Facilitar a integração do protótipo AppMan com diversas infraestruturas

Relatório do Projeto de Pesquisa

Rômulo Bandeira Rosinha e Eder Stone Fontoura

Dezembro 2006

Capítulo 1

Introdução

O modelo de gerenciamento hierárquico de aplicações em ambiente de computação em grade GRAND, recebeu uma implementação parcial (protótipo) chamada de Appman. Para a aproximação do protótipo ao modelo idealizado, algumas melhorias ainda são necessárias.

Durante os testes realizados com o protótipo, foram identificadas algumas deficiências do protótipo e do protótipo em relação a seu modelo. Este trabalho procura tratar de dois problemas. O primeiro é a integração do Appman com escalonadores locais, no caso específico deste trabalho o OpenPBS. O segundo trata-se da utilização do protótipo em clusters que não possuem sistema de compartilhamento de arquivos entre seus nodos.

A seção de Introdução apresenta o modelo GRAND, o protótipo Appman e o escalonador local OpenPBS. No capítulo 2 são apresentados os modelos propostos para a solução destes dois problemas. No capítulo 3 é apresentada a aplicação Geneal que foi utilizada nos testes dos modelos propostos. No capítulo 4 apresenta os testes realizados com os modelos propostos. Por fim são apresentadas as conclusões e as sugestões de tranalhos futuros.

1.1 GRAND

O GRAND (Grid Robust Application Deployment) é um modelo de gerenciamento hierárquico de aplicações em ambiente de computação de grade. O modelo considera:

- ambiente heterogêneo;
- grande quantidade de tarefas que podem apresentar dependência através do compartilhamento de arquivos;
- tarefas não se comunicam por troca de mensagens;

- podem ser utilizados arquivos grandes;
- grande número de arquivos pode ser manipulado pelas tarefas;
- cada nodo da grade possui o seu gerenciador de recursos local;
- trata apenas o gerenciamento de arquivos de entrada e saída do processamento (stage in e stage out);

O GRAND possui um gerenciador de aplicação, chamado de *Application Manager* que fica localizado na máquina de onde as tarefas são submetidas (*Submit Machine*). O *Application Manager* ou AM, submete as tarefas a outros gerenciadores, disparados pelo próprio AM ou préviamente inicializados pelo sistema responsável pela escolha da máquinas que executarão as tarefas (Exehda), e aguarda o resultado final da execução das tarefas.

No que diz respeito ao gerenciamento de dados, os dados de entrada são transferidos para o local onde a tarefa será executada, os resultados são enviados de forma controlada ao usuário para evitar congestionamento da rede e o escalonamento prioriza a localidade no disparo de tarefas.

No que diz respeito ao gerenciamento e controle da aplicação, o modelo adota uma hierárquia com os seguintes níveis:

- **nível 0**: usuário submete a aplicação em uma máquina através do *Application Manager*;
- **nível 1**: os *Application Managers* enviam aos *Submission Managers* a descrições das tarefas;
- **nível 2**: os *Task Managers* são instanciados sob demanda pelos *Submission Managers* para controlar a submissão de tarefas aos escalonadores específicos da grade (RMS);
- **nível 3**: os escalonadores específicos recebem as requisições dps *Task Managers* e fazem a execução das tarefas.

Este modelo com uma hierárquia de gerenciadores de submissão acima do meta escalonador, realizando um balanceamento de carga computacional necessário para controlar a execução das tarefas, evitando a sobrecarga de uma máquina de submissão.

1.2 AppMan

O AppMan (Application Manager) [1] é uma implementação simplificada do modelo GRAND. O AppMan é implementado em Java para possibilitar portabilidade. Ele usa a ferramenta JavaCC para analisar gramaticalmente e interpretar a linguagem GRID-ADL. O ambiente de execução do AppMan usa os serviços fornecidos pelo middleware Exehda que possibilitam execução remota e monitoração. O AppMan implementa as principais funções do modelo GRAND, incluindo a submissão distribuída de tarefas e a monitoração da aplicação, fornecendo retorno para o usuário.

Uma instância do AppMan e do Exehda precisa estar presente em cada nó participante da grade. Todos os nós podem submeter e executar tarefas. Os arquivos de dados de entrada devem estar disponíveis em um servidor web. O grafo da aplicação é construído em memória e o Application Manager (AM) é iniciado. O AM particiona o grafo usando um algoritmo de agregação. Então, o AM instancia um ou mais Submission Managers (SM) e distribui sub-grafos para os SMs. Os arquivos de entrada e os executáveis são obtidos do servidor web. Os SMs atualizam o AM com o progresso da execução das tarefas. Cada SM, independentemente, verifica a lista de nós disponíveis e aleatoriamente escolhe um para executar a tarefa. Imediatamente um Task Manager (TM) é instanciado, ficando responsável por criar a tarefa remota e monitorá-la até que a sua execução termine.

Esses são os principais passos na implementação atual. No entanto,uma deficiência do protótipo AppMan é não possibilitar o escalonamento através de gerenciadores de recursos (RMS), como idealizado no GRAND. Nesse cenário mais desejável apresentado na figura 1.1, o TM em um nó da grade contata diretamente o gerenciador de recursos em um nó remoto da grade para executar as tarefas.

Como citado anteriormente, o Exehda e o Appman necessitam estar presentes em todos os nodos participantes da grade. Os nodos principais de cada cluster que compõem a grade são chamados de base. Cada base possui um conjunto de nodos abaixo dela. Para o correto funcionamento do Appman, é pressuposto que existe um sistema de compartilhamento de arquivos entre os nodos e a base. Isto porque os arquivos de entrada da aplicação são colocados no nodo base e posteriormente copiados para cada nodo para a execução de uma determinada tarefa. Como esta cópia considera a existência de um sistema de compartilhamento de dados, a utilização de recursos espalhados na rede para formar um cluster, sem a utilização de um sistema de compartilhamento, é um impeditivo para a utilização do Appman, sendo esta então, uma segunda deficiência do protótipo Appman.

