Ambiente para animação de algoritmos distribuídos baseado em construções progressivas de *checkpoints* globais consistentes

Projeto de Iniciação Científica *

Candidato: Raphael Marcos Menderico

Orientadora: Profa. Dra. Islene Calciolari Garcia

Fevereiro de 2004

Resumo

A visualização de algoritmos pode auxiliar no projeto dos mesmos, na depuração de programas e no ensino. Uma das técnicas utilizadas para permitir uma visualização de alto nível sobre a execução de um algoritmo é construir visões progressivas a partir da execução real de um algoritmo e utilizar um ambiente gráfico para gerar uma animação, mostrando uma sequência de ações que foi executada pelo algoritmo. O ponto central deste projeto é a implementação de um ambiente de animação de algoritmos distribuídos partindo de construções progressivas de *checkpoints* globais consistentes obtidos durante a execução de uma aplicação distribuida. Esse ambiente será disponibilizados de acordo com a filosofia de software livre.

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

*Esse projeto detalha o pedido de renovação de bolsa do projeto *Ambiente para visualização de protocolos de checkpointing* - Processo FAPESP $n^o 03/00483-0$

1 Introdução

Visualizar algoritmos pode ser útil em diversos sentidos. Pode, por exemplo, auxiliar no projeto de algoritmos, na depuração de programas e no ensino. Para obtermos uma visualização de alto nível, uma das técnicas possíveis é construir visões progressivas a partir da execução real de um algoritmo, e então utilizar um ambiente gráfico para gerar uma animação, que permitirá a observação da sequência de ações que o algoritmo executou.

Entretanto, quando tratamos de ambientes distribuídos assíncronos, não é possível obter uma visão global da execução real dos algoritmos. Cada processo participante do ambiente somente tem conhecimento sobre a sua própria execução e pode obter informações dos outros processos através da troca de mensagens. Um visualizador não é exceção, uma vez que ele também é um processo em um ambiente distribuído. A abordagem nesse caso é obter um conjunto de estados, um de cada processo, que poderia ser obtida por um observador onisciente externo [7]. Esse conjunto de estados locais é chamado de estado global consistente.

Para obter um estado global consistente podemos utilizar os estados gravados pela aplicação (ou *checkpoints*). O conjunto de todos os *checkpoints* gravados pelos processos participantes do ambiente é chamado de padrão de *checkpoint*. Esse padrão pode ser obtido pela própria aplicação ou induzido por um protocolo de *checkpointing*. A partir de determinadas classes de padrões de *checkpoint* é possível construir visões progressivas da computação [8].

Atualmente, existe um conjunto muito grande de visualizadores e ambientes de animação para sistemas distribuídos [6, 2, 15, 4], porém, nenhum deles utiliza a construção progressiva de *checkpoints* para obter seus estados globais consistentes e poucos trabalham com o fato do usuário escolher quais pontos da aplicação devem ser exibidos, simplesmente mostrando todos os estados possíveis.

O objetivo desse trabalho é a especificação e implementação de um ambiente para animação de algoritmos distribuídos, baseado em construções progressivas de *checkpoints* globais consistentes, que permitam a construção de uma sequência de ações a partir da execução real de uma aplicação distribuída. Em uma primeira etapa, esse ambiente permitirá analisar o funcionamento dos algoritmos de *checkpointing* e da construção de visões progressivas, complementando as análises já obtidas com o ambiente anterior. Posteriormente, ele pode ser utilizado como ferramenta de ensino e de análise de algoritmos distribuídos. O ambiente, bem como seu código fonte, serão disponibilizados 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 e a Seção 5 encerra este documento indicando como este trabalho poderá ser expandido.

2 Objetivo

Neste segundo projeto, o objetivo é construir uma animação do algoritmo baseado em construções progressivas de *checkpoints* globais consistentes. A animação será construida a partir de um arquivo de histórico gerado pela aplicação. Essa animação não será construída em tempo real, pois isso dificultaria a visualização dos estados globais consistentes, e traria também uma complexidade maior para o projeto.

