UMA IMPLEMENTAÇÃO DE LINDA UTILIZANDO NUVEM COMO ESPAÇO DE TUPLAS

LUCAS EDUARDO BRETANA[1]; ALANA SCHWENDLER[2]; GERSON GERALDO H. CAVALHEIRO[3]

[1]Universidade Federal de Pelotas – lebretana@inf.ufpel.edu.br
[2]Universidade Federal de Pelotas – aschwendler@inf.ufpel.edu.br
[3]Universidade Federal de Pelotas – gerson.cavalheiro@inf.ufpel.edu.br

1. INTRODUÇÃO

Programas com grande carga de dados a serem processados podem fazer uso de paradigmas de processamento paralelo e distribuído para melhor o tempo de execução e tirar maior vantagem do processador. O processamento paralelo consiste basicamente em dividir um programa em partes a serem computadas simultaneamente, em diferentes fluxos de execução. Essas tarefas devem compartilhar os seus resultados de modo a desenvolver o problema por completo.

Em ambientes com memória distribuída, além do paralelismo permitir o aumento de desempenho em termos de tempo de processamento, também permitem a construção de aplicações que manipulem uma quantidade de dados maior que aquela possível de ser armazenada em um único sitio físico. Neste caso, a colaboração entre as partes da aplicação, suas tarefas, deve se dar com apoio de algum mecanismo de comunicação, como por troca de mensagens (CÁCERES, MONGELLI, SONG, 2002), RMI (SARMENTO, 2003) ou DSM (COX, YU, 1997). Nesse trabalho damos foco ao modelo baseado em Espaço de Tuplas (TS, do inglês Tuple Space) cujo modelo apresenta um espaço de compartilhado entre processos trabalhadores (workers) onde os dados são manipulados no formato de tuplas na forma: <key, content>. O elemento key consiste em uma chave única de identificação da tupla, e o content é o conteúdo desta tupla, correspondendo ao dado efetivo ou mesmo representando uma tarefa a ser executada.

O TS deve ser implementado de modo a ser visível e acessível a todos os *workers* (máquinas) que estão realizando as tarefas que operam concorrentemente e acessam o espaço de tuplas para troca de dados. Para tal, geralmente esse papel é realizado por um modelo de clientes e servidores, usando de modos de computação em rede (Grid, Nuvem etc). A computação em nuvem é, em essência, um modelo de computação paralela, e utilizar esta técnica de nuvem como o TS é vantajosa por vários motivos, ressaltando principalmente questões como ubiquidade, custo a longo prazo e segurança dos dados.

Um dos modelos para implementação de um TS para programação paralela é o LindaTS (Linda Tuple Space), que consiste em primitivas simples que suportam um estilo desacoplado de programação paralela (AHUJA, CARRIERO, GELERNTER, 1986). Neste modelo a interação com o TS é feita com quatro primitivas de comunicação: `read`, `in`, `out` e `eval` que são explicadas no decorrer do artigo.

Com estes conceitos em mente, foi criada uma implementação de LindaTS em forma de biblioteca utilizando o Dropbox como espaço de tuplas com o intuito de que outros projetos possam usufruir dessa arquitetura.

O presente trabalho foi realizado com apoio do Programa Nacional de Cooperação Acadêmica da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – CAPES/Brasil

2. METODOLOGIA

Para trabalhar com o espaço de tuplas, foram analisados outros projetos que implementam um modelo de compartilhamento de dados para evolução colaborativa da computação de um problema, como o Seti@Home. O problema considera a existência de um ambiente de pesquisa sob o qual são desenvolvidos projetos colaborativos. Neste ambiente, existe um sitio no qual um projeto colaborativo foi instanciado. O papel de administrador deste projeto é atribuído ao seu proprietário no momento da instanciação de um novo projeto. No modelo de aplicação previsto, dados são compartilhados, podendo ser manipulados por diferentes heurísticas visando a obtenção de um resultado global. Colaboradores podem ser associados ao projeto para, compartilhando o acesso aos dados, aplicar suas próprias heurísticas de manipulação, compartilhando, por sua vez, os resultados por eles obtidos.

A figura a seguir ilustra ambiente de desenvolvimento de projetos colaborativos concebidos.

