UNIVERSIDADE FEDERAL FLUMINENSE

RAFAEL BURLAMAQUI AMARAL

UMA ABORDAGEM AUTÔNOMA PARA GERENCIAR WORKFLOWS CIENTÍFICOS EM AMBIENTES COMPUTACIONAIS HETEROGÊNEOS E COMPARTILHADOS

NITERÓI

2016

UNIVERSIDADE FEDERAL FLUMINENSE

RAFAEL BURLAMAQUI AMARAL

UMA ABORDAGEM AUTÔNOMA PARA GERENCIAR WORKFLOWS CIENTÍFICOS EM AMBIENTES COMPUTACIONAIS HETEROGÊNEOS E COMPARTILHADOS

Tese de Doutorado apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do Grau de Doutor em Computação. Área de concentração: Redes e Sistemas Distribuídos e Paralelos.

Orientador:
MARIA CRISTINA SILVA BOERES

Co-orientador:
EUGENE FRANCIS VINOD REBELLO

NITERÓI

2016

RAFAEL BURLAMAQUI AMARAL

UMA ABORDAGEM AUTÔNOMA PARA GERENCIAR *WORKFLOWS* CIENTÍFICOS EM AMBIENTES COMPUTACIONAIS HETEROGÊNEOS E COMPARTILHADOS

Tese de Doutorado apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do Grau de Doutor em Computação. Área de concentração: Redes e Sistemas Distribuídos e Paralelos

Aprovada em 14 de dezembro de 2016.

BANCA EXAMINADORA

Profa. Dra. Maria Cristina Silva Boeres - Orientadora, UFF

Prof. Dr. Eugene Francis Vinod Rebello - Orientador, UFF

Prof. Dr. Antonio Augusto de Aragão Rocha - UFF

Prof. Dr. Daniel Cardoso Moraes de Oliveira - UFF

Prof. Dr. Bruno Richard Schulze, LNCC

Prof. Dr. Felipe Maia Galvão França - UFRJ

Niterói

2016

Resumo

Nas áreas da física, biologia, ecologia, entre outras, existe a necessidade crescente de executar experimentos que requerem ambientes de processamento de alto desempenho, por tratarem de aplicações computacionalmente complexas ou por envolverem um grande volume de dados. Esses experimentos, no contexto de *e-Science*, são formados por inúmeras tarefas ou aplicações que normalmente são executadas diversas vezes e respeitam uma sequência de execução pré-determinada. Esses experimentos são frequentemente denominados como *workflows* científicos. Devido à complexidade dessas aplicações, geralmente é necessário unir ambientes computacionais de alto desempenho para atender as necessidades dos *workflows*, por exemplo, para aumentar a capacidade de processamento, formando ambientes heterogêneos e compartilhados.

Dado que o problema de escalonamento de tarefas em ambientes homogêneos é considerado da classe NP-Completo, o problema de escalonamento de *workflows* científicos em diferentes ambientes computacionais com sistemas de gerenciamento de tarefas distintos, formado por máquinas heterogêneas e sem garantia de exclusividade, torna a obtenção de um escalonamento eficiente ainda mais difícil por tratar de aplicações intrinsecamente complexas e de grande porte, em ambientes compartilhados com comportamentos pouco previsíveis. Outro ponto a ser considerado é o gerenciador de tarefas de cada ambiente, que são independentes e nem sempre oferecem apoio a *workflows* científicos. Além disso, na maioria das vezes, esses gerenciadores são centralizados, uma abordagem que torna-se pouco eficiente para o gerenciamento de quantidades significativas de *workflows* e de recursos em grande escalas.

Nesse contexto, diferentemente dos sistemas gerenciadores atuais, esse trabalho propõe a criação de uma abordagem de escalonamento e gerenciamento descentralizado de *workflows* científicos, portável para diferentes e-infraestruturas, respeitando as políticas de acesso de cada ambiente e permitindo requisições simultâneas de múltiplos usuários. Mais especificamente, a abordagem proposta tem por objetivo adaptar um Sistema de Gerenciamento de Aplicação (SGA) para criar *workflows* científicos autônomos que consigam se autogerenciar e explorar as características intrínsecas de aplicações *e-Science* e dos recursos alocados, visando otimizar a utilização das e-infraestruturas para reduzir o tempo de execução dos *workflows* científicos. Enquanto o SGA não necessita ser instalado nos recursos que serão utilizados, também oferece mecanismos de tolerância a falhas, escalonamento de tarefas e balanceamento de carga dinâmico. Desta forma, é possível executar *workflows* em ambientes que antes não tinham suporte para workflows. Essa abordagem parece ser mais escalável do que as abordagens existentes e mais capacitada para encarar as principais características dos ambientes computacionais atuais que são dinâmicos (com variação na capacidade e na disponibilidade), instáveis e não exclusivos (recursos compartilhados entre diferentes usuários).

Palavras-chave: Computação autônoma; *Workflows* científicos; Escalonamento dinâmico e distribuído; Sistemas de gerenciamento; E-infraestruturas.

Abstract

In scientific fields of study like physics, biology, ecology, among others, there is a growing need to perform experiments that require high performance computing environments because the research involves computationally complex applications or large amounts of data. In the context of e-Science, these experiments, often referred to as scientific workflows, are typically made up of numerous tasks that are executed several times while respecting a predetermined execution sequence. With the increasing complexity of these applications, scientists face difficulties obtaining access to sufficient computational resources. It may even be necessary to aggregate distinct computational environments to address the needs of these workflows. More often than not, the most commonly avaliable e-infrastructures are heterogeneous and shared.

Given that the classic task scheduling problem, even in homogeneous environments, is considered to be NP-Complete, the problem of efficiently scheduling multiple scientific workflows in different computational environments, formed by heterogeneous and shared machines, is even more difficult given one must deal with intrinsically complex applications running in large scale environments whose job management systems often operate independently and generally do not support scientific workflows. If one considers that nearly all such management systems are centralized, it is not hard to see that this traditional approach will become extremely inefficient when having to manage significant numbers of workflows and resources.

In this context, unlike current management systems, this work proposes the adoption of a decentralized scheduling and management approach for scientific workflows – one that is portable to different e-infrastructures, respects the access policies of each environment and allows simultaneous requests from multiple users. More specifically, the proposed approach aims to adapt an existing Application Management System (AMS) to create autonomous scientific workflows that can self-manage and exploit the intrinsic characteristics of its e-Science applications and the available resources, in order to optimize the use of the e-infrastructure and to reduce the execution time of the workflow. While the AMS does not need to be pre-installed on the resources, it also provides mechanisms for fault tolerance, task scheduling, and dynamic load balancing. In this way, it is possible to harness e-infrastructures that previously did not support workflows. This approach seems to be more scalable than existing ones and better able to address the main characteristics of today's computational environments that are dynamic (varying in capacity and availability), unstable and shared amongst multiple users.

Keywords: Autonomous Computing; Scientific Workflows; Distributed and dynamic Scheduling; Management systems; E-infrastructure.

Lista de Figuras

2.1 Execução Sequencial em mais de um computador. 10

2.2 Modelo de memória compartilhada. Fonte: [83] 12

2.3 Execução Paralela. 13

2.4 DAG de uma execução em sequência: (a) perspectiva de fluxo de controle. (b) perspectiva de fluxo de dados e controle. 16

2.5 DAG de uma execução com divisão paralela: (a) perspectiva de fluxo de controle. (b) e (c) perspectiva de fluxo de dados e controle. 16

2.6 DAG de uma execução com sincronização: (a) perspectiva de fluxo de controle. (b) perspectiva de fluxo de dados e controle. 17

2.7 DAG de uma execução com divisão paralela: (a) perspectiva de fluxo de controle. (b) perspectiva de fluxo de dados e controle. 17

2.8 *Workflow* do SciEvol. Adaptado de [46]. 18

2.9 *Workflow* do SciEvol. Fonte: [46]. 19

2.10 Arquitetura de um sistema de computação em *cluster*. Fonte: [39]. 20

2.11 Modelo de um sistema de computação em *grid*. Fonte: [39]. 21

2.12 Arquitetura HTCondor. Fonte: [1] 26

2.13 Hierarquia de escalonadores dinâmicos do EasyGrid AMS. Fonte: [19]. . . . 28

2.14 *Workflow* associado a criação do modelo de nicho ecológico do OpenModeller. Fonte: [32]. 30

2.15 *Workflow* modelado com o Kepler. Fonte: [42] 31

2.16 Arquitetura Hierárquica do sistema de grade. Fonte: [51] 36

2.17 Hierarquia de processos gerenciadores do EasyGrid AMS. Fonte: [19] . . . 38

3.1 Fatores do Processo de Distribuição de Espécies. Fonte: [68]. 43

3.2 Exemplos de aplicações de ENM. Fonte: [48]. 44

3.3 Abordagem correlativa do OpenModeller. (1) pontos de ocorrência das espécies, (2) camadas ambientais, (3) algoritmo de modelagem, (4) modelo de espaço ambiental, (5) projeção do modelo. Fonte: [60]. 47

3.4 Visão geral do experimento. 51

3.5 Tempo de execução dos algoritmos do openModeller. 53

3.6 Execução concorrente da aplicação OM_MODEL em uma máquina do tipo HT2. 54

3.7 Execução concorrente da aplicação OM_TEST em uma máquina do tipo HT2. 55

3.8 Execução concorrente da aplicação OM_PROJECT em uma máquina do tipo HT2. 56

3.9 Variação da quantidade de layers. 58

4.1 Visão geral da arquitetura de gerenciadores proposta. 62

5.1 Visão geral da abordagem proposta com o estudo de caso. 70

5.2 Exemplo de divisão de *workflow*. (a)*workflow* original (formado por A1, B1, B2, C1, C2, C3 e C4), (b) *subworkflow* (formado por A1, B1, C1 e C2), (c) *subworkflow* (formado por A1, B2, C3 e C4). 72

5.3 Sistema de Gerenciamento do Usuário. 75

5.4 Sistema de Gerenciamento de Recursos. 76

5.5 Sistema de Gerenciamento da Aplicação. 78

5.6 Escalonamento das tarefas obtidos com o meta-escalonador (UMS). 80

5.7 Avaliação do tempo de execução das três abordagens com a aplicação da projeção sequêncial e paralela. 81

5.8 Execução concorrente da aplicação OM_PROJECT em oito máquina do tipo HT1. (a) Meta-escalonador, (b) HTCondor-DAGMan e (c) EasyGrid AMS. 82

5.9 Experimento 5E com EasyGrid AMS em 3 *hosts*. 83

5.10 Experimento 5E com EasyGrid AMS em 4 *hosts*. 84

5.11 Experimento 5E com HTCondor-DAGMan em 6 *hosts*. 85

5.12 Experimento 5E com EasyGrid AMS em 6 *hosts*. 86

6.1 Avaliação da quantidade de memória RAM necessária. 90

6.2 Variação da quantidade de VRAM. 91

6.3 Análise do experimento com diferentes quantidade de VRAM. (A) Utilizando 1GB de VRAM; (B) Utilizando 2GB de VRAM. 92

6.4 Variação da quantidade de processos concorrentes. 92

6.5 Tipos de virtualização. 94

6.6 Virtualização baseada em container. Fonte:[4] 95

6.7 Comparação entre MV x Docker x Host 95

6.8 Tempo de Execução do AMS variando a quantidade de hosts 97

6.9 Tempo de Execução do AMS variando a quantidade de hosts 97

6.10 Variação da quantidade de tarefas concorrentes. 102

Lista de Tabelas

2.1 Possíveis estados dos recursos computacionais. 26

2.2 Diferentes formas de gerenciamento de tarefas. Fonte: [19] 33

2.3 Abordagens de escalonamento em aplicações compostas por *bag-of-tasks* [26, 87]. 35

2.4 Abordagens de escalonamento de *workflow* baseadas em heurísticas 40

2.5 Abordagens de escalonamento de *workflows* baseados em meta-heurísticas. 41

3.1 Comportamento do servidor para cada operação do OWMS. Fonte: [60]. . . 48

3.2 Comportamento do servidor para cada operação do OWMS. Fonte: [60]. . . 49

3.3 Características de cada espécie. 56

3.4 Média do tempo (em segundos) da execução sequencial de cada espécie com 5 algoritmos diferentes do openModeller. 57

3.5 Variação da quantidade de *Layers* no om_model 57

3.6 Variação da quantidade de *Layers* no om_test 57

3.7 Variação da quantidade de *Layers* no om_project 58

5.1 Média de tempo de execução de cada uma das diferentes abordagens. . . . 82

6.1 Ambientes computacionais utilizados. 89

6.2 Média de tempo de execução, em segundos, do AMS. 96

6.3 Descrição dos testes com compartilhamento de recursos 98

6.4 Média de tempo de execução de cada uma das diferentes abordagens. . . . 99

6.5 Execução do AMS com meta. 100

6.6 Comparação, em segundos, entre a projeção sequencial e paralela. 103

6.7 Infraestrutura Apropriada para uso do OpenModeller. 104

Lista de Abreviaturas e Siglas

AMS : *Application Management System*;
CGI : *Common Gateway Interface*
CRIA : Centro de Referência em Informação Ambiental;
CPU : *Central Processing Unit*;
DAG : *Directed Acyclic Graph*;
ENM : *Ecological Niche Modelling*;
GBIF : *Global Biodiversity Information Facility*;
HVB : Herbário Virtual Brasileiro;
I/O : *Input/Output*;
KVM : *Kernel-based Virtual Machine*;
LD : *Local Disk*
LSF : *Load Sharing Facility*
MAU : *Memory Access Unit*;
MNE : Modelagem de Nichos Ecológicos
MPI : *Message Passing Interface*;
MPMD : *Multiple Program Multiple Data*;
MV : Máquina Virtual;
NFS : *Network File System*
OMWS : *OpenModeller Web Service*;
PBS : *Portable Batch System*;
PRAM : *Parallel Random Machine*;
RMS : *Resource Management System*;
SOAP : *Simple Object Access Protocol*;
SGA : Sistema de Gerenciamento de Aplicação
SGE : *Sun Grid Engine*
UMS : *User Management System*;
VCPU : *Virtual CPU*;
VRAM : *Virtual Random Access Memory*;
WfMS : *Workflow Management Systems*;

Sumário

1 Introdução **1**

1.1 Motivação 2

1.2 Objetivos 4

1.3 Contribuições 5

1.4 Organização 7

2 Aplicações em Ambientes de Alto Desempenho **9**

2.1 Tipos de Aplicações 9

2.1.1 Aplicações Sequenciais 9

2.1.2 Aplicações Paralelas 11

2.1.3 *Workflows* Científicos 14

2.1.4 Relação entre os tipos de aplicações 17

2.2 Ambientes de Execução 19

2.2.1 *Cluster* 19

2.2.2 Grades Computacionais 20

2.2.3 Computação em Nuvem 21

2.3 Sistemas de Gerenciamento 24

2.3.1 Gerenciadores de Recursos 24

2.3.1.1 *Portable Batch System - PBS* 24

2.3.1.2 HTCondor 25

2.3.2 Gerenciadores de Aplicação 27

2.3.2.1 EasyGrid AMS 27

2.3.3 Gerenciadores de *workflows* existentes 29
2.3.3.1 Taverna 29
2.3.3.2 Kepler 30
2.3.3.3 HTCondor - DAGMan 31
2.3.4 Orientação 32
2.3.4.1 Sistema de Gerenciamento de Usuário 32
2.3.4.2 Sistemas Gerenciadores de Recursos 32
2.3.4.3 Sistema de Gerenciamento de Aplicação 33
2.4 Escalonamento de Tarefas 33
2.4.1 Escalonamento de Sistemas Distribuídos 34
2.4.1.1 Aplicações *Bag-of-Tasks* 34
2.4.2 Escalonamento em Grades Computacionais 34
2.4.2.1 Escalonador Centralizado 35
2.4.2.2 Escalonador Hierárquico 36
2.4.2.3 Escalonador Distribuído 37
2.4.3 Escalonamento de *Workflows* 38

3 Estudo de Caso: Modelagem de Nichos Ecológicos com openModeller 42

3.1 Modelagem de Nichos Ecológicos 43
3.2 openModeller 45
3.2.1 oM Server 1.0 48
3.2.2 oM Server 2.0 49
3.3 Descrição de um experimento científico para criação de um modelo de nicho ecológico 50
3.4 Análise da execução do openModeller 52
3.4.1 Comparação dos algoritmos 52
3.4.2 Impacto da variação dos *Layers* 56

3.5 Conclusões 58

4 Uma Arquitetura para Integrar Ambientes Computacionais para Executar *Workflows* Científicos 60

4.1 Descrição do Problema de Escalonamento 60

4.2 Uma Arquitetura Multinível 61

4.2.1 Pré-requisitos 62

4.2.2 Etapa 1: *Resource Broker* 65

4.2.3 Etapa 2: Conector de cada ambiente computacional 66

4.2.4 Etapa 3: Gerência autônoma 67

4.3 Conclusão 68

5 Avaliação das Abordagens de Gerenciamento para *Workflows ENM* 69

5.1 Aplicação da abordagem proposta no estudo de caso 69

5.1.1 Descrição da e-infraestrutura 70

5.2 *Script* meta-escalonador 74

5.3 HTCondor e DAGMan 75

5.4 EasyGrid AMS 77

5.5 Uma Comparação UMS x RMS x AMS 79

5.5.1 Avaliação da primeira versão do openModeller - OMWS 1 79

5.5.2 Avaliação da segunda versão do openModeller - OMWS 2 81

5.6 Conclusões 86

6 Avaliação do Desempenho do EasyGrid AMS com *Workflows ENM* 88

6.1 Ambiente computacional 88

6.1.1 Infraestrutura 88

6.1.2 Análise da memória RAM 89

6.1.2.1 Consumo de memória RAM 90

6.1.2.2 Variação da VRAM - sem execução concorrente de MV . . 90

6.1.3 Quantidade de processos concorrentes 92

6.1.4 Local de armazenamento dos dados 93

6.1.5 Flexibilidade de infraestruturas 93

6.2 Desempenho do AMS com o OpenModeller 96

6.2.1 Flexibilidade na execução do OpenModeller 100

6.2.2 Estratégia Paralela para a Projeção de Modelo de Nicho Ecológico . 102

6.3 Conclusões 103

7 Conclusão **105**

Referências **108**

Capítulo 1

Introdução

A evolução tecnológica dos últimos anos permitiu que diversos domínios da ciência (biologia, física, astronomia, ecologia, entre outras) explorassem novos tipos de experimentos científicos, como, por exemplo, simulações do fluxo atmosférico [35] ou da dinâmica molecular de nanoestruturas [33]. Em termos gerais, são experimentos baseados em simulações que objetivam estudar o comportamento de um sistema real específico. Para realizar simulações do mundo real em ambientes computacionais, estes experimentos frequentemente fazem uso de modelos complexos e que, na maioria das vezes, podem consumir e/ou produzir uma grande quantidade de dados. Experimentos científicos completos podem ser compostos de vários sub-experimentos independentes, como no caso de avaliações *parameter-sweep*. A realização de um sub-experimento, pode ser uma tarefa complexa, formada por inúmeras aplicações menores que possuem um fluxo coerente de execução. Desta forma, a obtenção dos resultados dos experimentos, em tempos hábeis, fica cada vez mais dependentes da utilização de recursos computacionais de alto desempenho (*High Performance Computing - HPC*).

