# ILUCTUS: Uma Biblioteca para o Apoio ao Processamento Colaborativo de Dados

**Lucas Eduardo Bretana [*], Alana Schwendler [†], Gerson Geraldo H. Cavalheiro**

[1]Laboratory of Ubiquitous and Parallel Systems
Programa de Pós-Graduação em Computação
Universidade Federal de Pelotas
Pelotas, RS – Brasil

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

***Resumo.*** *Evoluir grandes volumes de dados requer sítios com capacidade de processamento e armazenamento de informação. O processamento distribuído utiliza tecnologias que permitem o compartilhamento de recursos e custos de processamento, e necessitam de ferramentas que façam a comunicação entre aplicação e 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 e aplicações.*

## 1. Introdução

Este trabalho está relacionado à manipulação, produção e compartilhamento de grandes volumes de dados. Vários estudos e pesquisas que trabalham manipulando grandes volumes de dados necessitam do processamento distribuído e compartilhamento de custos para a obtenção de resultados conclusivos. A demanda por sítios capazes de produzir e armazenar dados é constantemente notada no meio científico e acadêmico. O surgimento do protocolo HTTP (*The Hypertext Transfer Protocol*) advindo da necessidade do CERN (*Conseil Européen pour la Recherche Nucléaire*) [Berners-Lee et al. 1994] em armazenar dados gerados por pesquisas e que fiquem disponíveis para processamento é um exemplo histórico desta situação.

A demanda por processamento de grandes quantidades de dados pode ser ressaltada, também, no contexto de *Open Data*[Ebrahim and Irani 2005], onde bases de dados públicas são disponibilizadas na Internet. O objetivo é o compartilhamento destes dados para que a comunidade os manipule de forma a gerar resultados e, portanto, novos conhecimentos. No entanto, as políticas usualmente aplicadas nestas bases de dados não preveem que o resultado dos processamentos utilizados sobre os dados disponibilizados sejam agregados a estas. Como consequência, os resultados podem ter um impacto menor que o desejado no crescimento do conhecimento além de haver o claro risco de duplicidade de uso de recursos de processamento quando outra pesquisa/pesquisador visa obter o mesmo resultado, reproduzindo o mesmo trabalho.

[*]Bolsista IC Capes
[†]Bolsista IC Capes

Neste artigo é apresentada uma alternativa à gestão de bases de dados *Open Data*. A contribuição central é permitir que a oferta de dados receba como contrapartida dos pesquisadores beneficiados, os resultados obtidos, permitindo o crescimento da própria base. Outro ponto a ser considerado é que os custos de armazenamento são compartilhados entre os pesquisadores que estão utilizando a base de dados para evolui-la. A proposta segue a linha de pensamento das licenças como o *Creative Commons* [Commons 2016] e *GNU General Public License* (GPL) [Gough 2009] para evolução de dados por uma comunidade de interesse. A proposta oferece um modelo de negócio entre o proprietário da base de dados e seus usuários, que reza que os dados obtidos a partir do processamento sobre os dados originais também serão compartilhados entre todos aqueles que também fazem, ou venham a fazer, uso da base de dados original.

A partir de estudos sobre os trabalhos relacionados como licenças de compartilhamento, entre outros, tornou-se possível o desenvolvimento de uma biblioteca que explora os recursos de nuvem [Armbrust et al. 2010] utilizando o modelo Espaço de Tuplas (TS, do inglês *Tuple Space*) para coordenação e comunicação entre as atividades.

A utilização de nuvem em união com o conceito de Espaço de Tuplas permite construir um modelo para garantir evolução de dados de maneira distribuída. Um dos benefícios que podem ser obtidos é a colaboratividade na evolução de dados em uma aplicação com uma larga escala de informações que necessitam ser processadas. Uma vez que exista uma aplicação com a necessidade de processamento distribuído, ela pode ser compartilhada num espaço em nuvem onde existirão colaboradores intencionados a evoluir os dados da pesquisa para conseguir obter os resultados desejados, dividindo custos de processamento e, eventualmente, de armazenamento. Dentre exemplos de aplicações que fariam uso desse modelo de programação, citam-se o treinamento de redes neurais, cálculo do escavamento de túneis e sequenciamento do DNA.

