



UNILASALLE



CENTRO UNIVERSITÁRIO LA SALLE

CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

**INTEGRAÇÃO DO SISTEMA APPMAN DE
GERENCIAMENTO DE APLICAÇÕES PARA AMBIENTE
DE GRADE COM DIFERENTES SISTEMAS DE
GERENCIAMENTO DE RECURSOS**

TONISMAR RÉGIS BERNARDO

Canoas, junho de 2008

TONISMAR RÉGIS BERNARDO

**INTEGRAÇÃO DO SISTEMA
APPMAN DE GERENCIAMENTO
DE APLICAÇÕES PARA
AMBIENTE DE GRADE COM
DIFERENTES SISTEMAS DE
GERENCIAMENTO DE RECURSOS**

Trabalho de conclusão apresentado à banca examinadora do curso de Ciência da Computação do Centro Universitário La Salle - Unilasalle, como exigência parcial para obtenção do grau de Bacharel em Ciência da Computação, sob orientação da Profa. DSC. Patrícia Kayser Vargas Mangan.

Canoas, junho de 2008

TERMO DE APROVAÇÃO

TONISMAR RÉGIS BERNARDO

INTEGRAÇÃO DO SISTEMA APPMAN DE GERENCIAMENTO DE APLICAÇÕES PARA AMBIENTE DE GRADE COM DIFERENTES SISTEMAS DE GERENCIAMENTO DE RECURSOS

Trabalho de conclusão aprovado como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação do Centro Universitário La Salle - Unilasalle, pela seguinte banca examinadora:

Prof. Me. Marcos Ennes Barreto
Centro Universitário La Salle - Unilasalle

Prof. Me. Mozart Lemos de Siqueira
Centro Universitário La Salle - Unilasalle

Prof. Dsc. Patrícia Kayser Vargas Mangan
Centro Universitário La Salle - Unilasalle

Canoas, junho de 2008

AGRADECIMENTOS

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS	5
LISTA DE FIGURAS	7
LISTA DE TABELAS	8
1 INTRODUÇÃO	9
1.1 Motivação	10
1.2 Objetivo	11
1.2.1 Gerais	11
1.2.2 Específicos	11
1.3 Metodologia de Pesquisa	11
1.4 Estrutura do Trabalho	12
2 GERENCIAMENTO DE APLICAÇÕES EM GRADE	13
2.1 Gerenciamento de Recursos	14
2.2 Portable Batch System - PBS	16
2.3 PBS - Cliente	17
2.4 PBS - Servidor	17
2.5 Processo de Escalonamento	17
3 MODELO GRAND	19
3.1 Característica das Aplicações	20
3.2 Gerenciamento de Dados	20
3.3 Particionamento	21
3.4 Modelo de Gerenciamento	21
3.5 Protótipo AppMan	23
4 COMPONENTE DE INTEGRAÇÃO	26
4.1 Categorias de Tarefas	27
4.2 Submissão, Monitoramento e Controle	27

4.3	DRMAA para PBS	28
4.4	A Java <i>Binding</i> API	29
5	INTEGRAÇÃO DO APPMAN COM A ESPECIFICAÇÃO DR- MAA	30
5.1	Criação do Ambiente Computacional	30
5.2	Componente DRMAA (submissão e monitoração)	31
5.3	AppMan com DRMAA	32
5.3.1	Classe GridTaskDrmaa	33
5.3.2	Arquivo de configuração	33
	REFERÊNCIAS	35

LISTA DE ABREVIATURAS E SIGLAS

RMS	<i>Resource Management System</i>
DRMS	<i>Distributed Resource Management System</i>
DRM	<i>Distributed Resource Management</i>
GRAND	<i>Grid Robust Application Deployment</i>
DRMAA	<i>Distributed Resource Management Application</i>
API	<i>Application Program Interface</i>
ISAM	Infra-estrutura de Suporte às Aplicações Móveis
EXEHDA	<i>Execution Environment for Highly Distributed Applications</i>
LDAP	<i>Lightweight Directory Access Protocol</i>
UML	<i>Unified Modeling Language</i>
OGF	<i>Open Grid Forum</i>
GT	<i>Globus Toolkit</i>
PBS	<i>Portable Batch System</i>
MDS	<i>Metacomputing Directory Services</i>
GRAM	<i>Globus Resource Allocation Manager</i>
OGSA	<i>Open Grid Services Architecture</i>
WSRF	<i>Web Services Resource Framework</i>
AP	<i>Application Manager</i>
SM	<i>Submission Manager</i>
TM	<i>Task Manager</i>
DAG	<i>Direct Acyclic Graph</i>
DSC	<i>Dominant Sequence Clustering</i>
GRID-ADL	<i>Grid Application Description Language</i>

JavaCC	<i>Java Compiler Compiler</i>
IDL	<i>Interface Definition Language</i>
GFD	<i>Grid Forum Draft</i>
IFL	<i>Batch Interface Library</i>
MOM	<i>Machine Oriented Mineserver</i>
IPC	<i>Inter-Process Communication</i>
UML	<i>Unified Modeling Language</i>
JVM	<i>Java Virtual Machine</i>
JNI	<i>Java Native Interface</i>

LISTA DE FIGURAS

Figura 2.1: Escalonamento em um nó	18
Figura 3.1: Categorias de aplicações de grade: (a) <i>independent tasks</i> , (b) <i>loosely-coupled task(phase)</i> , (c) <i>loosely-coupled tasks (pipeline)</i> e (d) <i>tightly-coupled tasks</i>	20
Figura 3.2: Principais componentes do modelo hierárquico de gerenciamento de tarefas	22
Figura 3.3: Diagrama UML simplificado do AppMan	23
Figura 3.4: Arquivo de entrada para DAG	24
Figura 3.5: AppMan executando principais passos	25
Figura 5.1: Diagrama de classes do pacote appman.rmswrapper.pbs.drmaa	32
Figura 5.2: Método <i>execute</i> da nova classe <i>appman.GridTaskDrmaa</i>	33
Figura 5.3: Arquivo de configuração <i>gridnodes.properties</i>	34

LISTA DE TABELAS

1 INTRODUÇÃO

Grades computacionais (computational grid) é uma das formas mais recentes de ambiente para processamento geograficamente distribuído, que conta com uma grande infra-estrutura de redes e pode ser empregada em troca de programas, dados e serviços. Segundo Dantas (1), pode-se dizer, também, que Computação em Grade (Grid Computing) representa uma forma estendida dos serviços Web permitindo que recursos computacionais possam ser compartilhados. Podemos definir grades como uma plataforma computacional heterogênea distribuída geograficamente fornecendo serviços e recursos às organizações participantes da plataforma (1).

Um sistema de gerenciamento de recursos (*Resource Management System* - RMS) é a parte central de um sistema distribuído fornecendo um mecanismo de enfileiramento de tarefas, políticas de escalonamento, esquemas de prioridades e monitoramento de recursos proporcionando controles adicionais sobre inicialização, escalonamento e execução de tarefas. Também coordena a distribuição dessas tarefas entre as diferentes máquinas em uma rede (2; 3; 4). Devido a heterogeneidade das grades alguns problemas são apresentados, tais como, a alocação dos nós para um grande número de tarefas, gerenciamento de dados e sobrecarga em nós de submissão. O modelo de gerenciamento de aplicações denominado GRAND (*Grid Robust Application Deployment*) (4) visa permitir um particionamento flexível e utilizar uma hierarquia de gerenciadores que realizam a submissão das tarefas. Baseado nesse modelo um protótipo, AppMan (*Application Manager*) (5), foi implementado objetivando garantir o escalonamento das tarefas bem como a autorização e autenticação para tarefas executadas. Ele foi avaliado apresentando bons resultados referente ao gerenciamento de dados e serviços. Esse protótipo consiste em um gerenciador de aplicações que dispara e controla cada aplicação nos nós baseando-se nas informações indicadas pelo gerenciador de submissão que tem como função, além da já citada, criar e monitorar os gerenciadores de tarefas. Os gerenciadores de tarefas são responsáveis pela comunicação com o escalonador de um determinado domínio

garantindo a execução remota e ordem das tarefas de acordo com a dependência de dados (4).

O modelo GRAND permitiria que qualquer sistema gerenciador de recurso fosse usado nos nós, porém o AppMan funciona unicamente com seu próprio sistema de gerenciamento de recursos. Nenhum estudo mais detalhado foi realizado até o momento de como possibilitar que o protótipo suporte a integração com diferentes RMSs.

Uma interface de aplicação (*Application Program Interface* - API) denominada DRMAA (*Distributed Resource Management Application API*) foi desenvolvida com o objetivo de facilitar a integração das aplicações para diferentes RMSs (6).

Este trabalho pretende avaliar se a especificação DRMAA atende as necessidades do AppMan bem como realizar a integração do AppMan ao menos com um RMS.

