ILUCTUS: Uma implementação de Linda usando Cloud como Espaço de Tuplas.

Lucas Eduardo Bretana, Alana Schwendler, Gerson Gerlado H. Cavalheiro

[1]Laboratory of Ubiquitous and Parallel Systems – UFPEL
Pelotas, RS - Brasil

{lebretana, aschwendler, gerson.cavalheiro}
@inf.ufpel.edu.br

Resumo. *Processamento distribuído usa tecnologias que permitem o compartilhamento de recursos de processamento. O espaço de tuplas é um modelo de programação concebido sobre essas tecnologias que foi retomado neste trabalho como uma alternativa para o desenvolvimento de aplicações em ambiente de nuvem. A biblioteca ILUCTUS, é apresentada neste artigo, bem como um estudo dos custos de suas operações elementares.*

1. Introdução

As tecnologias de comunicação promovidas pela Internet fomentaram a criação de novos mecanismos de colaboração entre as pessoas. Alguns destes mecanismos permitiram formalizar metodologias de desenvolvimento de conhecimento e/ou produtos, tal o *Creative Commons* e o *GNU Public License* que auxiliaram o desenvolvimento da pesquisa científica. Este trabalho propõe uma ferramenta de mesmo propósito de modo compartilhado. A proposta pode ser vista como uma extensão do modelo de computação voluntária, como Seti@Home [Anderson et al. 2002]. Na proposta, projetos colaborativos são criados e abertos à contribuições públicas. A abertura de um projeto implica da disponibilização de um conjunto de dados para processamento. As contribuições consistem na manipulação dos dados, geração e disponibilização de resultados.

Em ambientes de memória distribuída, o paralelismo permite o aumento de desempenho e também a construção de aplicações que manipulem uma quantia de dados maior que a suportada em um único sítio físico. Neste caso, a colaboração entre tarefas deve se dar com apoio de algum mecanismo de comunicação, como por troca de mensagens [Cáceres et al. 2001], RMI [ORACLE 2016]; ou DSM [Yu and Cox 1997]. Nesse trabalho o foco é dado ao Espaço de Tuplas (TS, do inglês *Tuple Space*)[Wells et al. 2004]. Um TS é composto por um espaço de dados compartilhado entre processos trabalhadores (*workers*). O dados são manipulados na forma de tuplas: <*key*, *content*>. O elemento *key* consiste em uma identificação única da tupla, e o *content* é o conteúdo da respectiva tupla, para a implementação fica sendo um objeto serializado.

O modelo de TS, de 1985, foi suplantado pelos outros métodos de comunicação. Porém, o uso de novas tecnologias de nuvens computacionais permite a criação de uma ferramenta de compartilhamento de dados produzidos por diferentes usuários. Dessa forma a colaboração se dá pelo compartilhamento do processamento, resultados e custos de armazenagem dos dados.

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

Considerando estes conceitos, foi criada a ferramenta ILUCTUS, uma implementação de LindaTS em forma de biblioteca utilizando o Dropbox como espaço de tuplas com o intuito de que outros projetos usufruam dessa arquitetura.

2. Metodologia

Foram analisados outros projetos que implementam um modelo de compartilhamento de dados para evolução de um problema, como o Seti@Home. O problema considera a existência de um ambiente de pesquisa sob o qual são desenvolvidos projetos colaborativos. O papel de administrador 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 aplicar suas próprias heurísticas de manipulação compartilhando, por sua vez, os resultados por eles obtidos.

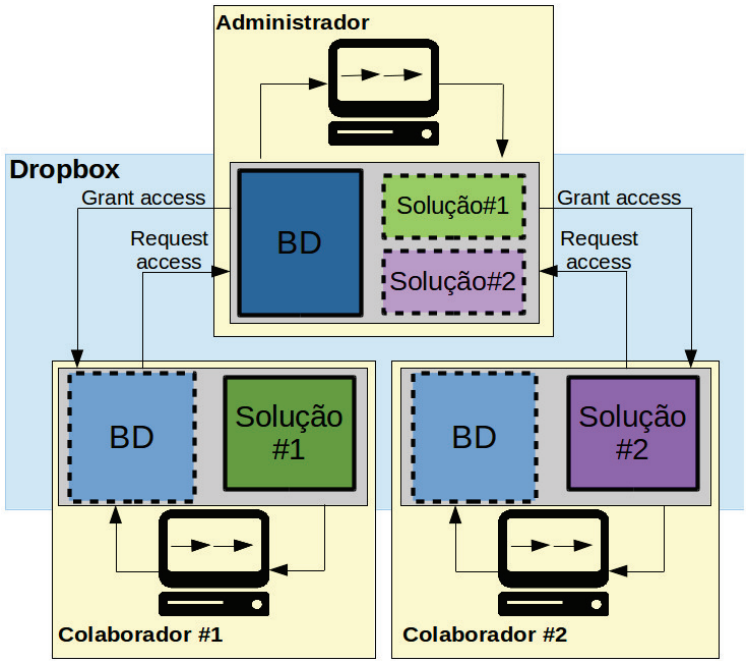

Figura 1. Ilustração da arquitetura.

