Ambiente para a Visualização de Protocolos de *Checkpointing*

Projeto de Iniciação Científica

Candidato: Raphael Marcos Menderico

Orientadora: Profa. Dra. Islene Calciolari Garcia

Fevereiro de 2003

Resumo

Checkpoints são utilizados em sistemas distribuídos para auxiliar tarefas como monitorização e recuperação por retrocesso de estado. No entanto, entender o comportamento de um protocolo de *checkpointing* pode ser uma tarefa bastante complexa visto que o seu funcionamento pode depender da interação entre vários processos. O objetivo deste trabalho é a especificação e implementação de um ambiente que permita a visualização gráfica de protocolos para *checkpointing*. Acreditamos que este ambiente será útil primeiramente para auxiliar a depuração dos protocolos durante a fase de implementação dos mesmos. Além disso, o ambiente servirá como ferramenta de pesquisa, permitindo a exploração de detalhes do comportamento dos protocolos. O código-fonte desenvolvido para o ambiente será disponibilizado de acordo com a filosofia de *software* livre.

Palavras-chave: Algoritmos distribuídos, tolerância a falhas, *checkpointing*, visualização de algoritmos

1 Introdução

Soluções para vários problemas em sistemas distribuídos, em particular problemas relacionados à monitorização e recuperação por retrocesso de estado, estão baseadas na utilização de *checkpoints*. Um *checkpoint* é um estado de um processo que foi selecionado para ser gravado em memória estável. Um *checkpoint* global consistente representa uma fotografia da computação distribuída que poderia ter sido obtida por um observador onisciente externo [8]. Um protocolo de *checkpointing* coordena a seleção de *checkpoints* pelos processos de maneira a garantir a construção de *checkpoints* globais consistentes [10, 17].

Entender o comportamento de um protocolo de *checkpointing* pode ser uma tarefa complexa, visto que temos de levar em conta a interação entre vários processos. As descrições dos protocolos presentes na literatura estão sempre acompanhadas de diagramas espaço-tempo semelhantes aos ilustrados nas Figuras 1 e 2. Nestes diagramas, as linhas horizontais representam a execução dos processos com o decorrer do tempo, as setas menores representam as mensagens e os quadrados *checkpoints*.