1.1 Motivação

Gerenciar aplicações consiste em preparar, submeter e monitorar o progresso de execução de todas as tarefas as quais compõem uma aplicação. Cada nó da grade possui seu gerenciador de recursos, os quais podem possuir inúmeros atributos. No modelo estudado foram considerados como atributos o controle exclusivo e a dependência de trabalhos (4). Verificando esses atributos, determinadas ações podem ser agendadas para as aplicações. O GRAND respeita as políticas bem como as especificações administrativas do domínio escolhido considerando que a maioria dos clusters acadêmicos ou redes locais estarão prontos para serem integrados em uma grade, já possuindo usuários locais, submissão local de tarefas e um administrador de sistema, assim sendo o menos intrusivo possível nos diferentes RMSs existentes em uma grade (4).

A especificação DRMAA tem por objetivo facilitar a interface entre aplicações de RMSs com aplicações de diferentes desenvolvedores abstraindo as relações fundamentais da tarefa do RMS provendo um modelo fácil de usar (easy-to-use) para desenvolvedores tanto de aplicações como de RMSs encorajando, desse modo, adoção dos mesmos (6).

Como já dito anteriormente, qualquer RMS pode ser usado nos nós da grade de acordo com o modelo GRAND, porém o protótipo AppMan funciona apenas com seu próprio gerenciador de recursos (4).

Considerando as vantagens do DRMAA (7) e as questões citadas, acredita-se motivador o estudo e desenvolvimento de uma integração com ao menos um RMS. Maiores detalhes serão esclarecidos no capítulo seguinte.

1.2 Objetivo

1.2.1 Gerais

Permitir que o AppMan exporte tarefas para nós da grade gerenciadas por diferentes RMSs e através disso proporcionar a sua integração e conseqüente utilização de recursos em outras instituições científicas.

1.2.2 Específicos

- Verificar se a especificação DRMAA atende todas necessidades esperadas pelo AppMan assim como confirmar a aptidão do AppMan na integração com a DRMAA;
- Integrar AppMan com pelo menos um RMS;
- Avaliar o sobrecusto (overhead) da solução implementada através da verificação do desempenho da integração comparando com o escalonador atual;

1.3 Metodologia de Pesquisa

Com o propósito de verificar a viabilidade da exportação de tarefas do AppMan para RMS diferentes, foi realizado um estudo da especificação DRMAA para o desenvolvimento desta integração. Também foi feito um estudo aprofundado no modelo GRAND onde foi desenvolvido o protótipo, bem como da implementação atual do AppMan.

O desenvolvimento feito na linguagem Java que foi também a linguagem desenvolvida o AppMan facilitando a portabilidade. A métrica de avaliação usada na verificação das vantagens de um RMS diferente comparado com o escalonador do próprio AppMan foi a escalabilidade, tendo como base um estudo teórico aprofundado incluindo os itens citados na bibliografia presente neste trabalho.

Outros trabalhos que proporcionam integração entre diferentes RMSs foram estudados para averiguação das soluções adotadas.

Também foram executados todos procedimentos de instalação do ambiente EXEHDA - ISAM e do AppMan. Alguns problemas foram encontrados com o servidor Lightweight Directory Access Protocol (LDAP) necessário para o funcionamento do EXEHDA. Uma engenharia reversa das classes do projeto AppMan gerando o diagrama UML foi feita para facilitar o estudo da integração com a DRMAA.

Todo acompanhamento foi feito em companhia do professor orientador através de reuniões periódicas previamente estabelecidas conforme o cronograma previsto (colocar o cronograma?).

1.4 Estrutura do Trabalho

A estrutura deste trabalho é dividida em Xs capítulos e esta Introdução. Os capítulos são estruturados da seguinte forma:

CAPÍTULO 2: *Gerenciamento de Aplicações em Grade*. Este capítulo apresenta uma descrição dos conceitos de Grades Computacionais e também de Sistemas Gerenciadores de Recursos.

CAPÍTULO 3: *O Modelo GRAND*. O capítulo 3 explica as características do modelo GRAND e também o protótipo AppMan desenvolvido com base neste modelo.

CAPÍTULO 4: *A Especificação DRMAA*. Neste capítulo será explanado os conceitos da especificação DRMAA, experiências de implementações e suas vantagens.

CAPÍTULO 5: *Portable Batch System - PBS*. Este capítulo conceitua o PBS, suas principais características e sua utilização.

CAPÍTULO 6: *Implementação*. O capítulo 6 explica a metodologia mais detalhada, os motivos da escolha das tecnologias e como foi efetivamente implementado o trabalho.

CAPÍTULO 7: *Resultados Experimentais*. O presente capítulo explicará de que forma foram feitos os testes e avaliações dos resultados.

CAPÍTULO 8: *Conclusão*. Finalmente serão apresentadas as conclusões e propostos trabalhos futuros.

2 GERENCIAMENTO DE APLICAÇÕES EM GRADE

A idéia de computação em grade para processamento de aplicações em paralelo veio, por consequência, dos inúmeros avanços no desempenho de redes de computadores.

Atualmente o uso dessas redes tem aumentado exponencialmente. Muitas dessas redes são distribuídas de forma geograficamente separadas precisando de uma complexa infra-estrutura de software e hardware para gerenciá-las e conectá-las. Dentre as diversas soluções existentes a grade computacional (grid computing) possui características que viabiliza essa conexão.

O Open Grid Forum (OGF) uma comunidade fórum com milhares de indivíduos representando mais de 400 organizações em mais de 50 países criou e documentou (8) especificações técnicas e experiências de usuários. O OGF definiu grades computacionais como um ambiente persistente o qual habilita aplicações para integrar instrumentos, disponibilizar informações em locações difusas. Desde lá esta não é a única e precisa definição para o conceito de grades. Foster (9) define um sistema em grade propondo um *checklist* de três pontos.

1. coordenar recursos os quais não são direcionados para um controle central.
2. usar protocolos e interfaces padronizados, abertos para propósitos gerais.
3. oferecer QoS (qualidade de serviço) não triviais tais como: autenticação, escalonamento de tarefas, disponibilidade.

Uma definição formal do que um sistema em grade pode prover foi definido por Foster et al. em (10). Focando na sua semântica, mostrando que grades não são apenas uma modificação de um sistema distribuído convencional. Podem apresentar recursos heterogênicos como sensores e detectores e não apenas nós computacionais. Abaixo uma lista de aspectos que evidenciam uma grade computacional (11):

- heterogeneidade
- alta dispersão geográfica
- compartilhamento (não pode ser dedicado a uma única aplicação)
- múltiplos domínios administrativos (recursos de várias instituições)
- controle distribuído

Usualmente os usuários usam linguagens de descrição para descreverem a aplicação. A aplicação pode ser representada por um grafo onde cada vértice representa uma tarefa e suas arestas representam a ordem de precedência entre elas. O capítulo seguinte explicará mas aprofundadamente o conceito de representação em grafos utilizado no modelo GRAND. Quando um sistema de gerenciamento de aplicação recebe o grafo da aplicação ele precisa disparar as tarefas na ordem e controlar sua execução.

A distribuição das tarefas entre recursos é um problema que diminui o desempenho. Uma aplicação distribuída pode ser particionada pelo agrupamento de tarefas relacionadas visando a redução de comunicação. Este particionamento deve ser tal que as dependências entre tarefas e localidade dos dados sejam mantidas (4).

A grade deve estar preparada para lidar com todo o dinamismo e variabilidade, procurando obter a melhor performance possível adaptando-se ao cenário no momento.

2.1 Gerenciamento de Recursos

Devido à grande escala, ampla distribuição e existência de múltiplos domínios administrativos, a construção de um escalonador de recursos para grades é praticamente inviável, até porque, convencer os administradores dos recursos que compõem a grade abrirem mão do controle dos seus recursos não é uma tarefa nada fácil. Escalonadores têm como características receber solicitações de vários usuários, arbitrando, portanto, entre os usuários, o uso dos recursos controlados.

Casavant (12) considera escalonar como um problema de gerenciamento de recursos. Basicamente um mecanismo ou uma política usada para, eficientemente e efetivamente, gerenciar o acesso e uso de um determinado recurso. Porém, de acordo com o OGF's (8), escalonamento é o processo de ordenar tarefas sobre os recursos computacionais e ordenar a comunicação entre as tarefas, assim sendo, ambas aplicações e sistemas devem ser escalonadas.