O restante do artigo está organizado como segue. A próxima seção, seção 2, caracteriza o modelo de negócio concebido para desenvolver o trabalho, considerando o contexto de compartilhamento de informações. Seguidamente, a seção 3 descreve como se deu a implementação da biblioteca sugerida e relata quais ferramentas foram utilizadas em 3.1. Posteriormente, a seção 4 fala sobre o ambiente de armazenamento e compartilhamento dos dados da aplicação. Por fim, seguem as seções 5 que descreve alguns testes que foram realizados para aferir o funcionamento deste trabalho, a seção 6 traz alguns trabalhos e pesquisas semelhantes à apresentada neste artigo e na seção final são apresentadas as conclusões e trabalhos futuros.

## 2. Modelo de negócio

O modelo de negócio identificado se desenvolve no contexto do processamento e armazenamento colaborativo de grandes quantidades de dados gerados por uma pesquisa e que serão manipulados por uma comunidade de interesse em comum. O modelo de aplicação concebido considera a existência de um ambiente de pesquisa sob o qual são desenvolvidos **Projetos Colaborativos**. Um Projeto Colaborativo requer um ambiente em nuvem que proveja recursos para armazenamento de dados, políticas próprias para autentificação de usuários e compartilhamento de dados e ainda uma interface de programação aplicativa (API) com primitivas que permitam a manipulação dos arquivos na nuvem.

Um Projeto Colaborativo é instanciado por um ator do modelo de negócios iden-

tificado como **Administrador**. Este Administrador, além de criar o Projeto Colaborativo, também disponibiliza o primeiro conjunto de dados, o qual será operado segundo as **Políticas de Colaboração** do sistema. São estas políticas que definem as regras de como os **Colaboradores**, outros atores do modelo, irão participar do projeto acessando os dados compartilhados e oferecendo acesso aos resultados por eles próprios produzidos.

Políticas de Colaboração consistem em um conjunto de regras que definem as relações de compartilhamento de informação entre os participantes bem como operações de compartilhamento dos dados entre os diferentes atores que compõem o projeto. As regras iniciais destas políticas definem como se dá o processo de autentificação na aplicação, e para isso é necessária a troca de *tokens* de identificação. Da parte do Administrador, esses *tokens*, ou chaves, identificam o projeto e autorizam uma aplicação a ter acesso aos dados compartilhados na nuvem. *Tokens* são únicos para cada Colaborador em um Projeto Colaborativo. Pelo lado do Colaborador, o *token* identifica-o dentro do projeto, oferecendo, desta forma, acesso aos seus resultados na nuvem. Nesta troca inicial de informações entre o Administrador e Colaborador, além dos *tokens*, também é feita a troca de quaisquer outras informações necessárias para dar inicio ao processamento dos dados, tais como identificação da base de dados primária, lista de bases já processadas por Colaboradores pré-existentes, permissões sobre estas bases etc.

Além dos processos iniciais de troca de informações, as Políticas de Colaboração também rege o acesso a uma base de dados que já compõe o Projeto, protegendo o sistema de ter as bases de dados alteradas inadvertidamente por qualquer Colaborador. A proposta não inclui sistema de recuperação, considerando que cada Colaborador é responsável pelos dados gerados pela sua contribuição e que este não tem acesso destrutivo aos dados produzido pelos demais atores.