A Figura 1 ilustra a arquitetura do ambiente, que é composta por uma aplicação distribuída, um *monitor* e um *ambiente de animação*. A aplicação distribuída é formada por um conjunto de processos $P_1 \dots P_n$ 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 arquivo de *definições* é escrito pelo usuário do ambiente e define o conjunto de alterações que devem ocorrer para cada mudança de estado. O *ambiente de animação* lê esses arquivos e constrói uma animação referente à execução da aplicação.

Para facilitar a implementação do ambiente de animação, ele foi dividido em três camadas:

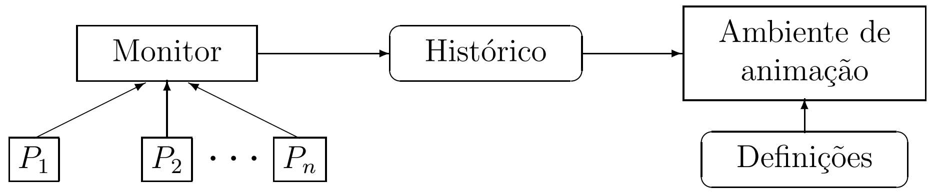

Figura 1: Arquitetura do ambiente

(i) camada de interpretação, (ii) camada de construção progressiva de estados e (iii) camada de animação. A camada de interpretação lê o arquivo de *histórico* e transforma-o em objetos que podem ser utilizados pelo ambiente de animação. A camada de construção progressiva de estados utiliza os objetos gerados pela camada anterior para construir visões progressivas da computação [8]. Essa camada também é responsável por interpretar as alterações entre os estados e informar as alterações necessárias na interface de animação. Por fim, a camada de animação lê o conjunto de alterações necessárias e mostra a próxima imagem na sequêcia de animação. Essa arquitetura pode ser vista na Figura 2

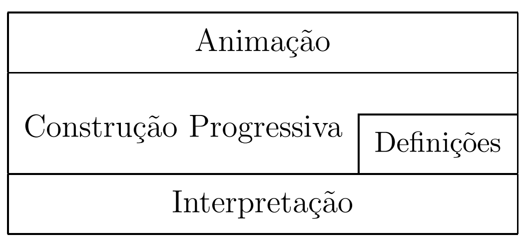

Figura 2: Estrutura do ambiente de animação

Por fim, serão utilizadas as aplicações desenvolvidos no projeto anterior, que incluem um conjunto de processos e aplicações de teste — implementadas utilizando a linguagem C/C++ [9, 14] e o ambiente LAM-MPI [3] —, uma biblioteca de protocolos de *checkpointing* [13, 19, 10, 16, 11, 18, 17] e um processo monitor, que coleta os estados e escreve o arquivo de *histórico*.

3 Estágio atual do trabalho

O candidato a bolsa já possui possui conhecimentos a respeito do funcionamento e da implementação de protocolos de *checkpointing*, a implementaçao de sistemas distribuídos utilizando a linguagem C/C++ [9, 14]e o ambiente LAM-MPI [3], e também da implementação de ambientes gráficos utilizando a linguagem Python [12] e o ambiente Tkinter [5]. Esses conhecimentos foram adquiridos durante o desenvolvimento do projeto *Ambiente de visualização de protocolos de checkpointing*.

Atualmente, o candidato iniciou os estudo de construções progressivas de *checkpoints* globais consistentes [8].

4 Plano de Trabalho e Cronograma

A Tabela 1 mostra o cronograma de trabalho. Nos primeiros dois meses o candidato continuará os estudos de construções progressivas de *checkpoints* globais consistentes e, também, o ambiente para interface gráfica, agora voltados para a realização de animação. Em seguida, o candidato deverá implementar um ambiente de animação para os protocolos da classe ZPF (Z-Path Free) [13, 19]. Nessa primeira versão será possível apenas desenhar um tipo de objeto e somente uma operação será permitida para o mesmo. As mudanças de estado não serão realizadas de forma atômica nessa etapa (ou seja, uma mudança de estado será decomposta em etapas).