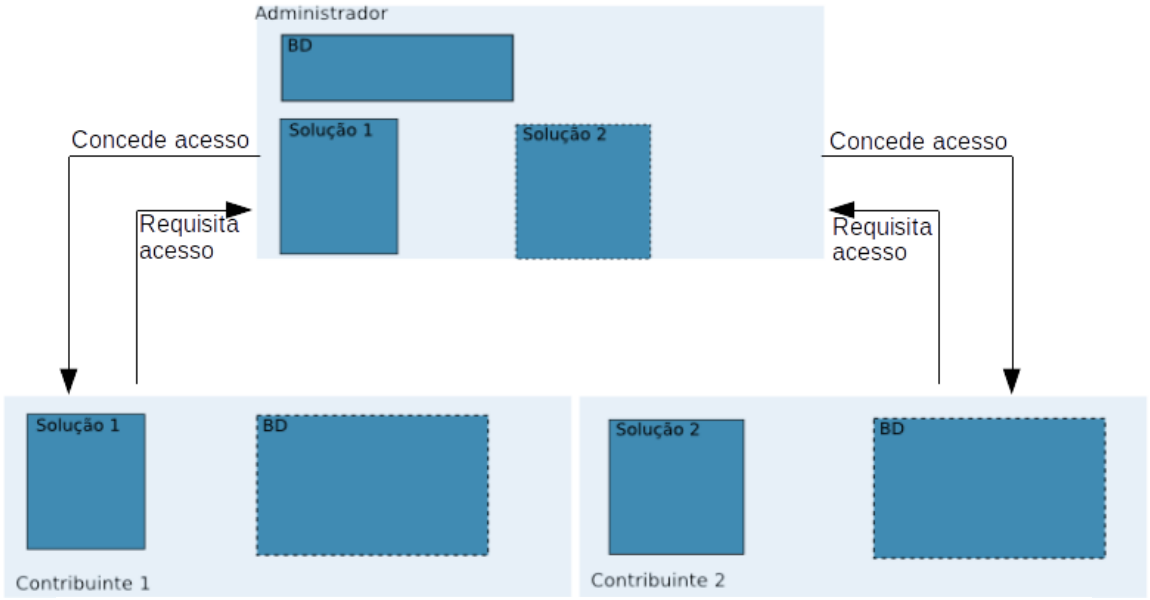

Figura 1: Os sitios são representados pelas caixas azul claro e as bases de dados são representadas pelas caixas azul escuro. Nas bases de dados, a borda continua representa a base de dado original e a borda pontilhada mostra que a base foi compartilhada. O compartilhamento é representado pelas setas que o mostram nas duas etapas *request access* e *grant acess*.

O desenvolvimento de um modelo de espaço de tuplas para compartilhar dados na programação paralela se baseia em tarefas básicas como leitura, escrita e deleção. Uma vez que se tem a disponibilidade dessas tarefas, são desenvolvidas primitivas para manipular o TS e que serão disponibilizadas pelo usuário deste modelo.

O espaço de tuplas disponibiliza uma possibilidade de compartilhamento de dados e informações. Nele serão armazenados os dados a serem processados e os resultados obtidos a partir deste processamento. No caso deste projeto, a nuvem que será utilizada é o Dropbox. Para isso, utilizamos a Dropbox Core API SDK 1.8.2 API (Application Programming Interface ou Interface de Programação de Aplicativos) que é disponibilizada pela empresa. Usamos o Dropbox pela facilidade de uso da API devido à documentação ser mais completa e compreensível que outras oferecidas, além de um considerável número de

trabalhos usando essa API disponibilizados como software livre. O projeto foi desenvolvido na linguagem de programação Java na versão 8 do JDK.

Estas primitivas complexas são quatro: read, in, out e eval. A seguir, a explicação da funcionalidade de cada uma delas:

- **Read**: lê uma tupla do TS com base em um critério de identificação;
- **In**: lê e retira uma tupla do TS com base em um critério de identificação;
- **Out**: atribui uma nova tupla no TS para ser computada;
- **Eval**: recebe uma método e uma tupla, aplica a método à mesma e salva o resultado no TS, no formato de uma tupla.

O projeto foi desenvolvido no formato de API a ser disponibilizado para quem quiser desenvolver um projeto que posso utilizar LindaTS como espaço de compartilhamento. A implementação de cada uma das primitivas foi feita como descrito a seguir:

- **Read**: recebe apenas um parâmetro que serve de parâmetro para identificar a tupla procurada. Esse parâmetro é uma função lambda tipada como uma *FunctionalInterface*, introduzida no Java 8, de modo a ser aplicada em cada tupla existente no TS até que uma seja dado *match* (correspondente) em alguma, e esta é retornada pela função. Optou-se pela busca não bloqueante, onde caso não seja encontrada um tupla certa, é retornado um *null*.
- **In**: funciona basicamente da mesma forma que a primitiva Read (também recebe um função lambda de *match* e é implementada de modo não bloqueante), porém após ser retornada a instancia da tupla, a sua referencia no TS é apagada.
- **Out**: essa primitiva tem como parâmetro uma tupla onde o campo *content* é de qualquer tipo, desde que seja passível de serialização.
- **Eval**: nessa primitiva a função a ser aplicada sobre o valor é uma função lambda, também tipada como uma *FunctionalInterface* e uma tupla. A função é aplicada sobre o campo *content* da tupla e o resultado dessa função é colocado no TS usando como o *key* o valor que a tupla original tinha.