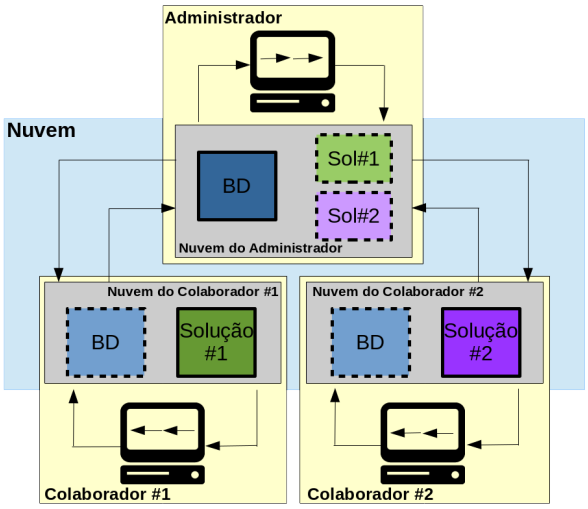


**Figura 1. Ilustração do modelo de negócio.**

A Figura 1 ilustra o modelo de comunicação entre Administrador e Colaboradores no ambiente de desenvolvimento de Projetos Colaborativos concebidos. Nas bases de dados (BD), a borda contínua representa a base de dados original na nuvem onde foi primeiro formada. O compartilhamento é representado pelas setas entre as nuvens de origem e destino. O conjunto destas BDs representa a nuvem de dados manipulada pelo

Projeto Colaborativo, oferecendo amplo acesso a toda coleção de dados, sem distinção se o dado foi produzido pelo Administrador ou algum Colaborador.

São considerados dois cenários de desenvolvimento de um Projeto Colaborativo. O primeiro é caracterizado como um **Projeto Fechado**, no qual o Administrador disponibiliza uma **aplicação**, que já inclui as Políticas de Colaboração e automatiza os processos de autentificação. Tal aplicação é fechada, e aplica uma determinada heurística para o processamento dos dados, e tem por objetivo explorar o poder de processamento e de armazenamento disponibilizado pelos Colaboradores. O processamento desta aplicação se dá nos recursos próprios aos Colaboradores, sendo, então, na nuvem, disponibilizados os resultados do processamento realizados.

O segundo cenário é descrito por um **Projeto Aberto**. Neste cenário, o Colaborador não apenas oferece seus recursos de processamento, mas também sua *expertise* no tema, oferecendo novas heurísticas, na forma de aplicações por ele próprio desenvolvidas, para manipular os dados. Em um Projeto Aberto, os *tokens* de autentificação recebidos pelo Colaborador permitem que sua aplicação seja conectada a um Projeto Colaborativo.

## 3. Contexto de Implementação

Para o desenvolvimento deste trabalho foi necessário o estudo de meios comunicação que servissem de apoio para o compartilhamento de dados de modo distribuído entre diferentes sítios de processamento. Outra necessidade era definir uma tecnologia de nuvem que fornecesse a mecânica básica de comunicação por meio de uma API, bem como documentação clara e completa do uso e funcionamento dessa API. Por último, uma linguagem de programação que pudesse fazer uso desta API e fornecesse o necessário para desenvolver o modelo de comunicação escolhido.

### 3.1. Ferramentas

Uma das necessidades encontradas era definir um mecanismo de comunicação que permitisse a colaboração de tarefas de forma distribuída. Exemplos de mecanismos possíveis para essa comunicação são troca de mensagem [Cáceres et al. 2001], RMI [ORACLE 2016], DSM [Yu and Cox 1997], Espaço de Tuplas (TS, do inglês *Tuple Space*). O modelo de comunicação escolhido foi o Espaço de Tuplas, que, apesar de apresentar altos sobrecustos de comunicação nos ambientes para processamento distribuído utilizados anteriormente ao advento da tecnologia de nuvem, oferece uma camada de abstração para colaboração com uma semântica bastante clara. As questões de desempenho, em um ambiente distribuído de larga escala, diluem-se com os benefícios obtidos pela camada de desenvolvimento.

Um TS consiste basicamente de um espaço de endereçamento compartilhado onde os dados são manipulados como tuplas que seguem o formato <*key*, *content*>. O campo *key* de uma tupla é um identificador e deve ser único dentro do TS. O *content* é o dado da tupla em si, sendo este um dado de tipo genérico. O Espaço de Tuplas prevê a existência de operações simples de leitura, escrita e remoção para manipulação das tuplas. Estas operações devem então ser concretizadas pelas implementações de TS. A escolha do Espaço de Tuplas como o modelo a ser utilizado se deu principalmente por ter uma semântica conhecida e que está apta para o modelo de memória distribuída.