E-Science ou *eScience* inicialmente se referiu à ciência computacionalmente intensiva e que é normalmente realizada em ambientes distribuídos, ou ciência que usa imensos conjuntos de dados que exigem HPC. Desde então, a *e-Science* tem sido mais amplamente interpretada como "a aplicação da tecnologia de computação na realização de investigação científica moderna, incluindo a preparação, experimentação, coleta de dados, disseminação de resultados, armazenamento a longo prazo e acessibilidade de todos os materiais gerados através do processo científico". A descrição do processo para realizar o referido objetivo científico é frequentemente representada por um *workflow científico*, que expressa as etapas computacionais ou de análise de dados necessárias e suas dependências entre si. As tarefas de um *workflow* científico são organizadas (em tempo de configuração) e orquestradas (em

tempo de execução) de acordo com o fluxo de dados e, possivelmente, outras dependências, conforme especificado pelo cientista [22].

1.1 Motivação

Tanto a Europa como o Brasil reiteraram que as mudanças climáticas, a energia, a biodiversidade e o desenvolvimento sustentável são os desafios mais relevantes da atualidade e que exigem uma resposta global urgente. Juntamente com isso, a Europa e o Brasil reconheceram a necessidade de trabalhar juntos para alcançar o objetivo global de reduzir significativamente o atual índice de perda de biodiversidade e atualizar o plano estratégico da Convenção sobre Diversidade Biológica para o período pós-2010[1]. Uma posição que foi aplaudida e aprovada pela Assembléia Geral das Nações Unidas quando proclamou, na Resolução 65/161, o período de 2011 a 2020 como a Década das Nações Unidas para a Biodiversidade e convidou os estados membros a financiar as atividades da década[2].

A ciência da biodiversidade engloba um amplo espectro de áreas acadêmicas, estudando a vida nas mais variadas escalas, desde genes à espécies e de ecossistemas até paisagens. Cada vez mais, os cientistas da biodiversidade precisam considerar as múltiplas escalas temporais por meio das quais os ecossistemas e os serviços prestados operam e são impactados. Cada vez mais cientistas da biodiversidade têm de considerar dinâmicas não-lineares, incertezas, múltiplas dimensões e inter-relações de sistemas socioecológicos complexos. Para enfrentar os desafios globais associados à ciência da biodiversidade, são necessários mecanismos para a realização de pesquisas colaborativas em larga escala. Para atingir esse objetivo, os cientistas precisam de infraestruturas que ofereçam suporte à aplicações e-Science de forma eficiente para conduzir pesquisas interdisciplinares de alta qualidade, essas infraestruturas são denominadas e-infraestruturas [5, 3].

Devido à complexidade e a grande demanda de processamento e armazenamento exigido por *workflows* científicos atuais de diversas áreas de pesquisa, muitas vezes, a aquisição, operação e manutenção de sistemas computacionais que atendam os requisitos podem ter custos bastante elevados. Para reduzir os custos associados aos ambientes computacionais de alto desempenho, instituições tendem a juntar-se, através de acordos, dividindo os custos e compartilhando o uso dos recursos e, ao longo do tempo, novos equipamentos podem ser adquiridos e adicionados aos antigos recursos, formando um ambiente compu-

[1] Bruxelas, 6 de Outubro de 2009, 14137/09 (Presse 285), Third European Union-Brazil Summit Joint Statement.

[2] UN Notification 11 Janeiro 2011, http://www.cbd.int/doc/notifications/2011/ntf-2011-004-undb-en.pdf

tacional cada vez mais heterogêneo.

Um caminho alternativo, conhecido como computação em grade [29], amplamente adotado uma década atrás, onde a agregação de recursos computacionais distintos (e então heterogêneos) em instituições geograficamente distribuídos foi incentivada para formar um ambiente com maior capacidade de processamento e armazenamento para ser compartilhado entre seus usuários.

Devido ao alto custo monetário para adquirir e manter computadores com grande capacidade computacional (manutenção e energia elétrica), na última década, infraestruturas computacionais vem sendo oferecidas como um serviço. A Computação em Nuvem permite que o usuário adquira uma capacidade de computação redimensionável, sem preocupar-se com instalação e gerência dos recursos, pagando somente pelo tempo de uso dos recursos. Por exemplo: Amazon AWS [3].

Devido as tendências tecnológicas atuais, os servidores, o principal componente de e-infraestruturas (infraestruturas para *e-Science*), estão sendo projetados cada vez com mais recursos (núcleos de processamento, memórias, canais de entrada/saída, *etc.*) na tentativa de atender os requisitos energéticos e de desempenho esperados na era *exascale* [44]. Enquanto os três sistemas arquiteturais anteriores são de áreas de pesquisas distintas, leva-se em consideração suas tendências evolucionárias, no qual, pode-se identificar as prováveis características dos futuros ambientes computacionais, são elas: uma quantidade maior de sistemas heterogêneos, compartilhados, cada vez com uma escala maior, tanto em capacidade de processamento dos servidores, quanto em quantidade.

Os proprietários destes sistemas (tanto *clusters*, grades ou nuvens computacionais) têm um objetivo em comum que é maximizar o retorno de investimento. Do ponto de vista dos administradores, isso pode ser obtido maximizando a utilização dos recursos computacionais disponíveis. Infelizmente, nem sempre esse objetivo está em conformidade com os interesses de um usuário individual. Devido o compartilhamento de recursos, enquanto um conjunto de experimentos pode gastar menos tempo no total, individualmente um destes experimentos pode demorar muito mais tempo do que quando executado sozinho.

O desempenho de aplicações em servidores está fortemente relacionado com o sistema de gerenciamento de tarefas utilizado e a política de escalonamento adotada. Buyya em [87], destaca diversos métodos de escalonamento para ambientes computacionais dedicados e homogêneos, e considera como desafio o escalonamento de aplicações de *workflows* em ambientes computacionais com as seguintes características: ambiente computacional

[3] https://aws.amazon.com/pt/ec2/

compartilhado com muitos usuários competindo por recursos entre si; recursos que não são controlados exclusivamente por um único escalonador; recursos heterogêneos; recursos heterogêneos que podem não ter o mesmo desempenho para qualquer tarefa; *workflows* científicos que fazem uso intensivo de processamento e utilizam grandes conjuntos de dados entre suas tarefas. Além disso, é importante destacar que existem diversos sistemas gerenciadores de *workflows*, porém a maioria desses gerenciadores são centralizados e podem sofrer uma sobrecarga de processamento ao gerenciar uma grande quantidade de *workflows* ou recursos.

1.2 Objetivos

"O Brasil está emergindo no cenário global e pode ser mais que uma potência convencional. Ele abriga um quinto de todas as espécies conhecidas e dois terços das florestas tropicais existentes. Essa rica variedade de plantas e animais, ou a biodiversidade, pode fazer do Brasil uma potência verde." Alan Charlton, Embaixador Britânico ao Brasil[4]

A biodiversidade de animais, fungos, micróbios e espécies vegetais é fundamental para a sobrevivência da vida, mas os cientistas estão lutando para compreender a enorme variedade e riqueza que existe na terra. Em todo o mundo, mas em nenhum outro lugar mais agudo do que no Brasil, os cientistas estão lidando com a mudança climática e as mudanças no uso da terra, que irão alterar a distribuição e disponibilidade dessa biodiversidade - algumas espécies migrarão para outros lugares e outras tornar-se-ão extintas - mas como pode-se saber e planejar com antecedência?

Infelizmente, o ritmo de extinção de espécies é considerado bem mais acelerado que o ritmo da ciência na identificação e descrição de novas espécies. O Centro Nacional de Conservação da Flora (CNCFlora) do Jardim Botânico do Rio de Janeiro é referência nacional na geração, coordenação e difusão de informação sobre biodiversidade e na conservação da flora brasileira ameaçada de extinção. Até o ano de 2020, umas das atribuições do CNCFlora é o de avaliar o grau de risco de extinção de todas as espécies da flora do Brasil.

A modelagem de nicho ecológico (*Ecological Niche Modelling - ENM*) é uma das principais atividades realizadas pelos cientistas da biodiversidade para entender a distribuição das espécies. Os nichos ecológicos podem ser entendidos como espaços multidimensionais, com diferentes ótimos para cada espécie (por exemplo, temperatura, umidade) projetados sobre áreas geográficas como camadas. Os pontos de ocorrência das espécies são superpostos sobre as camadas e as correlações de ocorrências com fatores ambientais podem ser calculadas usando vários algoritmos.

[4]Extrato do seu artigo intitulado "O Brasil e sua Biodiversidade", no jornal Folha de São Paulo, 23 de maio de 2010.

Diferentes algoritmos mostram benefícios particulares, mas a escala na qual os cálculos são feitos é muito importante [27]. O ENM tem sido utilizado com sucesso para fazer predições e resolver problemas em muitas áreas, incluindo o manejo de espécies invasoras [67], o impacto das mudanças climáticas na biodiversidade [56], o planejamento da conservação da biodiversidade [66] e a biogeografia de doenças [65].

As ferramentas de modelagem são um outro tipo de recurso amplamente utilizado pelos cientistas da biodiversidade. Embora tradicionalmente os cientistas tenham usado apenas ferramentas desenvolvidas internamente, existe agora uma crescente demanda por ferramentas reutilizáveis. OpenModeller [70] é um dos primeiros e mais relevantes exemplos de ferramentas de acesso aberto na área da biodiversidade. Tem como objetivo proporcionar um ambiente flexível, amigável e multi-plataforma onde todo o processo de um experimento de modelagem de nichos pode ser realizado. O *software* inclui apoio para a leitura dos pontos de ocorrências das espécies e dos dados ambientais, permite a seleção de camadas ambientais nas quais o modelo deve basear-se para a criação de um modelo de nicho ecológico, além disso, realiza a projeção do modelo em um cenário ambiental. Dado que a avaliação do nicho ecológico de uma única espécie envolve a criação e teste de vários modelos, um experimento completo é formado por inúmeras tarefas com dependências entre si e sua representação pode ser dada por meio de um *workflow* científico.

Usando ENM como um estudo de caso, neste trabalho de doutorado, foi proposta uma abordagem para gerenciar *workflows* científicos em ambientes computacionais heterogêneos e compartilhados, buscando uma solução para:

- Integrar diferentes ambientes computacionais respeitando suas políticas locais sem ter que alterar o sistema de gerenciamento adotado em cada ambiente; e
- Prover uma forma mais eficiente de executar, em paralelo *workflows* científicos, inclusive, em ambientes que não oferecem suporte a essa classe de aplicação; e
- Melhorar a utilização dos recursos do ambiente, através de um mecanismo distribuído de gerência que permita a execução concorrente de múltiplos *workflows* científicos, compartilhando os mesmos recursos.

1.3 Contribuições

EUBrazilOpenBio - *Open Data and Cloud Computing e-Infrastructure for Biodiversity* (2011-2014) foi um projeto de pesquisa aplicado, financiado pela Comissão Europeia no âmbito do programa de cooperação, *Framework Programme Seven* (FP7), e pelo Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) do Ministério da Ciência e Tecnologia (MCT) do governo brasileiro, sob a chamada correspondente MCT/CNPq 066/2010. O projeto teve

como objetivo compreender como lidar com os desafios da pesquisa na área de biodiversidade de forma mais eficaz, através da criação e implantação de uma nova e-infraestrutura híbrida de dados e computação em nuvem. A e-infraestrutura foi projetada para suportar as necessidades e exigências da comunidade de pesquisa da biodiversidade, possibilitando a geração de novos conhecimentos através do desenvolvimento de dois estudos de casos. Esta tese de doutorado relata alguns dos trabalhos e resultados obtidos durante e depois a realização deste projeto, em relação a um destes estudo de casos - modelagem de nicho ecológico. Alguns dos principais destaques do trabalho são:

- A implantação da e-infraestrutura híbrida do EUBrazilOpenBio que:
  - inclui instalações de serviços de computação que permitem a execução otimizada de um experimento de ENM através da adoção de uma estratégia de gerenciamento de *workflows* não convencional (sistema de gerenciamento centralizado no próprio *workflow*);
  - é capaz de alavancar uma série de tipos de recursos distribuídos, tais como: clusters HPC (sem modificação de seus próprios sistemas de gerenciamento de carga de trabalho tradicionais, como, por exemplo, a plataforma *Load Sharing Facility* - LSF, *Portable Batch System* - PBS e *Sun Grid Engine* - SGE); HTCondor *pools* (também usados para a computação oportunista), e; infraestrutura de nuvem. Esta diversidade de recursos considerados tem como objetivo refletir sobre o cenário mais provável de tipos de recursos de infraestrutura que estariam disponíveis para a comunidade de biodiversidade.
- A ferramenta openModeller [70] - um *framework* genérico de *software* livre bastante utilizado pela comunidade de ecologia e de biodiversidade no mundo inteiro - foi adotada e teve a Etapa de Projeção otimizada para explorar as características intrínsecas de e-infraestruturas distribuídas.
  - Uma análise extensiva das diversas configurações possíveis foi realizada para avaliar a eficiência das implementações consideradas. Os principais resultados relacionado ao Projeto EUBrazilOpenBio foram resumidos no documento *Performance Profiling Report for the Ecological Niche Modelling Tool openModeller*.
  - Uma nova implementação do serviço *web* do openModeller foi desenvolvida (denominada *OpenModeller Web Service* - Versão 2, ou, simplesmente, OMWS2), capaz de receber requisições de *workflows* (tarefas envolvidas e suas dependências associadas). Desta forma, o gerenciamento das tarefas do *workflow* pode ser realizado diretamente no ambiente HPC, no qual o OMWS2 está localizado. Na primeira versão do OMWS, as requisições não possuem qualquer informação sobre as dependências entre as tarefas, exigindo que esse controle seja realizado exclusivamente pelo usuário.

- Um melhor aproveitamento dos ambientes distribuídos, explorando o uso da execução concorrente de *workflows* de ENM, com um gerenciamento mais próximo das máquinas de execução.
  - Em geral, as abordagens de escalonamento alocam, somente, um processo por núcleo de processamento. Enquanto a alocação de mais de um processo por núcleo da CPU pode melhorar a utilização de um servidor, esta técnica pode causar uma degradação significativa no desempenho da aplicação. O desafio é encontrar o equilíbrio correto entre essas duas estratégias (aumentar para melhorar a utilização do servidor ou reduzir para melhorar o desempenho individual da aplicação), na execução de múltiplos *workflows* que compartilham o mesmo ambiente computacional distribuído.

Este estudo de caso foi inspirado e construído com base na experiência do INCT - Herbário Virtual da Flora e dos Fungos[5], que utiliza um procedimento padronizado de modelagem de distribuição de plantas, que ocorrem no Brasil, após avaliação cuidadosa da qualidade dos dados de ocorrência disponíveis na rede speciesLink [80].

1.4 Organização

Esta proposta de tese está organizada da seguinte forma:

- O Capítulo 2 descreve e exemplifica os principais conceitos abordados nessa proposta (tais como: aplicação sequencial, aplicação paralela e *workflows* científicos), relaciona-se esses conceitos e apresenta-se os principais ambientes computacionais e sistemas de gerenciamentos associados, além dos métodos de escalonamento mais relevantes para cada tipo de aplicação.
- No Capítulo 3 descreve-se o estudo de caso adotado nesse trabalho, baseado em um problema real encontrado pelos pesquisadores, de diferentes países, da área de biodiversidade.
- No Capítulo 4 é descrito o problema de escalonamento de *workflows* em ambientes computacionais distintos, logo a seguir, apresenta-se a abordagem proposta para escalonar e gerenciar *workflows* científicos, integrando diferentes ambientes computacionais.
- O Capítulo 5 apresenta duas formas tradicionais de abordar o gerenciamento de *workflows* do problema exposto no Capítulo 4, por fim, apresenta-se a solução proposta nesse trabalho e uma correlação entre as três abordagens apresentadas.
- O Capítulo 6 trás alguns experimentos que foram utilizados na definição da infraestrutura computacional e na avaliação da abordagem proposta.

[5] http://inct.florabrasil.net/

- Por fim, o último capítulo traz as conclusões obtidas nesse trabalho e a indicação dos trabalhos futuros.

Capítulo 2

Aplicações em Ambientes de Alto Desempenho

Esse trabalho envolve a execução de aplicações que fazem uso intensivo de recursos computacionais, portanto, necessitam de ambientes computacionais de alto desempenho para obter um desempenho satisfatório. Esse capítulo abordará os principais temas relacionados nesse contexto, servindo de base para a compreensão dos conceitos envolvidos na abordagem proposta.