A Figura 1 ilustra ambiente de desenvolvimento de projetos colaborativos concebidos. Nas bases de dados (BD), a borda contínua 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 espaço de tuplas possibilita o compartilhamento de dados e informações. Para interagir com a nuvem do Dropbox, foi utilizada a Dropbox Core SDK 2.0.1 API (*Application Programming Interface*). A opção pelo Dropbox considerou a facilidade de manipulação de sua API e a acessibilidade a outros trabalhos e documentações disponi-

bilizadas. O projeto foi desenvolvido na linguagem de programação Java na versão 8 do JDK.

O desenvolvimento da ferramenta ILUCTUS baseou-se em quatro primitivas de funcionamento, que derivam da tecnologia LindaTS. São elas: `read`, `in`, `out` e `eval`. A seguir, a explicação da funcionalidade de cada uma delas:

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

A ILUCTUS foi desenvolvida no formato de API, que permite a construção de contribuições a um projeto colaborativo. Suas primitivas são:

1. **Read:** recebe apenas um parâmetro que identifica a tupla procurada. Esse parâmetro é uma função *lambda* tipada como uma *FunctionalInterface* de modo a ser aplicada em cada tupla existente no TS até que uma seja dado *match* (correspondente), sendo retornada. Realiza busca não bloqueante, onde caso não ocorra *match*, é retornado um *null*.
2. **In:** de operação similar ao *Read*, porém após ser retornada a instancia da tupla, sua referência no TS é apagada.
3. **Out:** tem como parâmetro uma tupla onde o campo *content* é de qualquer tipo, desde que seja passível de serialização.
4. **Eval:** a função a ser aplicada sobre o valor é uma função *lambda* e uma tupla. A função é aplicada no campo content da tupla e seu resultado é colocado no TS usando como o key o valor original da tupla.

Dentro do TS, as tuplas são tratadas como arquivos, sendo que seu nome é o valor *key* da tupla. Já o conteúdo é o *content* da tupla, escrito usando a *stream* de *bytes* gerada 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 TS 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 utiliza da ILUCTUS para fazer a evolução das bases de dados, 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) Requisita o compartilhamento das bases de dados. 3)Inicia a execução de tarefas e seus resultados são escritos no TS em uma nova base de dados. 4) Ao término, seja por não ter mais tarefas ou porque o usuário encerra, o acesso à base é finalizado, os resultados 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 isto, qualquer usuário do Dropbox pode contribuir para projetos que necessitem de um maior processamento de dados.

Para aferir a funcionalidade da ferramenta, a Tabela 1 apresenta avaliações sobre os tempos de latência associados às execuções de suas primitivas. Deve ser considerado que o Dropbox exige um tempo mínimo entre as requisições e que os presentes resultados correspondem a uma média de 30 chamadas a cada primitiva a partir de um computador localizado na rede da UFPel.

A manipulação das tuplas é feita pelas primitivas descritas. Logo, foram feitas execuções das primitivas `Read`, `In` e `Out` de modo a fazer a coleta dos seus tempos de execução em alguns cenários variados. Para `Read` e `In` foram realizados buscas no TS com poucas tuplas (dez), uma quantidade média (cinquenta) e uma quantidade grande (cem). Para `Out` não se fazem necessárias essas variações, já que sua ação é destrutiva. Os testes com a primitiva `Eval` foram dispensados pois a mesma se trata de uma execução sequencial de outras primitivas.

Tabela 1. Tempos médios e desvio padrão de cada cenário de execução

	Out	Read			In		
	-	P	M	G	P	M	G
μ	1.09s	3.78s	16.3s	32.33s	7.27s	16.29s	33.15s
δ	0.36s	0.37s	0.65s	1.57s	0.77s	0.49s	0.99s

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 que necessitem de um maior processamento de dados. Esses projetos devem compartilhar dados e tarefas no formato de tuplas e utilizarem a ILUCTUS para manipulação das mesmas.

Em trabalhos futuros pretende-se desenvolver variações das primitivas atuais, como, escrita não destrutivas, leituras que retornam conjuntos de tuplas e leituras bloqueantes. Além disso serão desenvolvidos casos de testes para melhor avaliar o projeto.

Referências

Ahuja, S., Curriero, N., and Gelernter, D. (1986). Linda and friends. *Computer;(United States)*, 19(8).

Anderson, D. P., Cobb, J., Korpela, E., Lebofsky, M., and Werthimer, D. (2002). Seti@ home: an experiment in public-resource computing. *Communications of the ACM*, 45(11):56–61.

Cáceres, E. N., Mongelli, H., and Song, S. W. (2001). Algoritmos paralelos usando cgm/pvm/mpi: uma introduçao. In *XXI Congresso da Sociedade Brasileira de Computaçao, Jornada de Atualizaçao de Informática*, pages 219–278.

ORACLE (2016). Trail: Rmi. java documentation. tutorials. Disponível em: <http://docs.oracle.com/javase/tuorial/rmi>. Acesso em: Agosto de 2016.

Wells, G. C., Chalmers, A. G., and Clayton, P. G. (2004). Linda implementations in java for concurrent systems. *Concurrency and Computation: Practice and Experience*, 16(10):1005–1022.

Yu, W. and Cox, A. (1997). Java/dsm: A platform for heterogeneous computing. *Concurrency: Practice and Experience*, 9(11):1213–1224.