O gerenciamento de recursos de um sistema centralizado possui informação completa e atualizada do status dos recursos gerenciados. Este difere do sistema distribuído, o qual não tem conhecimento global de recursos dificultando assim, o ge-

renciamento. O ambiente em grade introduz cinco desafios para o problema de gerenciamento de recursos em ambientes distribuídos (13):

1. autonomia: os recursos são, tipicamente propriedades e operados por diferentes organizações em diferentes domínios administrativos.
2. heterogeneidade: diferentes lugares podem usar diferentes sistemas de gerenciamento de recursos (RMS).
3. estender as políticas: suporte no desenvolvimento de nova aplicação de mecanismos de gerência num domínio específico, sem necessitar de mudanças no código instalado nos domínios participantes.
4. co-alocação: algumas aplicações tem necessidades de recursos os quais só podem ser satisfeitos apenas usando recursos simultâneos com vários domínios.
5. controle online: RMSs precisam suportar negociações para adaptar necessidades de aplicações para recursos disponíveis.

Sistemas computacionais, na sua grande parte, falham ao tratar dois problemas (4):

- gerenciamento e controle de um grande número de tarefas;
- o balanceamento da carga da máquina de submissão e do tráfego da rede.

Uma distribuição dinâmica de dados e tarefas em uma hierarquia de gerenciadores poderia ajudar o gerenciamento de aplicações. O modelo GRAND baseado na submissão e controle particionados e hierárquicos foi proposto em (4). Uma melhor explanação deste modelo será dada no próximo capítulo.

Grande parte das pesquisas sobre escalonamento de tarefas em grades seguem uma organização hierárquica ou centralizada tais como: Globus (14), Condor (2), ISAM (15) e PBS (16).

Globus (17), um dos projetos mais referenciados na literatura, tem como principal software o Globus Toolkit (GT). Como o nome indica, o GT não é uma solução completa e sim um conjunto de serviços que podem ser combinados para a construção de um *middleware* de grade. O Globus (14) tem seu modelo de escalonamento centralizado. Não fornece suporte nativo as políticas de escalonamento mas permite que gerenciadores externos adicionem esta capacidade. O Globus (GT versão 3) oferece serviços de informação através de uma rede hierárquica chamada *Metacomputing Directory Services* (MDS) (18). O gerenciamento de cada recurso é feito por uma instância do *Globus Resource Allocation Manager* (GRAM) (19). GRAM é o responsável por instanciar, monitorar e reportar o estado das tarefas alocadas para

o recurso. A GT4 (20) disponibiliza tais serviços em uma arquitetura baseada em *Web Services* a *Open Grid Services Architecture* (OGSA) junto com *Web Services Resource Framework* (WSRF). A GT4 é focada na qualidade, robustez, facilidade de uso e documentação.

Outro projeto bastante importante é o Condor (2), cujos trabalhos originais são voltados para redes de computadores, mas atualmente também contemplam *clusters* e grades. O Condor trabalha com a descoberta de recursos ociosos dentro de uma rede alocando esses recursos para execução das tarefas. Condor possui uma arquitetura de escalonamento centralizado, ou seja, uma máquina especial é responsável pelo escalonamento. Todas as máquinas podem submeter tarefas a máquina central que se responsabiliza de encontrar recursos disponíveis para execução da tarefa. Tanto o Condor quanto o Globus perdem pontos no quesito tolerância a falhas e escalabilidade devido ao fato de terem um controle centralizado onde um problema na máquina central comprometeria o sistema por inteiro. Além disso, para o Globus, são necessárias negociações com os donos de recursos além da necessidade do mapeamento dos clientes para usuários locais.

Já o projeto ISAM (15) possui uma arquitetura organizada na forma de células autônomas cooperativas. Sua proposta é fornecer uma infra-estrutura tanto para a construção quanto execução de aplicações pervasivas (15). Concebida para habilitar as aplicações a obter informações do ambiente onde executam e se adaptam às alterações que ocorrem durante o transcurso da execução. O ISAM, diferente do Globus e do Condor possui um modelo de escalonador de tarefas descentralizado ajudando o sistema alcançar um bom nível de tolerância a falhas e escalabilidade.

Um dos gerenciadores que podem ser integrados com o Globus é o PBS (16). O PBS é um RMS que tem por propósito prover controle adicionais sobre a execução de tarefas em batch. O sistema permite um domínio definir e implementar políticas tais como os tipos de recursos e como esses recursos podem ser usados por diferentes tarefas.

2.2 Portable Batch System - PBS

Desenvolvido para fornecer processamento em lote, ao contrário de colocar um processo em segundo plano, o processamento em lote do PBS abrange o escalonamento de múltiplas tarefas conforme as políticas do RMS. Cada tarefa pode conter inúmeros processos. Essas tarefas podem ser direcionadas para rotear processos através de nós em uma rede. É possível a reserva de recursos para determinada tarefa antes do início da sua execução (21).

2.3 PBS - Cliente

Comandos devem estar de acordo com a especificação POSIX.15. É possível especificar mais de uma operação na linha de comando onde essas serão interpretadas uma de cada vez. A geração de um erro de operando em determinado servidor será colocado na saída de erro padrão. O comando continua executando demais operandos. Sendo que o recebimento de qualquer erro por qualquer operando, resultará em um *status* final maior que zero.

2.4 PBS - Servidor

O servidor principal tem como foco principal a comunicação dos clientes e gerenciar as tarefas. O servidor é o coração do sistema e suas principais responsabilidades são (21):

- possuir e controlar tarefas em lotes.
- possuir e controlar filas.
- recuperar estado de tarefas e filas.
- executar serviços em nome de clientes baseado em um serviço de requisição de lote.
- executar serviços adiados em nome de tarefas baseadas em eventos externos.
- iniciar seleção de tarefas para execução baseado em um grupo local definido de políticas de regras.
- estabelecer recursos reservados e usar limites para tarefas iniciadas no local de execução.
- colocar uma tarefa de lote em execução e monitorar seu progresso.
- executar o processamento e a limpeza de tarefas.

2.5 Processo de Escalonamento

O processo de escalonamento se caracteriza por uma escolha que elege qual tarefa será executada. Para isto, esta tarefa deve constar em uma fila de execução.

O PBS fornece um programa separado como um processo de seleção que maximiza a flexibilidade na implementação das políticas locais. O escalonador comunica-se com o servidor via um *socket* IPC. Desta forma é possível possuir escalonador e servidor em diferentes nós.

Para cumprir seu papel de gerenciador de recursos, o escalonador se comunica com outro processo chamado *Machine Oriented Mineserver* (MOM). Assim é possível recuperar informações sobre a carga do sistema do nó. Nestas informações podem conter detalhes do uso de memória, carga de CPU, entre outras. Esta relação entre PBS, o escalonador e o MOM é demonstrado na figura 2.1 (16).

1. Evento avisa o servidor para iniciar um ciclo de escalonamento.
2. Servidor envia comando de escalonamento para Escalonador.
3. Escalonador requisita informação de recursos do MOM.
4. MOM retorna informações solicitadas.
5. Escalonador requisita informações do servidor.
6. Servidor envia informações de *status* da tarefa para o escalonador. Escalonador realiza política para executar a tarefa.
7. Escalonador envia solicitação de execução para o servidor.
8. Servidor envia tarefa para o MOM executar.

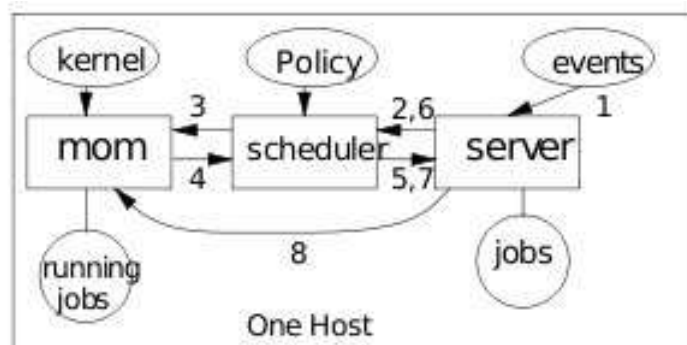


Figura 2.1: Escalonamento em um nó

Um ponto negativo no PBS é o fato deste ter um servidor centralizando, onde uma pane neste nó, afetaria o sistema por inteiro.

Para acesso as funcionalidades que o PBS oferece são fornecidas duas maneiras, um conjunto de *scripts* de linha de comando e uma biblioteca de programação. A biblioteca escrita na linguagem C denominada *Batch Interface Library* (IFL).

O próximo capítulo faz uma análise sobre o modelo GRAND, suas características bem como o protótipo AppMan desenvolvido nos padrões do modelo GRAND.