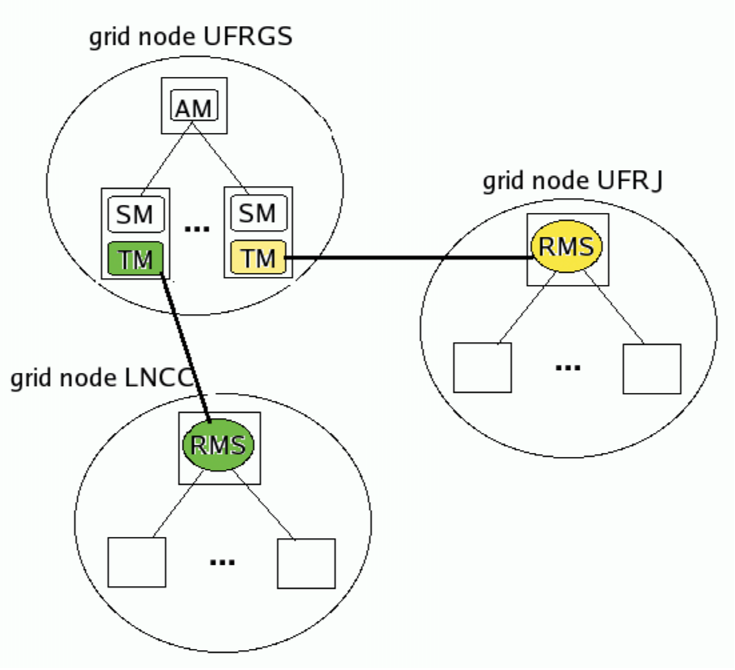

Figura 1.1: Cenário com o AppMan utilizando vários gerenciadores de recursos

1.3 OpenPBS

O OpenPBS (Portable Batch System) [2] é um sistema que estende um sistema operacional POSIX ou Unix e que permite a submissão e controle de tarefas em um ambiente de batch, independente de um ambiente interativo.

O propósito do OpenPBS é de oferecer controles adicionais sobre a inicialização e o escalonamento da execução de trabalhos tipo batch, além de permitir que essas tarefas sejam direcionados para diferentes nós de uma rede. O OpenPBS permite que um site defina e implemente políticas sobre que tipos de recursos e quando de cada recurso pode ser usado por diferentes tarefas. O OpenPBS também fornece um mecanismo com o qual o usuário pode garantir que o job terá acesso aos recursos necessários para que ele seja completado.

Para interface com o OpenPBS, são oferecidas duas maneiras: um conjunto de scripts para linha de comando e uma biblioteca de programação. Ambas permitem acesso a todas as funcionalidades que o OpenPBS oferece. A biblioteca é escrita usando a linguagem C e é chamada Batch Interface Library (IFL), possibilita a construção de novos clientes de acordo com a necessidade de cada usuário.

1.4 DRMAA

A especificação DRMAA (Distributed Resource Management Application API) [3] define uma interface de programação genérica para gerenciadores de recursos e tem como objetivo principal facilitar a integração de aplicações. O escopo da especificação é limitado à submissão, monitoração e controle de tarefas, além da obtenção do estado final da execução de uma tarefa. A DRMAA oferece aos programadores de aplicações e as desenvolvedores de gerenciadores de recursos um modelo de programação que possibilita a construção de aplicações distribuídas fortemente acopladas com os gerenciadores de recursos usados. A interface de programação também preserva a flexibilidade no momento da implantação do sistema, permitindo que sejam utilizados os diversos gerenciadores de recursos disponíveis.

A especificação DRMAA foi desenvolvida por um grupo de trabalho do Global Grid Forum e a sua primeira versão foi publicada em junho de 2004. Inicialmente desenvolvida tendo como base a linguagem C, a especificação apresentou mais tarde a definição da interface de programação para a linguagem Java. O desenvolvimento dessa interface em Java foi liderada pela equipe de desenvolvimento do gerenciador de recursos da Sun, o Sun Grid Engine. Existem versões disponíveis publicamente da interface de programação tanto na linguagem C quanto em Java. No entanto, as implementações encontradas publicamente oferecem suporte apenas para o Sun Grid Engine, nenhuma para o OpenPBS.

Capítulo 2

Modelagem das soluções propostas

Conforme o exposto no capítulo anterior, o Appman requer a melhoria em pelo menos dois pontos (deficiências):

- transferência de dados para os nodos de um cluster que não possui NFS instalado;
- integração com tecnologias para submissão de tarefas.

As seções abaixo apresentam um detalhamento de cada proposta.