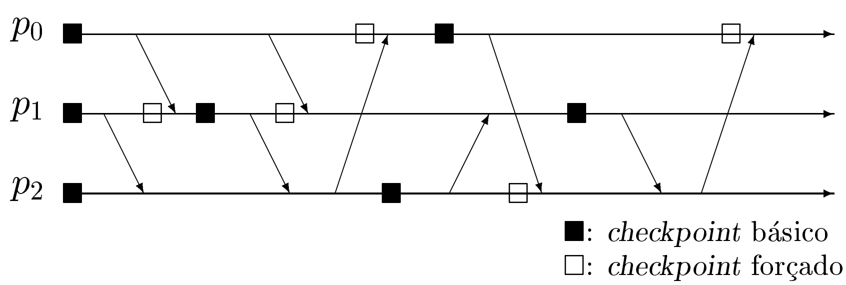

Figura 1: Diagrama espaço-tempo ilustrando um protocolo quase-síncrono

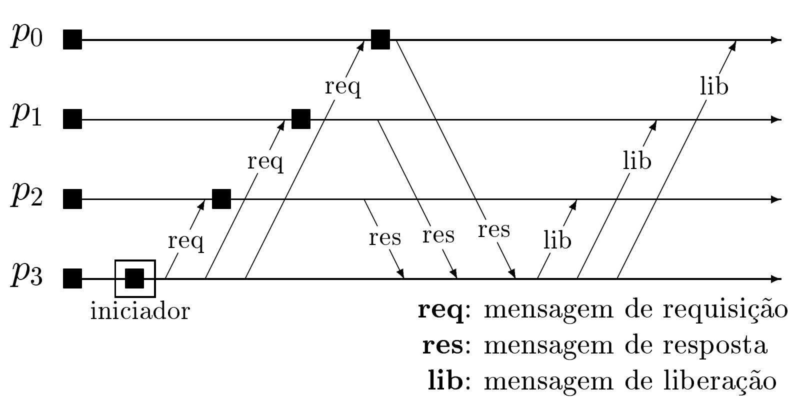

Figura 2: Diagrama espaço-tempo ilustrando um protocolo síncrono bloqueante

A Figura 1 ilustra um protocolo quase-síncrono de *checkpointing*, no qual podemos distinguir dois tipos de *checkpoints*: os que são selecionados de maneira autônoma pelos processos, denominados *checkpoints* básicos, e os que são induzidos pelo protocolo, denominados *checkpoints* forçados. No cenário exibido, os processos são induzidos a gravar um *checkpoint* forçado imediatamente antes de todo evento de recepção de mensagem [17]. Esta abordagem é muito simples, mas claramente induz um número bastante alto de *checkpoints* forçados. A literatura da área apresenta vários protocolos mais complexos que permitem reduções significativas no número de *checkpoints* forçados [18, 17].

A Figura 2 ilustra o comportamento de um protocolo síncrono bloqueante [14]. Nesta classe de protocolos, a obtenção de um *checkpoint* global consistente é coordenada por meio da transmissão de mensagens de controle. Durante a fase de coordenação, os processos interrompem o processamento das mensagens da aplicação e cuidam apenas das mensagens de controle. Um processo denominado iniciador grava um *checkpoint* e envia mensagens de requisição para que os outros processos da aplicação também gravem *checkpoints*. Quando o processo iniciador recebe uma mensagem de cada um dos outros processos indicando que os *checkpoints* foram gravados, ele envia mensagens de liberação para todos eles, de modo que estes possam retomar o processamento da aplicação. Para a classe de protocolos síncronos também existem variações mais complexas descritas na literatura, inclusive protocolos que permitem que a coordenação seja feita sem a interrupção do processamento da aplicação [7].

Dada a variedade de protocolos existentes e a complexidade de se entender o comportamento de alguns deles, consideramos que o desenvolvimento de uma ferramenta para visualização destes protocolos poderia ser bastante útil. Verificamos a existência de alguns visualizadores para algoritmos distribuídos [3, 5], mas que não são específicos para *checkpointing* e que não atendem adequadamente os nossos requisitos.

O objetivo deste trabalho é a especificação e implementação de um ambiente para a visualização de protocolos de *checkpointing* que permita a construção de diagramas espaço-tempo a partir de uma execução real de uma aplicação distribuída. Em uma primeira etapa, estes visualizadores poderão auxiliar a depuração dos protocolos durante a fase de implementação dos mesmos. Em uma segunda fase, o ambiente servirá como ferramenta de pesquisa, permitindo a exposição de detalhes do comportamento global dos protocolos. Será possível visualizar diferenças fundamentais de comportamento entre classes de protocolos e diferenças mais sutis entre protocolos de uma mesma classe. Além disso, os diagramas espaço-tempo poderão indicar qual é o padrão de mensagens gerado por uma determinada aplicação de maneira a auxiliar a escolha do protocolo mais adequado para esta aplicação. O código-fonte desenvolvido para o ambiente será disponibilizado de acordo com a filosofia de *software* livre [1].

O restante deste documento está estruturado da seguinte forma. A Seção 2 detalha o objetivo deste projeto. A Seção 3 descreve o estágio atual de desenvolvimento do trabalho proposto. A Seção 4 descreve o plano de trabalho e o cronograma. A Seção 5 encerra este documento indicando como este trabalho poderá ser expandido.

2 Objetivo

O objetivo deste trabalho é a especificação e implementação de um ambiente para a visualização de protocolos de *checkpointing* que permita a construção de diagramas espaço-tempo a partir de uma execução real de uma aplicação distribuída. Neste projeto inicial, o diagrama espaço-tempo será construído estaticamente a partir de um arquivo de histórico gerado pela aplicação; não estamos considerando que será feita uma animação da execução, pois isto seria bem mais complexo.

