Um estudo de caso da ferramenta ILUCTUS utilizando o cálculo de Fibonacci

Alana Schwendler, Lucas Eduardo Bretana, Ana Marilza S. Pernas, Gerson Geraldo H. Cavalheiro

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

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

Resumo. *Aplicações de processamento distribuído necessitam de meios para realizar a comunicação entre aplicação e processamento. Para isto, apresenta-se uma nova aplicação de modelo colaborativo que utiliza o espaço de tuplas para armazenar e compartilhar dados. Este modelo usa algumas primitivas básicas para manipular as informações. Será apresentado um estudo de caso do cálculo de Fibonacci utilizando este modelo de comunicação.*

1. Introdução

A obtenção de resultados do processamento de dados muitas vezes pode ser um processo difícil, trabalhoso e demorado pela demanda de recursos computacionais para execução de uma determinada tarefa. Esta demanda de processamento pode exceder a capacidade de processamento disponível de um usuário, representado por uma pessoa física, tal como um pesquisador em um centro de pesquisa, ou jurídica, ou um departamento de uma empresa. Nestes casos, o processamento distribuído (Navaux 1989) representa uma alternativa interessante, quando permite que diferentes sítios, dotados de recursos computacionais, ofereçam a capacidade de processamento necessária (Ahuja et al. 1986). Neste contexto, este trabalho aborda uma aplicação da ferramenta ILUCTUS (Bretana et al. 2017), desenvolvida pelos autores deste artigo, a qual visa explorar a infraestrutura de nuvem do Dropbox como um substrato para execução colaborativa de projetos de pesquisa.

O cenário identificado é aquele onde um grupo de pesquisadores se interessa no processamento de algum tipo de dado em busca de desenvolvimento científico. Neste cenário, um conjunto de dados é processado segundo alguma heurística, gerando como resultado, novos conhecimentos na forma de um novo conjunto de dados. É de se esperar, neste cenário, que diferentes heurísticas possam ser empregadas e que os resultados produzidos alimentem um novo ciclo de produção de conhecimento. Neste modelo colaborativo existe o princípio do compartilhamento dos dados e distribuição do processamento. Os papéis de colaboração identificados na ILUCTUS são o de Administrador e o dos Colaboradores.

Um Administrador é o pesquisador responsável pela instanciação de um Projeto Colaborativo, sendo também aquele que oferece o primeiro conjunto de dados a serem manipulados em um Espaço de Dados Compartilhado (aqui chamado de ambiente compartilhado). Cabe ao Administrador também definir as regras de colaboração e de permissões de acesso. Um Colaborador é um pesquisador que, segundo alguma heurística,

processa em seus recursos de processamento próprios, os dados compartilhados no Projeto e oferece, em contrapartida, os resultados de seu processamento também no mesmo ambiente compartilhado. Um Administrador também pode exercer papel de Colaborador.

O ambiente compartilhado manipulado pela ILUCTUS consiste em áreas de dados alocadas para cada participante, Administrador e Colaboradores, do Projeto Colaborativo. Assim, não apenas existe o compartilhamento dos dados e a distribuição do processamento em diferentes sítios, mas também há o compartilhamento dos custos associados ao armazenamento dos dados.

Neste trabalho é exemplificado o uso da ILUCTUS pela apresentação do desenvolvimento Projeto Colaborativo. Neste Projeto, é considerada uma aplicação sintética onde deseja-se crescer o conhecimento em uma base de dados. O problema é representado pelo cálculo da série de Fibonacci. No exemplo proposto, o Administrador cria uma base de dados inicial no ambiente compartilhado com os valores para cada posição da série de Fibonacci e libera para Colaboradores efetivarem seu preenchimento.

2. Aplicação

A aplicação de compartilhamento foi desenvolvida utilizando a linguagem de programação Java (Arnold et al. 2000) na versão 8 do JDK com apoio da ILUCTUS sobre o Dropbox. O funcionamento dessa ferramenta se dá pelas tarefas de leitura, escrita e deleção de dados no ambiente compartilhado, sendo que os dados estão descritos por tuplas contendo uma chave de identificação e a informação propriamente dita. A chave de identificação pode ser dada por uma expressão regular representada por uma expressão lambda. As primitivas disponíveis são: `read`, que lê um dado de acordo com a chave de identificação fornecida; `in`, que lê um dado e o retira do ambiente compartilhado de acordo com a identificação fornecida; `out`, insere uma nova tupla no ambiente compartilhado; `eval` que recebe uma tupla e um método, retira esta tupla do ambiente compartilhado, aplica o método na tupla e atualiza ela com o novo valor no ambiente compartilhado.

Para representar a produção de conhecimento, foi implementada uma aplicação sintética reproduzindo o cálculo recursivo de Fibonacci dado pela Equação 1. Opta-se pelo método recursivo para que seja possível gerar uma maior carga de processamento e assim perceber melhor o desempenho da aplicação.

$$F_{(n)} = \begin{cases} 0, & \text{if n = 0} \\ 1, & \text{if n = 1} \\ F_{(n-1)} + F_{(n-2)}, & \text{if n > 1} \end{cases} \quad (1)$$

A identificação da tupla é a posição desejada da série e seu dado é o respectivo valor da posição identificada. Se a tupla estiver apenas com a identificação preenchida e seu conteúdo estiver nulo, significa que ela está disponível para o processamento. Caso contrário, aquela posição da série já foi calculada e disponibilizada.