3 MODELO GRAND

No modelo GRAND são tratados três aspectos do gerenciamento de dados: transferência automática dos dados de entrada para o local onde o arquivo será necessário; o envio de resultados é controlado evitando congestionamento da rede; priorização de localidade no disparo de tarefas para não haver transferências desnecessárias de dados degradando o desempenho. Através de uma hierarquia de gerenciadores (Figura 1) é feito o disparo e controle das aplicações. o *Application Manager* (AP) recebe uma submissão de aplicação através de um usuário, os APs mandam os *Submission Managers* (SM) descrições de tarefas assim, sob demanda, são instanciados os *Task Managers* (TM) para controlar a submissão de tarefas a escalonadores de domínios específicos da grade, esses escalonadores recebem requisições dos TMs fazendo a execução das tarefas propriamente ditas.

Pelo fato de que, na atualidade, ambientes grades envolvem principalmente instituições de ensino em aplicações usualmente classificadas como aplicações científicas, o escopo do GRAND é limitado as seguintes itens.

1. heterogeneidade, lembrando que isto afeta diretamente a política de escalonamento por necessitar de saber as características distintas de hardware e software;
2. grande número de submissão de tarefas, referindo-se a aplicações que geram centenas ou milhares de processos;
3. ausência de comunicação por troca de mensagens, pelo fato da necessidade de inúmeros aspectos nas fases de agrupamento e mapeamento serem considerados;
4. interdependência de tarefas, devido ao compartilhamento de arquivos;
5. manipulação de grande número de arquivos pelas tarefas;

6. o uso de arquivos grandes, através de técnicas como *staging* e *caching*, minimizando a perda de desempenho em função da latência de transmissão;
7. segurança, assume-se que exista uma conexão segura entre os nós da grade;
8. descoberta dinâmica de recursos;
9. gerenciador de recursos local em cada nó;
10. uma tarefa é executada em um RMS até sua finalização;

3.1 Característica das Aplicações

Baseando-se em aplicações típicas para ambientes de grades, comunicando-se via troca de arquivo, conhecimento do número de processo a serem criados e sem troca de mensagens, é proposta a seguinte taxonomia de aplicações (5; 22; 4)

- tarefas independentes *independent tasks* ou seja, as tarefas não possuem dependência entre si. Muitas vezes chamado de *bag-of-tasks*.
- tarefas fracamente acopladas *loosely-coupled tasks* tarefas com poucos pontos de compartilhamentos, aplicações divididas em fases ou sequência.
- tarefas fortemente acopladas *tightly-coupled tasks* grafos complexos, aplicações em lógica com restrições.

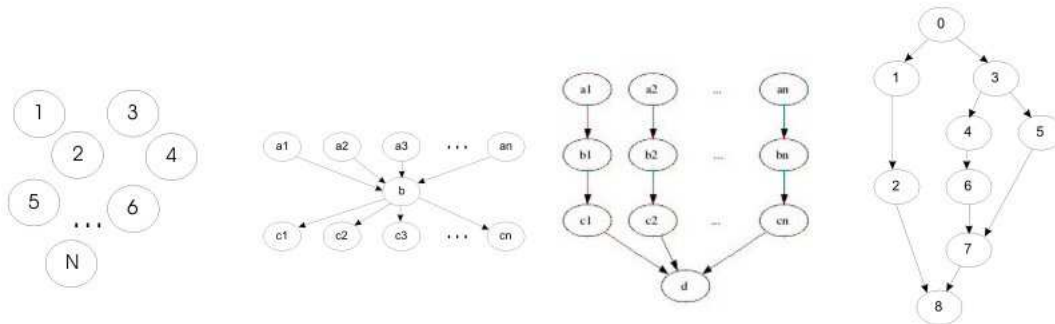


Figura 3.1: Categorias de aplicações de grade: (a) *independent tasks*, (b) *loosely-coupled task(phase)*, (c) *loosely-coupled tasks (pipeline)* e (d) *tightly-coupled tasks*

3.2 Gerenciamento de Dados

Ainda há muito o que ser feito nesta questão, até então o tratamento dos dados é resolvido através de soluções distintas. A preocupação principal de vários trabalhos (23; 24) é com dados em forma de arquivos.

No GRAND, o gerenciamento de arquivos é apenas com *stage in* e *stage out*, ou seja, enviar os dados para o local de processamento e retorno de resultados.

A máquina de submissão (*submit machine*) é a máquina que o usuário submete um grande número de tarefas. Nesta máquina exige um gerenciador encarregado de submeter tarefas para outros gerenciadores guardando o resultado final das tarefas sem preocupar-se com os detalhes. Esse gerenciador executa ou utiliza gerenciadores do sistema que fazem o escalonamento preocupando-se em privilegiar o local dos dados. O envio para a máquina (*home*) é controlado para evitar um congestionamento.

A liberação de carga da máquina que foi usada para disparar a aplicação é a principal vantagem deste esquema.

3.3 Particionamento

O problema tratado no contexto pressupõe que as aplicações submetidas são formadas por tarefas passíveis de dependência na ordem de execução, determinadas pelos dados de entrada e saída, não realizando comunicação por troca de mensagem.

Uma aplicação pode ser representada por um grafo direcionado acíclico (DAG - *Directed Acyclic Graph*) onde o vértice representa uma tarefa e as arestas sua ordem de precedência entre outras tarefas. Este DAG é particionado em sub-grafos para que eles possam ser alocados de forma eficiente em diferentes processadores. O algoritmo DSC (*Dominant Sequence Clustering*) de complexidade $O((v+e) \log v)$ onde v é o número de tarefas e e o número de arestas.

3.4 Modelo de Gerenciamento

Três aspectos do gerenciamento de dados são tratados no modelo GRAND:

- transferência dos dados de entrada automaticamente para o local onde o arquivo é necessário;
- envio de resultados de forma controlada, evitando congestionamento, levando em conta o grande volume de dados;
- para evitar transferências desnecessárias de dados transientes, o escalonamento prioriza localidade no disparo de tarefas e degradação do desempenho;

Uma hierarquia de gerenciadores é usada para o disparo e controle das aplicações conforme ilustrado na figura 3.2.

- **nível 0** aplicação submetida pelo usuário em uma máquina através do *Application Manager*(AP).

- **nível 1** os APs enviam aos *Submission Managers*(SM) descrições de tarefas.
- **nível 2** os *Task Managers*(TK) são instanciados sob demanda pelos SMs com o propósito de controlar a submissão de tarefas para escalonadores de domínios específicos da grade.
- **nível 3** os TKs enviam requisições para os escalonadores a executar de fato as tarefas.

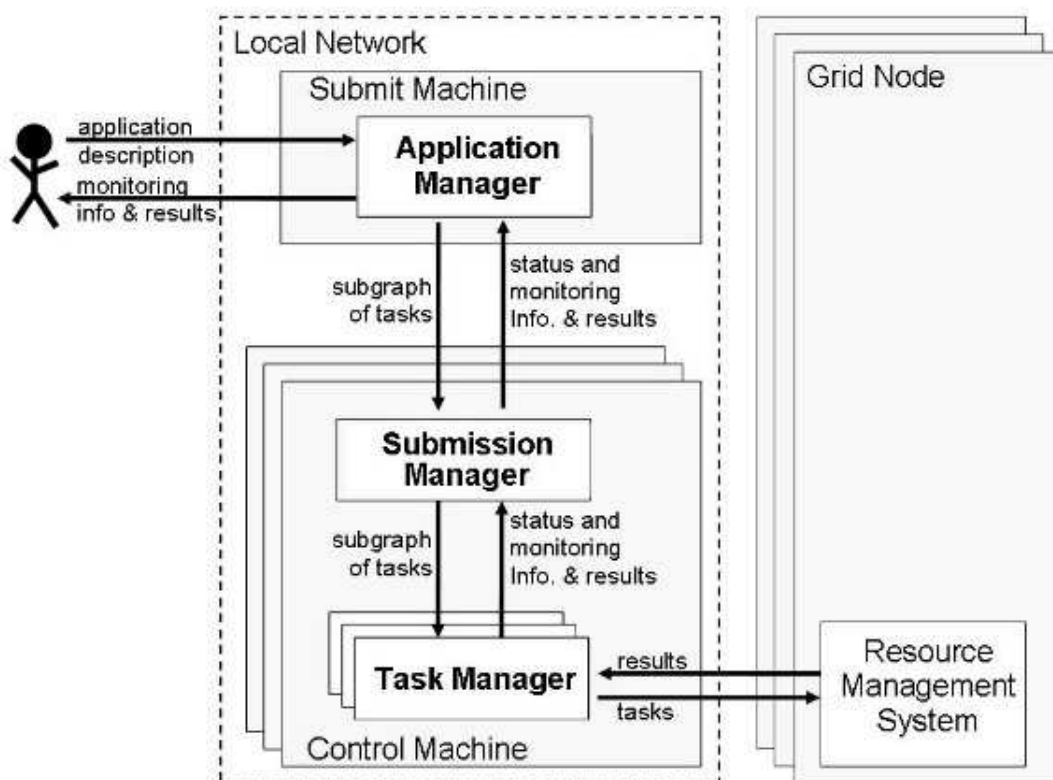


Figura 3.2: Principais componentes do modelo hierárquico de gerenciamento de tarefas

O AP encarrega-se de receber um arquivo de entrada com a descrição da tarefa, particiona as tarefas em sub-grafos para enviá-los para SMs distintos visando manter a localidade dos dados diminuindo a comunicação entre os SMs e apresenta ao usuário informações do estado da execução da aplicação.