A Figura 3 ilustra a arquitetura do ambiente, que é composta por uma aplicação distribuída, um *monitor* e um *visualizador*. A aplicação distribuída é formada por um conjunto de processos $P_1 \ldots P_n$ que se comunicam por meio de troca de mensagens e que gravam periodicamente seus estados de acordo com algum protocolo de *checkpointing*. O processo *monitor* recebe dos processos os eventos de interesse para a visualização e os armazena em um arquivo *histórico*. O processo *visualizador* lê este arquivo e constrói um diagrama espaço-tempo referente à execução da aplicação.

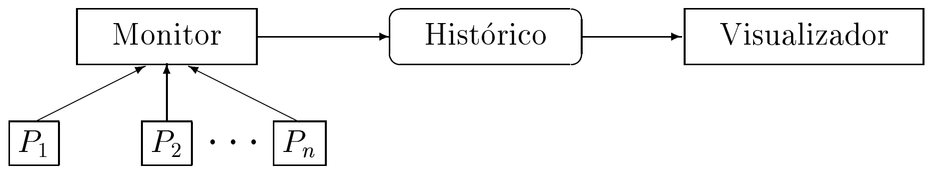

Figura 3: Arquitetura do ambiente

Para que esta arquitetura seja implementada facilmente, consideramos que cada processo tem sua estrutura interna dividida em três partes principais: (i) camada de código específico da aplicação, (ii) camada de *checkpointing* e (iii) camada de comunicação. O código específico da aplicação será implementado utilizando-se a linguagem C [13] e o subsistema de comunicação será implementado utilizando-se LAM-MPI [2]. A camada de *checkpointing* deve encapsular as chamadas de envio e recepção de mensagens da aplicação de maneira a acrescentar as informações necessárias para a execução do protocolo. Esta estrutura permite que se possa testar uma mesma aplicação com vários protocolos e um mesmo protocolo com vários tipos de aplicação.

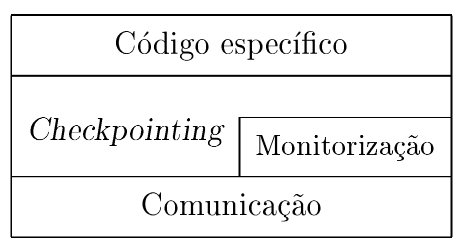

Figura 4: Estrutura de um processo da aplicação

A camada de *checkpointing* também é responsável pelo envio de informações de interesse para o processo *monitor*. O processo *monitor* tem como tarefa principal processar as mensagens vindas de todos os processos da aplicação e produzir o arquivo *histórico* que facilitará a construção do diagrama espaço-tempo pelo *visualizador*. O processo *visualizador* é o responsável pela parte gráfica do ambiente e será programada em Python [16] e Tkinter [4].

3 Estágio atual do trabalho

Inicialmente, o candidato à bolsa leu artigos sobre precedência causal, relógios lógicos e estados globais consistentes [9, 8, 15]. Em seguida, o candidato estudou o ambiente LAM/MPI [2] e implementou pequenas aplicações distribuídas que exercitaram estes conceitos.

Atualmente, o candidato iniciou o estudo de protocolos quase-síncronos de *checkpointing* [17] e também os recursos fornecidos pelo ambiente gráfico escolhido [4]. A Figura 5 mostra uma imagem obtida a partir de um teste implementado pelo candidato que ilustra o comportamento do protocolo proposto por Briatico, Ciuffoletti e Simoncini [6].