Dentre os modelos que descrevem uma implementação de Espaço de Tuplas o escolhido foi o modelo de Linda [Ahuja et al. 1986]. A escolha se deu pela simplicidade com que os métodos de comunicação descritos por esse modelo se adaptam à solução do problema de compartilhamento de dados. Os meios para manipulação destas tuplas são por operações básicas de de manipulação dos dados. É importante notar que o Linda é um modelo e por isso ao ser implementado pode-se apresentar outras primitivas de comunicação. Linda define quatro primitivas de interação com os dados, sendo elas:

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.

Para a escolha da nuvem se levou em consideração aquelas que dispõem de uma API para gerenciamento de autentificação e manipulação dos dados. Outra característica importante era facilidade da aquisição de uma nuvem particular por um usuário comum, aumentando o número de Colaboradores em potencial. Dentre as nuvens que atendiam aos requisitos necessários a escolha foi pela nuvem do Dropbox [Drago et al. 2012]. Esta decisão se deu pois, na ocasião da implementação, foi aquela que apresentou um processo de autentificação robusto e a melhor documentação sobre a API.

A opção por Java se deu considerando tanto a popularidade desta como pela portabilidade oferecida pelo seu modelo de implementação baseado em máquina virtual. Outro aspecto positivo é a existência de uma API do Dropbox desenvolvida para esta linguagem.

### 3.2. Implementação

A implementação do modelo de negócio descrito foi feita no formato de uma biblioteca que ganhou o nome de ILUCTUS. Esta biblioteca permite que sejam desenvolvidos Projetos que utilizem a nuvem do Dropbox como memória compartilhada, no formato de Espaço de Tuplas, para o processamento distribuído de dados. Para isso, se desenvolveu métodos para manipular os dados usando operações de leitura, escrita e remoção de tuplas. A ILUCTUS concretiza essas operações em primitivas baseadas nas descritas pelo modelo de Linda.

Para a criação das primitivas foi definida a representação de uma tupla dentro do sistema de arquivos oferecido pelo Dropbox. Para este trabalho definiu-se que o campo *key* de uma tupla é usado como o nome do arquivo que contém os dados referentes àquela tupla. O campo de dados da tupla, *content*, é o conteúdo do arquivo no sistema de arquivos do Dropbox. A escrita e leitura dos objetos Java para arquivos é feita usando recursos internos da Máquina Virtual Java (*Java Virtual Machine* ou JVM). Para isso basta que o tipo do campo *content* implemente a interface *Serializable* da API Java. Usando as ferramentas da JVM um objeto Java pode ser convertido para uma *stream* de *bytes*, o que torna trivial a sua escrita em arquivo. É importante ressaltar que operação inversa também existe e é correspondente, operando uma *stream* de *bytes* válida a JVM retorna um objeto Java.

Nesta biblioteca o meio pelo qual é feita a identificação de uma tupla é utilizando o recurso de expressões *lambda*. Dentro do sistema de Java uma expressão lambda é tipada sobre um Interface que recebe a anotação de *FunctionalInterface* e nesta Interface

se generaliza o funcionamento da expressão *lambda*. Para a expressão *lambda* que faz a identificação de uma tupla dentro do TS a Interface foi desenvolvida de forma a conter um método que recebe como parâmetro uma tupla e retorna um valor *boolean*. O funcionamento do procedimento que identifica a tupla (*match*) fica abstraído e deve ser concretizado em cada uso das primitivas que fazem busca no TS.

As operações de leitura do TS são realizadas pelas primitivas `Read` e `In`. `Read` faz uma busca dentro do TS utilizando uma expressão *lambda* recebido como argumento para identificar a tupla desejada. A implementação do `Read` é não bloqueante, desta forma a primitiva retorna a primeira tupla que for identificada, ou *null* em caso do dados buscado não ser encontrado. O funcionamento da primitiva `In` é semelhante à `Read`, porém uma vez que a tupla é encontrada a sua referência dentro do TS é removida.

