previsão de falhas
 - temperatura
 - bateria
 - uso de memória
- migração da computação

- Kenneth Birman. *Reliable Distributed Systems*. Springer-Verlag, 2005.
- William Gropp, Ewing Lusk. Fault Tolerance in Message Passing Interface Programs. *International Journal of High Performance Computing Applications*, 18(3), 363-372, 2004.
- E. N. Elnozahy, L. Alvisi, Y. Wang, D. B. Johnson. A survey of rollback-recovery protocols in message-passing systems. *ACM Computing Surveys*, 34(3), 2002.
- Camargo, R. , Goldchleger, A., Kon, F., and Goldman, A. Checkpointing BSP parallel applications on the InteGrade. *Concurrency and Computation: Pract. Exper.* 18(6), 567-579, 2006.