No SM é decidido a localidade dos sub-grafos com base em informações dinâmicas sobre os recursos computacionais, indicação do estado da execução e falhas de comunicação periódicas com o SMs, recuperação de falhas através de um registro persistente (*log*), criação e monitoração dos TKs e verificar a continuação ou não da execução de tarefas em máquinas que forem liberadas ou que houverem falhas.

As responsabilidades dos TKs são comunicar-se com um escalonador de um determinado domínio garantindo a execução remota, garantir a ordem da execução conforme dependência de dados e garantir a disponibilidade dos dados de entrada bem como o recebimento dos dados de saída.

A hierarquia do modelo proposto realiza um balanceamento da carga computacional utilizada para controlar a execução das tarefas, evitando seu sobrecarregamento. Deste modo a perda de dados por congestionamento da rede é evitado. E, também, é permitido que escalonadores já existentes sejam integrados em um único ambiente de escalonamento.

3.5 Protótipo AppMan

O protótipo possui um *Application Manager* que dispara e monitora as aplicações em máquinas de uma rede local e cada máquina possui um *Submission Manager*. Estes SMs não se comunicam nem reconhecem a localização de outros SMs. Assim sendo, as aplicações só funcionam em situações que o particionamento gere grafos totalmente disjuntos, ou seja, sem dependência entre SMs e cada sub-grafo executa até o final sem falhas. Desenvolvido em Java visando a portabilidade, a figura 3.3 demonstra diagrama UML básico do desenvolvimento do AppMan.

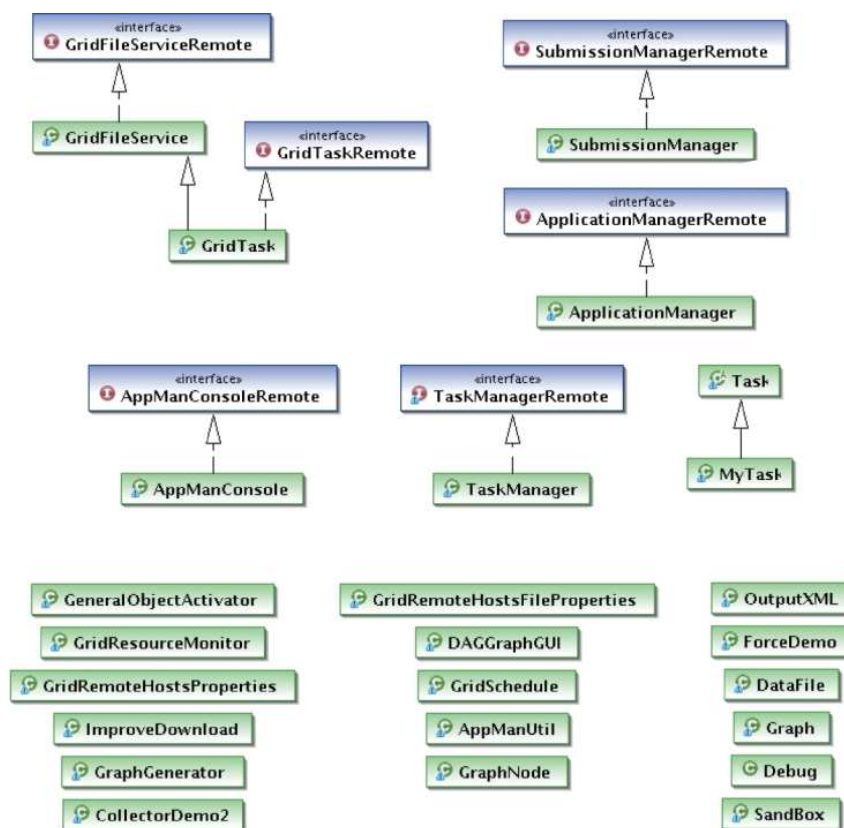


Figura 3.3: Diagrama UML simplificado do AppMan

A especificação da aplicação é feito pela linguagem GRID-ADL (*Grid Application Description Language*), onde o usuário descreve apenas características das tarefas incluindo arquivos manipulados. O *parser* para leitura do arquivo de descrição e a inferência do DAG é implementado na ferramenta JavaCC (25). Assim que obtém-se o DAG a execução é iniciada. Na implementação das fases de submissão e controle da execução usou-se a linguagem Java dentro do ambiente EXEHDA (*Execution Environment for High Distributed Application*) (26).

A figura 3.4 é um exemplo de apenas três linhas uma definição de números de tarefas. Neste exemplo é definido um grafo independente com seis tarefas. A aplicação irá executar seis simulações Monte Carlo. Cada uma recebendo um arquivo de entrada diferente.

```

1 graph independent
2 foreach TASK in 1..6 {
3   task -e mcarlo -i ${TASK}.in -o ${TASK}.out
4 }

```

Figura 3.4: Arquivo de entrada para DAG

EXEHDA é um modelo voltado a aplicações distribuídas, contemplando mobilidade de hardware e software. As aplicações são baseadas no paradigma de programação do projeto ISAM (Infra-estrutura de Suporte às Aplicações Móveis).

O passo um da figura 3.5 representa o usuário submetendo um arquivo de descrição em GRID-ADL. É feito um *parser* no arquivo e o grafo da aplicação é colocado em memória. Um algoritmo de cluster é executado e os sub-grafos são gerados. O AM é inicializado então os AMs instanciam SMs (passo 2), distribuindo alguns sub-grafos para SMs. O arquivo de entrada e o executável são adquiridos pelo *webservice* (passo 3). Um novo SM é instanciado para cada aplicação em cada máquina especificada. Após a criação dos SMs, o AP atribui sub-grafos para cada SM. Os SMs reportam para o AM o progresso das tarefas. Cada SM, independente, verifica a lista de máquinas disponíveis e escolhe onde irá executar, através de um *round-robin* (passo 4).

Um TM é instanciado, o qual cria tarefas remotas e monitora até estarem completas. Antes de iniciar a execução de uma tarefa, AppMan transfere todos arquivos de entrada para um diretório temporário na máquina remota onde a tarefa será executada. Cada SM atualiza informação, através do serviço de informação do EXEHDA, do nós disponíveis.

Alguns mecanismos de tolerância à falhas foram implementados. Na transferência dos arquivos de entrada, não foi possível enviar um arquivo, assume-se uma falha temporária de comunicação e tenta-se novamente a transferência. Se, ao término de

uma tarefa, o arquivo de saída não é gerado, a tarefa é submetida novamente. Em ambos casos, o processo é repetido em um determinado número de vezes. Caso este número é excedido, uma falha permanente é assumida e a aplicação é abortada.

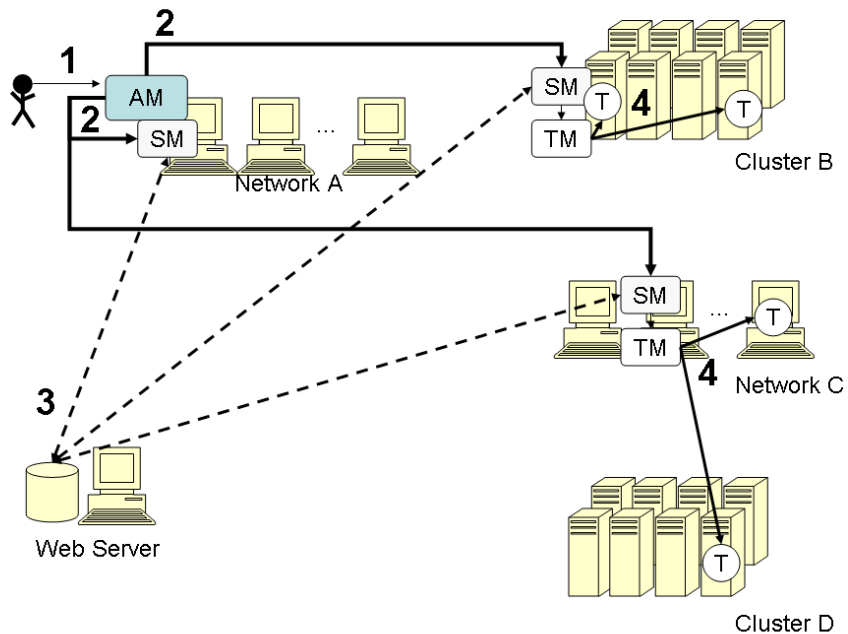


Figura 3.5: AppMan executando principais passos

O capítulo seguinte demonstra as características da interface DRMAA usada na implementação do projeto junto com a *Java Binding*.