2.1 Modelo para tansferência de dados entre recursos de um cluster

A implementação atual do Appman, que conta com o `Exehda` para a submissão de tarefas, pressupõe que os nodos pertencentes à uma célula (Base) possuem um sistema de compartilhamento de arquivos (NFS). Sendo assim, as transferências de dados entre a base e os nodos se dá como uma cópia em disco local. No entanto, alguns experimentos foram impedidos de serem realizados durante a realização dos testes de validação do protótipo Appman, devido a não existência de compartilhamento de arquivos em alguns clusters que seriam inicialmente utilizados. Além disso, a inclusão de um `Desktop`, como um nodo de uma base do `Exehda`, não seria possível pelo mesmo problema. Notou-se então, a necessidade de solucionar este problema através de um mecanismo que permitisse a transferência de arquivos para recursos que não possuem um sistema de compartilhamento de arquivos, porém, nos casos em que o sistema de compartilhamento existir, utilizá-lo principalmente para objetivar desempenho.

```
212 public synchronized void installURLFile(String url,
213         String localFile, boolean chmod) throws RemoteException {
214    try {
215         String dir = defaultdir;
216         String[] cmd = { "/bin/bash", "--login", "-c", "mkdir -p " + dir };
217         Runtime.getRuntime().exec(cmd).waitFor();
218         String localpath = dir + "/" + localFile;
219         File f = new File(localpath);
220         if (f.exists()) {
221            if (chmod) {
222               Runtime.getRuntime().exec("chmod u+x " + localpath).waitFor();
223            }
224            return;
225         }
226         if ((url.indexOf("http") != -1) || (url.indexOf("ftp") != -1)) {
227            BufferedOutputStream out = new BufferedOutputStream(
228                        new FileOutputStream(new File(localpath)));
229            BufferedInputStream in;
230            URLConnection conn = (new URL(url)).openConnection();
231            conn.connect();
232            in = new BufferedInputStream(conn.getInputStream());
233            int c;
234            while ((c = in.read()) != -1) {
235                 out.write(c);
236            }
237            in.close();
238            out.close();
239         }
240         else {
241            try {
242               FileChannel srcChannel = new FileInputStream(url).getChannel();
243               FileChannel dstChannel = new FileOutputStream(localpath).getChannel();
244               dstChannel.transferFrom(srcChannel, 0, srcChannel.size());
```

Figura 2.1: Trecho do código da classe GridFileService que efetua a cópia do arquivo de entrada.

2.1.1 Modelo Atual

No modelo atual, uma tarefa é representada por uma instancia de um objeto da classe `MyTask`. Este objeto é instanciado e gerênciado pelo `TaskManager`, apresentado anteriormente. Para fazer o download dos arquivos necessários, o `MyTask` instancia um objeto da classe `GridFileService`, que trata da trasferência de arquivos no Appman, e invoca o método `installURLFile`. Neste método é realizada a cópia do arquivo para o recurso local. Esta cópia conta com a existência do diretório onde se encontra o arquivo de entrada e ao tentar realizar a cópia, o erro acontece.

A figura 2.1 apresenta o trecho do código que efetua a cópia do arquivo de entrada. Na linha 226 da figura, o sistema verifica se é uma cópia para a base ou para nodo. Se for para base, o Appman considera que o arquivo está localizado em um servidor http qualquer, conectando-se à ele e buscando o arquivo de entrada. Na linha 242 ele prepara a transferência para nodos considerando a existênca de um sistema de compartilhamento de arquivos.

2.1.2 Modelo Proposto

Para a solução do problema, o modelo proposto requer a alteração do código da classe `GridFileService` e a criação das classes `DMServerImpl`, `DMAppmanImpl`

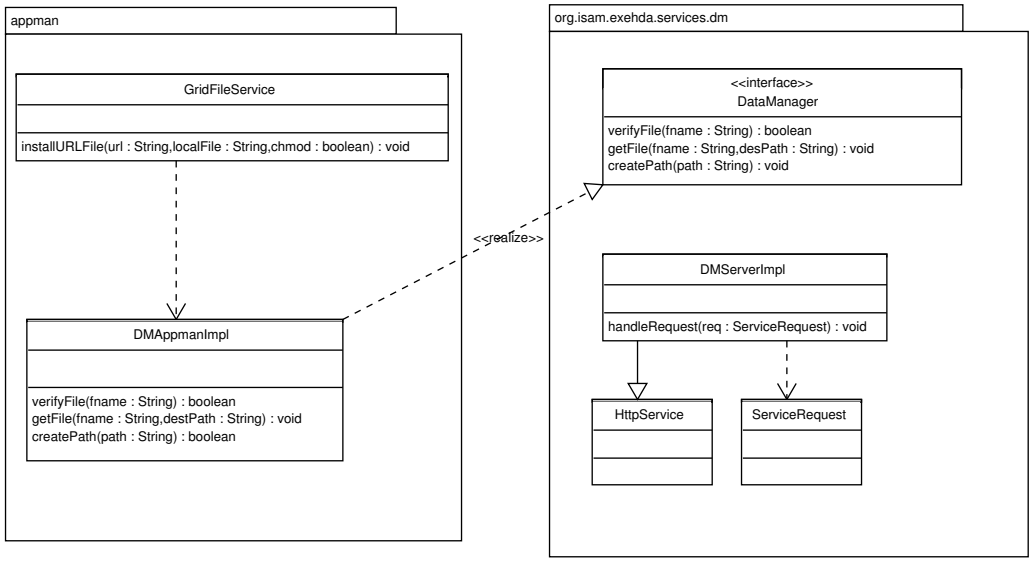

Figura 2.2: Diagrama de classes do modelo proposto.

e da interface `DataManager`. A figura 2.2 apresenta o diagrama de classes contendo as classes propostas para a solução, seus principais métodos e algumas classes já existentes no modelo atual.

A idéia do modelo proposto é disponibilizar um serviço de transferência de arquivos, chamado de `DataManagerService`, que será instanciado em cada recurso base do cluster pelo próprio `midleware Exehda`, como um mini-servidor http que aceitará requisições para transferência de arquivos.

A classe que representa este serviço é a `DMServerImpl` e parte de seu código é apresentado na figura 2.3. O trecho de código apresentado é o responsável por tratar uma requisição de leitura de um arquivo. Na linha 103, o mini-servidor invoca o método de verificação do usuário que está fazendo a requisição. Se o usuário possuir a devida permissão, na linha 116 o arquivo é enviado como uma resposta à requisição.

A interface `DataManager` possui os métodos `getFile`, `createPath` e `verifyFile` que servem para transferir o arquivo, criar um caminho no recurso local e verificar se um arquivo existe no recurso local, respectivamente. Esta interface serve para determinar os métodos que devem ser implementados por classes que venham a ser implementadas para a integração do Appman com outros sistemas de transferência de arquivos. Ainda existem algumas tarefas que devem ser adicionadas ao `DataManagerService` e por isso, a interface `DataManager` ainda poderá ser expandida.

