Um estudo do Modelo de Atores aplicado à concorrência de recursos

Fellipe Souto Sampaio

Orientadora: Prof. Dra. Ana Cristina Vieira de Melo

Departamento de Ciência da Computação - Instituto de Matemática e Estatística - Universidade de São Paulo

Introdução

Podemos entender o Modelo de atores como:

- Um modelo matemático para descrever a **computação concorrente**
- Um método alternativo para lidar com **recursos compartilhados**
- Uma forma alternativa de modelar problemas clássicos de concorrência e suas soluções, como o **Jantar dos filósofos**
- A base para a biblioteca **Akka**, que permite criar sistemas concorrentes baseados em atores que sejam: **assíncronos**, **imutáveis** e **orientados a mensagens**

Objetivos

- **Aprender** os fundamentos do modelo de atores
- **Entender** a concorrência de recursos através do problema do Jantar dos Filósofos
- **Modelar** o problema dos Filósofos e **implementar** três soluções diferentes utilizando a biblioteca Akka e a linguagem de programação Scala
- **Simular** o problema e **avaliar** o desempenho dos algoritmos a partir de diferentes métricas

O Modelo de Atores

Modelo matemático para programação concorrente idealizado por Carl Hewitt em 1973 **[1]**. Atores são unidades funcionais com **processamento**, **armazenamento** e **comunicação** e que podem **[2]**:

- Tomar decisões locais e alterar variáveis internas
- Mandar mensagens para si, ou para outros atores
- Criar novos atores
- Mudar seu comportamento interno

O Jantar dos Filósofos

O problema pode ser descrito como:

- Cinco filósofos estão dispostos em uma mesa circular, com um prato de espaguete na frente de cada um
- Os filósofos alternam entre pensar e comer
- Para comer um filósofo precisa de dois garfos ao mesmo tempo
- Existe apenas um garfo entre cada prato
- Filósofos adjacentes não podem comer ao mesmo tempo