Inicialmente, na Seção 2.1, define-se os diferentes tipos de aplicações (sequencial, paralela, *workflow*), que compõem os termos fundamentais dessa proposta. Logo em seguida, ilustra-se como os conceitos anteriores estão fortemente relacionados entre si. A Seção 2.2 mostra as caraterísticas dos três principais ambientes de execução quando busca-se alto desempenho. Na sequência, a Seção 2.3 mostra alguns dos principais gerenciadores dos ambientes vistos e apresenta três distintas abordagens de gerenciamento. A Seção 2.4 apresentam alguns métodos de escalonamento em Sistemas Distribuídos, Grades Computacionais e *Workflows* Científicos, respectivamente.

2.1 Tipos de Aplicações

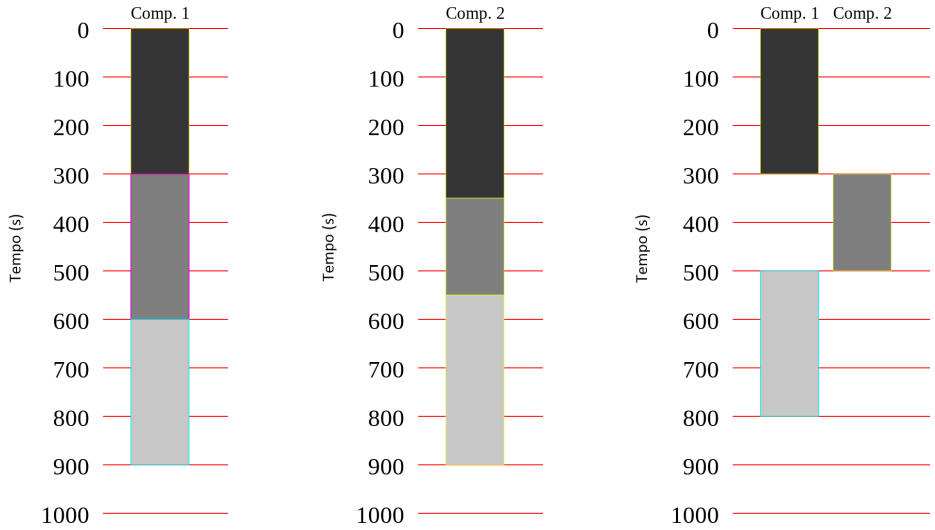
Essa seção tem como objetivo definir uma terminologia comum quando se refere os tipos de aplicações que serão abordadas nesse trabalho, assim como o relacionamento entre esses conceitos.

2.1.1 Aplicações Sequenciais

Aplicação é um programa que possui um objetivo específico. O termo "programa", possui a seguinte definição clássica, amplamente conhecida, que programa é um conjunto de instruções que descrevem uma tarefa a ser realizada pelo computador. O que diferencia as aplicações sequenciais tradicionais das aplicações sequências de ambientes de alto desempenho é a forma como essas

instruções são organizadas, buscando explorar as características dos recursos disponíveis para execução. O termo "recurso" será associado nesse trabalho, a qualquer recurso computacional que possa ser utilizado por uma aplicação, destaca-se: memória RAM, disco rígido, processador e até mesmo, pode-se fazer uso deste termo ao referenciar-se a computadores como um recurso que é formado por um conjunto de outros recursos.

A implementação modular é comum em aplicações complexas para simplificar o desenvolvimento e entendimento. Seguindo a máxima "dividir para conquistar", é possível reduzir tarefas complexas em módulos (blocos de instruções) menores. Desta forma, aplicações sequenciais complexas podem ser compostas por diversos módulos (menores que o original e mais simples de serem programados e compreendidos), além disso, pode-se explorar separadamente diferentes máquinas de um ambiente distribuído (utilizando o recurso que oferece o melhor desempenho para as características intrínsecas de cada módulo). Por exemplo, se um módulo exige uma grande quantidade de operações com pontos flutuantes e outro módulo necessita de um intenso acesso a disco, cada módulo pode ser encaminhado para ser executado no recurso que melhor atende suas características.



(a) Execução sequencial no computador 1.

(b) Execução sequencial no computador 2.

(c) Execução sequencial explorando as características de 2 computadores.

Figura 2.1: Execução Sequencial em mais de um computador.

A Figura 2.1 apresenta um exemplo de uma aplicação sequencial desenvolvida em módulos. Em (a), tem-se a representação da execução de uma única aplicação sequencial em um computador (identificado apenas como: Comp. 1), essa aplicação é formada por 3 etapas (módulos) e, nesse computador, foi necessário 300 segundos para executar cada uma das tarefas (representadas por cores diferentes). A parte (b) apresenta a execução da mesma aplicação em outro computador (identificado como: Comp. 2), onde, necessitou-se de 900 segundos para executar as três tarefas envolvidas. Porém, ao analisar o tempo de execução das tarefas nos dois computadores,

é possível observar que no Comp. 1, cada tarefa foi executada em exatamente 300 segundos; no Comp. 2, a primeira e última etapa foram executadas em 350 segundos e a segunda etapa necessitou de 200 segundos. Essa diferença de tempo está relacionada com as características da aplicação e do recurso computacional (por exemplo, uso de I/O, CPU, entre outros). A parte (c) da Figura 2.1 mostra o melhor escalonamento para essa aplicação sequencial, dado a disponibilidade apenas dos dois recursos analisados, obteve-se uma redução de 100 segundos (ganho de 11% em relação a execução em uma única máquina). Observa-se com esse exemplo, que a decisão do local de execução é primordial para obter o melhor desempenho da aplicação.

As aplicações sequenciais, por definição, podem possuir apenas um módulo ativo de cada vez (em um único processador) e possuem uma sequencia de execução que sempre deve ser mantida. Em contrapartida, as aplicações paralela, serão abordadas na Seção 2.1.2, são formadas por diversos módulos ativos em diferentes processadores (ao mesmo tempo).

Cada módulo é comumente conhecido por uma *tarefa*. O escalonamento, atividade responsável pela atribuição de cada *tarefa* em um recurso (processador) específico, é o principal desafio da computação distribuída. Esse problema surge sempre que as tarefas de uma aplicação precisam ser distribuídos entre vários computadores interligados, normalmente, busca-se minimizar os custos da execução de uma aplicação. O custo pode ser tempo, dinheiro, ou alguma outra medida de uso de recurso.

2.1.2 Aplicações Paralelas

Quando trabalha-se em ambientes computacionais de alto desempenho, normalmente busca-se o uso da Computação Paralela para realizar mais de uma tarefa ao mesmo tempo, empenhando-se em melhorar o desempenho da aplicação e, consequentemente, reduzir o tempo de execução. Isso implica que computação paralela envolve aplicações que executam instruções de forma concorrente (compartilhando recursos) ou de forma paralela (sem compartilhamento de recursos). Em alguns casos, o tempo total de uma aplicação paralela pode ser maior que uma sequencial, isso ocorre, normalmente, quando as tarefas paralelizadas são menores que o *overhead* associado com a criação e gerenciamento da paralelização.

A principal motivação de computação paralela é acelerar uma aplicação com a execução simultânea de parte das instruções existentes na aplicação [83]. Porém, o tempo não é a única medida no qual obtem-se vantagem com o uso da abordagem paralela. Existem cenários computacionais, nos quais a única chance de terminar um cálculo e alcançar uma solução é possuir um número determinado de processadores trabalhando simultaneamente em uma tarefa específica.

As duas formas mais usuais de aplicações paralelas, exploram o modelo de compartilhamento de memória e/ou o modelo de redes interconectadas, onde esses modelos diferenciam-se basica-

mente pelo grau de compartilhamento de memória primária entre os elementos de processamento que executam as tarefas.

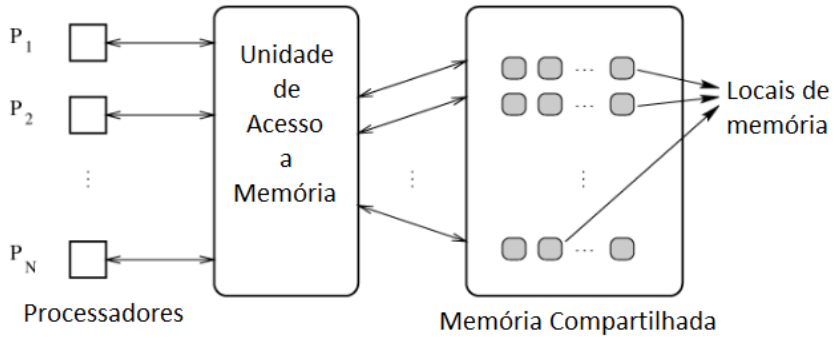


Figura 2.2: Modelo de memória compartilhada. Fonte: [83]

Aplicações que exploram o modelo de compartilhamento de memória, são baseadas em computadores formados por diversos núcleos de processamento que são conectados diretamente em uma memória global compartilhada. Por exemplo, na Figura 2.2, a unidade de acesso a memória (MAU - Memory Access Unit) permite que qualquer um dos N processadores acessem qualquer posição de memória, tanto para a escrita, como para a leitura. Essa memória é compartilhada e pode ser acessada, de forma assíncrona, por diferentes processos que estejam sendo executados de forma concorrente entre os processadores.

Aplicações que exploram, exclusivamente, esse tipo de modelo não são tão usuais pois essa arquitetura não é facilmente escalável para grande quantidade de memória e/ou grande quantidade de processadores. Aplicações que exploram esse tipo de paralelismo podem ser desenvolvidas, por exemplo, com o uso da *application programming interface (API)* OpenMP (*Open Multi-Processing*[1]).

Aplicações que utilizam redes interconectadas (*Interconnection Networks*) tentam evitar o problema encontrado no modelo de memória compartilhada (associado ao gargalo da utilização do MAU ao gerenciar muitos processadores com muitas posições de memória compartilhada), descentralizando o gerenciamento de memória e processador entre diversos computadores de uma rede.

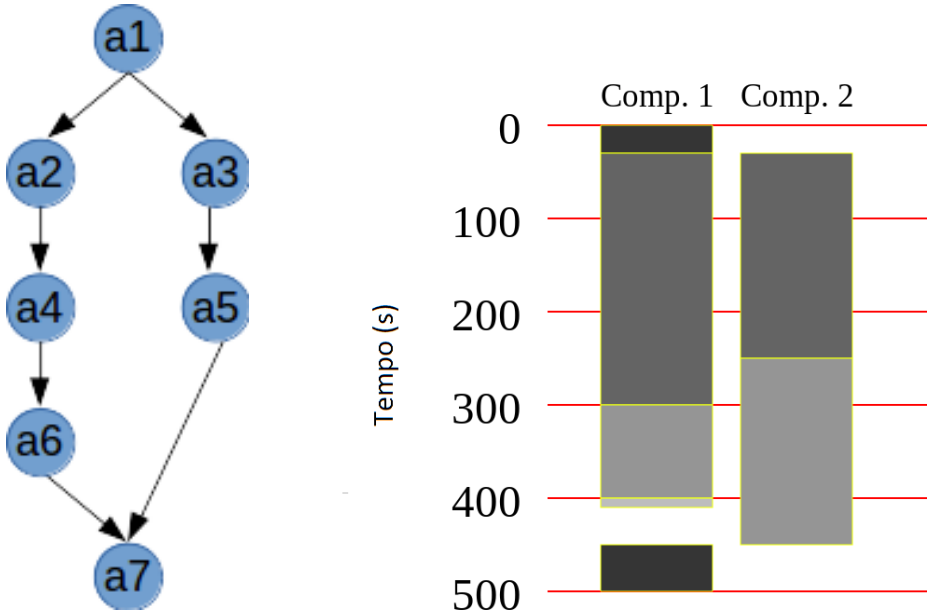
Nesse modelo, cada processador possui sua própria memória local (privada) e todas as solicitações para dados acontecem por comunicações entre os processadores, que ocorrem por meio de mensagens através dos links diretos que estão conectando-os (rede de computadores). É importante, nesse tipo de modelo, garantir comunicações rápidas entre computadores para não prejudicar os benefícios de execução paralela, e para isso necessita-se de uma boa topologia de rede.

Esse tipo de aplicação que envolve troca de mensagens entre computadores que estão inter-

[1]http://openmp.org

conectados são, frequentemente, implementadas com o uso da biblioteca MPI (*Message Passing Interface*). No padrão MPI, uma aplicação é constituída por um ou mais processos que se comunicam, acionando-se funções para o envio e recebimento de mensagens entre os processos. Inicialmente, na maioria das implementações, um conjunto fixo de processos é criado. Porém, esses processos podem executar diferentes programas. Por isso, o padrão MPI é algumas vezes referido como MPMD (*Multiple Program Multiple Data*). Para fazer uso do padrão de comunicação do MPI, cada aplicação precisa ser compilada com a biblioteca do MPI.

A relação das tarefas de uma aplicação paralela pode ser descrita em termos de um grafo de fluxo de dados ou grafo acíclico dirigido (GAD) (em inglês, DAG (*Directed Acyclic Graph*)) [43]. Em cada DAG, um nó representa uma *tarefa* e cada aresta entre dois nós representa a dependência de dados entre elas. Um DAG pode ser representado por $G = (V, E, \epsilon, \omega)$, onde G representa a aplicação paralela em si, formada pelos nós (ou nodos), identificados por $V = \{a1, a2, ..., a_n\}$, onde cada nó $a \in V$ corresponde a uma tarefa do grafo com peso $\epsilon(a)$, que indica a quantidade de operações realizadas por tarefa a. Enquanto que a aresta $(a1, a2) \in E$ representa a relação de precedência entre as tarefas $a1$ e $a2$, ou seja, a tarefa $a1$ deve completar sua execução e entrega os dados requeridos por tarefa $a2$ antes que a $a2$ comece. Associado a esta aresta pode existir um peso, $\omega(a1, a2)$, que informa a quantidade de dados enviada de $a1$ para $a2$.



(a) Representação de uma aplicação paralela com DAG.

(b) Escalonamento de uma aplicação paralela em dois computadores.

Figura 2.3: Execução Paralela.

A Figura 2.3 (a) mostra o DAG G de uma aplicação paralela formado por 7 tarefas, omitindo-se os pesos associados aos nós e arestas (ϵ e ω, respectivamente), onde G=($\{a1, a2, a3, a4, a5, a6, a7\}$,

$\{(a1, a2), (a1, a3), (a2, a4), (a3, a5), (a4, a6), (a5, a7), (a6, a7)\}$). Um possível escalonamento é apresentado em (b), onde o tempo total necessário para executar completamente essas tarefas é de 500 segundos. Observa-se, em especial, as três principais situações que ocorrem com aplicações paralelas, são elas:

(a) Sequencial: A execução de uma tarefa só pode acontecer após a execução da tarefa que a precede. Por exemplo, a Figura 2.3, apresenta a sequência de tarefas $a2$, $a4$ e $a6$; e a sequência $a3$ e $a5$.

(b) Divisão paralela (*fork*): Quando duas tarefas são executadas somente ao término de outra. Na Figura 2.3, a execução de $a1$ libera $a2$ e $a3$.

(c) Sincronização (*join*): Ocorre quando uma tarefa só pode ser executada quando duas ou mais concluírem. Na Figura 2.3, a7 só inicia após a execução completa de a5 e a6.

2.1.3 *Workflows* Científicos

Workflow é um termo amplo que historicamente está associado à automação total ou parcial de um processo de negócio onde documentos, informações ou atividades são passadas entre participantes de acordo com um conjunto bem definido de regras para alcançar ou contribuir para um objetivo global de negócios [72, 41]. Um Sistema de Gerenciamento de *Workflow* é responsável por definir, gerenciar e executar *workflows*, onde essa execução é baseada em uma representação lógica de todo *Workflow* [36]. Essa tecnologia é utilizada por diferentes áreas da ciência, como a biologia, física, astronomia, entre outras que não são comerciais [35, 33]. Essas áreas que normalmente realizavam (várias vezes os mesmos) experimentos manualmente e adotaram o uso de *workflows* como uma abstração para a especificação de seus experimentos, deram origem ao termo *workflow* científico, que hoje é usado para descrever *workflows* que compartilham as características de manipulação de grandes volumes de dados e que necessitam de processamento de alto desempenho [61, 81, 23]. Outras características que distinguem os *workflows* comerciais dos científicos são descritas em [49]: *Workflows* comerciais realizam poucas trocas de mensagens entre as tarefas e possuem complexos fluxos de controle; *workflows* científicos podem possuir fluxos de controle, mas são, principalmente, orientados ao fluxo de dados. Tipicamente, esses dados são volumosos, heterogêneos, distribuídos e são obtidos através da análise de outro conjunto de dados.

Um *workflow* científico tem seu ciclo de vida formado, basicamente, em 4 fases [10]:

- Projeto: A partir de uma hipótese científica a ser testada, elabora-se uma representação e uma abordagem que caracterize todas as etapas envolvidas com o objetivo experimental que deseja-se alcançar.

- Instanciação: Após a etapa que define o projeto, é necessário instância-lo com dados de entrada e os parâmetros definidos pelo cientista para execução.
- Execução: Essa fase corresponde, justamente, a execução de cada atividade (ou tarefa) que compõem o *workflows*. Durante a execução, normalmente, as tarefas geram dados que são utilizadas como dados de entrada para outras tarefas.
- Análise: A última etapa do ciclo de vida de um *workflow*, está relacionado com a avaliação dos resultados obtidos, que pode levar a rejeição da hipótese científica avaliada, revisão de alguma etapa do *workflow*, ou até mesmo, a rejeição do *workflow*.

Importante destacar que os *workflows* científicos são utilizados para gerenciar apenas atividades automatizáveis de experimentos, portanto, não envolvem (pouco ou) nenhum tipo de atividades manuais.

Da mesma forma que as aplicações paralelas, *workflows* científicos são frequentemente modelados com DAGs, devido a simplicidade dessa representação, onde os nós representam as *atividades* computacionais individuais e as arestas podem representar o fluxo de controle (ordem de execução) e/ou o fluxo de dados (forma como os dados são transferidos) entre as atividades [58]. A diferença entre as representações seria o nível de abstração dos nós. Enquanto em *workflows* as atividades são tipicamente instâncias de programas ou aplicações completos, nas aplicações paralelas, as tarefas são geralmente blocos de código ou processos da mesma aplicação. Nada impede que uma atividade de uma *workflow* seja uma aplicação paralela.