No caso da integração do Appman com o serviço de transferência de arquivos do `Exehda`, o `DataManagerService`, a classe `DMAppmanImpl` imple-

```
 97    protected void handleDMRead( ServiceRequest req )
 98        throws IOException
 99        {
100            String fileName = req.getProperty(PROP_DM_RESOURCE);
101            String userName = req.getProperty(PROP_DM_USER);
102
103            if ( checkAccess(fileName, userName, DM_OP_READ) ) {
104                File file = new File(docRoot+fileName);
105
106                ServiceResponse resp =
107                    new ServiceResponse(req,
108                                        ServiceResponse.OK);
109                resp.putHeader(CONTENT_LENGTH, ""+file.length());
110
111                byte[] buff = new byte[4096];
112                InputStream ins = new FileInputStream(file);
113
114                int nr;
115
116                while ( (nr = ins.read(buff)) != -1 ) {
117                    resp.putData(buff, 0, nr);
118                }
119
120                resp.close();
121            }
122            else {
123                ServiceResponse resp =
124                    new ServiceResponse( req,
125                                         ServiceResponse.BAD_REQUEST);
126                resp.close();
127            }
128        }
```

Figura 2.3: Trecho do código da classe DMServerImpl.

menta a interface `DataManager`. Um objeto desta classe é instanciado no método `installURL` da classe `GridFileService` e é utilizado para efetivar a transferência de arquivos em casos onde não se tem um sistema de compartilhamento de arquivos.

A figura 2.4 apresenta parte do código da classe DMAppmanImpl e a figura 2.5 apresenta parte do código da classe `GridFileService` já com as novas alteralções. Na linha 79 da figura 2.4 é realizada a conexão do nodo com o `DataManagerService` e nas linhas posterires, é iniciado o processo de transferência do arquivo remoto para o sistema local. Na linha 216 da figura 2.5 o objeto que fará a comunicação com o `DataManagerService` é instanciado. Com ele, é verificado se o cluster possui um sistema de compartilhamento de arquivos , através do método `verifyFile`, que verifica se localmente está visível o arquivo de entrada que foi criado na base. Se não estiver, significa que não existe um sistema de compartilhamento de arquivos e o método para busca do arquivo remoto é invocado, conforme apresentado na linha 236.

2.1.3 Integração com o AppMan

A integração do modelo ao Appman causou pouco impacto ao protótipo. Conforme citado anteriormente, foi adicionada a classe `DMAppmanImpl` e efetuada a alteração da classe `GridFileService`. No entanto, mudanças no

```
68   public void getFile(String fname, String destpath){
69
70           URLConnection dmconn;
71           BufferedInputStream br;
72           BufferedOutputStream bo;
73           FileOutputStream fo;
74           int lbyte;
75
76           try{
77              urlfile = new URL(
78              ctAddress + ":" + port + "/" + fname);
79              dmconn = urlfile.openConnection();
80              dmconn.setDoOutput(true);
81              dmconn.setRequestProperty("Isam-User","marky@ramones");
82              dmconn.setRequestProperty("Isam-dm-Operation","read");
83              br = new BufferedInputStream(dmconn.getInputStream());
84              fo = new FileOutputStream(destpath);
85              bo = new BufferedOutputStream(fo);
86
87              while((lbyte=br.read())!= -1){
88                 bo.write(lbyte);
89              }
90              br.close();
91              bo.flush();
92              bo.close();
93           }catch(IOException ioe){
94              System.out.println( "Erro ao transferir arquivo - " + ioe.getMessage());
95       }
96   }
```

Figura 2.4: Trecho do código da classe DMAppmanImpl.

```
209         public synchronized void installURLFile(String url, String localFile,
210                         boolean chmod) throws RemoteException {
211            try {
212                 boolean existe;
213                 boolean criado;
214                 String dir = defaultdir;
215                 String localpath = dir + "/" + localFile;
216                 DMAppmanImpl dm = new DMAppmanImpl();
217                 existe = dm.verifyFile(url);
218                 criado = dm.createPath(dir);
219                 if (!existe){
220                    if ((url.indexOf("http") != -1) || (url.indexOf("ftp") != -1)) {
221                       BufferedOutputStream out = new BufferedOutputStream(
222                              new FileOutputStream(new File(localpath)));
223                       BufferedInputStream in;
224                       URLConnection conn = (new URL(url)).openConnection();
225                       conn.connect();
226                       in = new BufferedInputStream(conn.getInputStream());
227                       int c;
228                       while ((c = in.read()) != -1) {
229                         out.write(c);
230                       }
231                       in.close();
232                       out.close();
233                    }
234                    else
235                    {
236                       dm.getFile(url, localpath);
237                       File f = new File(localpath);
238                       System.out.println("******Testando se a cópia funcionou");
239                       if (f.exists()) {
240                          Debug.debug("GridFileService File already installed.", true);
241                          if (chmod) {
242                             Runtime.getRuntime().exec("chmod u+x " + localpath)
243                                                 .waitFor();
244                          }
245                          return;
246                       }
247                    }
```

Figura 2.5: Trecho do código da classe GridFileService com as novas alterações.