O Administrador disponibiliza diversos valores que possam ser calculados e que sejam relevantes para o seu problema, podendo realizar isto com a primitiva `out`. A maneira com que o Colaborador vai calcular o valor de Fibonacci é de sua escolha. Duas heurísticas possíveis são:

1. Utilizar a primitiva `in` passando uma expressão lambda que retorne apenas as tuplas com o valor nulo, chama novamente a primitiva `in` para verificar se o valor do índice anterior já foi calculado e chama também para verificar o índice precedente ao anterior já foi calculado. Caso estes valores estejam nulos, são realizados primeiro os seus cálculos, fazendo sempre esta checagem recursivamente, até que seja possível calcular o valor real desejado. Com o detalhe de que cada vez que a primitiva `in` retira uma tupla do ambiente compartilhado e calcula o valor, a primitiva `out` é chamada para escrever o valor que foi processado;
2. Utilizar a primitiva `in` passando uma expressão lambda que retorne apenas as tuplas com o valor nulo, chamar a primitiva `read` para verificar se o valor anterior e precedente ao anterior já foram calculados. Caso estes valores não tenham sido processados ainda, é chamado o método de cálculo, porém estes valores não são escritos no ambiente compartilhado. O único valor que será escrito é o valor real que deseja ser calculado.

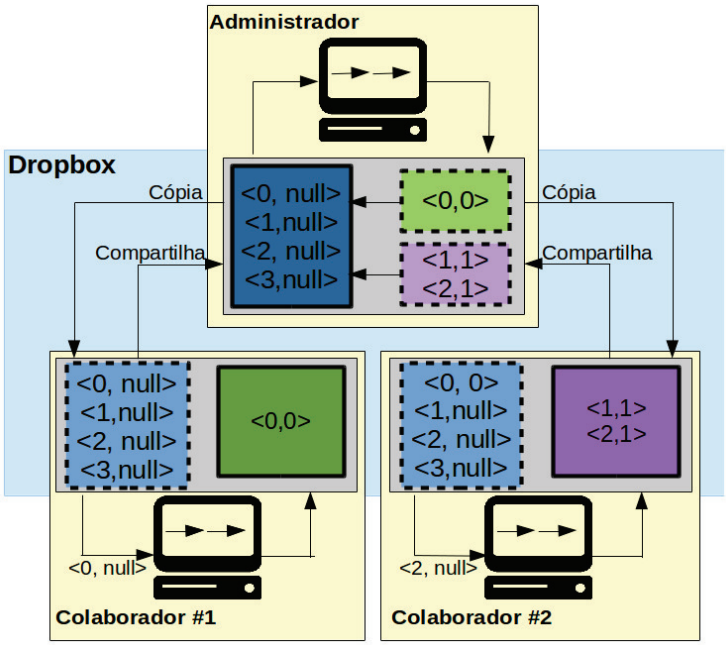

Figura 1. Exemplo da heurística 1.

Na Figura 1 é apresentado um esquema representando uma instância do Projeto Colaborativo proposto. Nela o Administrador disponibiliza, em seu espaço no Dropbox, valores para serem calculados da série de Fibonacci e os Colaboradores consomem estes dados, produzindo os valores das posições necessárias segundo heurísticas próprias.

3. Análise de desempenho

Foram feitas cinco execuções para cada uma das propostas de colaboração, e em cada execução foram calculados os valores da série de Fibonacci de zero até cem. Cada heurística obteve um tempo de execução diferente. Os dados coletados foram registrados de acordo com as propostas sendo executadas em dois sítios. Com os resultados, foram registrados alguns dados como tempo médio (μ) e desvio padrão (δ). No caso, os tempos foram elevados pois foi executado um método recursivo que exige muito processamento, logo, os resultados são tempos mais demorados.

Local	H#1		H#2	
-	μ	δ	μ	δ
Sítio#1	203.19min	3,46	170,70min	4,02min
Sítio#2	207.35min	1,38	169,90min	4,36min

4. Conclusão e trabalhos futuros

Com a coleta dos dados, é possível perceber que ocorre uma diferença de tempo de execução entre as heurísticas. Os tempos são elevados pois a execução se deu de forma sequencial, sem que houvesse a distribuição do trabalho entre colaboradores. Contudo, a ferramenta permite o compartilhamento de processamento e custos, então futuramente serão realizados testes onde mais de um sítio executa ao mesmo tempo, a fim de melhorar o desempenho e conseguir compartilhar os custos, gerando novos resultados. Com isto, há uma previsão que os tempos serão diminuídos, mas para isso são necessários mais estudos de casos e testes em situações de colaboração. No atual estágio de desenvolvimento, o objetivo é entender a operacionalidade do ambiente.

Novos testes serão desenvolvidos de forma a tirar mais vantagens de futuras versões da biblioteca ILUCTUS. Como novos estudos de caso, pretende-se abordar o problema com mais heuristicas e paralelamente e com um número maior de execuções o que ampliará a visão do funcionamento dos projetos colaborativos nesta nova abordagem de colaboração.

Referências

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

[Arnold et al. 2000] Arnold, K., Gosling, J., and Holmes, D. (2000). *The Java programming language*, volume 2. Addison-wesley Reading.

[Bretana et al. 2017] Bretana, L. E., Schwendler, A., and Cavalheiro, G. G. H. (2017). Iluctus: An implementation of linda using cloud as tuple space.

[Navaux 1989] Navaux, P. O. (1989). Introdução ao processamento paralelo. *RBC-Revista Brasileira de Computação*, 5(2):31–43.