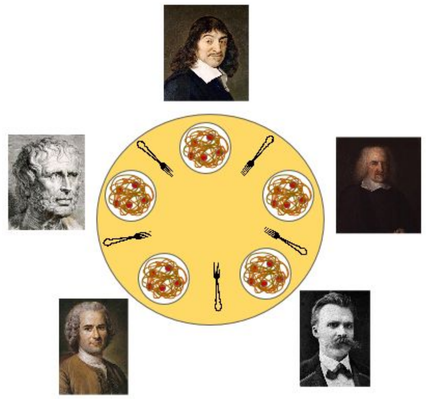

Fig 1: Uma ilustração do problema dos filósofos

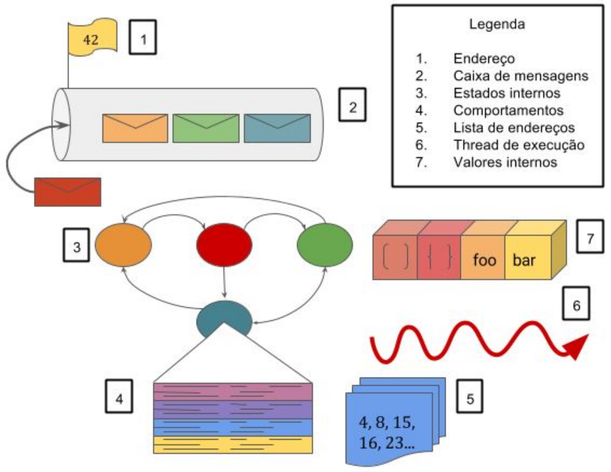

Fig 2: Estrutura interna de um ator

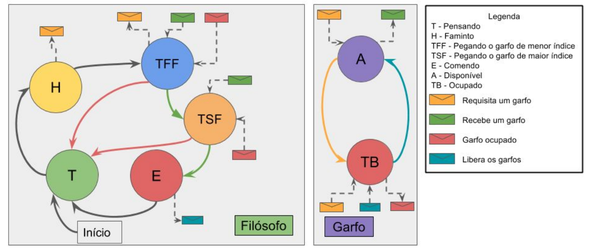

Fig 3: Diagrama de estados do algoritmo Hierarquia de Recursos

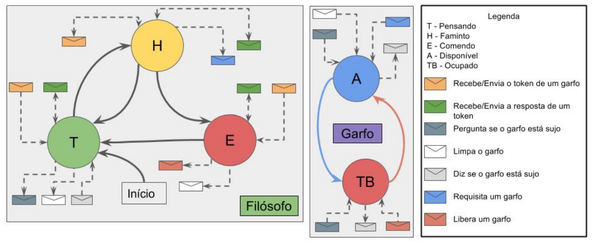

Fig 4: Diagrama de estados do algoritmo Chandy-Misra

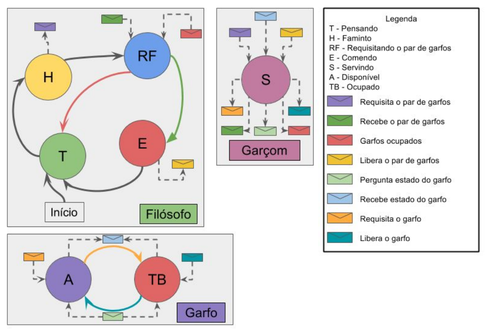

Fig 5: Diagrama de estados do algoritmo Waiter

Algoritmos

Uma solução válida para o problema deve verificar as seguintes propriedades:

- Fornecer um comportamento para os filósofos
- Possibilitar que todos os filósofos consigam comer
- Evitar que os filósofos morram de fome
- Coordenar o acesso aos recursos compartilhados
- Ser justa, de forma que todos comam em média o mesmo e esperem uma quantidade equivalente de tempo

Os algoritmos Hierarquia de Recursos **[3]** **[4]**, Waiter **[5]** e Chandy-Misra **[6]** verificam as propriedades apresentadas.

Conclusão

Através da modelagem do problema e das simulações podemos concluir que:

- Cada filósofo, garfo e garçom é um ator, e seu comportamento é definido como um máquina de estados
- Cada um dos algoritmo define um comportamento para os filósofos
- Todos os filósofos se alimentaram, isso evidencia a ausência de impasses, como deadlocks e livelocks
- Um tempo de espera quase uniforme indica que nenhum filósofo morreu de fome
- A ausência de impasses indica que o acesso ao recurso foi coordenado de forma correta e simétrica entre os filósofos
- Cada algoritmo coordena o acesso aos recursos compartilhados de formas diferentes

Resultados

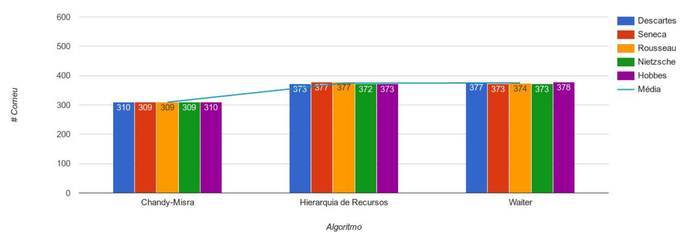

Fig 6: Quantidades de vezes que os filósofos comeram

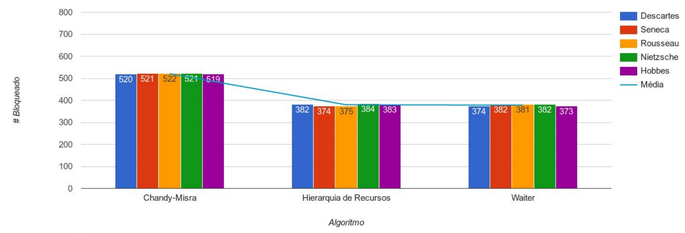

Fig 7: Quantidades de vezes que os filósofos foram impedidos de comer

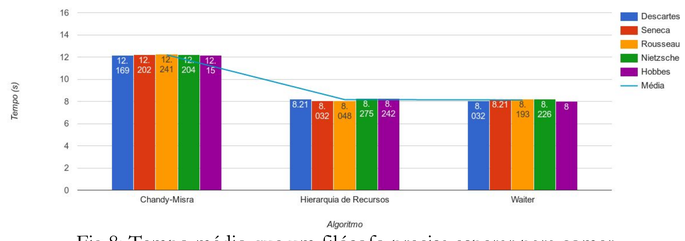

Fig 8: Tempo médio que um filósofo precisa esperar para comer

Referências

[1] Carl Hewitt et al. A universal modular actor formalism for artificial intelligence. IJCAI'73, 1973, pp. 235–245.

[2] Gul A. Agha. Actors: a Model of Concurrent Computation in Distributed Systems. MIT Artificial Intelligence Laboratory, 1985, pp. 12-31.

[3] Edsger W. Dijkstra. Hierarchical ordening of sequential processes. EWD310, p. 7

[4] Edsger W. Dijkstra. Two starvation-free solutions of a general exclusion problem. EWD625

[5] C.A.R Hoare. Communication Sequential Process. Prentice Hall; 1 edition, 1985., pp. 57-61.

[6] K.M. Chandy et al. The drinking pilosopher problem. ACM Transactions on Programming Languages and Systems, Vol. 6, No. 4, 1984, pp. 632–646

Para mais informações sobre o trabalho, consulte:
https://github.com/fllsouto/mac0499
ou envie um email para fllsouto@gmail.com