middleware Exehda, utilizado pelo Appman, forma necessárias. Houve a inclusão da classe DMServerImpl, da interface DataManager e foi necessária a alteração do arquivo de configuração exehda-services.xml, que descreve a base, seus serviços e seus nodos, além de outras informações necessárias durante a inicialização do Exehda. A alteração foi basicamente a inclusão de um novo serviço à ser inicializado pelo Exehda, que é o DataManagerService, representado pela classe DMServerImpl. A figura 2.6 apresenta uma parte do código do arquivo exehda-services.xml. Na linha 8 o serviço DataManagerService é declarado para ser iniciado durante a inicialização do Exehda. Nas linhas 2,3 e 4 a base gradep é declarada.

```
1 <EXEHDA>
2    <PROFILE name="gradep.localhost">
3       <PROP name="localhost.id" value="0"/>
4       <PROP name="localhost.cell" value="gradep"/>
5          .
6          .
7          .
8       <SERVICE name="dm" loadPolicy="boot">
9           <PROP name="impl" value="org.isam.exehda.services.DMServerImpl"/>
10          <PROP name="docroot" value="/tmp/appman-dev/"/>
11          <PROP name="contactAddress" value="http://gradep.inf.ufrgs.br"/>
12      </SERVICE>
13          .
14          .
15          .
16   </PROFILE>
17</EXEHDA>
```

Figura 2.6: Exemplo de um arquivo de configuração do AppMan

2.2 Modelagem da solução para integração do protótipo com outras tecnologias para a submissão de tarefas

Como apresentado no capítulo anterior, a implementação atual do AppMan faz um escalonamento aleatório das tarefas nos nós disponíveis de uma grade. Um cenário mais desejável também foi mostrado, onde o AppMan seria integrado com gerenciadores de recursos, possibilitando a execução em ambientes maiores com menos complexidade de instalação do EXEHDA.

Resumindo, essa integração do AppMan com um gerenciador de recursos possibilitaria:

- utilizar um número maior de nós de processamento
- integrar grades dispersas geograficamente para execução de uma aplicação com baixa complexidade de software
- facilitar a realização de testes experimentais

A especificação DRMAA tenta preencher esse espaço, oferecendo uma maneira padrão para que programadores possam desenvolver aplicações que façam uso de diversos gerenciadores de recursos com menor esforço. No entanto, não foi encontrada nenhuma implementação em Java dessa especificação que ofereça uma interface entre o AppMan e o OpenPBS, gerenciador de recursos mais popular e disponível em vários nós acessíveis pelos integrantes do projeto em que o AppMan faz parte.

O objetivo desta modelagem é definir uma interface em Java para possibilitar que o AppMan contate um gerenciador de recursos local para a submissão e monitoração da execução de tarefas. Essa modelagem deve permitir também que, na medida do necessário, novas implementações ofereçam suporte a outros gerenciadores de recursos além do OpenPBS, alvo particular deste trabalho.

2.2.1 Modelo do componente de submissão e monitoração

O modelo do componente é definido através de um conjunto de classes Java e está disponível na forma de um diagrama como ilustrado pela figura 2.7. A classe principal do modelo, `appman.rmswrapper.pbs.drmaa.SessionImpl`, concentra praticamente todos os métodos necessários para a submissão e monitoração de uma tarefa. A definição desses métodos é feita pela interface `org.ggf.drmaa.Session`, parte da especificação DRMAA. Do ponto de vista do AppMan, os métodos mais relevantes dessa classe são:

- `String runJob(JobTemplate jobTemplate)` método que submete uma tarefa e retorna um identificador único para ela que será usado em futuras chamadas.
- `JobInfo wait(String jobName, long timeout)` método que espera o fim da execução da tarefa identificada pelo parâmetro `jobName` por um tempo máximo de espera pelo fim da execução

O modelo é composto, ainda, pelas seguintes classes auxiliares:

- `appman.rmswrapper.pbs.drmaa.JobTemplateImpl` classe responsável por encapsular a definição de uma tarefa para submissão