Em 1999, pesquisadores de Eindhoven University of Technology e Queensland University of Technology criaram o *The Workflow Patterns*[2], uma iniciativa que busca descrever detalhadamente o padrão de modelos de *workflows* para diferentes perspectivas relevantes, tais como: fluxo de controle e fluxo de dados. A Perspectiva de Controle de Fluxo descreve os aspectos relacionados com o fluxo de dependência entre as tarefas [73]. Enquanto a Perspectiva de Dados objetiva a passagem de informação entre as tarefas. Originalmente foram propostos mais de vinte padrões diferentes, atualmente, contabiliza-se 43 padrões distintos. Pode-se destacar, para ambas perspectivas, os seguintes padrões estruturais [74]:

a) Sequência: A Figura 2.4 (a), apresenta uma perspectiva de fluxo de controle, uma sequência de três tarefas é ilustrada, a tarefa subsequente só é habilitada após a execução da antecessora. Em (b), apresenta-se um DAG com a perspectiva de fluxo de dados e controle, portanto, a tarefa T1 recebe o dado D1 e gera o dado D2, que será utilizado como entrada para a tarefa T2.

b) Divisão Paralela(*fork*): A Figura 2.5 (a) mostra que após a execução de uma das tarefas, duas outras tarefas paralelas são habilitadas para execução. Em (b) e (c) é possível observar a

[2]http://www.workflowpatterns.com

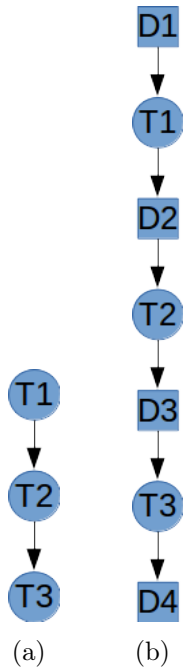


Figura 2.4: DAG de uma execução em sequência: (a) perspectiva de fluxo de controle. (b) perspectiva de fluxo de dados e controle.

representação do fluxo de controle e dados ao mesmo tempo, onde em (b) a tarefa a1 gera duas saída, uma para a2 e outra para a3, enquanto em (c), apenas uma saída é gerada para as tarefas subsequentes. Essa divisão pode ser realizada baseada em, basicamente, dois propósitos, são eles: produzir dados que serão utilizados por múltiplas tarefas ou para particionar um grande conjunto de dados em grupos menores, para serem processados simultaneamente por diferentes tarefas.

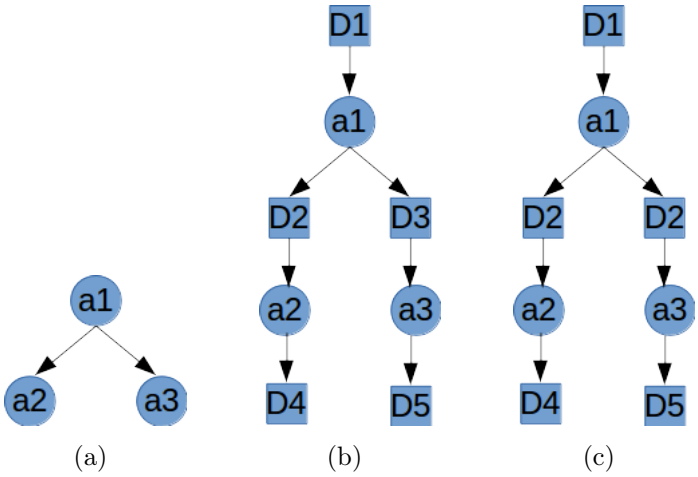


Figura 2.5: DAG de uma execução com divisão paralela: (a) perspectiva de fluxo de controle. (b) e (c) perspectiva de fluxo de dados e controle.

c) Sincronização (*join*): Duas tarefas precisam ser executadas para permitir a execução da tarefa subsequente (Figura 2.6). Essa atividade pode estar relacionada com a agregação de vários

resultados que serão utilizados com entrada para uma tarefa.

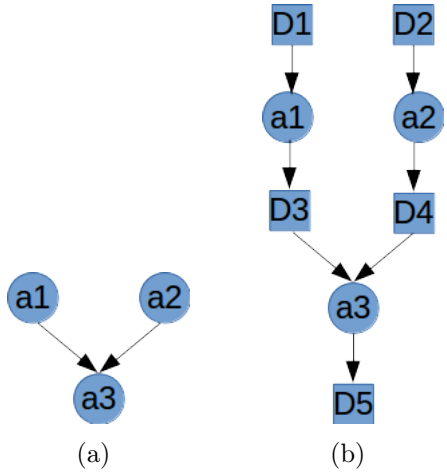


Figura 2.6: DAG de uma execução com sincronização: (a) perspectiva de fluxo de controle. (b) perspectiva de fluxo de dados e controle.

d) Escolha Exclusiva: Uma tarefa pode selecionar, de forma exclusiva, qual será sua próxima tarefa. Por exemplo, na Figura 2.7 ao término da tarefa *a1*, a tarefa *a2* ou *a3* serão executadas, mas não ambas.

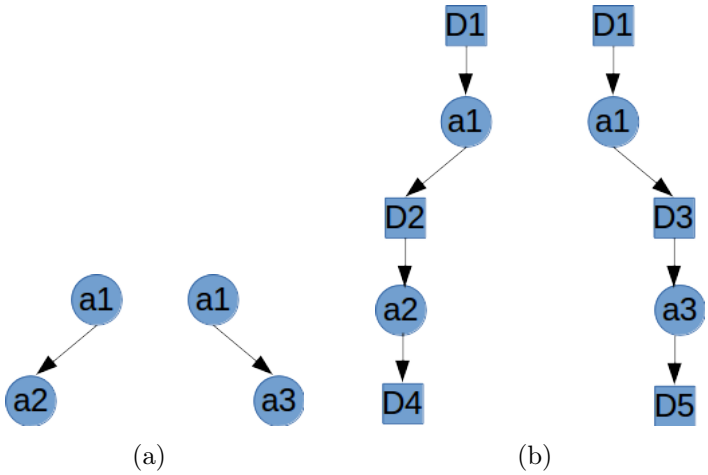


Figura 2.7: DAG de uma execução com divisão paralela: (a) perspectiva de fluxo de controle. (b) perspectiva de fluxo de dados e controle.

2.1.4 Relação entre os tipos de aplicações

Cada vez mais pesquisadores tentam resolver problemas por meio de simulações ou análise de um grande volume de dados. Pela complexidade intrínseca desses problemas, existe a necessidade de criar aplicações modulares que facilitam, tanto o entendimento do problema, como facilitam a implementação e validação das aplicações. Portanto, o uso dos três conceitos anteriores são

amplamente utilizados em conjunto (aplicações sequenciais, aplicações paralelas e *workflows*). A Seção 2.1.1 mostra que aplicações complexas, criadas de forma modular, podem tirar proveito de ambientes heterogêneos sendo submetidos para o recurso que melhor atende a cada tarefa; A Seção 2.1.2 apresenta a vantagem do uso de paralelizar aplicações, permitindo que tarefas independentes sejam executadas de forma paralela (concorrendo ou não por recursos), buscando minimizar, desta forma, o tempo total de execução. Por fim, a Seção 2.1.3, fornece um conceito organizacional, que permite um entendimento global do fluxo de trabalho das "tarefas" envolvidas. Essas três abordagens são utilizadas em conjunto na execução de *workflows* científicos.

DAGs são utilizados para modelar *workflows* científicos de diferentes áreas, tais como aplicações de astronomia, biodiversidade, ecologia, meteorologia, entre outras [23]. Como exemplo, será apresentado, brevemente, um exemplo da bioinformática, descrito em [46], que introduz o SciEvol [57], um *workflow* científico para a reconstrução da evolução molecular que visa inferir relações evolutivas em dados genômicos. O SciEvol para ser executado no SGWfC (Sistemas Gerenciadores de Workflows Científicos) Chiron [59], que será abordado na Seção 2.3.3, consiste em 7 fases, conforme DAG da Figura 2.8.

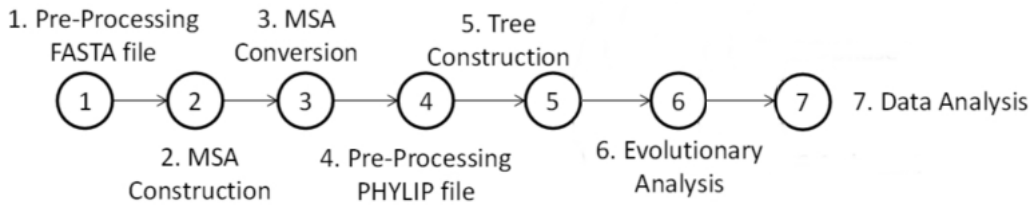


Figura 2.8: *Workflow* do SciEvol. Adaptado de [46].

A Fase 6 (*Evolutionary Analysis*) é responsável por analisar a árvore filogenética com parâmetros correspondentes a cada um dos seis modelos de substituição de códon, que são usados para verificar se os grupos de genes estão sob a seleção positiva de Darwin. Como essa fase explora a mesma aplicação, apenas explorando diferentes parâmetros, a Fase 6 pode ser paralelizada com uma tarefa independente para cada modelo de substituição de códon (conforme Figura 2.9).

A Figura 2.9 mostra o uso de DAG para ilustrar o fluxo de controle do *workflow* do SciEvol, composto por aplicações sequenciais (etapas 1 à 5), divisão paralela ao término da etapa 5, aplicações paralelas na etapa 6 e sincronização de todas as tarefas paralelas na etapa 7.

Com problemas cada vez mais complexos, cientistas frequentemente necessitam executar múltiplas aplicações paralelas ou *workflows*. Hoje, *jobs*, *Workflows* científicos e aplicações paralelas usufruem de Sistemas Distribuídos na tentativa de obter um melhor desempenho. Por motivos econômicos, provedores oferecem os mesmos ambientes computacionais distribuídos para estas três classes de trabalho, podendo ser formado por diversos recursos heterogêneos, sem a necessidade de estarem próximos fisicamente e, visando uma maior eficiência, permitem o compartilhamento dos recursos.

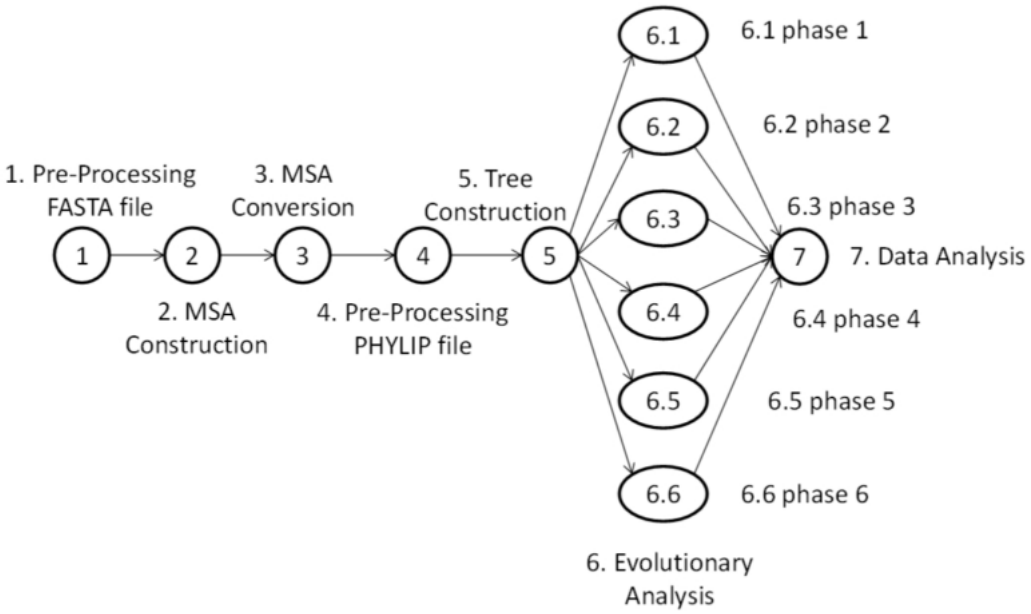


Figura 2.9: *Workflow* do SciEvol. Fonte: [46].

2.2 Ambientes de Execução

Computação de alto desempenho (*High Performance Computing - HPC*) refere-se geralmente à prática de agregação de poder computacional de uma forma que oferece um desempenho muito maior do que se poderia obter de um típico computador *desktop* ou estação de trabalho, a fim de resolver grandes problemas em, por exemplo, ciência, engenharia ou negócios. A chave de ter um sistema efetivo de HPC é de conseguir que os nós individuais trabalhem juntos facilmente para resolver um dado problema. Nas últimas três décadas, esses ambientes computacionais podem ser classificados em três principais categorias: *Clusters*, Grades Computacionais e Computação em Nuvem (*Cloud Computing*) [39, 25].

2.2.1 *Cluster*

A computação em *cluster* é formada pelo uso de vários computadores, com dispositivos de armazenamento e interligações redundantes, formando um único sistema com alta disponibilidade. A redundância aumenta a disponibilidade dos recursos e permite o balanceamento de carga em sistemas de alto tráfego.

O objetivo da computação em *cluster* é fornecer um ambiente eficiente e com alta disponibilidade, de forma transparente aos usuários. Os sistemas de gerenciamento de *cluster* permitem o balanceamento de carga, processamento paralelo, metodologias de escalonamento, monitoramento e tolerância a falhas. Os algoritmos de balanceamento de carga são projetados essencialmente para distribuir a carga em processadores e maximizar a utilização enquanto minimizam o

tempo total de execução da tarefa. O objetivo é minimizar o custo total de execução e comunicação encontrado nas tarefas.

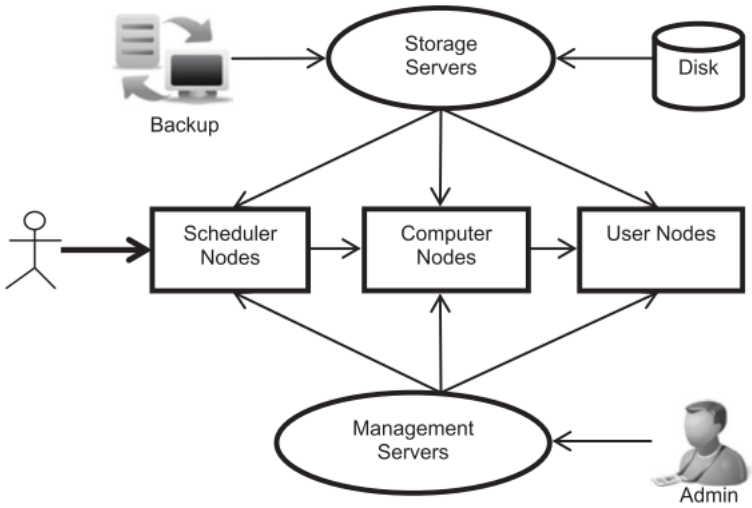


Figura 2.10: Arquitetura de um sistema de computação em *cluster*. Fonte: [39].

A Figura 2.10 mostra a arquitetura usual de um *cluster*, onde existe um servidor de armazenamento (*storage server*) que gerencia o acesso aos disco de forma transparente para todos os nós computacionais existentes. Além disso, esse ambiente necessita de um sistema de gerenciamento, capaz de permitir o acesso de usuários, o funcionamento em conjunto dos recursos e o escalonamento das tarefas submetidas pelos usuários (monitoramento, manutenção, entre outras tarefas). Como exemplo de gerenciadores de *clusters*, pode-se citar: HTCondor, PBS, LSF, OpenSSI, entre outros [39].

2.2.2 Grades Computacionais

O conceito de computação em grade é mais amplo, agregando *clusters* distintos e até mesmo computadores pessoais de baixo custos através do uso de redes de longa distância, como a internet, para permitir ampla disponibilidade.

Uma grade computacional é uma infraestrutura de hardware e software que fornece acesso confiável, consistente, difundido e barato a recursos computacionais de ponta [29]. Uma grade computacional diz respeito, sobretudo, ao agrupamento em grande escala de recursos, sejam ciclos de processamento, dados, sensores ou pessoas. Tal agrupamento requer quantidades significativas de *hardware* para conseguir as interconexões e *software* necessários para monitorar e controlar o conjunto resultante. O sistema de gerenciamento de grades mais conhecido é o Globus Toolkit [39].

Diferentemente do modelo de uso exclusivo adotado por *clusters*, descrito em 2.2.1, ambientes de grades sao compartilhados por ser formado por diferentes redes com diferentes domínios

administrativos, e portanto, com políticas de acesso e usuários independentes. Um exemplo desse ambiente é ilustrado na Figura 2.11.

Um grupo de indivíduos e instituições formam uma organização virtual (*Virtual Organization*). Nessas organizações, os indivíduos e as instituições definem regras para o compartilhamento de recursos (essas regras podem estar relacionadas com a restrição de arquivos ou recursos, conforme o usuário). Sistemas em Grade permitem acesso seguro através de certificados X.509 para a identificação de usuários e recursos computacionais. Grades computacionais diferenciam-se dos sistemas convencionais de HPC (*cluster*), por serem menos acopladas, e mais distribuídas, heterogêneas e geograficamente dispersas.