4 COMPONENTE DE INTEGRAÇÃO

Alguns trabalhos relacionados a integração de RMS foram encontrados no desenvolvimento da presente monografia, dentre eles podemos citar o Sun HPC ClusterTools (27) que é um grupo de ferramentas para desenvolvimento paralelo que estende o *Sun Network Computing* para aplicações de alto nível em memória distribuída. Oferece apenas três RMS para integração: *Sun N1 Grid Engine V6*, *Load Sharing Facility* (LSF) HPC V 6.2 e *OpenPBS Portable Batch System* (PBS) 2.3.16. Porém esta integração não é genérica é apenas fornecida para o *Sun HPC*.

Outro trabalho relacionado é o ambiente de programação *Distributed Execution and Communication Kernel* (DECK) (28) que tem por objetivo o suporte à execução de aplicações paralelas regulares e irregulares em agregados (*clusters computing*). O DECK visa oferecer recursos para criação remota de *threads*, suporte a múltiplas interfaces de rede, tolerância a falhas e balanceamento de carga através de recursos de comunicação e multiprogramação. Contudo, o escopo do DECK é direcionado para aplicações em agregados onde todos os nós correspondem a computadores, diferentemente de grades que os nós podem ser heterogêneos enquanto que em agregados os nós são homogêneos.

Devido as limitações dos trabalhos citados, a *Distributed Resource Management Application API* (DRMAA), que possui inúmeros trabalhos recentes relacionados (29; 30; 31; 32) possui os atributos necessário para o objetivo deste trabalho. A idéia é que a API seja implementada em múltiplas linguagens. Implementada inicialmente em C/C++, tendo como segunda opção linguagens de *scripts* tais como *Perl* e *Python*. O desenvolvimento da *Interface Definition Language* (IDL) soluciona o problema através de uma *interface* que serve para múltiplas linguagens.

Uma biblioteca ideal deveria servir, dinamicamente e estaticamente, para todos RMS e suas versões. Porém, é possível implementar múltiplos mas não todos RMSs.

A idéia da implementação e aplicações distribuídas em larga escala são para um ambiente DRM e suas políticas. Nem sempre isto é possível, então uma solução

menos intrusiva com uma "categoria de tarefa" e "especificação nativa", tem sido proposta. As políticas específicas do lugar são abstraídas/agregadas em simples *strings* que são interpretadas pela implementação DRMAA. DRMAA não possui um mecanismo explícito de organização de arquivos (6).

4.1 Categorias de Tarefas

Experiências com integrações em um DRM mostra que mesmo uma aplicação de mesmo *Independent Software Vendor* (ISV) possuem políticas diferenciadas através de usuários. Essas políticas afetam atributos específicos tais como os recursos usados pela tarefa, preferência de localidade da execução da tarefa e onde a tarefa irá ser escalonada com relação a outras tarefas.

DRMAA fornece formas distintas para "*job categories*" que encapsulam detalhes das tarefas específicas, escondendo estes detalhes da aplicação. É possível criar uma categoria de tarefa específica para uma aplicação a ser despachada pelo DRMS, o nome associado a categoria deve ser especificado como um atributo de submissão de tarefa. A DRMAA pode, então, usar o nome da categoria para gerenciar recursos específicos do local e requisitos funcionais das tarefas.

Um problema típico que dificulta a escrita para muitos ISVs é o não conhecimento prévio da configuração do DRMS, ainda assim a DRMAA facilita a escrita de aplicações (33).

4.2 Submissão, Monitoramento e Controle

Existem duas versões de rotinas de submissão. Uma versão para submissão individual de tarefas e uma versão para submeter tarefas em volume.

Monitoramento de tarefas e grupo de controle da API precisa manipular:

- liberação, cancelamento, recomeço e finalização de tarefas;
- checagem do código de saída de uma tarefa remota finalizada;
- checagem do estado de uma tarefa remota;
- aguardar a tarefa remota até o final da execução;
- aguardar por todas as tarefas, ou seja, subconjunto da sessão corrente de tarefas terminar a execução (mecanismo de sincronização)

Os sinais Unix e Windows são repassados com a rotina de controle de tarefas que possuem diferenças nos sistemas DRM/RMS. O único não tradicional característica é a passagem da *string* DRMAA_JOB_IDS_SESSION_ALL como um *job_id* para indicar operações em todos os *job_ids* no processo corrente.

A tarefa remota pode estar nos seguintes estados:

- mantido pelo sistema;
- mantido pelo usuário;
- mantido pelo sistema e pelo usuário simultaneamente;
- ativo na fila;
- suspenso pelo sistema;
- suspenso pelo usuário;
- suspenso pelo usuário e sistema simultaneamente;
- em execução;
- completo com/sem sucesso;

Ainda é necessário adicionar nesta lista uma possibilidade da implementação DRMAA não ser capaz de determinar o estado da tarefa remota. Visto que, uma tarefa rejeitada não possui um *job_id* e, conseqüentemente não possui um estado. Existe ainda rotinas necessárias para o rastreamento da execução e a representação textual do erro. Esse rastreamento é especialmente útil em situações onde existem múltiplos processos espalhados em diferentes níveis (33).

4.3 DRMAA para PBS

A biblioteca cobre aproximadamente toda a especificação DRMAA 1.0 com as exceções listadas (34). Ela passou no teste oficial da *suíte* DRMAA com exceção dos testes que requerem *status* de término de tarefa. Todos os obrigatórios e alguns atributos opcionais de tarefas são implementados.

- com PBS, recuperar o *status* de uma tarefa terminada é impossível. Por isso tarefas terminadas são marcadas como completas com código de retorno 0 desde que não tenham terminadas através da biblioteca.
- apenas são aceitas tarefas submetidas sob a sessão corrente (especificação diz que devem ser aceitas tarefas identificadas da sessão anterior).
- o estado do término da tarefa, quando realizado pelo PBS, é marcado como "abortado" e "marcado" qualquer que seja o estado.
- `drmaa-wcoredump()` sempre retorna *false*.

A natureza da implementação DRMAA, como uma boa biblioteca, torna-se uma boa candidata para inclusão em um *Web Server* adquirindo suporte para um Portal DRMS/RMS.

A implementação DRMAA pode ser:

- ligada por uma coleção de *scripts* CGI que são referenciados por páginas *web* residentes;
- ligada em um *Web Server* como um módulo separado;
- feita como um módulo Perl que:
 - incluído no módulo modperl;
 - acessado de scripts CGI Per;

A questão sobre estado de manutenção, segurança e autenticação/autorização necessita que a DRMAA seja vista apenas como um componente de um Portal DRM/RMS.

4.4 A Java *Binding* API

Apesar da especificação GFD.022 ter uma tendência para linguagem C, há uma liberdade para melhorar o ajuste com uma linguagem orientada a objetos como o Java. Alguns métodos sugerido na especificação não constam nos *bindings* de linguagens orientadas a objetos. Os métodos `drmaa_get_attribute()`, `drmaa_set_attribute()`, `drmaa_get_vector_attribute()`, `drmaa_set_vector_attribute()` e `drmaa_get_vector_attribute_names()` não são necessários pois a especificação do *binding* Java especifica uma propriedade ***getter*** e ***setter*** para cada atributo DRMAA. Um *getter* é um método para recuperar valores e um método *setter* para atribuir valores. Essas propriedades permitem, em tempo de compilação, checar os atributos DRMAA permitindo especial tratamento dos atributos que são melhores representados como algo diferente de uma *String* (35).

Alguns estudos realizados (31; 32) a fim de validar a especificação DRMAA constatam total viabilidade em usá-la. Mostrando-se totalmente flexível e útil, torna-se uma ferramenta viável para desenvolvimento tanto de ISV como para aplicações de usuários final.

No capítulo seguinte é feito uma análise do PBS, RMS usado na integração com o protótipo AppMan.

5 INTEGRAÇÃO DO APPMAN COM A ESPECIFICAÇÃO DRMAA

O protótipo AppMan ainda está em desenvolvimento, portanto nem todos os requisitos descritos no modelo GRAND constam no protótipo e, seu funcionamento ainda apresenta alguns comportamentos inesperados.

Desenvolvido em grupo com seu código fonte aberto, seus fontes estão acessíveis para qualquer um que deseja visualizá-los.

Em um primeiro momento foi feita uma análise das versões existentes no repositório com controle de versões gerenciado pelo *software Subversion*. Dentre as versões analisadas foi encontrado em uma ramificação (*branch*) deste repositório uma versão do AppMan que constava uma implementação da integração, teoricamente, realizada e funcional com o uso da DRMAA. Ao efetuar testes com esta versão foi constatado que não estava funcional. Alguns ajustes, que serão explicados posteriormente, foram necessários para o funcionamento correto desta versão.