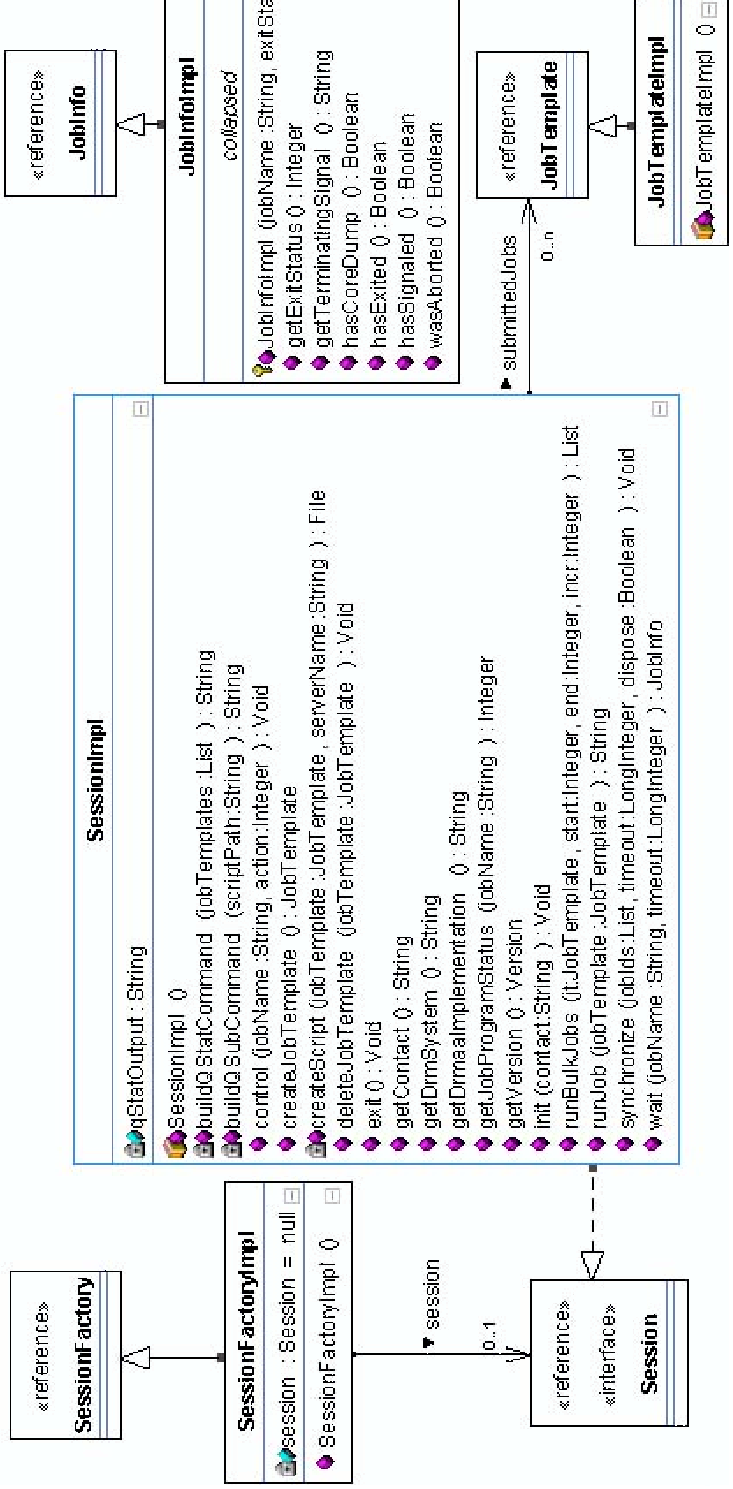

Figura 2.7: Diagrama de classes do pacote `appman.rmswrapper.pbs.drmaa`

- `appman.rmswrapper.pbs.drmaa.JobInfoImpl` classe responsável por representar uma tarefa após a sua execução

2.2.2 Integração com o AppMan

A integração da solução proposta com o AppMan foi realizada através de duas modificações na versão original:

1. A primeira modificação consistiu em alterar o mecanismo de configuração do protótipo com o objetivo de indicar qual componente seria responsável por executar as tarefas da aplicação: o componente original, delegando essa atribuição ao *middleware* EXEHDA; ou o novo, delegando a atribuição ao gerenciador de recursos disponível.

2. A segunda modificação implicou na criação de uma nova classe, chamada `appman.GridTaskDrmaa`, que utiliza a interface definida pelo componente apresentado no capítulo anterior para submeter e monitorar as tarefas da aplicação.

Mecanismo de configuração

A estrutura do arquivo de configuração do AppMan foi alterada para que um nodo possa saber qual componente deve ser usado para submeter as tarefas para execução. Uma nova propriedade foi adicionada à configuração do AppMan para cumprir com esse fim, definindo qual componente deve ser usado em cada nodo onde serão disparados os *SubmissionManagers*.

Por exemplo, o arquivo de configuração da figura 2.8 indica ao AppMan que cada um dos dois *SubmissionManagers* a serem criados devem usar componentes diferentes para a submissão das tarefas. Esse exemplo define que o *SubmissionManager* criado no host `1.integridade` use o componente GridTaskDrmaa para submissão enquanto que o host `1.gradep` utilize o componente GridTask para a mesma finalidade. Desse modo é possível montar rapidamente cenários distintos aproveitando os recursos disponíveis em cada nodo da grade.

Classe `appman.GridTaskDrmaa`

A classe `appman.GridTaskDrmaa` foi criada para fazer a integração do protótipo com o novo componente responsável por submeter as tarefas para um gerenciador de recursos. Essa classe tem o seu comportamento definido pela mesma interface, nomeada `appman.GridTaskRemote`, que a classe original,

```
# AppMan configuration
grid.targetHosts.submissionmanagers.hosts = 1.integridade, 1.gradep
grid.targetHosts.host-final-results = 1.integridade
grid.targetHosts.localscheduler = exehda

# host configuration
grid.targetHosts.submissionmanagers.\
   hosts[1.integridade].concreteTaskClassName=GridTaskDrmaa
grid.targetHosts.submissionmanagers.\
   hosts[1.gradep].concreteTaskClassName=GridTask
```

Figura 2.8: Exemplo de um arquivo de configuração do AppMan

chamada `appman.GridTask`. Dessa maneira, o impacto da modificação é mínimo no restante do código do AppMan.

Nessa nova classe, a alteração mais significativa se encontra no método `execute`, responsável por fazer as chamadas ao componente de submissão. O novo método (figura 2.9) realiza todos os passos determinados pela especificação DRMAA para submissão e monitoração de uma tarefa.

```
private int execute() throws Exception {
 int jobExitStatus = -1;

 // get a session to call a rms
 SessionFactory factory = SessionFactory.getFactory();
 Session session = factory.getSession();

 // initialize the session
 session.init(null);

 // create a job template
 JobTemplate jobTemplate = session.createJobTemplate();

 // copy all attributes from this class to the newly created job template
 copyAttributesTo(jobTemplate);

 // submit the job
 String jobId = session.runJob(jobTemplate);

 // delete job template to free resource
 session.deleteJobTemplate(jobTemplate);

 // wait for job completion
 JobInfo jobInfo = session.wait(jobId, Session.TIMEOUT_WAIT_FOREVER);

 // get job exit status
 jobExitStatus = jobInfo.getExitStatus();

 // exit the session
 session.exit();

 return jobExitStatus;
}
```

Figura 2.9: Método `execute` da nova classe `GridTaskDrmaa`

Capítulo 3

Aplicação GeneAl

A aplicação GeneAl [4] implementa um algoritmo distribuído que realiza alinhamento de seqüências de nucleotídeos, recebendo como entrada uma seqüência de nucleotídeos e busca os melhores alinhamentos com essa seqüência em diversas bases de dados genéticos. Essa aplicação foi implementada utilizando uma abordagem mestre-trabalhador [5], onde um mestre tinha a responsabilidade de dividir o trabalho, enviar trabalho para os trabalhadores, coletar os resultados parciais e gerar o resultado final.