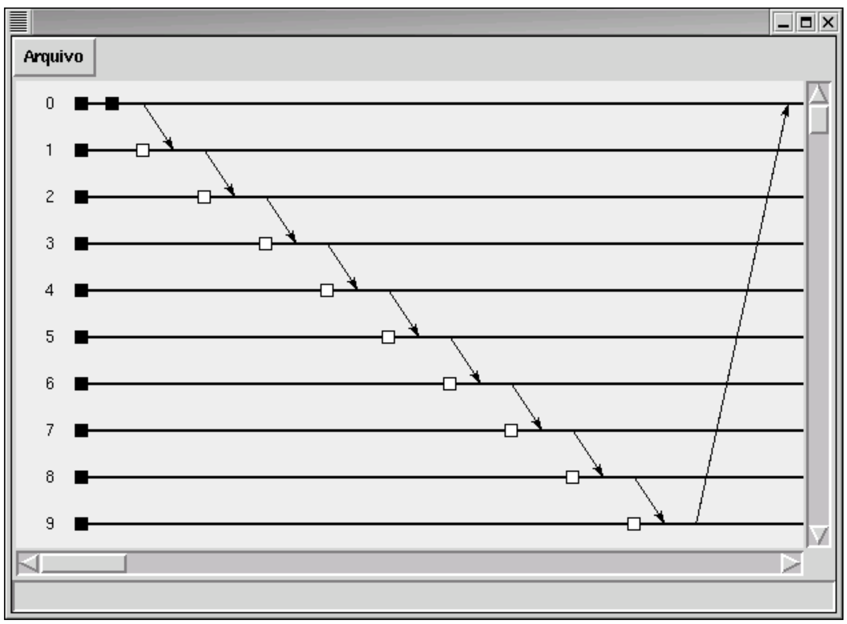

Figura 5: Protótipo da interface

4 Plano de Trabalho e Cronograma

A Tabela 1 mostra o cronograma de trabalho. Nos primeiros dois meses, o candidato continuará o estudo dos protocolos quase-síncronos e do ambiente para a interface gráfica. Em seguida, o candidato deverá implementar uma aplicação distribuída que utilize o o protocolo proposto por Briatico, Ciuffoletti e Simoncini (BCS) [6]. Nesta fase, consideramos que o código da aplicação pode ser o mais simples possível, como uma troca aleatória de mensagens entre os processos da aplicação acrescida da passagem de uma ficha. Neste mesmo período será feita a especificação do formato do arquivo de *histórico* que faz a interface entre os processos *monitor* e o *visualizador*. Desta forma, após seis meses de trabalho teremos um primeiro protótipo do ambiente.

Terminada a implementação do protocolo BCS, que é baseado em índices, serão implementados protocolos baseados em vetores de dependência e que obedecem à propriedade de Rollback-Dependency Trackability (RDT) [19, 12].

Após a visualização dos protocolos quase-síncronos, será iniciada a fase de estudo dos protocolos síncronos. A princípio, será implementado e visualizado apenas o protocolo proposto por Koo e Toueg (KT) [14]. A implementação de outros protocolos está prevista na Seção 5.

Finalmente, os últimos meses do cronograma estão reservados para melhorias na interface com o usuário do ambiente de visualização e para a implementação de outras aplicações teste.

5 Trabalhos Futuros

Este projeto é uma primeira tentativa de se utilizar visualizadores gráficos para auxiliar o entendimento de protocolos para *checkpointing*. Nesta fase inicial, selecionamos apenas alguns protocolos quase-síncronos e um protocolo síncrono para a visualização. Em uma próxima etapa, gostaríamos de completar o conjunto de protocolos a serem implementados com outros protocolos presentes na literatura e também com protocolos em desenvolvimento pelo grupo de pesquisa.

A arquitetura escolhida para a implementação do ambiente (Figura 3) inclui além dos processos da aplicação um processo *monitor*. Um outro projeto poderia fazer uma visualização mais detalhada do processo *monitor*, incluindo a seqüência de *checkpoints* globais consistentes construídos [11] e os predicados globais avaliados [9]. Outra possibilidade seria a simulação de falhas e a visualização da recuperação da aplicação por retrocesso de estado.