5.1 Criação do Ambiente Computacional

Para o desenvolvimento do trabalho foi necessário a configuração de um servidor LDAP (APENDICE A) pré-requisito para a execução do *middleware* EXEHDA/ISAM onde é executado o AppMan. Alguns problemas foram encontrados na configuração deste servidor. Um dos motivos foi o fato da documentação existente para a instalação e configuração do protótipo estar desatualizada com as versões do LDAP e distribuições do sistema operacional Linux existente atualmente.

Também foi criado um ambiente computacional com um RMS OpenPBS instalado possuindo três nós para submissão e execução de tarefas. A instalação do OpenPBS (APENDICE B) foi feita em um computador com processador AMD Turion(tm) 64 X2 Mobile Technology TL-50 1.6Ghz e 1.5Gb de memória RAM. Nesta configuração/instalação do OpenPBS foi usado um passo à passo, porém foi necessário a alteração nos parâmetros de configuração (**-with-scp**) para a compilação do

RMS. Essa alteração faz com que o PBS deixe de fazer o *stage-in/stage-out* com o comando padrão **pbs_rcp** para usar o comando **scp** que é o comando padrão para cópia segura dos sistemas operacionais Linux.

O Servidor LDAP foi instalado em um computador AMD Athlon(tm) 64 X2 Dual Core Processor 4000+ 2.2Ghz e memória RAM de 2.0Gb. Este computador também está na lista de nós do OpenPBS além de mais dois outros. O primeiro possui processador AMD Sempron(tm) 2500+ 1.8Ghz e memória RAM de 512Mb e o outro possui um processador Intel(R) Celeron(R) 2.00GHz com 512Mb de memória RAM. Em todos os computadores o sistema operacional é Linux Ubuntu 7.10 com Kernel 2.6.22.

Outro pré-requisito para o funcionamento do protótipo é a instalação do servidor *Network File System* (NFS), usado para compartilhamento de arquivos no Linux. A versão atual do protótipo necessita que o diretório onde as tarefas são criadas seja compartilhado. Esta é uma necessidade que não faz parte do modelo GRAND.

Para compilação e alterações nos fontes do protótipo foi usado as IDEs NetBeans(36) e Eclipse(37), para geração de diagramas UML bem como a engenharia reversa do AppMan, foi usado, além das IDEs citadas, o software Umbrello(38).

5.2 Componente DRMAA (submissão e monitoração)

Baseado na implementação desenvolveu-se um modelo definido por um conjunto de classes conforme figura 6.1. Os principais métodos necessários para submissão e monitoração de uma tarefa estão na classe *appman.rmswrapper.pbs.drmaa.SessionImpl* que implementa a *interface org.ggf.drmaa.Session*.

A DRMAA java *binding* foi implementada com a *JAVA Native Interface* (JNI). JNI é uma *interface* de padrão de programação para escrever métodos Java sob uma máquina virtual Java (JVM), fazendo chamadas recebendo chamadas de aplicações nativas. Essas aplicações nativas, são aplicações específicas do sistema operacional. Dentre as linguagens, pode-se fazer chamadas em C++, C e *Assembly* (39).

Dentre os inúmeros métodos existentes na classe *SessionImpl* os mais relevantes para o AppMan são:

- *String runJob(JobTemplate jobTemplate)* submete uma tarefa e tem como retorno um identificador para ser usado se necessário.
- *JobInfo wait(String jobName, long timeout)* espera o final da execução por um determinado tempo máximo.
- *appman.rmswrapper.pbs.drmaa.JobTemplateImpl* encapsula a definição de uma tarefa para submissão.

- *appman.rmswrapper.pbs.drmaa.JobInfoImpl* representa a tarefa após a sua execução.

A implementação de DRMAA usada foi a mesma usada na implementação com o GridWay 1.0 (31) sendo implementada apenas as *interfaces* essenciais para a submissão das tarefas.

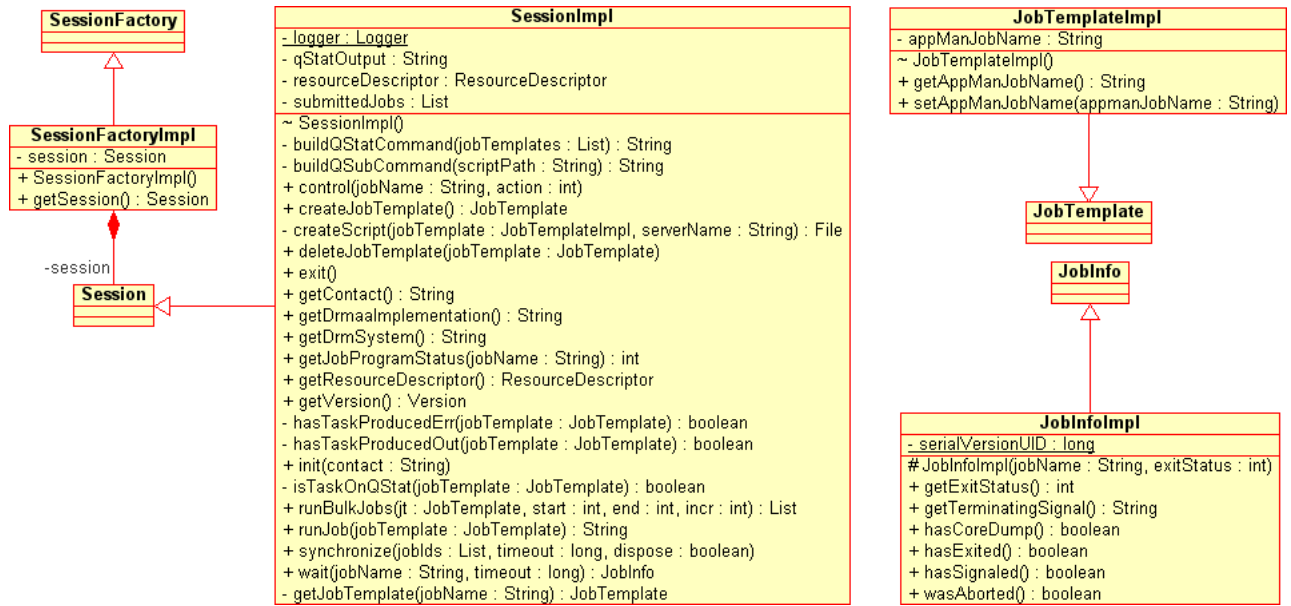


Figura 5.1: Diagrama de classes do pacote **appman.rmswrapper.pbs.drmaa**

Apesar de, não tão relevante quanto os métodos citados acima, o método *createScript* sofreu uma pequena modificação. Este método retorna um arquivo (*File*) que é o *script* que será submetido para o PBS. Neste método foi feita uma alteração na geração do *script* (ANEXO A) que é submetido ao PBS. Em sua forma inicial, notou-se que as tarefas estavam sendo submetidas para o nó padrão de submissão do PBS. Para que as tarefas fossem distribuídas de forma pseudo-randômica entre os nós disponíveis adicionou-se a seguinte linha no arquivo de submissão:

```
#PBS -l nodes=x
```

Onde **x** é um número gerado pseudo-randomicamente entre 1 e 3 que são os nós disponíveis.

5.3 AppMan com DRMAA

Conforme análise do código encontrado no repositório, a integração ficou concentrada em duas mudanças. A primeira é a criação de uma classe nova *appman.GridTaskDrmaa* e a outra foi uma alteração no arquivo de configuração que indica em qual componente é responsável para execução da tarefa, ou *middleware*

EXEHDA, ou nova implementação. Com isso, o impacto da modificação torna-se relativamente baixo.

5.3.1 Classe GridTaskDrmaa

A nova classe implementa a mesma *interface* que a classe padrão *appman.GridTask*. A maior alteração encontra-se no método *execute* onde são feitas as chamadas ao componente de submissão. O método *execute* figura 5.2, é onde são realizados os passos indicados pela especificação DRMAA na submissão e monitoração de tarefas.