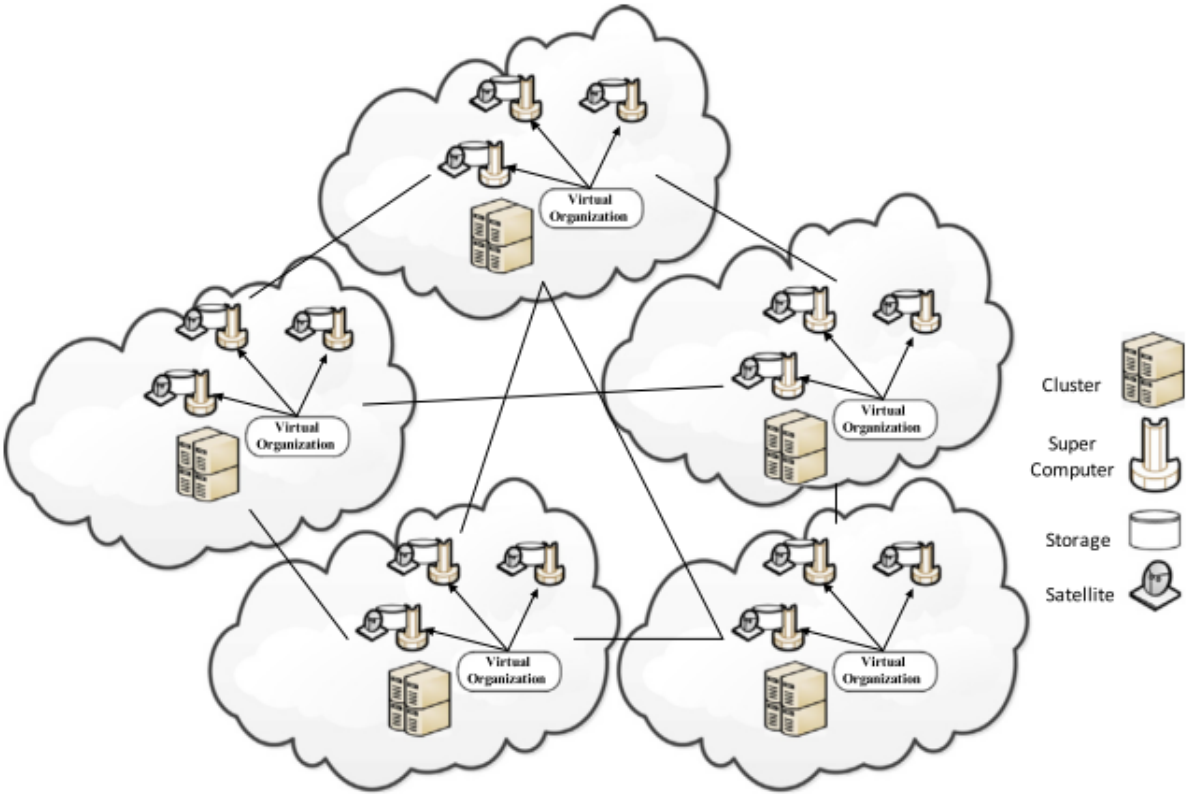


Figura 2.11: Modelo de um sistema de computação em *grid*. Fonte: [39].

2.2.3 Computação em Nuvem

A Computação em Nuvem é um modelo para permitir onipresente e conveniente acesso sob-demanda pela rede para um conjunto compartilhado de recursos computacionais configurável (por exemplo, redes, servidores, armazenamento, aplicações e serviços) que podem ser fornecidos e liberados rapidamente com um mínimo do esforço do cliente e interação do provedor de serviço[3].

[3]Mell, P., Grace, T.: The NIST Definition of Cloud Computing (Draft), National Institute of Standards and Technology (Janeiro 2011), http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf

As cinco características essenciais da computação em nuvem são:

1. Auto-atendimento sob demanda: Um consumidor pode unilateralmente obter capacidades de computação, como ciclos de CPU do servidor e armazenamento de rede, conforme necessário, automaticamente e sem necessidade de interação humana com cada provedor de serviços;

2. Acesso à rede: as capacidades estarão disponíveis por meio da rede e serão acessadas através de mecanismos padrão que promovem a utilização por plataformas diversas de (por exemplo, celulares, *tablets*, computadores portáteis e estações de trabalho);

3. Agrupamento de recursos: Os recursos de computação do provedor são agrupados para atender múltiplos consumidores usando um modelo de vários locatários, com diferentes recursos físicos e virtuais dinamicamente atribuídos e reatribuídos de acordo com a demanda do consumidor. Há uma sensação de independência de localização no sentido de que o cliente geralmente não tem controle ou conhecimento sobre a localização exata dos recursos fornecidos, mas pode especificar a localização em um nível mais alto de abstração (por exemplo, país, estado ou datacenter). Exemplos de recursos incluem armazenamento, processamento, memória e largura de banda da rede.

4. Elasticidade: As capacidades podem ser elasticamente provisionadas e liberadas, em alguns casos automaticamente, para escalar rapidamente para fora e para dentro proporcionais à demanda. Para o consumidor, os recursos disponíveis para fornecimento muitas vezes parecem ser ilimitados e podem ser apropriados em qualquer quantidade a qualquer momento.

5. Serviço medido: Os sistemas em nuvem controlam e otimizam o uso de recursos automaticamente, alavancando uma capacidade de medição em algum nível de abstração apropriado ao tipo de serviço (por exemplo, armazenamento, processamento, largura de banda e contas de usuário ativas). O uso de recursos pode ser monitorado, controlado e relatado, proporcionando transparência para o provedor e o consumidor.

A computação em nuvem pode ser classificada em diferentes formas, as três principais são:

1. IaaS (*Infrastructure as a Service* ou Infraestrutura como serviço): Oferece uma infraestrutura computacional conforme a necessidade do usuário. O exemplo mais conhecido é o *Amazon Elastic Compute Cloud* (Amazon EC2)[4], capaz de fornecer um ambiente computacional dinâmico e sob demanda de máquinas virtuais. No IaaS, assim como no PaaS e SaaS, é utilizado o modelo *pay-per-use*, onde a cobrança é baseada no serviço e não em

[4]aws.amazon.com

produto, ou seja, no caso da Amazon EC2, se houver a necessidade de utilizar 10 servidores por um mês, a Amazon EC2 disponibiliza esses servidores por este período determinado, através de um contrato de aluguel.

2. PaaS (*Plataform as a Service* ou Plataforma como Serviço): Prove plataformas específicas para o usuário. PaaS pode fornece um conjunto de serviços básicos, incluindo banco de dados, armazenamento e servidores virtuais. É possível usar esses serviços para criar aplicativos sobre essas plataformas, com as ferramentas e APIs fornecidas pela plataforma PaaS, com isso, os usuários finais dos aplicativos não precisam fazer download, instalar ou fazer qualquer tipo de manutenção no sistema ou nos servidores. Por exemplo, o Google App Engine[5] é uma plataforma para criação de aplicativos da Web dimensionáveis e *back-ends* móveis. O App Engine dimensionará o aplicativo de acordo com a quantidade de tráfego que ele necessitar. Desta forma, o desenvolvedor paga somente pelos recursos utilizados.

3. SaaS (*Software as a Service* ou Software como Serviço): Nuvens desse tipo oferecem um software online para seus usuários. Um *Software* como serviço é uma aplicação que não é instalada em nenhum computador localmente, é usado como um serviço, pela internet. Esses serviços podem ser pagos ou gratuitos. Exemplo de SaaS gratuitos: Gmail, Dropbox, Google Drive, Overleaf[6], entre outros. Algumas vezes esses serviços gratuitos podem oferecer funcionalidades pagas, como é o caso do Gmail que oferece mais espaço de armazenamento e o Overleaf que oferece sincronização com Dropbox, projetos maiores, entre outras funcionalidades, através de uma taxa mensal de uso.

A infraestrutura de nuvem refere-se aos componentes de *hardware* e *software* - como servidores, armazenamento, rede e software de virtualização - necessários para suportar os requisitos de computação de um modelo de computação em nuvem. Além disso, as infraestruturas em nuvem incluem uma camada de abstração de software que virtualiza recursos e apresenta logicamente aos usuários através de meios programáticos. Os recursos virtualizados são hospedados por um provedor de serviços em *clusters* dentro de um *datacenter* e entregues aos usuários através de uma rede ou da Internet. Os usuários não sabem em quais tipos de máquinas suas solicitações vão ser realizadas e quanta concorrência teria para cada máquina. Em principio, os provedores só garantem uma capacidade de processamento e armazenamento mínima.

[5]cloud.google.com
[6]www.overleaf.com

2.3 Sistemas de Gerenciamento

O uso de ambientes de alto desempenho exige que a realização de seus *jobs* seja coordenada numa forma eficiente (e de preferência sem a necessidade de interação com o usuário). Desta forma, existem diferentes gerenciadores automatizando etapas específicas ao longo da execução de aplicações em ambientes HPC.

2.3.1 Gerenciadores de Recursos

O problema de alocação de tarefas se enquadra nas categorias dos problemas NP-Difíceis, e que torna árdua a busca de encontrar uma solução ótima em tempo hábil. Avanços tecnológicos, como servidores de múltiplos processadores com cada vez mais núcleos, está motivando a tendência de consolidação de servidores e um aumento no grau de execução concorrente de aplicações. Infelizmente, estes avanços não tem conseguido aumentar a capacidade de todos os recursos de servidor igualmente, por exemplo, acesso a memória principal ou secundário. O aumento de concorrência por estes recursos tem impacto negativo no desempenho das aplicações e que então vão requerer atenção especial do escalonador. Será apresentado a seguir os principais tipos de gerenciadores de recursos envolvidos nesse trabalho.

2.3.1.1 *Portable Batch System - PBS*

O sistema PBS (*Portable Batch System - PBS*) é responsável por alocar os *jobs* em recursos computacionais disponíveis. Frequentemente utilizado em ambientes de *cluster* UNIX. Desde que começou a ser utilizado pela NASA em 1994, recebeu diversas melhorias e modificações, dividindo-se em três diferentes versões:

1. OpenPBS : Versão original (*open source*) lançada em 1998 e que atualmente foi descontinuada.
2. TORQUE *Resource Manager* : Baseou-se na Versão OpenPBS e é mantido pela Adaptive Computing [7].
3. PBS *Professional* : Versão comercial oferecida por Altair Engineering[8].

O *Terascale Open-source Resource and QUEue Manager* (TORQUE) é um software *open source* capaz de gerenciar *jobs* em recursos computacionais distribuídos. Desde a separação do OpenPBS, avançou-se muito nas áreas de escalabilidade, confiabilidade (tolerância a falhas)

[7] http://www.adaptivecomputing.com/products/open-source/torque/
[8] http://www.pbspro.org/

e funcionalidades (escalonamento, usabilidade, entre outras), sendo atualmente utilizado em dezenas de milhares de ambientes governamentais, acadêmicos e comerciais em todo o mundo.

O escalonador de *job* (*job scheduler*) é uma aplicação responsável por alocar e executar os *jobs* de forma autônoma. Essa execução não interativa de processos é chamada de *batch processing* [1]. Os escalonadores gerenciam filas de *jobs* no *cluster* de computadores. Para cada fila é possível associar um conjunto de recursos e uma política de escalonamento, que será responsável por definir a prioridade de cada *job* para ser adiciona a fila e posteriormente submetida aos recursos.

2.3.1.2 HTCondor

HTCondor [37] é um sistema de gerenciamento de carga de trabalho para aplicações de computação intensiva em clusters e largamente utilizado em sistemas de recursos distribuídos (dedicados e/ou compartilhados). O HTCondor foi lançado em 1984 pela Universidade de Wisconsin-Madison, e fornece mecanismos de enfileiramento e priorização de tarefas, políticas de escalonamento e monitoração de recursos.

As aplicações paralelas ou sequenciais submetidas pelos usuários são enfileiradas pelo sistema HTCondor que baseado em uma política escolhe quando e onde vai executar os trabalhos. É possível configurar a quantidade de *slots* disponíveis em cada recurso, isso representa a quantidade de processos que serão executados simultaneamente em cada máquina.

O escalonador HTCondor é capaz de gerenciar um conjunto de nós computacionais dedicados ou não dedicados (O HTCondor possui mecanismos que permitem tirar proveito dos computadores apenas quando estiverem ociosos) [1]. Na Figura 2.12 é apresentada a arquitetura do HTCondor que é formada por um nó computacional que é chamado de gerenciador central, e nós de execução e de submissão (um mesmo nó pode assumir mais de uma dessas funções), esse conjunto de nós computacionais é chamado de "Condor Pool".

Cada tipo de nó está composto por um conjunto de programas com as seguintes características [1]:

- Gerenciador Central (*Central Manager*): Para cada *pool* computacional do HTCondor existe um Gerenciador Central (*Central Manager*), responsável por monitorar os recursos computacionais do *pool* coletando informações de características e uso (mecanismo realizado através do *daemon condor_collector*). A partir dessas informações, o *daemon condor_negotiator* aloca um recurso do *pool* para a execução de um *job* que esteja na fila, pronto para execução.

- Nós de submissão (*Submit Machine*): O *daemon_schedd* é responsável por submeter as tarefas para a fila de execução. Posteriormente, o *daemon shadow* fica responsável por

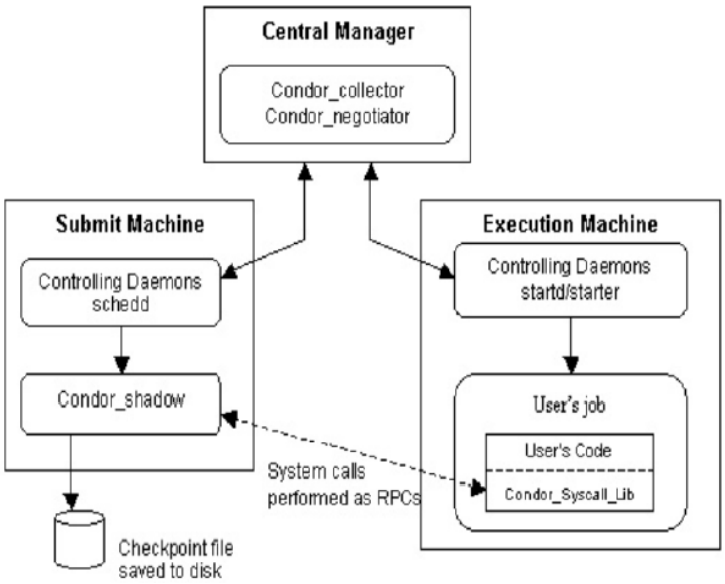


Figura 2.12: Arquitetura HTCondor. Fonte: [1]

gerenciar a execução remota das tarefas e controlar o estado de cada tarefa para efeito de *checkpointing* (que em caso de falha poderá reescalonar a tarefa incompleta).

- Nós de execução (*Execution Machine*): O *daemon_startd* é utilizado para executar as tarefas no recurso (em nome do usuário) e informar ao gerenciador central as informações relacionadas ao uso do recurso.

Os nós de execução podem passar por diferentes estados, como apresenta a Tabela 2.1:

Tabela 2.1: Possíveis estados dos recursos computacionais.

Estado	**Descrição**
Idle	Sem jobs em execução.
Busy	Executando Job.
Suspended	Job está sendo suspenso.
Vacating	Job está sendo checkpointed.
Killing	Job está sendo finalizado.
Benchmarking	Executando benchmark, ou seja, avaliando a capacidade do host.

O usuário precisa basicamente de três etapas para executar uma tarefa no HTCondor:

1. Definir o universo que será utilizado: Universos são os ambientes de execução. Destacam-se três tipos básicos de universos: o *standard*, o *Vanilla* e o *Parallel*. O universe *standard* fornece recursos avançados como migração das tarefas e serviço de *checkpoint*. Porém, o uso desse universo exige que os programas sejam relinkados com as bibliotecas do HTCondor, com o uso do comando condor_compile.

O universo *Vanilla* não exige que os programas sejam relinkados para serem executados. Entretanto, não é possível utilizar os recursos avançados existentes no universo *standard*.

O uso do universo *Parallel* permite que várias máquinas sejam alocadas para a execução dos programas submetidos.

2. Criação de um arquivo de submissão de Tarefa: Esse arquivo é essencial para especificar todas as necessidades e características da tarefa, contendo informações como: comando a ser executado, arquivo de entrada, plataforma requerida, entre outras informações relevantes.

3. Submissão da Tarefa: O arquivo criado na etapa anterior é submetido com o uso do comando condor_submit. Com isso o HTCondor se encarregará de gerenciar a execução da tarefa, monitorando o seu progresso. Quando a execução chegar ao fim, o HTCondor irá informar ao usuário algumas estatísticas de desempenho.

2.3.2 Gerenciadores de Aplicação

Diferentemente dos Sistemas Gerenciadores de Recursos que objetivam maximizar a utilização dos recursos disponíveis no ambiente computacional, oferecendo serviços como escalonamento e tolerância a falhas, baseando-se apenas nos estados dos recursos. O Sistema de Gerenciamento de Aplicação (AMS - *Application Management System*) utiliza o ambiente computacional conforme sua disponibilidade e as características específicas de cada aplicação.

2.3.2.1 EasyGrid AMS

O EasyGrid AMS é um *middleware* para a execução de aplicações MPI em ambientes computacionais de alto desempenho, típicamente, grades computacionais. Seu objetivo é encapsular aplicações MPI, tornando-as autônomas, com características autoadaptativas, eficientes e robustas para serem executadas em ambientes compartilhados. O EasyGrid AMS é embutido na aplicação que deseja-se executar, em tempo de compilação, sem que nenhuma alteração seja realizada no código fonte original do usuário. Além disso, foi desenvolvido utilizando uma implementação padrão do MPI, o que garante a portabilidade da aplicação gerada.

A arquitetura do SGA EasyGrid é formada por diferentes escalonadores dinâmicos, distribuídos hierarquicamente, conforme Figura 2.13. O GDS é o Escalonador Dinâmico Global, responsável por gerenciar as tarefas entre diferentes sites. O SDS é o Escalonador Dinâmico de Site, responsável por gerenciar as tarefas entre os *hosts* de um mesmo conjunto de computadores, e, por último, o HDS, Escalonador Dinâmico de *host*, responsável por gerenciar as tarefas na própria máquina.