Dentro do TS, as tuplas são tratadas como arquivo, sendo que seu nome é o valor *key* da tupla. Já o conteúdo do arquivo é o *content* da tupla que é escrito usando a *stream* de *bytes* geradas pela Máquina Virtual Java (Java Virtual Machine ou JVM), isso porque o *content* da tupla deve implementar a *Interface Serializable* da API Java.

3. RESULTADOS E DISCUSSÃO

A biblioteca criada permite que sejam desenvolvidos projetos que utilizem o espaço de tuplas com uma memória compartilhada para o processamento paralelo de tarefas. Primeiramente é criada uma base de dados iniciais de tarefas a serem processadas no formato de TS dentro do Dropbox. Uma vez que um novo integrante entre no projeto, ele recebe um programa que deverá ser executado e esse usuário deve manter uma conta no Dropbox .

O funcionamento da aplicação se dá da seguinte maneira: 1) O programa requisita acesso a uma conta do Dropbox. 2) Uma vez que o acesso é concedido, faz uma requisição de compartilhamento das bases de dados. 3) Tendo acesso a essas bases, inicia-se a execução de tarefas e seus resultados são escritos no TS em uma nova base de dados, sendo esses resultados novas tarefas ou dados finais. 4) Quando a aplicação termina, seja por não ter mais tarefas ou porque o

usuário a fecha, o acesso à base de dados é finalizado, as informações que ela colocou no TS ficam disponíveis para o projeto em si e todas os arquivos referentes ao TS são deletados do Dropbox do contribuinte, para não onerar custos individuais. Com esta sequência de passos, qualquer usuário do Dropbox pode contribuir para projetos que necessitem de um maior processamento de dados.

4. CONCLUSÕES

Desenvolver uma biblioteca para manipular um espaço de tuplas, dentro do Dropbox, usando as primitivas do modelo LindaTS é viável e pode ser utilizado em projetos necessitem de um maior processamento de dados. Esses projetos devem disponibilizar um programa a ser executado em diferentes máquinas de modo a distribuir o processamento das tarefas. As tarefas devem ser dispostas no formato de tuplas e compartilhadas seguindo os modelos apresentados.

Atualmente a API do Dropbox utilizada, versão 1.8.2, limita algumas partes do projeto, sem no entanto inviabilizar o modelo. Por exemplo, o compartilhamento das pastas referentes ao TS deve ser feito manualmente pelo dono do projeto, assim como a interrupção de tal compartilhamento, não existe por ora a possibilidade de realizar este trabalho de forma automatizada. Além disso, a pasta do TS é sincronizada no Dropbox dos contribuintes em modo oculto, embora isso não torna impossível que ela seja vista pelo dono da conta.

Em trabalhos futuros pretende-se dar manutenção ao nosso software, tornando-o compatível com novas versões de API do Dropbox, trazendo mais segurança aos processos de autenticação e possibilidade de gerenciar o compartilhamento de arquivos dentro da nossa biblioteca. Além disso, pretende-se criar um biblioteca voltada a contribuintes desenvolvedores para que eles possam programar seus próprios softwares que vão trabalhar em cima de bases de dados compartilhadas, existentes no projeto.

5. REFERÊNCIAS BIBLIOGRÁFICAS

AHUJA, S.; CARRIERO, N.; GELERNTER, D. Linda and Friends. **IEEE Journals & Magazines.** Los Alamitos, CA, USA. v. 19. p. 26 - 34. 1986.

CÁCERES, E. N.; MONGELLI, H.; SONG, S. W. Algoritmos Paralelos usando CGM/PVM/MPI: Uma introdução. **XXI Congresso da Sociedade Brasileira de Computação Jornada de Atualização de Informática.** São Paulo, BRASIL. P, 219 – 278. 2001.

COX, A; YU, W. Java/DSM: A platform for heterogeneous computing. **Concurrency: Practice and Experience**. Houston, TX. v. 9, n. 11, p. 1213-1224. 1997.

ORACLE. **Trail: RMI**. Java Documentation. Tutorials. Acessado em 02 de agosto de 2016. Online. Disponível em: https://docs.oracle.com/javase/tutorial/rmi/.

KORPELA, E.; WERTHIMER, D.; ANDERSON, D.; COBB, J.; LEBOFSKY, M. SETI@HOME - MASSIVELY DISTRIBUTED COMPUTING FOR SETI. **IEEE Journals & Magazines.** Piscataway, NJ, USA. v. 3. p. 78 - 81. 2001.