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

    // sets the provider for SessionFactory
    // TODO: make it externally configurable
    System.setProperty("org.ggf.drmaa.SessionFactory", "appman.rmswrapper.pbs.drmaa.SessionFactoryImpl");
    // to solve an issue with SessionFactory not finding provider class
    Thread.currentThread().setContextClassLoader(this.getClass().getClassLoader());

    // 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();

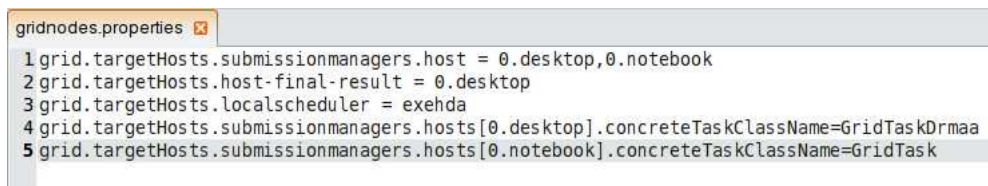
    // exit the session
    session.exit();

    return jobExitStatus;
}
```

Figura 5.2: Método *execute* da nova classe *appman.GridTaskDrmaa*

5.3.2 Arquivo de configuração

A alteração no arquivo de configuração foi feita para que o nó saiba que componente usar na submissão da tarefa. Esta alteração consiste na criação de uma propriedade que recebe o valor que indica qual componente usar. A figura x exemplifica o arquivo de configuração onde o *Submission Manager* criado no nó **0.desktop** usará a *GridTaskDrmaa* para submeter tarefas e o nó **0.notebook** usará *GridTask* para submeter.

A screenshot of a text editor window titled 'gridnodes.properties'. The window contains five lines of configuration code. The first line is '1 grid.targetHosts.submissionmanagers.host = 0.desktop,0.notebook'. The second line is '2 grid.targetHosts.host-final-result = 0.desktop'. The third line is '3 grid.targetHosts.localscheduler = exehda'. The fourth line is '4 grid.targetHosts.submissionmanagers.hosts[0.desktop].concreteTaskClassName=GridTaskDrmaa'. The fifth line is '5 grid.targetHosts.submissionmanagers.hosts[0.notebook].concreteTaskClassName=GridTask'.

```
gridnodes.properties
1 grid.targetHosts.submissionmanagers.host = 0.desktop,0.notebook
2 grid.targetHosts.host-final-result = 0.desktop
3 grid.targetHosts.localscheduler = exehda
4 grid.targetHosts.submissionmanagers.hosts[0.desktop].concreteTaskClassName=GridTaskDrmaa
5 grid.targetHosts.submissionmanagers.hosts[0.notebook].concreteTaskClassName=GridTask
```

Figura 5.3: Arquivo de configuração *gridnodes.properties*

Para que esta nova configuração fosse reconhecida foi criado o método *load-ConcreteTaskClassName* na classe *TaskManager* (**vou colocar uma figura do método**) o qual lê a propriedade no arquivo de configuração e retorna o nome do componente configurado para onde submeter a tarefa. Na implementação atual não foi possível, conforme sugerido, a criação de um *Submission Manager* que submete tarefas via componente DRMAA e um *Submission Manager* usando o componente padrão ao mesmo tempo.

REFERÊNCIAS

- [1] M. A. R. Dantas, *Computação Distribuída de Alto Desempenho: Redes, Clusters e Grids Computacionais*. 2005.
- [2] A. O. o. t. C. S. High Throughput Computing (CONDOR), “High throughput computing (condor), an overview of the condor system,” 2007. acessado em Agosto de 2007.
- [3] A. Bayucan, R. L. Henderson, C. Lesiak, B. Mann, T. Proett, and D. Tweten, “Numerical aerospace simulation systems division nasa ames research center,” p. 281, 2007.
- [4] P. K. V. Mangan, “Grand: Um modelo de gerenciamento hierárquico de aplicações em ambiente de computação em grade,” p. 150, 2006.
- [5] P. K. Vargas, I. de Castro Dutra, and C. F. R. Geyer, “Hierarchical resource management and application control in grid environments,” p. 8, 2003. Relatório Técnico ES-608/03, COPPE/Sistemas UFRJ.
- [6] H. Rajic, R. Brobst, W. Chan, F. Ferstl, J. Gardine, A. H. ans Bill Nitzber, and J. Tollefsrud, “Open grid forum documents, distributed resource management application api specification 1.0 (drmaa),” p. 29, Junho 2004.
- [7] P. Tröger and B. Gietzel, “Condor drmaa 1.0 implementation - experience report (gfd-103),” tech. rep., Hasso-Plattner-Institute and University of Wisconsin-Madison, Fevereiro 2007.
- [8] M. Roehrig, W. Ziegler, and P. Wieder, “Grid scheduling dictionary of terms and keywords,” November 2002.
- [9] I. Foster, S. Tuecke, and C. Kesselman, “The anatomy of the grid enabling scalable virtual organizations,” p. 25, 2001.
- [10] I. Foster, C. Kesselman, J. M. Nick, and S. Tuecke, “The physiology of the grid: An open grid services architecture for distributed systems integration,” p. 31, Junho 2002.

- [11] W. Cirne, “Grids computacionais: Arquiteturas, tecnologias e aplicações,” p. 46, 2002.
- [12] T. L. Casavant and J. G. Kuhl, “A taxonomy of scheduling in general-purpose distributed computing systems,” p. 37, 1996.
- [13] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke, “A resource management architecture for metacomputing systems,” p. 19, 1998.
- [14] I. Foster and C. Kesselman, “The globus project: A status report,” p. 15, 1998.
- [15] ISAM, “Apresentação do isam - <http://www.inf.ufrgs.br/isam/>.”
- [16] A. Bayucan, R. L. Henderson, C. Lesiak, B. Man, T. Proett, and D. Tweten, “Portable batch system - external reference specification,” p. 281, Agosto 1998.
- [17] T. G. A. <http://www.globus.org/alliance/publications/papers.php>, “The globus aliance.”
- [18] L. A. S. Santos, M. T. Rebonatto, P. K. Vargas, and C. F. R. Geyer, “Uma proposta de escalonamento colaborativo de aplicações em um ambiente de computação em grade,” p. 8.
- [19] N. Andrade, “Acesso em grids computacionais: estado da arte e prespectivas,” p. 16, 2002.
- [20] J. E. M. León, “Análisis comparativo gt 2.4 - gt 4.0,” *Semana de Cómputo Científico - Supercómputo, Visualización y Realidad Virtual*, p. 60, 2006.
- [21] A. Bayucan, C. Lesiak, B. Mann, R. L. Henderson, T. Proett, and D. Tweten, “Pbs - internal design specification,” p. 867, 1998.
- [22] P. K. Vargas, I. de Castro Dutra, V. D. do Nascimento, L. A. S. Santos, L. C. da Silva, C. F. R. Geyer, and B. Schulze, “Hierarchical submission in a grid environment,” December 2005.
- [23] T. G. Project, “Gridftp - universal data transfer for the grid,” p. 10, September 2000.
- [24] B. S. White, M. Walker, M. Humphrey, and A. S. Grimshaw, “LegionFS: A secure and scalable file system supporting cross-domain high-performance applications,” p. 10, 2001.
- [25] “Javacc <https://javacc.dev.java.net/>.”

- [26] EXEHDA, “Apresentação do exehda - <http://www.inf.ufrgs.br/exehda/index.php>.”
- [27] “Sun hpc clustertools <http://docs.sun.com/source/819-4131-10/Introduction.html>.”
- [28] M. E. Barreto, “Deck: Um ambiente para programação paralela em agregados de multiprocessadores,” Abril 2000.
- [29] L. Ciesnik, P. Domagalski, K. Kurowski, and P. Lichocki, “Pbs/torque drmaa 1.0 implementation - experience report (gfd-117),” tech. rep., Poznan Supercomputing and Networking Center and Poland FedStage Systems Inc., Setembro 2007.
- [30] A. Haas, “Drmaa state of c binding/implementation drmaa implementation compliance test,” tech. rep., Sun Microsystems, Março 2004.
- [31] J. Herrera, E. Huedo, R. S. Montero, and I. M. Llorente, “Gridway drmaa 1.0 implementation – experience report (gfd-104),” tech. rep., Universidad Complutense de Madrid, Fevereiro 2007.
- [32] D. Templeton and A. Haas, “N1TM grid engine drmaa 1.0 implementation – experience report (gfd-105),” tech. rep., Sun Microsystems, Inc., Maio 2006.
- [33] H. Rajic, R. Brobst, C. D. Systems, W. Chan, F. Ferstl, J. Gardiner, A. Haas, B. Nitzberg, and J. Tollefsrud, “Distributed resource management application api specification,” p. 20, 2002.
- [34] D. for PBS, “Fedstage drmaa for pbs - <https://www.fedstage.com/wiki/>.”
- [35] D. Templeton, P. Tröger, R. Brobst, A. Haas, H. Rajic, and J. Tollefsrud, “Distributed resource management application api javatm language bindings 1.0,” p. 56, 2003.
- [36] “Netbeans ide <http://www.netbeans.org>.”
- [37] “Eclipse <http://www.eclipse.org>.”
- [38] “Umbrello uml modeller <http://uml.sf.net>.”
- [39] “Java native interface - jni <http://java.sun.com/j2se/1.4.2/docs/guide/jni/>.”