O sistema de comunicação entre os processos da aplicação foi alterado e não existe comuni-

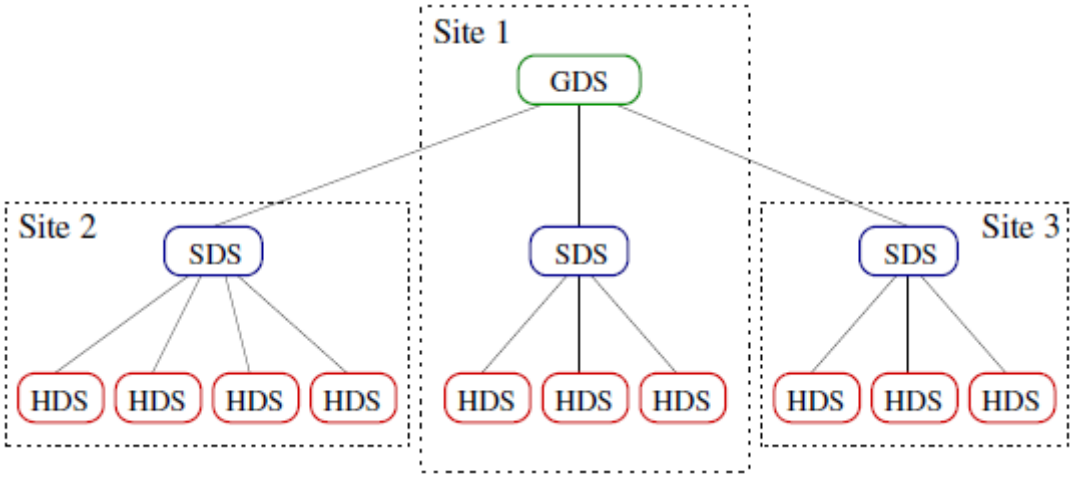


Figura 2.13: Hierarquia de escalonadores dinâmicos do EasyGrid AMS. Fonte: [19].

cação direta entre os processos, toda troca de mensagem é interceptada pelos gerenciadores, que analisam a mensagem e repassam para o destinatário. Essa interceptação das mensagens serve para colaborar com o monitoramento hierárquico da execução das tarefas. Sempre que uma tarefa termina, o seu gerenciador direto e superiores são comunicados, facilitando o balanceamento de carga entre as máquinas envolvidas.

Cada nível de escalonador é independente e pode possuir uma politica de escalonamento independente [77, 76, 54, 17, 52, 19, 18, 15]. Para executar o EasyGrid AMS é necessário criar arquivos de configuração para cada nível de escalonamento, definindo a forma de gerenciamento das entidades abaixo.

- Arquivo de configuração do GDS: O arquivo de configuração desse nível precisa indicar quantos SDS ele irá gerenciar, qual a politica de escalonamento adotada e se irá utilizar tolerância a falhas.
- Arquivo de configuração do SDS: Cada linha desse aquivo representa um HDS e deve indicar a politica de escalonamento associada e a quantidade máxima de processos concorrentes.
- Arquivo de configuração do HDS: Esse arquivo possui o escalonamento estático inicial, indicando onde cada tarefa deverá executar. Se a quantidade de processos concorrentes for alcançada durante a execução, os demais processos serão colocados em uma lista de espera e seu escalonador acima poderá enviar a solicitação do processo para outra máquina ou site.

2.3.3 Gerenciadores de *workflows* existentes

Um sistema de gerenciamento de *workflow* (WfMS - *Workflow Management System*) é um sistema que define, gerencia e executa *workflows* controlando a ordem das atividades representadas, por exemplo, por um DAG [11].

Existem diversos Sistemas de Gerenciamento de *Workflows* que são voltados a HPC, tais como: E-Science Central [85], Askalon [28, 71], Pegasus [21, 20], Swift/T [84], entre muitos outros. Nesse trabalho específico, priorizou-se destacar alguns WfMS que são utilizados pela comunidade científica que utiliza o OpenModeller (estudo de caso deste trabalho), são eles: Taverna, Kepler e HTCondor - DAGMan, que serão, respectivamente, descrito nas próximas seções.

2.3.3.1 Taverna

O Taverna[9] [82, 38, 6] é um conjunto de ferramentas open-source criado pelo grupo myGrid[10] [50] para projetar, executar e monitorar *workflows* científicos, mesmo que o usuário não tenha um extenso conhecimento em programação (com *scripts* e serviços Web). Muito utilizado em diferentes áreas científicas, como: bioinformática, biodiversidade, química, astronomia, mineração de dados, análise de documentos e imagens, entre outros. As ferramentas do Taverna incluem o Workbench (aplicativo desktop cliente), a ferramenta de linha de comando (para rápidas execuções de *workflows* a partir do terminal), o server (que gerencia o *workflow*) e um player (interface *web* para submissão dos *workflows*).

A construção de *workflows* no Taverna é feita através de componentes disponibilizados publicamente pela comunidade científica na forma de serviços. A rede social MyExperiment [32] facilita o compartilhamento e a reutilização de *workflows* científicos dentro da comunidade científica. Atualmente existem mais de dois mil *workflows* para o Taverna. Inclusive, a Figura 2.14 mostra o *workflow* associado a criação do modelo de nicho ecológico do openModeller (objeto de estudo desse trabalho). Os dados de entrada e saída são apresentados em linhas pontilhadas, que indicam todas as informações associadas. O fluxo de execução das tarefas internas pode ser visto na parte mais interna da figura.

Um *workflow* definido graficamente é representado textualmente na linguagem Scufl (*Simplified Conceptual Workflow Language*), baseada em XML. A execução do *workflow* ocorre quando o arquivo codificado em Scufl é interpretado pelo Motor de Execução do Taverna.

[9] http://www.taverna.org.uk
[10] http://www.mygrid.org.uk

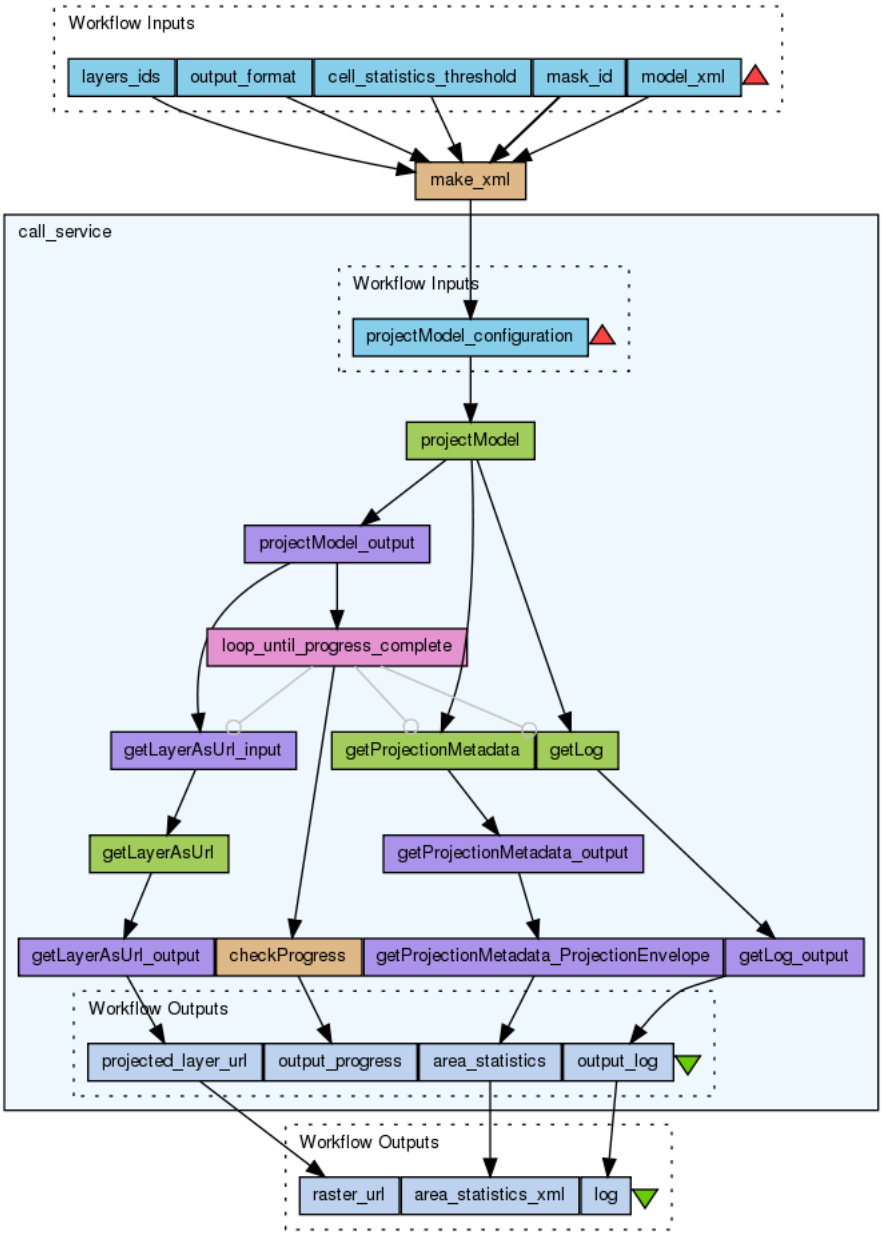


Figura 2.14: *Workflow* associado a criação do modelo de nicho ecológico do OpenModeller. Fonte: [32].

2.3.3.2 Kepler

O projeto Kepler [42] é dedicado a apoiar aplicações da comunidade científica, fornecendo apoio a dados armazenados em diferentes formatos, dados locais ou remotos, e é um ambiente eficaz para a integração de outros *softwares* comumente utilizados, tais como: *script* "R" com o código compilado na linguagem C, facilitando a execução remota e a execução de modelos distribuídos.

O Kepler utiliza um paradigma orientado a atores, onde cada ator pode desempenhar uma atividade que pode ser atômica ou composta. Os atores projetados podem consumir ou produzir

dados e podem comunicar-se com outros atores através de canais. O Kepler também permite o uso de uma entidade independente, denominada diretor, responsável por definir a ordem de execução dos atores e a comunicação. Diferentes tipos de diretores são disponibilizados na interface gráfica, e cada diretor possui um tipo de modelo de computação associado, por exemplo: diretores que permitiram que os autores sejam executados de forma sequencial ou paralela.

É possível integrar o Kepler com grades computacionais que utilizam o Globus. Diferentes atores são criados para cada atividade, um *Job* do Globus é representado no Kepler através da criação do ator *RunJobGridClient*. A Figura 2.15 apresenta a interface do Kepler ao modelar um *workflow* como exemplo. Qualquer aplicação acessível por linha de comando pode ser integrado ao *workflow* com o uso do ator *CommandLine*.

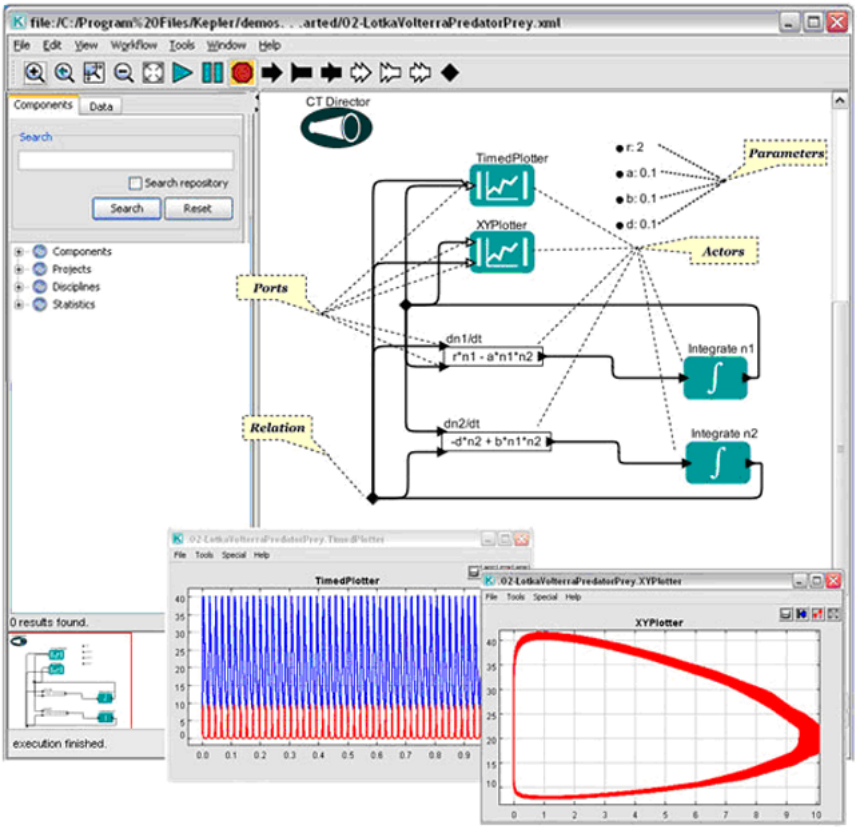


Figura 2.15: *Workflow* modelado com o Kepler. Fonte: [42]

2.3.3.3 HTCondor - DAGMan

O DAGMan (*Directed Acyclic Graph Manager*) é um meta-escalonador do HTCondor capaz de gerenciar as dependências entre os Jobs em um nível superior ao escalonador HTCondor. Um DAG pode ser usado para representar um conjunto de tarefas onde a entrada, saída ou execução de uma ou mais tarefas são dependentes de outras tarefas. O DAGMan, por definição, não é um escalonador de *workflows* científicos, mas possui todas as características necessárias para

gerenciar o controle das dependências de um *workflow*.

Cada *workflow* que será submetido para execução, necessitará de um arquivo que descreva o DAG (com suas respectivas dependências). O DAGMan utiliza esse arquivo para controlar as tarefas estão prontas para execução (sem dependência ou já executaram as suas dependências). Para submeter o *job* para o *pool* HTCondor, necessita-se do mesmo arquivo de submissão descrito na Seção 2.3.1.2. Pois o DAGMan é apenas um controlador de dependências que é executado um nível acima do HTCondor, submetendo as tarefas que não possuem dependências.

2.3.4 Orientação

Na prática, gerenciamento pode ser realizado focando em interesses de vários pontos de vistas. Essa seção aborda os principais paradigmas de execução dos sistemas de gerenciamento. Podem ser classificados basicamente em três formas distintas [19]:

2.3.4.1 Sistema de Gerenciamento de Usuário

O Sistema de Gerenciamento de Usuário (UMS - *User Management System*) está relacionado com a visão do usuário, no qual a aplicação é gerenciada com suas características internas e com os requisitos indicados pelo usuário. Esse tipo de sistema costuma trabalhar com aplicações ou *jobs* do tipo *bag-of-task* e é considerado um sistema de gerenciamento descentralizado, pois tipicamente cada usuário submete e gerencia sua própria aplicação. Por definição, os sistemas UMS não são considerados totalmente autônomos porque sistemas autogerenciáveis devem ser capazes de tomar decisões sem a influência dos usuários. Como exemplo desse tipo de gerenciamento, pode-se considerar o *script* de um usuário ou portais de grades que fornece o *script* que precisa realizar todo o gerenciamento de dependências. Portanto, o *script* submete uma aplicação para execução em um recurso e fica monitorando o término para obter os dados de saída e ter condições de utilizá-lo como dados de entrada para outra aplicação, que será submetida logo em seguida (respeitando as dependências pré-existentes).

2.3.4.2 Sistemas Gerenciadores de Recursos

A maioria dos ambientes de alto desempenho utilizam Sistemas Gerenciadores de Recursos (RMS - *Resource Management System*). Esse tipo de sistema permite que o usuário não se preocupe com a forma de execução das aplicações na infraestrutura, e tendem a maximizar a utilização do ambiente, independentemente dos requisitos e características internas das aplicações. Esses gerenciadores costumam ser centralizados, tendo total controle sobre todo o sistema e são autogerenciáveis, sem necessitar da intervenção do usuário para tomar decisões. Por exemplo, HTCondor e TORQUE são gerenciadores do tipo RMS.

2.3.4.3 Sistema de Gerenciamento de Aplicação

O Sistema de Gerenciamento de Aplicação (AMS - *Application Management System*) não fica junto com o usuário (igual ao UMS), e não precisa ser instalado e configurado em todo o ambiente computacional (igual o RMS). Esta abordagem destaca-se por criar uma aplicação, contendo todas as tarefas que precisam ser gerenciadas. Desta forma, pela proximidade da aplicação (gerenciador), tarefas que devem ser executadas e *hosts*, as aplicações AMS podem ser gerenciadas segundo os seus próprios requisitos e sua topologia interna, facilitando o uso de políticas customizadas que podem ser ajustadas durante a execução, conforme o desempenho da aplicação na infraestrutura.

Além disso, por estarem acopladas a aplicação que será executada, o escalonamento é descentralizado em relação a todo ambiente, pois cada aplicação gerencia apenas as suas próprias tarefas. A Tabela 2.2 apresenta um quadro comparativo entre essas três formas de gerenciamento distintas:

Tabela 2.2: Diferentes formas de gerenciamento de tarefas. Fonte: [19]

	Orientação	**Gerenciamento**	**Escalonamento**	**Instalação**
UMS	usuário	descentralizado por usuário	centralizado	instalado nos recursos
RMS	sistema	centralizado	centralizado	instalado nos recursos
AMS	aplicação	descentralizado por aplicação	descentralizado por aplicação	embutido na aplicação

2.4 Escalonamento de Tarefas

O escalonamento de processos ou tarefas é a etapa de associar um processo para ser executado em uma CPU (*Central Processing Unit*). Em sistemas distribuídos, o escalonamento é uma das etapas mais importantes, visto que uma gerência errada pode sobrecarregar um recurso e deixar outro ocioso, afetando muito o desempenho da aplicação. Um *resource broker* ou escalonador é um componente intermediário que atua como mediador entre os usuários e os recursos computacionais, tendo como objetivo alocar (e/ou escalonar) recursos e gerenciar a execução do *workflow* ou aplicação de um ou vários usuários [62]. A seguir, será apresentado os principais métodos de escalonamento de acordo com o ambiente e/ou tipo de aplicações.