A escolha dessa aplicação para a realização de testes com o AppMan foi motivada pela intenção de utilizar uma aplicação real para validar a aplicabilidade do protótipo. Contribuiu para a escolha a familiaridade com a aplicação, pelo fato dela ter sido desenvolvida dentro do nosso grupo de pesquisa. Finalmente, a possibilidade de fatorar a aplicação em um número possivelmente grande de tarefas também foi decisivo, pois vai ao encontro do cenário que o modelo GRAND vislumbra.

No entanto, a aplicação da forma como descrita em [4] não pode ser utilizada diretamente com o AppMan. Para adequar a aplicação ao uso com o AppMan, ela foi dividida em três classes, cada uma responsável por uma fase da aplicação:

1. **GenealInitialTask** responsável por dividir a base de dados em segmentos que serão processados por cada trabalhador.

2. **GenealProcessTask** responsável por encontrar em um segmento da base de dados as melhores seqüências de nucleotídeos.

3. **GenealFinalTask** responsável por coletar os resultados parciais e gerar o resultado final.

Para definir a dependência entre as tarefas e os arquivos de entrada e

saída foi escrito um script em Grid-ADL (figura 3.1), a linguagem definida pelo GRAND para tal.

```
graph low-coupled

geneal-init="/etc/alternatives/java -cp ./geneal-appman.jar GenealInitialTask"
geneal-process="/etc/alternatives/java -cp ./geneal-appman.jar GenealProcessTask"
geneal-final="/etc/alternatives/java -cp ./geneal-appman.jar GenealFinalTask"

web-base-url="http://gradep.inf.ufrgs.br/geneal"

number-of-tasks=100
input-sequence="TTTTCCTAACCCTAACCCTAACCCTAACCCTAACCCTA..."
process-input=""
process-output=""

foreach task_id in 1..100 {
 process-input = ${process-input} + ";database-segment-" + ${task_id} + ".txt"
}

task i -e "${geneal-init} ${number-of-tasks}" \
 -i ${web-base-url} + "/geneal-database.txt;" + ${web-base-url} + "/geneal-appman.jar" \
 -o ${process-input}

foreach task_id in 1..100 {
 task ${task_id} -e "${geneal-process} ${task_id} ${input-sequence}" \
  -i "database-segment-" + ${task_id} + ".txt;" + ${web-base-url} + "/geneal-appman.jar" \
  -o "result-" + ${task_id} + ".txt"
 process-output = ${process-output} + ";result-" + ${task_id} + ".txt"
}

task f -e "${geneal-final} ${geneal-number-of-tasks}" \
 -i ${process-output} + ";" + ${web-base-url} + "/geneal-appman.jar" \
 -o final-result.txt
```

Figura 3.1: Aplicação GeneAl descrita com a linguagem GRID-ADL

Capítulo 4

Experimentos realizados

Os experimentos realizados com a versão do AppMan integrada a um gerenciador de recursos, no caso o OpenPBS, tiveram como objetivo validar a solução proposta e comprovar as possíveis vantagens que essa integração traria para o usuário do protótipo.

O ambiente utilizado para a realização dos experimentos foi composto por um laptop (ktx) e um cluster (gradep) com 4 nós (gradep, c0-0, c0-1, c0-2). Considerando a organização celular do ISAM, duas células foram criadas, ktx-cell e gradep-cell, cada uma com o seu gerenciador de recursos, simulando duas organizações virtuais distintas. A configuração de software e hardware é detalhada na tabela 4.1.

Máquina	Processador	Memória	Rede	Software
ktx	Intel Centrino 1.7	1024 MB	100 Mbps	Linux Ubuntu Edgy OpenPBS v2.3.16 Server, Scheduler e MOM Java 1.5.0_09 EXEHDA
gradep	AMD Athlon XP 2400+	512 MB	100 Mbps	Linux Rocks Clusters 3.3.0 OpenPBS v2.3.16 Server, Scheduler e MOM Java 1.4.2_05 EXEHDA
c0-0	AMD Athlon XP 2400+	512 MB	100 Mbps	Linux Rocks Clusters 3.3.0 OpenPBS v2.3.16 MOM Java 1.4.2_05
c0-1	AMD Athlon XP 2400+	512 MB	100 Mbps	Linux Rocks Clusters 3.3.0 OpenPBS v2.3.16 MOM Java 1.4.2_05
c0-2	AMD Athlon XP 2400+	512 MB	100 Mbps	Linux Rocks Clusters 3.3.0 OpenPBS v2.3.16 MOM Java 1.4.2_05

Tabela 4.1: Configuração de software e hardware do ambiente

4.1 Experimento sobre a distribuição de tarefas

O primeiro experimento realizado consistiu em submeter seis vezes a aplicação GeneAl para execução. O *script* em GRID-ADL da aplicação foi modificado para dividir a base de dados em 50 segmentos. Dessa forma, a aplicação GeneAl nessa configuração consiste de uma tarefa inicial para divisão do trabalho, cinqüenta tarefas para o processamento do trabalho e de uma tarefa final para coleta e combinação dos resultados, totalizando 52 tarefas. A tabela 4.2 mostra a distribuição das tarefas em cada uma das seis execuções.

	Célula		Nó				
	ktx-cell	gradep-cell	ktx	gradep	c0-0	c0-1	c0-2
1	21	31	21	0	6	17	8
2	21	31	21	0	11	17	3
3	21	31	21	0	12	12	7
4	21	31	21	0	11	10	10
5	21	31	21	0	19	6	6
6	21	31	21	0	7	17	7

Tabela 4.2: Distribuição das tarefas

Como pode ser observado a distribuição das tarefas entre as duas células ocorre de forma determinística, em função do algoritmo de escalonamento utilizado pelo AppMan. Por outro lado, a distribuição de tarefas dentro da célula gradep-cell não é determinística, em virtude do algoritmo de escalonamento utilizado pelo OpenPBS. Esse algoritmo considera, entre outros fatores, a carga dos nós para decidir qual deles deverá receber uma tarefa para execução. Para reforçar esse ponto, pode-se observar que o nó gradep não recebe tarefa alguma em todas as execuções, provavelmente por estar sobrecarregado pela execução simultânea do EXEHDA e dos componentes servidor e escalonador do OpenPBS.