As operações para escrita no TS são as primitivas `Out` e `Eval`. A implementação da primitiva `Out` recebe uma tupla como argumento e insere esta tupla para o TS. Para esta implementação a escolha foi por uma abordagem destrutiva, onde se houver uma ocorrência da mesma chave no TS, o dado correspondente será sobrescrito. A primitiva `Eval` recebe uma expressão *lambda* para fazer a identificação de uma tupla dentro do TS e sobre esta tupla é aplicada uma função para modificar o seu estado. A função a ser aplicada pela primitiva também é recebida por argumento como uma expressão *lambda*. Esta *lambda* é tipada sobre uma interface que generaliza um método que tem como parâmetro uma tupla e que retorna, também, uma tupla. Uma vez que tupla é identificada e processada, ela é atribuída novamente ao Espaço de Tuplas usando o mesmo identificador da tupla original.

## 4. Armazenagem e compartilhamento

A Figura 2 ilustra como ocorrem os processos de armazenagem e compartilhamento das bases de dados. Nesta figura, as bases de dados são representadas com borda pontilhada ou contínua, referenciando respectivamente, à base na sua nuvem original ou uma cópia proveniente do compartilhamento. Já os processos de compartilhamento são ilustrados pelas ações **Requisita acesso**, onde é feita a requisição de uma base de dados, e **Concede acesso**, onde o acesso é garantindo e então a base é copiada para a nuvem do requisitante.

O processamento distribuído dos dados ocorre quando existe o compartilhamento das bases de dados que são então processadas separadamente. Neste compartilhamento é feita a cópia dos dados requisitados para a nuvem do Colaborador. A base de dados que fornece os materiais para processamento fica na nuvem do Colaborador até o fim da execução da aplicação e então é removida, de forma a não onerar sobrecustos de armazenagem, uma vez que o programa terminou. Na figura isso fica claro ao notar a cópia da base de dados *init* na nuvem dos Colaboradores.

Por outro lado a base de dados criada a partir dos resultados da computação no Colaborador existe unicamente na nuvem deste Colaborador até que a aplicação termine. Desta forma temos, além do processamento distribuído, a divisão dos custos de armazenamento dos dados. Na figura, essa divisão dos custos é expressa pela ausência da base de dados *Solução 2* na nuvem do Administrador pois o Colaborador 2 ainda não terminou a execução e desta forma seus resultados não estão compartilhados.

