FEUP – FACULDADE DE ENGENHERIA DA UNIVERSIDADE DO PORTO

Gestão de jobs em "Grid Computing"

Algoritmos e Estruturas de Dados

Duarte Duarte - 110509101 - ei11101@fe.up.pt
Hugo Freixo - 110509086 - ei11086@fe.up.pt
Miguel Marques - 110509099 - ei11099@fe.up.pt

31 de dezembro de 2012

Turma 1 – Grupo 5 – 2MIEIC

Índice

Índice 1
O Grid Network 2
Solução Implementada 3
Diagrama de classes (UML) 4
Casos de Utilização 5
Dificuldades 6
Distribuição do trabalho pelos elementos do grupo 6

O Grid Network

O **GridNetwork** é um conjunto de redes de computação compostas por várias máquinas de diferentes características e de vários utilizadores.

Cada rede tem um **GridManager** que é o objeto que gere todos os utilizadores, máquinas e trabalhos dessa rede.

Os utilizadores podem ser utilizadores académicos, que têm um contador de trabalhos executados, ou utilizadores de empresas, que têm um orçamento para trabalhos.

Estes utilizadores podem criar novos trabalhos (*jobs*) que serão executados num determinado tempo especificado pelo utilizador que o criou.

Os trabalhos submetidos são colocados numa fila de espera até que uma máquina esteja disponível para os executar. Devido a uma arquitetura *multithreading* do programa, é possível submeter trabalhos por parte do utilizador enquanto que outros trabalhos são executados ou transferidos da fila de espera para as máquinas, tudo em simultâneo.

Existem máquinas de dois tipos: máquinas que respeitam a prioridade dos trabalhos (**Priority Machines**) e máquinas que não respeitam a prioridade dos trabalhos (**Machines**).

Os utilizadores que sejam eliminados de uma rede de computação são colocados numa tabela de dispersão (**IdleUserContainer**) na qual permanecem durante um certo tempo.

Solução Implementada

O **Grid Network** foi implementado através de uma árvore binária de pesquisa de **Grid** que organiza as grids por nome.

Cada **Grid** contém, além do nome, um tópico e um gestor.

O gestor, que também funciona como simulador, usa árvores binárias de pesquisa para organizar os seus utilizadores, máquinas e máquina com prioridade, árvores essas que ordenam por um *id* único em todas as redes.

Estando tudo integrado na mesma aplicação à medida que se vão adicionando mais trabalhos (**Job**), máquinas (**Machines**), máquinas com prioridade (**Priority Machines**) e utilizadores (**User**) o gestor (**GridManager**) vai atualizando os trabalhos, isto é, tendo em conta o tempo especificado aquando da criação de um novo trabalho o simulador irá apagar esse trabalho quando esse tempo tiver passado, tempo esse que é medido em segundos e em tempo real. Os utilizadores que forem eliminados de uma rede também serão adicionados a uma tabela de dispersão e serão retirados da tabela após um certo tempo.

O grupo recorreu a *templates* para a implementação das Máquinas, tendo sida criada um classe que recebe como argumento genérico um contentor onde irão ser armazenos os trabalhos. Foram então definidos dois tipos de Máquinas, **Machines** e **Priority Machines**, usando Fila e Fila de Prioridade (*queue* e *priority_queue*), respetivamente. O grupo decidiu que, para filtrar os trabalhos que um utilizador cria, todos os trabalhos cuja prioridade seja diferente de 0 serão adicionados a máquinas com prioridade, sendo os restantes adicionados a máquinas normais.

Para o gestor foi implementada uma estrutura de dados em que todas as classes são derivadas de interfaces (classes puras) que definem algumas funcionalidades básicas. São elas:

- **IPrint** - Imprimir para uma stream os dados de um objeto;
- **ISave** - Guardar num ficheiro binário os dados de um objeto.

Para o simulador foram implementadas duas interfaces:

- **IUpdate** - Atualizar os dados de um objeto com base numa diferença temporal;
- **Runnable** - Objeto que pode executar algum trabalho num *thread* separado.

O grupo implementou um *logger* para manter o utilizador atualizado acerca do sistema, bem como para fazer *debugging* de algumas funcionalidades implementadas.

Para gravar os dados da sessão optou-se pela utilização de ficheiros binários, visto que são mais compactos e menos propensos a erros de leitura, e para a sua leitura e escrita implementou-se uma classe (**ByteBuffer**) que lê e escreve, para um determinado ficheiro, diversos tipos de dados.

A interface do utilizador, toda ela feita através da consola, foi maioritariamente feita com recurso a menus (classes **IMenu**, **Menu** e **Menu::Item**) lidos de ficheiros de texto e que após o utilizador selecionar um **Menu::Item** retornam um valor inteiro que permite tratar (*handling)* a sua opção.

No decurso do desenvolvimento deste trabalho recorremos a funcionalidades do C++11, nomeadamente *lambdas*, *std::regex*, *std::thread*, *std::tuple* e outras, que permitem manter o código conciso e compacto.

Também de notar que não foram necessárias alterações fundamentais ao código entre a 1ª parte do projeto e a 2ª, em parte devido a uma razoável modularização do código.

Diagrama de classes (UML)

(Ver ficheiro uml.png em anexo.)

Casos de Utilização

- Adicionar Grids ao Grid Network
- Remover Grids do Grid Network
- Alterar nome das Grids
- Alterar tópico das Grids
- Procurar Grids por nome ou por tópico
- Listar todas as Grids
- Adicionar utilizadores ao sistema
- Remover utilizadores do sistema
- Alterar nome dos utilizadores
- Alterar orçamento dos utilizadores empresariais
- Procurar utilizadores no sistema através do nome
- Listar todos os utilizadores ou apenas utilizadores académicos ou empresariais
- Procurar utilizadores eliminados do sistema através do nome
- Listar todos os utilizadores eleminados ou apenas utilizadores académicos ou empresariais
- Adicionar máquinas ao sistema
- Remover máquinas do sistema
- Alterar nome das máquinas
- Alterar RAM das máquinas
- Alterar espaço do disco das máquinas
- Adicionar Software às máquinas
- Remover Software das máquinas
- Listar Software instalado numa máquina
- Alterar o número máximo de trabalhos de uma máquina
- Listar os trabalhos em execução numa máquina
- Remover um trabalho de uma máquina
- Remover todos os trabalhos de uma máquina
- Listar o Software necessário de um trabalho
- Listar o Software disponível numa máquina
- Procurar máquinas no sistema através do nome, RAM disponível, espaço do disco disponível ou número de trabalhos
- Listar todas as máquinas
- Criar novos trabalhos
- Procurar trabalhos no sistema através do nome, RAM necessária, espaço do disco necessário, prioridade ou tempo de execução
- Listar todos os trabalhos do sistema
- Alterar a prioridade de um trabalho que esteja armazenado num máquina com prioridade

Dificuldades

Não houve grandes dificuldades no planeamento nem na implementação do trabalho com exceção de alguns entraves na especialização de funções genéricas (com recurso a *templates*) e na execução de funções quando se fecha a consola abruptamente, casos que nos deram mais trabalho a resolver.

Distribuição do trabalho pelos elementos do grupo

O trabalho foi dividido por todos os elementos do grupo.

Todos os elementos ajudaram na implementação das estruturas de dados e da aplicação como também contribuiram com ideias para o projeto e para a interface do utilizador, com maior empenho por parte de Duarte e Miguel.

Tentou-se dividir o trabalho equitativamente por todos os elementos do grupo e pensamos que o resultado final foi bastante positivo.