4.2 Experimentos sobre a transferência de arquivos

Os testes para a transferência de arquivos em ambientes sem o sistema de compartilhamento de arquivos foi realizada através da inclusão de dois novos recursos à base gradep. Estes dois recursos são desktops convencionais, `indiana.inf.ufrgs.br` e `moe.inf.ufrgs.br`.

Os dois recursos foram inclusos como nodos da `base gradep`, tornando o ambiente com uma característica mista (com e sem sistema de compar-

tilhamento de arquivos), ou seja, a `base gradep` e os `nodos c0-0, c0-1 e c0-2` possuem um sistema de compartilhamento de arquivos e os nodos indiana e moe não. Neste caso, o Appman utilizará o método de transferência local entre a base e os `nodos c0-0, c0-1 e c0-2` e utilizará o serviço `DataManagerService` para a transferência entre a base e os outros dois nodos.

A figura 4.1 apresenta parte do arquivo de configuração `exehda-services.xml` contendo a configuração da base e dos nodos da base. A base é declarada na linha 2 e os nodos da base são declarados nas linhas 8, 14, 20, 26 e 33. Como pode ser observado, esta configuração foi a utilizada para os testes de transferência de arquivos (linhas 26 e 33).

```
1 <EXEHDA>
2    <PROFILE name="gradep.localhost">
3       <PROP name="localhost.id" value="0"/>
4       <PROP name="localhost.cell" value="gradep"/>
5          .
6          .
7   </PROFILE>
8   <PROFILE name="gradep.c0-0">
9      <PROP name="localhost.id" value="1"/>
10     <PROP name="localhost.cell" value="gradep"/>
11         .
12         .
13  </PROFILE>
14  <PROFILE name="gradep.c0-1">
15     <PROP name="localhost.id" value="2"/>
16     <PROP name="localhost.cell" value="gradep"/>
17         .
18         .
19  </PROFILE>
20  <PROFILE name="gradep.c0-2">
21     <PROP name="localhost.id" value="3"/>
22     <PROP name="localhost.cell" value="gradep"/>
23         .
24         .
25  </PROFILE>
26      <PROFILE name="gradep.indiana">
27      <PROP name="localhost.id" value="4"/>
28      <PROP name="localhost.cell" value="gradep"/>
29         .
30         .
31   </PROFILE>
32   </PROFILE>
33      <PROFILE name="gradep.moe">
34      <PROP name="localhost.id" value="5"/>
35      <PROP name="localhost.cell" value="gradep"/>
36         .
37         .
38   </PROFILE>
39</EXEHDA>
```

Figura 4.1: Arquivo de configuração de bases e nodos do Exehda

Para a realização do teste, a aplicação foi posta em execução neste ambiente e foi realizada a monitoração dos logs gerados pelo Appman e pelo Exehda. Com a execução e monitoração, constatou-se a efetividade da proposta, permitindo a execução completa da aplicação neste ambiente. Além disso, nos casos em que existia um sistema de compartilhamento de arquivos, o a proposta comportou-se como desejado, utilizando métodos de cópia local.

Capítulo 5

Conclusões

O objetivo principal deste projeto de pesquisa, possibilitar a integração do AppMan com o OpenPBS, foi atingido. Como apresentando neste relatório, a solução encontrada permite que o protótipo AppMan se aproxime mais do que o modelo GRAND define, possibilitando que um número maior de recursos seja empregado, com menor complexidade, na realização de testes.

Um segundo objetivo deste trabalho era o de possibilitar a execução de aplicações em ambientes que não possuem um sistema de compartilhamento de arquivos instalado. Este objetivo foi atingido e comprovado através da realização dos testes, possibilitando que o Appman seja utilizado nestes ambientes. O oferecimento da interface DataManager, permitirá a realização da transferência de arquivos por outras tecnologias existentes como, o GridFTP, bastando a implementação destes métodos.

A partir deste trabalho, o desenvolvimento do protótipo AppMan tem condições de evoluir mais rapidamente. Em especial, o algoritmo de escalonamento do AppMan tem margem para ser melhorado e a integração com gerenciadores de recursos auxilia no trabalho de desenvolvimento e testes do algoritmo, além de que a possibilidade de utilizar recursos sem a necessidade de um sistema compartilhamento de arquivos possibilita a montagem de um ambiente mais simples para a realização dos testes. Ainda dentro do escopo deste projeto de pesquisa, a implementação dos modelos propostos oferecendo suporte a outros gerenciadores de recursos ou sistemas de transferência de arquivos são trabalhos futuros que poderiam trazer benefícios interessantes.

Referências Bibliográficas

[1] APPMAN. 2006.
Disponível em: <http://www.cos.ufrj.br/~kayser/grand/appman/>. Acesso em: dezembro 2006.

[2] PBS. 2006. Disponível em: <http://www-unix.mcs.anl.gov/openpbs/>. Acesso em: dezembro 2006.

[3] DRMAA Specification. 2006. Disponivel em: <http://www.drmaa.org>. Acesso em: dezembro 2006.

[4] SCHAEFFER FILHO, A. E. et al. Applying the ISAM Architecture for Genetic Alignment in a Grid Environment. In: *WCGA 2005: Anais do IV Workshop on Computational Grids and Applications. Simpósio Brasileiro de Redes de Computadores (SBRC 2006)*. Curitiba: Sociedade Brasileira de Computação, 2005.

[5] YAMIN, A. C. et al. A framework for exploiting adaptation in high heterogeneous distributed processing. In: *Proceedings of the 14th Symposium on Computer Architecture and High Performance Computing*. [S.l.]: IEEE Computer Society Press, 2002. p. 125–132.