Nesse mesmo período o formato do arquivo de definições — utilizado para construir a animação —, será especificado, definindo quais os objetos existentes na interface e quais operações serão possíveis para esses objetos. Após seis meses, teremos todas as funcionalidades do ambiente especificadas, bem como uma primeira versão do ambiente de animação.

Após essa etapa, outras duas versões serão implementadas: a segunda versão implementará as mudanças atômicas de estado e, a terceira versão, implementa o conjunto completo de objetos e operações possíveis para esses objetos, além de implementar algoritmos que permitam a visualização de protocolos das classes ZCF e PZCF (Z-Cycle Free e Partially Z-Cycle Free, respectivamente) [13, 19] e dos protocolos síncronos [10, 16, 11, 18, 17] implementados no projeto anterior.

Por fim, os últimos meses do cronograma estão reservados para correções e pequenas melhorias no ambiente de animação, bem como a especificação de outras aplicações de teste, cada uma com um arquivo de *definições* de exemplo. Os resultados serão apresentados em um *site*, que conterá o código fonte da aplicação, os programas e um manual, mostrando ao usuário como criar suas próprias aplicações e seus arquivos de *definições*.

5 Trabalhos Futuros

Existem alguns caminhos a seguir: num futuro próximo, por exemplo, é possível a utilização de outros ambientes de animação — como o Polka [10] — , ao invés de trabalhar somente com ambientes desenvolvido pelo grupo de pesquisa. Outro possível caminho é construir *frameworks* em outras linguagens, ou mesmo ambientes de simulação, que sejam compatíveis com o sistema de animação proposto, de forma a expandir as possibilidades de uso do sistema. Por fim, também é válido utilizarmos esse ambiente em sala de aula, verificando seus reais usos no ensino de sistemas distribuidos.

	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 construções progressivas de *checkpoints* globais consistentes
2. Estudo do ambiente para animação
3. Especificação dos objetos existentes para animação
4. Especificação das ações permitidas para os objetos
5. Construção da primeira versão
6. Construção da segunda versão
7. Construção da terceira versão
8. Correções e pequenas melhorias no ambiente de animação
9. Construção de novas aplicações de teste, com arquivos de definições
10. Construção do site e do manual do usuário

Tabela 1: Cronograma de trabalho

Referências

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

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

[3] *MPI Primer / Developing With LAM*. Disponível em `http://www.lam-mpi.org`.

[4] Polka Animation System. Disponível em `http://www.cc.gatech.edu/gvu/softviz/parviz/polka.htm`.

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

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

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

[8] 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.

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

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

[11] P.-J. Leu and B. Bhargava. Concurrent Robust Checkpointing and Recovery in Distribuited Systems. In *In Proceedings of the 4th IEEE International Conference on Data Engineering*, pages 154–163, 1988.

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

[13] 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.

[14] V. V. Mizhari. *Treinamento em Linguagem C++*. Makron Books, 1994.

[15] Y. Moses, Z. Polunsky, A. Tal, and L. Ulitsky. Algorithm visualization for distributed environments. In *Proceedings IEEE Symposium on Information Visualization 1998*, pages 71–78, 1998.

[16] R. Prakash and M. Singhal. Minimal Global Snapshot and Failure Recovery using Infection. Technical Report OSU-CISRC-12/93-TR42, Department of Computer Science, The Ohio State University, 1993.

[17] T. C. Sakata. BCS-SNB: Um Protocolo de Checkpointing Síncrono Não Bloqueante. In *Workshop de Teses e Dissertaçoes (WTC), Latin-American Symposium on Dependable Computing(LADC)*, pages 73–78, São Paulo, SP, Oct. 2003.

[18] T. C. Sakata, I. C. Garcia, and L. E. Buzato. Checkpointing Síncrono Bloqueante Minimal com Iniciadores Concorrentes. In *Simpósio Brasileiro de Redes de Computadores*, pages 681–696, Natal, Rio Grande do Norte, May 2003.

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

Raphael Marcos Menderico

Islene Calciolari Garcia