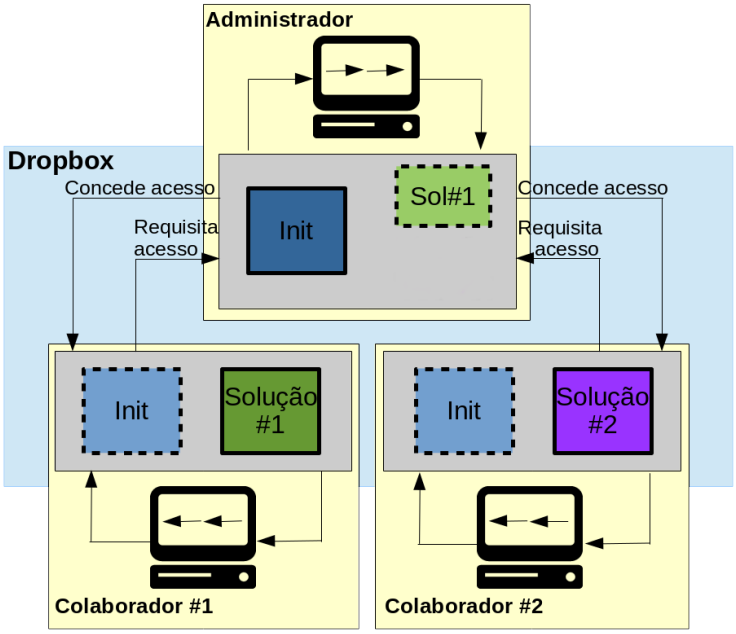


**Figura 2. Distribuição do armazenamento e fluxo de comunicação para compartilhamento de BDs.**

## 5. Caso de Teste

Para fazer a verificação da funcionalidade desta biblioteca apresentada no artigo, foram realizados dois tipos de testes. No primeiro caso, 5.1, foram feitas execuções repetidas das operações básicas com as quais a biblioteca opera, de forma a se obter os tempos médios de execução das mesmas. Em um segundo momento, 5.2 foi realizado um caso de teste para estressar a aplicação, utilizando a biblioteca desenvolvida em duas abordagens diferentes. Neste caso de teste também foi avaliado o tempo médio de execução com o desvio padrão.

### 5.1. Teste das primitivas

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 |
| $\mu$ | 1.09s | 3.78s | 16.3s | 32.33s | 7.27s | 16.29s | 33.15s |
| $\delta$ | 0.36s | 0.37s | 0.65s | 1.57s | 0.77s | 0.49s | 0.99s |

Deve ser levado em consideração que o Dropbox exige um tempo de espera entre requisições feitas pela API e que os resultados apresentados na Tabela 1 são referentes à 30 execuções de cada primitiva feitas a partir de computador na rede da UFPel. É possível observar que as primitivas de leitura, i.e., `Read` e `In`, tem um tempo médio maior que a primitiva `Out`. Isto coincide com o esperado, uma vez que na leitura é feito um número maior de requisições para a nuvem do Dropbox.

### 5.2. Caso de teste Fibonacci

Para representar a produção de informação, 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 calcular de maneira mais fiel o desempenho da aplicação. Com as execuções feitas, foram coletados os tempos em que os cálculos foram realizados.

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

Na execução dos casos de teste, a chave da tupla representa a posição da série de Fibonacci desejada, e o conteúdo da tupla é o respectivo valor da posição identificada na série. Se a tupla estiver apenas com a identificação preenchida e seu conteúdo estiver nulo, significa que seu valor ainda não foi calculado e que ela está disponível para o processamento. Caso o valor da tupla esteja preenchido, a posição da série correspondente àquele valor já foi calculada e disponibilizada para o resto da aplicação.

Inicialmente o Administrador disponibiliza diversos valores a serem calculados utilizando da primitiva `Out` da biblioteca. Uma vez que temos a base inicial, agora um Colaborador, ou o próprio Aministrador, utiliza de uma aplicação que fará o calculo para realizar as tarefas que estão no TS. Dentro das execuções do caso de teste foram propostas duas heurísticas:

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, i.e., verifica se os valores anteriores de Fibonacci já estão calculados. Caso estes valores estejam nulos, i.e. ainda não foram calculados, 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 ou o mesmo valor, caso a aquela estivesse com o seu valor calculado;
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.

As duas heurísticas listadas foram implementadas e os resultados obtidos podem ser observados na tabela 2. Na tabela, o tempo médio é representado pelo símbolo *mu* ($\mu$) e o desvio padrão é representado pelo símbolo *delta* ($\delta$). Percebe-se que a segunda heurística obteve melhor desempenho que a primeira heurística, já que é realizada a escrita do resultado apenas para o valor da série que é desejado, e não de todos os valores calculados como faz a primeira heurística. Para os dados apresentados na tabela deve-se considerar que o Dropbox exige um tempo mínimo entre as requisições e que os presentes resultados correspondem a média de 10 chamadas a cada primitiva a partir de um computador localizado na rede da UFPel.

**Tabela 2. Tempos de execução das heurísticas**

| Local | H#1 | | H#2 | |
|---|---|---|---|---|
| - | $\mu$ | $\delta$ | $\mu$ | $\delta$ |
| Sítio#1 | 203.19min | 3.46 | 170.70min | 4.02 |
| Sítio#2 | 207.35min | 1.38 | 169.90min | 4.36 |

Como foi mencionando anteriormente as primitivas retornam a primeira tupla cujo a expressão *lambda* faça *match* e a ordem na qual elas são verificas depende da organização dos arquivos dentro da nuvem. Na execução destes testes verificou-se que a ordem na qual a nuvem coloca os arquivos é de acordo com o *timestamp* da criação. Por isso as primeiras tuplas a serem verificadas foram as com maior índice na sequencia Fibonacci e desta forma a primeira heurística apresenta um sobre custo de reescrever as tuplas que já foram calculadas, o que explica porque seus tempo médio foi maior.