2.4.1 Escalonamento de Sistemas Distribuídos

Enquanto a utilização de sistemas distribuídos aumenta o poder computacional disponível, apenas o escalonador do sistema operacional não é capaz de resolver o problema de alocação entre as diferentes máquinas envolvidas. Sistemas distribuídos necessitam de um escalonador global para alocar a máquina que executará as tarefas, além de otimizar a execução dos processos objetivando diferentes metas dependendo da orientação do gerenciador, e que podem ser entre outras:

- Aumentar o *throughput* (vazão): A quantidade de tarefas finalizadas em uma unidade de tempo.
- Reduzir o tempo de resposta: Tempo total de execução da perspectiva do usuário.
- Otimizar a utilização de recursos: Busca utilizar ao máximo os recursos envolvidos, reduzindo o tempo de ociosidade dos recursos e a quantidade de recursos usados.
- Balancear a carga: Busca-se distribuir as cargas entre os recursos existentes para evitar que um recurso fique ocupado por muito tempo, enquanto os outros ficam ociosos.

Diferentemente dos escalonadores de sistemas operacionais, os escalonadores globais enfrentam a dificuldade de atuar com recursos heterogêneos, fraco acoplamento e gargalos no escalonador devido a necessidade de conhecer o estado de cada recurso do ambiente.

2.4.1.1 Aplicações *Bag-of-Tasks*

Aplicações *Bag-of-Tasks* são compostas por tarefas independentes, onde, de forma geral, corresponde as tarefas que podem ser executas em qualquer recurso, desde que seus arquivos de entrada estejam disponíveis. Essa classe de aplicações são mais simples de serem escalonadas, permitindo o uso de políticas baseadas no uso de CPU livre, velocidade da CPU, tamanho das tarefas a serem executadas, raramente considerando largura de banda existente. Na Tabela 2.3 apresenta-se algumas das principais políticas utilizadas no escalonamento de aplicações *Bag-of-Tasks*:

2.4.2 Escalonamento em Grades Computacionais

Frequentemente as políticas de escalonamento de tarefas de sistemas distribuídos são aplicadas em grades computacionais, porém, estes métodos de escalonamento de aplicações *Bag-of-Tasks* nem sempre oferecem o melhor resultado devido a característica intrínseca das grades computacionais. Características que devem ser levadas em conta ao escalonar tarefas em grades:

- Quantidade de recursos : O gerenciamento de grandes quantidades de recursos pode sobrecarregar o escalonador.

Tabela 2.3: Abordagens de escalonamento em aplicações compostas por *bag-of-tasks* [26, 87].

Método	**Descrição**
Round-Robin (RR)	É uma técnica que consiste em distribuir de forma cíclica aos recursos, as tarefas que forem sendo submetidas ao escalonador. Esse método é simples e rápido, porém não coleta nenhum tipo de informação dos recursos ou das tarefas, consequentemente, costuma gerar um grande desbalanceamento de carga entre os recursos.
Workqueue	Consiste em uma variação da política RR. Esse método, também, não necessita de informações dos recursos e das tarefas, porém, a alocação da tarefa ocorre de forma dinâmica, somente quando o recurso está disponível. O desbalanceamento é menor que o RR, porém, o escalonador pode submeter grandes tarefas para um recurso lento, no final da execução da aplicação.
Max-Min	Esse método atribui as maiores tarefas para o recurso que possui maior capacidade de concluir primeiro, criando um melhor balanceamento entre todos os recursos (comparado com as políticas RR e *Workqueue*). Porém, essa política necessita de conhecimento da aplicação e do recurso para tomar a decisão do melhor local para alocação da tarefa. Necessita basicamente de: tamanho da tarefa (quantidade de instruções), porção do recurso que está sendo utilizada (carga de trabalho atual do recurso) e velocidade do recurso (valor relativo a quantidade de execuções que consegue resolver por unidade de tempo).

- heterogeneidade de recursos: A diversividade de recursos deve ser considerada durante o escalonamento.
- Compartilhamento de recursos: O uso compartilhado de recursos pode afetar significativamente o desempenho da execução das tarefas.
- Movimentação de dados: Deve-se evitar a comunicação excessiva entre os recursos para evitar o congestionamento na rede, gerando degradação do desempenho.

Dada as características anteriores, observa-se que o escalonamento centralizado pode, facilmente, ser sobrecarregado com tantas informações e gerenciamentos. Portanto, uma alternativa para Grades é o uso de escalonadores hierárquicos e escalonadores descentralizados.

2.4.2.1 Escalonador Centralizado

Um escalonador centralizado é único em todo o sistema, portanto, ele deve gerenciar todas as requisições (novas, em execução e concluídas) e monitorar todos os recursos. Como, por definição, as Grades podem ser formadas por diversos conjuntos de computadores (com seus próprios

escalonadores), a quantidade de computadores pode, realmente, ser muito elevada, causando um gargalo no escalonador centralizado.

2.4.2.2 Escalonador Hierárquico

Escalonadores hierárquicos dividem as tarefas com outros escalonadores. Basicamente, existe um escalonador central para gerenciar outros escalonadores que serão responsáveis por gerenciar seus clusters ou máquinas locais.

Como um exemplo, o trabalho [51] apresenta uma estrutura hierárquica de escalonadores para melhorar a escalabilidade do ambiente de Grade. A grade computacional é formada por grupos de recursos, e cada grupo possui um escalonador local. Cada escalonador local primeiro tenta alocar e executar uma tarefa localmente em seu grupo, somente se não for possível, o escalonador comunica-se com outros escalonadores da grade para realizar a transferência ou recepção de tarefas.

Um modelo de escalonamento de recursos com a abordagem MTTR - *Minimum Total Time to Release* é apresentado. Conforme a Figura 2.16, essa abordagem possui duas camadas hierárquicas de escalonamento, a primeira camada é denominada *Grid Level*, a outra camada é denominada *Cluster Level*. Essa estratégia possui três passos: o primeiro passo está relacionado com a atualização do *Global Information System*; o segundo passo é a tomada de decisão, que avalia as tarefas que precisam ser executas e adota uma abordagem de escalonamento de acordo com os recursos disponíveis; e o terceiro passo associa-se a transferência de tarefas para algum grupo de computadores. Os resultados da simulação indicam que a arquitetura hierárquica resulta em um melhor tempo total (tempo de execução + tempo de espera + tempo de I/O) médio do que uma arquitetura centralizada empregando mesmo algoritmo.

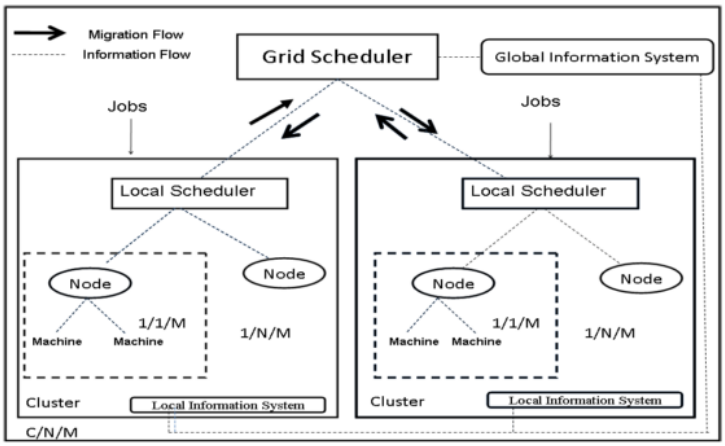


Figura 2.16: Arquitetura Hierárquica do sistema de grade. Fonte: [51]

Esse trabalho porém apresenta algumas ineficiências no seu escalonador hierárquico de dois níveis. O escalonador do *Grid Level* é responsável por decidir para qual *cluster* irá enviar as

tarefas baseando-se unicamente no poder computacional disponível (em MIPS), sem considerar, por exemplo, a memória disponível do recurso. E o escalonador do *Cluster Level* fica responsável pelo escalonamento em todos os recursos locais, aplicando a mesma política de escalonamento para todas as tarefas (o que nem sempre é eficiente). Esse trabalho também não considera recursos compartilhados no mesmo momento por mais de um usuário (situação típica de muitos ambientes computacionais), fator que pode prejudicar a eficiência do escalonamento.

2.4.2.3 Escalonador Distribuído

Em escalonadores distribuídos não existe um gerenciamento global, o gerenciamento ocorre de forma totalmente independente, evitando gargalos com sobrecarga de monitoramento e gerenciamento.

Um exemplo desse tipo de gerenciamento seria a execução de múltiplas aplicações paralelas cada um com seu EasyGrid AMS [19, 15]. O EasyGrid AMS é um sistema gerenciador de aplicação que é encapsulado junto com aplicações do tipo *bag-of-tasks* ou DAG [18]. Desta forma, cada aplicação pode autogerenciar-se de acordo com suas características e sua topologia interna e adequar-se a carga de trabalho de cada computador que está sendo utilizado.

Cada aplicação do EasyGrid AMS é formada por um conjunto de processos MPI com funções hierarquicamente distribuídas, composto por processos gerenciadores e agentes escalonadores que possibilitam políticas distintas entre as tarefas, além de prover a aplicação do usuário a capacidade de automonitoramento, autoconhecimento e conhecimento do ambiente. Esse monitoramento constante do ambiente de execução permite que a aplicação se auto-ajuste, otimizando a execução e se reconfigurando quando necessário.

O EasyGrid AMS possui características que oferecem escalabilidade (gerenciamento e escalonamento descentralizados entre várias aplicações que compartilham o mesmo ambiente computacional), flexibilidade (o escalonamento de tarefas pode ser dividido em níveis, tornando mais simples e flexíveis as políticas de escalonamento) e autonomia ao gerenciamento da aplicação nos recursos disponíveis [15, 19].

- Estrutura de gerenciamento do EasyGrid AMS: Os processos gerenciadores são encapsulados automaticamente na aplicação do usuário, permitindo que a aplicação seja capaz de se autogerenciar. A Figura 2.17, ilustra um exemplo de uma grade computacional com três sites, onde é possível visualizar a arquitetura de uma aplicação EasyGrid AMS composta por três níveis, os componentes que compõe essa hierarquia são:
  - GM: O gerenciador global fica no topo da hierarquia supervisionando os sites da grade.

– SM: O gerenciador de site é responsável por alocar os processos da aplicação no seu site.

– HM: O gerenciador de máquina existe em cada máquina e é responsável por escalonar, criar e executar os processos do usuário no seu respectivos recursos.

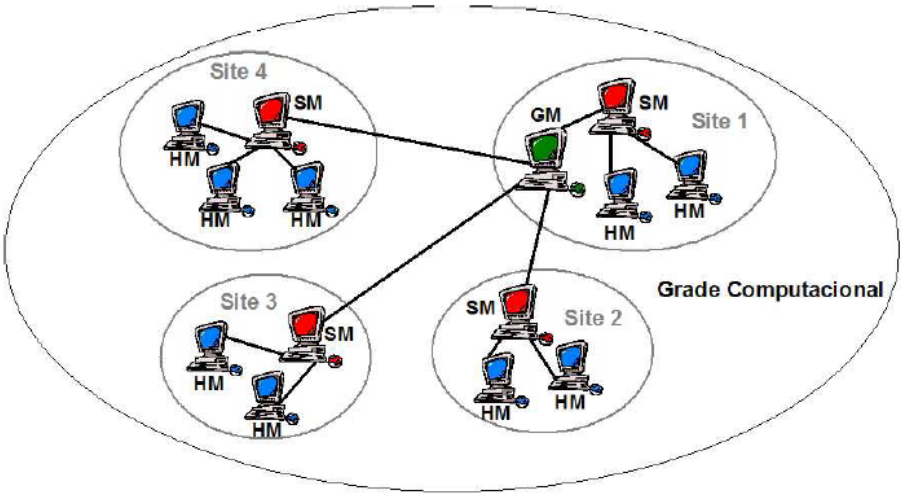


Figura 2.17: Hierarquia de processos gerenciadores do EasyGrid AMS. Fonte: [19]

O EasyGrid AMS foi desenvolvido para trabalhar com aplicações paralelas e não foi planejado para atuar especificamente com a complexidade dos *workflows* científicos. Porém, possui muitas características relevantes que podem ser aproveitadas nessa proposta de tese, são elas: capacidade de automonitoramento, autoconhecimento, conhecimento do ambiente, gerenciamento e escalonamento descentralizado, tolerante a falhas, entre outras. Além de todas essas características possui código aberto, o que permite que seja modificado para incluir as necessidades específicas para trabalhar com *workflows* científicos.

2.4.3 Escalonamento de *Workflows*

Escalonamento é necessário na fase de execução de *workflows*. *Workflows* em sistemas distribuídos podem ser vistos como um conjunto de tarefas relacionadas que são processadas em recursos distribuídos em uma ordem bem definida para atingir um objetivo específico [86]. Técnicas de gerenciamento de *workflows* têm sido desenvolvidas por muitos anos, no qual, muitas delas podem ser aplicadas no contexto de ambientes distribuídos, mas com as devidas restrições que o ambiente proporciona, tais como: recursos heterogêneos, múltiplos domínios de gerenciamento, disponibilidade e o uso de características dinâmicas [12], assim como os problemas relacionados com o fluxo de dados [88].

O processo de escalonamento de um *workflow* consiste em selecionar e definir recursos específicos para executar todas as tarefas que formam o *workflow* [8]. Esse planejamento envolve três

importantes características para a tomada de decisão [49], que são idênticas no escalonamento de aplicações paralelas:

a) A gerência da tomada de decisão: responsável por decidir a forma de mapeamento. Pode ser classificada em três formas: Centralizado (único escalonador central), Hierárquico (um escalonador central controla a execução do *workflow*, e mapeia partes do *workflow* para outros escalonadores) e Descentralizado (formado por vários escalonadores que se comunicam entre si sem possuir um escalonador central).

b) A abrangência do mapeamento está relacionada com o número de tarefas associadas ao *workflow*, essa decisão pode ser local (baseado nas informações da tarefa corrente), ou global (baseado em todas as tarefas correntes).

c) A tomada de decisão pode ser estática (o *workflow* é gerado antes da execução das tarefas) ou dinâmica (o *workflow* é gerado/modificado durante a execução das tarefas).

Por fim, o mapeamento pode ser orientado de acordo com um objetivo, e mantendo a relação de dependência existente. Esse objetivo pode ser, por exemplo, uma solução que forneça um menor tempo de execução do *workflow*, um menor número de computadores necessários, um menor custo financeiro (supondo uso de nuvens computacionais pagas), um menor volume e tempo de comunicação ou simplesmente a seleção de recursos que forneçam maior confiabilidade.

O escalonamento de *workflow* é uma das questões-chave no gerenciamento da execução dessas aplicações [69]. Por isso, muitas heurísticas têm sido desenvolvidas para escalonar tarefas interdependentes em ambientes dedicados e homogêneos [34]. Os algoritmos de escalonamento de *workflow* necessitam de um modelo de *workflow* abstrato que define as tarefas do *workflow*, sem especificar a localização onde essas tarefas serão executadas. Existem dois tipos de modelos abstratos de *workflow*: determinístico (as dependências das tarefas e os dados de entrada e saída são conhecidos com antecedência) e não-determinístico (conhecimento em tempo de execução). Esse trabalho aborda apenas o modelo de *workflow* abstrato determinístico representado como um Grafo Acíclico Direcionado (DAG). Os principais algoritmos de escalonamento baseado em melhor esforço são fundamentados em heurísticas ou meta-heurísticas [87] (custos monetários não são considerados). Esses algoritmos objetivam executar completamente todas as tarefas o quanto antes, ou minimizar o *makespan* da aplicação *workflow*. O *makespan* da aplicação é o tempo que decorre do início da aplicação até que todas as saídas estejam disponíveis ao usuário.

As abordagens baseadas em heurísticas são métodos que desenvolvem um algoritmo de escalonamento com o objetivo de encontrar soluções para um problema particular, enquanto que a abordagem baseada em meta-heurísticas desenvolvem algoritmos que proporcionam uma solução para atender um determinado tipo de problema. Ambas abordagens não oferecem garantia de

resultados eficientes [87]. Os algoritmos baseados em heurísticas podem ser classificados em três grandes grupo, conforme Tabela 2.4.

Além dos algoritmos baseados em heurística que possuem o objetivo de encontrar uma solução para um problema específico, tem-se as meta-heurísticas que buscam resolver uma classe de problemas, sendo aplicadas em problemas mais complexos, fornecem uma maneira eficiente de se mover rapidamente em direção a uma solução muito boa. Muitas meta-heurísticas têm sido aplicadas para resolver problemas de escalonamento de *workflows*, tais como: GRASP, Algoritmos Genéticos e *Simulated Annealing*, Tabela 2.5. Porém são abordagens geralmente caras computacionalmente e são usadas em escalonamentos estáticos. Como ambientes ficam cada vez mais compartilhados, algoritmos de qualidade e baixo custo são mais apropriados para escalonamento dinâmico.

Tabela 2.4: Abordagens de escalonamento de *workflow* baseadas em heurísticas

Algoritmo	**Descrição**
Escalonamento de Tarefas Individuais (*immediate mode*)	Esse é o método mais simples para escalonamento de aplicações *workflow*, abordagem gananciosa que faz o escalonamento com base apenas em uma única tarefa individual. O algoritmo Myopic[87] tem sido implementado em alguns sistemas de Grade, como, por exemplo, no DAGMan do HTCondor. O algoritmo escalona uma tarefa pronta, ainda não mapeada, para o recurso que espera-se que irá executar completamente a tarefa primeiro, esse procedimento repete-se até que todas as tarefas sejam escalonadas.
Escalonamento de Lista (*List Scheduling*)	Basicamente o *List Scheduling* faz uma lista ordenada de processos, com base em suas prioridades. Normalmente, existem duas grandes fases nesse tipo de algoritmo, a primeira fase está relacionada com a priorização das tarefas (define a prioridade de cada tarefa com um valor de classificação e gera uma lista de escalonamento) e a segunda fase é a seleção dos recursos (seleciona as tarefas na ordem de suas prioridades e mapeia cada tarefa selecionada em seu recurso ideal, se nenhum recurso atende a necessidade da tarefa, então seleciona-se a próxima tarefa da lista) [69, 47].
Escalonamento baseado em *Clusterização*	Os algoritmos dessa classe baseiam-se na localização dos dados, tentando evitar o tempo de transmissão dos resultados entre tarefas interdependentes de dados, desta forma é capaz de reduzir o tempo de execução. Esses algoritmos baseiam-se no escalonamento de grupos de tarefas, alocando um grupo de tarefas interdependentes para o mesmo recurso [13].