	2003								2004			
	Mai	Jun	Jul	Ago	Set	Out	Nov	Dez	Jan	Fev	Mar	Abr
1	■	■										
2	■	■										
3		■	■									
4			■	■	■							
5				■	■	■						
6						■	■	■				
7							■	■				
8									■	■		
9										■	■	■
10										■	■	■

1. Estudo de protocolos quase-síncronos para *checkpointing*
2. Estudo do ambiente para a visualização gráfica
3. Implementação de uma aplicação utilizando o protocolo BCS
4. Especificação da interface entre o monitor e o visualizador
5. Construção do primeiro protótipo
6. Implementação de protocolos quase-síncronos RDT
7. Estudo de protocolo síncrono KT
8. Implementação do protocolo KT
9. Melhorias para a interface de visualização
10. Implementação de outras aplicações de teste

Tabela 1: Cronograma de trabalho

Referências

[1] GNU General Public License. Disponível em `http://www.gnu.org/licenses/gpl.html`.

[2] LAM/MPI Parallel Computing. Disponível em `http://www.lam-mpi.org/`.

[3] LYDIAN: An Extensible Educational Animation Environment for Learning Distributed. Disponível em `http://www.cs.chalmers.se/~lydian/`.

[4] *Tkinter reference: A GUI for Python.* Disponível em `http://www.nmt.edu/tcc/help/pubs/tkinter.pdf`.

[5] ZADA. Disponível em `http://ls4-www.cs.uni-dortmund.de/RVS/zada.html`.

[6] D. Briatico, A. Ciuffoletti, and L. Simoncini. A Distributed Domino-Effect Free Recovery Algorithm. In *4th IEEE Symp. on Reliability in Distributed Software and Database Systems*, pages 207–215, 1984.

[7] G. Cao and M. Singhal. Mutable Checkpoints: A New Checkpointing Approach for Mobile Computing Systems. *IEEE Trans. on Parallel and Distributed Systems*, 12(2):157–172, Feb. 2001.

[8] M. Chandy and L. Lamport. Distributed Snapshots: Determining Global States of Distributed Systems. *ACM Trans. on Computing Systems*, 3(1):63–75, Feb. 1985.

[9] Ö. Babaoğlu and K. Marzullo. Consistent Global States of Distributed Systems: Fundamental Concepts and Mechanisms. In S. Mullender, editor, *Distributed Systems*, pages 55–96. Addison-Wesley, 1993.

[10] E. N. Elnozahy, D. Johnson, and Y.M.Yang. A Survey of Rollback-Recovery Protocols in Message-Passing Systems. Technical Report CMU-CS-96-181, School of Computer Science, Carnegie Mellon University, 1996.

[11] I. C. Garcia and L. E. Buzato. Progressive Construction of Consistent Global Checkpoints. In *19th IEEE Int. Conf. on Distributed Computing Systems*, Austin, Texas, EUA, June 1999.

[12] I. C. Garcia, G. M. D. Vieira, and L. E. Buzato. RDT-Partner: An Efficient Checkpointing Protocol that Enforces Rollback-Dependency Trackability. In *Simpósio Brasileiro de Redes de Computadores*, Florianópolis, Santa Catarina, May 2001.

[13] B. W. Kernighan and D. M. Ritchie. *C: A Linguagem de Programação.* Campus, 1986.

[14] R. Koo and S. Toueg. Checkpointing and Rollback-Recovery for Distributed Systems. *IEEE Trans. on Software Engineering*, 13:23–31, Jan. 1987.

[15] L. Lamport. Time, Clocks, and the Ordering of Events in a Distributed System. *Commun. ACM*, 21(7):558–565, July 1978.

[16] M. Luiz. *Programming Python*. O'Reilly, 1996.

[17] D. Manivannan and M. Singhal. Quasi-synchronous checkpointing: Models, characterization, and classification. *IEEE Transactions on Parallel and Distributed Systems*, 7(10):703–713, July 1999.

[18] G. M. D. Vieira. Estudo comparativo de algoritmos para *Checkpointing*. Dissertação de mestrado, Instituto de Computação—Universidade Estadual de Campinas, 2001.

[19] Y. M. Wang. Consistent Global Checkpoints that Contain a Given Set of Local Checkpoints. *IEEE Trans. on Computers*, 46(4):456–468, Apr. 1997.

Raphael Marcos Menderico

Islene Calciolari Garcia