## 6. Trabalhos Relacionados

Após uma pesquisa bibliográfica sobre o assunto foram encontradas trabalhos muito bem consolidados que serviram de embasamento para o desenvolvimento da proposta apresentada, destacando-se o Seti@Home, BOINC e o modelo EvoSpace. Vale destacar que uma das dificuldades deste trabalho foi encontrar trabalhos relacionados mais recentes.

O Seti@Home [Korpela et al. 2001] é uma tecnologia desenvolvida para realizar processamento distribuído de informações coletadas em sua pesquisa. Esta pesquisa é voltada para coleta de dados vindos do espaço. Esta tecnologia tem o seguinte funcionamento: o usuário que desejar ceder seu poder de processamento para a aplicação pode se cadastrar como colaborador, e sempre que seu computador estiver ocioso, alguns dados serão enviados para o seu sítio de processamento e a aplicação vai ser executada em segundo plano, enquanto houver processamento disponível para evoluir os seus dados. Uma vez que respostas são obtidas, elas são enviadas para a aplicação original, para seguir com o desenvolvimento dos dados da base original. O processo que está sendo executado no sítio de processamento fica como uma espécie de proteção de tela, permitindo que todo o poder de processamento ocioso se foque em trabalhar em cima da aplicação [Anderson et al. 2002]. Neste sentido, a base inicial da ILUCTUS foi dada pelo Seti@Home, pela sua ideia de processar os dados de maneira distribuída com vários colaboradores possíveis para a mesma aplicação. A biblioteca ILUCTUS se diferencia apresentando um modelo de Projeto Aberto onde que cada Colaborador pode aplicar sua própria heurística de evolução.

BOINC (Berkeley Open Infrastructure for Network Computing) [Anderson 2004] é uma ferramenta de gerenciamento de aplicações distribuídas, desenvolvida na Universidade de Berkeley - Califórnia. Assim como o Seti@Home, existem outras aplicações que fazem o mesmo tipo de processamento de dados porém com outros focos de pesquisa. O BOINC realiza uma espécie de ponte entre a aplicação e o sítio de colaboração. A partir dele, é possível escolher para qual aplicação o poder de processamento será cedido. Em união com a ideia do Seti@Home, o BOINC é uma interface de gerenciamento de aplicações distribuídos, o que baseia interface planejada para esta aplicação.

O modelo EvoSpace é uma plataforma evolutiva e distribuída baseada no modelo de Espaço de Tuplas [García-Valdez et al. 2013]. Esta plataforma é utilizada para o armazenamento de algoritmos evolutivos, utilizando o recurso da nuvem para isto, podendo ser utilizado como modelo de *Platform as a Service* (PaaS). Este modelo também trata de alguns problemas como redundâncias de trabalho, *starvation, starvation of the population pool*, insegurança de usuários conectados e um grande espaço de parâmetros.

Uma das aplicações de uso citadas é a de *Open Data*. As bases de dados abertas operam de maneira a serem preenchidas a partir de pesquisas de campo [Bouguettaya et al. 2001], onde os dados são coletados através de analises de comunidades precárias, censos, pesquisas demográficas etc. Pesquisas desta natureza geram grandes volumes de dados e, geralmente, precisam ser processadas para gerar estatísticas e resultados para trazer a melhoria das comunidades estudadas. Estas bases de dados são carregadas com quantidades exorbitantes de dados que precisam ser processados. Já que os dados estão disponíveis abertamente, podem ser usados por vários colaboradores para que os resultados necessários sejam alcançados.

A proposta deste artigo prevê a Colaboração e compartilhamento de custos de armazenamento em mais de um sítio. Estes trabalhos apresentam funcionalidades características que podem ser incorporadas à ILUCTUS. A proposta deste artigo utiliza os mesmos conceitos mas com tecnologias mais atuais, como o ambiente de nuvem servindo de ambiente de compartilhamento e a linguagem de programação Java em uma versão com mais funcionalidades agregadas.

## 7. Conclusões e trabalhos futuros

Neste trabalho foi apresentada ILUCTUS, uma biblioteca que permite a competência no processamento de dados em larga escala. Esta ferramenta foi concebida para atender uma crescente demanda do uso de nuvens computacionais para provisionamento de recursos para armazenamento de grandes coleções de dados. ILUCTUS foi implementada de forma a explorar a tecnologia de nuvem provida pelo DROPBOX como um Espaço de Tuplas distribuído em larga escala. Uma política de controle de acesso garante a integridade da base de dados construída na forma de um Projeto Colaborativo, no qual os papéis dos atores, Administrador e Colaborador, são claramente definidos. Nos cenários projetados, a adesão de Colaboradores aos Projetos Colaborativos permite racionalizar o uso de recursos computacionais, além de repartir custos de processamento e armazenamento dos dados.

Alternativas desta natureza de processamento distribuído significam um avanço científico, pois ocorre o compartilhamento de custos de processamento e armazenamento. A biblioteca desenvolvida neste trabalho traz estas possibilidades de compartilhamento de

custos. As tecnologias estão em evolução e, com isto, é dada a motivação para explorar os serviços de nuvem como um meio para compartilhar dados de pesquisas, garantindo com que novos avanços tecnológicos e pesquisas se beneficiem destes serviços.

Para trabalhos futuros serão desenvolvidos variações das primitivas. Podemos destacar a promoção de escritas não destrutivas que podem tirar vantagem do sistema de versionamento de arquivos disponibilizado pelo Dropbox. Além desta, tem-se a possibilidade de implementar leituras que retornam todas as tuplas que satisfazem os requisitos de assimilação e leitura bloqueantes. Além dos avanços para novas primitivas pretende-se desenvolver novas aplicações que usufruam da biblioteca para evolução de dados, tornando-se melhores métricas de avaliação da biblioteca.

## 8. Agradecimentos

Agradecemos à Universidade Federal de Pelotas (UFPel), aos órgãos de fomento CNPq e Fapergs e ao grupo de pesquisa LUPS.

## Referências

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

Anderson, D. P. (2004). Boinc: A system for public-resource computing and storage. pages 4–10.

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.

Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., et al. (2010). A view of cloud computing. *Communications of the ACM*, 53(4):50–58.

Berners-Lee, T., Dimitroyannis, D., Mallinckrodt, A. J., McKay, S., et al. (1994). World wide web. *Computers in Physics*, 8(3):298–299.

Bouguettaya, A., Ouzzani, M., Medjahed, B., and Cameron, J. (2001). Managing government databases. *Computer*, 34(2):56–64.

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.

Commons, C. (2016). Creative commons.

Drago, I., Mellia, M., M Munafo, M., Sperotto, A., Sadre, R., and Pras, A. (2012). Inside dropbox: understanding personal cloud storage services. pages 481–494.

Ebrahim, Z. and Irani, Z. (2005). E-government adoption: architecture and barriers. *Business process management journal*, 11(5):589–611.

García-Valdez, M., Trujillo, L., de Vega, F. F., Guervós, J. J. M., and Olague, G. (2013). Evospace: a distributed evolutionary platform based on the tuple space model. pages 499–508.

Gough, B. (2009). *GNU scientific library reference manual*. Network Theory Ltd.

Korpela, E., Werthimer, D., Anderson, D., Cobb, J., and Lebofsky, M. (2001). Seti@ home—massively distributed computing for seti. *Computing in science & engineering*, 3(1):78–83.

ORACLE (2016). Trail: Rmi. java documentation. tutorials. Disponível em: <http://docs.oracle.com/javase/tuorial/rmi>. Acesso em: Julho de 2017.

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