Tabela 2.5: Abordagens de escalonamento de *workflows* baseados em meta-heurísticas.

Método	**Descrição**
Greedy Randomized Adaptive Search Procedure (GRASP)	É uma técnica de pesquisa iterativa randomizada, comumente aplicada a problemas de otimização combinatória. Como diversos métodos construtivos, o algoritmo GRASP consiste em criar uma solução inicial e depois efetuar uma busca local para melhorar a qualidade da solução. A solução inicial é obtida de forma gulosa (*Greedy*), aleatória (*Randomized*) e adaptativa (*Adaptative*) [7].
Simulated Annealing	Consiste em uma técnica de busca local probabilística, e se fundamenta em uma analogia com a termodinâmica (processo de recozimento de metais, onde o sólido é aquecido além do seu ponto de fusão, onde o material passa por vários estados possíveis, e resfriado lentamente, obtendo-se uma estrutura cristalina em um estado de baixa energia). No algoritmo, a cada iteração do método, a solução atual é substituída por uma solução aproximada (na sua vizinhança no espaço de soluções), escolhida de acordo com a sua função objetivo. Essas iterações são repetidas até encontrar um resultado aproximado satisfatório [69].
Algoritmo Genérico	Essa meta-heurística não é determinística e trabalha com uma população de soluções simultaneamente, combinando-as em busca de soluções cada vez melhores [47]. Algoritmos genéticos diferem dos algoritmos tradicionais de otimização em basicamente quatro aspectos: • Baseiam-se em uma codificação do conjunto das soluções possíveis, e não nos parâmetros da otimização em si; • os resultados são apresentados como uma população de soluções e não como uma solução única; • não necessitam de nenhum conhecimento derivado do problema, apenas de uma forma de avaliação do resultado; • usam transições probabilísticas e não regras determinísticas

Capítulo 3

Estudo de Caso: Modelagem de Nichos Ecológicos com openModeller

Esse capítulo descreve o estudo de caso adotado para avaliar a abordagem de gerenciamento proposta. A escolha deve-se à relevância da pesquisa científica abordada por cientistas da biodiversidade e aos problemas computacionais encontrados durante o projeto EU-BrazilOpenBio. A importância, a nível mundial, desse estudo de caso fica evidente no plano estratégico assinado pelas Nações Unidas que declararam o período de 2011-2020 como a "Década da Biodiversidade"[1], que tem como objetivo travar e, eventualmente, reverter à perda de biodiversidade do planeta. Nesse contexto, surgiu o projeto denominado "*EUBrazilOpenBio - EU-Brazil Open Data and Cloud Computing e-infraestructure for Biodiversity*"[2], que tem como objetivo combinar a Ciência da Biodiversidade e o Movimento de Acesso Livre, promovendo o conceito de abertura para pesquisa científica. O projeto possui a meta de implantar uma plataforma de acesso aberto integrando as infraestruturas e recursos Europeus e Brasileiros, dando passos significativos no sentido de apoiar plenamente as necessidades e exigências da comunidade científica em biodiversidade. Este capitulo apresenta o estudo de caso deste projeto - o problema cientifico da modelagem de nichos ecológicos. A Seção 3.1 define e contextualiza o processo de Modelagem de Nichos Ecológicos. Logo em seguida, na Seção 3.2, é apresentado o *software* openModeller, uma das principais ferramentas mundiais de modelagem de nichos ecológicos. Na Seção 3.3, descreve-se dois experimentos reais da modelagem de nichos ecológicos de espécies da flora brasileira. Por fim, a Seção 3.4 apresenta uma análise do comportamento do openModeller na criação de um modelo de nicho ecológico.

[1] http://www.cbd.int/2010
[2] http://www.eubrazilopenbio.eu/

3.1 Modelagem de Nichos Ecológicos

A Modelagem de Nichos Ecológicos (ENM - *Ecological Niche Modelling*) é utilizada como uma ferramenta para a predição da distribuição geográfica de uma espécie, onde o modelo leva em conta os pontos de ocorrência, um conjunto de condições ambientais e recursos nos quais a espécie é capaz de sobreviver, crescer e reproduzir. Com o conhecimento desses dados é possível predizer os prováveis locais de ocorrência da espécie [48]. Além dos dados citados anteriormente, existem outros fatores que podem influenciar no processo de modelagem de nichos ecológicos [68], conforme a ilustração da Figura 3.1.

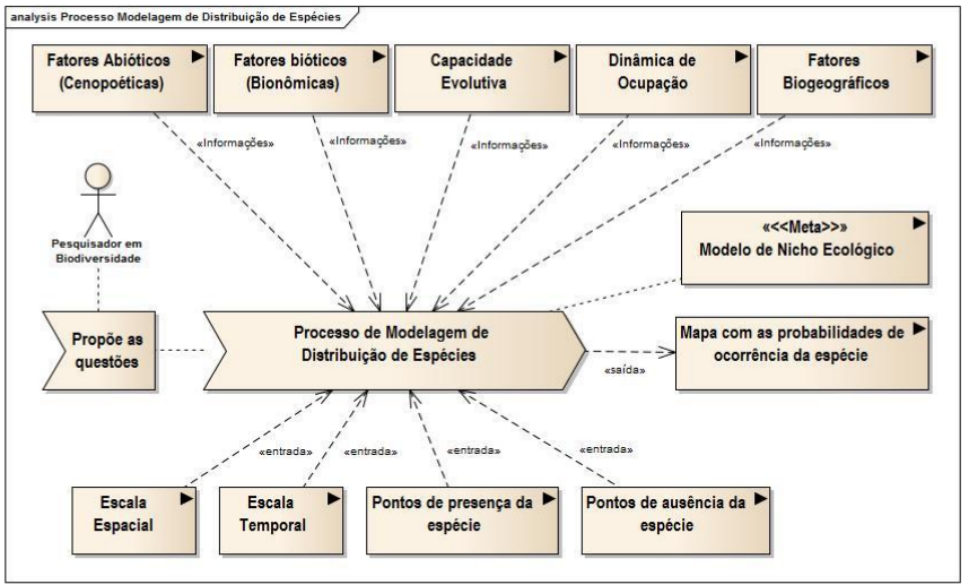


Figura 3.1: Fatores do Processo de Distribuição de Espécies. Fonte: [68].

O usuário, especialista, precisa definir os fatores que influenciam na criação do modelo desejado, conforme a espécie analisada. Destaca-se alguns dos fatores que serão utilizados no framework que será apresentado na próxima seção:

- Fatores Abióticos: São dados que informam as características ambientes de uma região, por exemplo, temperatura, umidade, precipitação média anual, entre outras.
- Fatores Bióticos: São informações relacionadas a interação entre as espécies que ocupam a mesma região, a dinâmica de consumo de recursos, como por exemplo, predadores e competidores.
- Pontos de ocorrência e ausência das espécies.

O processo de ENM pode ser dividido em cinco grandes etapas [68]:

1. Construção da Hipótese Científica: Essa etapa visa definir as hipóteses estatísticas e científicas da pesquisa, assim como planejar a pesquisa e revisar os problemas relacionados. Além de coletar os pontos de ocorrência e ausência da espécie.

2. Pré-Análise: Antes de criar o modelo, precisa-se validar os dados da fase anterior, portanto, realiza-se um teste das hipóteses estatísticas e analise-se os dados gerados.

3. Modelagem: O objetivo dessa fase é selecionar o algoritmo mais adequado e os parâmetros associados. A Figura 3.2, apresenta alguns dos casos mais relevantes do uso de ENM, em diferentes áreas científicas e variando o método (algoritmo), conforme o experimento analisado. Nota-se um domínio nas atividades ligadas com a biologia.

4. Predição: Após a etapa anterior, que cria um modelo de distribuição, baseado em uma região específica. Essa etapa faz uso do modelo para estimar a distribuição de uma espécie em uma nova área.

5. Validação da Hipótese Científica: Essa etapa é responsável por validar a hipótese ecológica formulada na Etapa 1. Os modelos e predições de distribuição podem ser combinados. Destaca-se que essa etapa deve ser realizada por um especialista da espécie avaliada.

ÁREA	MÉTODO*
Predição de distribuição de espécies raras ou ameaçadas de extinção	Maxent, GARP
Guiar levantamentos para detectar espécies novas ou raras e novos padrões de distribuição	GARP, Distância Euclidiana
Escolha de espécies para recuperação de áreas degradadas	GARP
Escolha de áreas prioritárias para conservação	GARP, ENFA, GLM
Determinação de áreas com maior risco de invasão por espécies exóticas	GARP, BIOCLIM, Maxent
Análise do efeito das mudanças climáticas globais sobre a biodiversidade	GARP, BIOCLIM, GLM
Predição de áreas ideais para plantio	GARP, Maxent, BIOCLIM, DOMAIN

Figura 3.2: Exemplos de aplicações de ENM. Fonte: [48].

3.2 openModeller

O projeto EU-BrazilOpenBio adotou o uso do openModeller [70] por ser um reconhecido *framework*, de código aberto para geração de modelos de distribuição potencial de espécies, também conhecido como ambiente computacional de modelagem de nichos ecológicos, desenvolvido pelo Centro de Referência em Informação Ambiental - CRIA, junto com outros parceiros nacionais e internacionais [2, 16, 3].

Fica evidente a necessidade de um ambiente computacional de alto desempenho ao observar um experimento típico dos pesquisadores que utilizam o openModeller (experimento semelhante ao descrito na Seção 3.3), pois, utilizando os recursos computacionais do HVB - Herbário Virtual Brasileiro, necessita-se cerca de 50 minutos para criar um modelo de nicho ecológico para uma única espécie. Atualmente são reconhecidos mais de 46 mil espécies na Lista da Flora Brasileira[3] e seria necessário mais de 12 meses, executando continuamente, para gerar modelos para apenas 30% dos angiospermas conhecidos (cerca de 33 mil espécies).

Modelagem de nichos ecológicos é largamente utilizada para auxiliar na predição e compreensão da distribuição das espécies. Exemplos de aplicação: na indicação de áreas prioritárias para a conservação da biodiversidade, na avaliação de invasões potenciais de espécies exóticas, no estudo de impactos de mudanças climáticas na biodiversidade, no acompanhamento de vetores de doenças infecciosas, entre outras. Para o openModeller realizar este tipo de modelagem são necessários dois tipo de dados (dados bióticos e dados abióticos) e o uso dos algoritmos de modelagem.

a) Dados bióticos

A rede speciesLink [80] é um sistema de informação que integra em tempo real dados primários de coleções científicas. O speciesLink conta com 578 milhões de registros recuperados pelos usuários da rede speciesLink em 2016, organizados em 448 coleções e sub-coleções. O portal *Global Biodiversity Information Facility* - GBIF[30] conta com mais de 639 milhões de pontos de ocorrências de quase 1,7 milhões de espécies.

b) Dados abióticos

O Worldclim [4] é um conjunto de dados climáticos globais (camadas climáticas) de temperatura (máximas, médias e mínimas), precipitação e altitude em 4 resoluções (10', 5', 2.5' e 30').

Estão disponíveis no openModeller os seguintes algoritmos para modelagem:

- BIOCLIM: Bioclimatic envelopes.

[3] http://www.floradobrasil.jbrj.gov.br/
[4] http://www.worldclim.org/

- CSMBS: Climate Space Model.
- GARP: Genetic Algorithm for Rule-set Production (single run) - new openModeller implementation.
- GARP BS: GARP with Best Subsets - new openModeller implementation.
- DG GARP: Desktop GARP implementation.
- DG GARP BS: GARP with best subsets - Desktop GARP implementation.
- ENFA: Ecological-Niche Factor Analysis.
- ENVSCORE: Envelope Score.
- ENVDIST: Environmental Distance.
- MAXENT: Maximum Entropy.
- NICHE MOSAIC: Niche Mosaic.
- ANN: Artificial Neural Network.
- RF: Random Forests.
- SVM: Support Vector Machines.
- AQUAMAPS: algorithm for aquatic organisms.

A abordagem do openModeller é ilustrada na Figura 3.3: (1) a modelagem de nichos ecológicos é gerada relacionando os pontos de ocorrência conhecidos de uma espécie, (2) com as variáveis ambientais que podem afetar a sua distribuição, (3) aplica-se um algoritmo para criar um modelo, (4) gera-se um modelo de espaço ambiental, (5) por fim, o modelo criado é projetado indicando com a cor vermelha a alta probabilidade de existência, e com a cor azul a baixa probabilidade de existência. Esse método é conhecido como método correlativo [79]. O modelo resultante, basicamente, tenta encontrar uma representação das condições que são adequadas para as espécies. Tais modelos podem ser projetados em diferentes regiões geográficas e com diferentes cenários ambientais, tornando-se possível prever o impacto das mudanças climáticas sobre a biodiversidade [64].

Além das tarefas de criação e projeção de modelo de nichos ecológicos apresentados na Figura 3.3, itens (3) e (5), respectivamente. O openModeller permite vários outros comandos relacionados, são eles:

- om_algorithm (retorna informações sobre os algoritmos disponíveis)

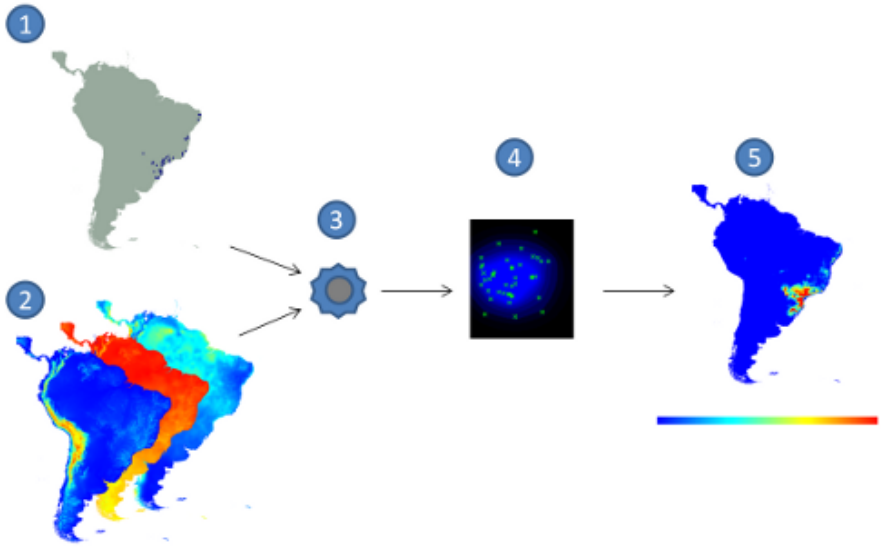


Figura 3.3: Abordagem correlativa do OpenModeller. (1) pontos de ocorrência das espécies, (2) camadas ambientais, (3) algoritmo de modelagem, (4) modelo de espaço ambiental, (5) projeção do modelo. Fonte: [60].

- om_evaluate (retorna os valores do modelo dado um conjunto de pontos e o cenário ambiental)
- om_model (cria um modelo de distribuição de um nicho ecológico)
- om_niche (disponível apenas para GNU/Linux com X11) (ferramenta para visualizar um espaço ambiental em um modelo openModeller)
- om_points (recupera pontos de ocorrência de um local específico)
- om_project (projeta um modelo de distribuição do openModeller)
- om_pseudo (gera pontos aleatórios - latitude e longitude)
- om_sampler (gera um vetor de dados ambientais associados a um ponto no espaço geográfico)
- om_test (testa o modelo de distribuição)
- om_viewer (disponível apenas para GNU/Linux com X11) (ferramenta para visualizar as camadas ambientais e o modelo projetado)

O *framework* openModeller ainda oferece uma versão destinada para servidor *web*, denominado OMWS (openModeller *Web Service*), desenvolvido usando gSOAP v2.8.15 e C++, esse serviço permite a execução de aplicações CGI (*Common Gateway Interface*). O OMWS oferece algumas das operações da Tabela 3.1 de forma síncrona (respondendo imediatamente a solicitação) e também permite uma comunicação de forma assíncrona, extraindo parte do XML existente

dentro da mensagem SOAP (*Simple Object Access Protocol*) e armazenando-a no sistema de arquivo para ser processado de forma independente posteriormente. Esse arquivo XML armazenado no sistema de aquivo do servidor, contém os dados de entrada necessários para executar as etapas do OpenModeller para criar um ENM

No início do Projeto EUBrazilOpenBio, o openModeller possuía somente uma versão *web*, ao longo de várias discussões e análises, percebeu-se a necessidade da criação de uma nova versão, compatível com a primeira. Atualmente o OMWS possui duas versões (oM Server 1.0 e oM Server 2.0, comumente denominados OWMS 1 e OMWS 2, respectivamente) que permitem formatos diferentes de submissão de requisições. São elas:

3.2.1 oM Server 1.0

A primeira versão do openModeller para servidor (que ainda está em funcionamento), disponibiliza as operações apresentadas na Tabela 3.1:

Tabela 3.1: Comportamento do servidor para cada operação do OWMS. Fonte: [60].

Operação	Comportamento do servidor
ping	Envolve a chamada direta para uma biblioteca local do OpenModeller e requer acesso ao sistema de arquivos para verificar a configuração.
getAlgorithms	Executa uma chamada direta a biblioteca local do OpenModeller.
getLayers	Requer acesso ao sistema de arquivos para listar os layers disponíveis localmente.
createModel testModel projectModel	Extrai o XML da mensagem SOAP, coloca-o no sistema de arquivos e retorna o respectivo *ticket* (Cada requisição desse tipo, recebe um *ticket* para ser identificado e monitorado).
getProgress getLog getModel getTestResult getProjectionMetadata getLayerAsUrl	Requisita acesso ao sistema de arquivos para ler as informações do *ticket*.

Dentre as operações destacadas na Tabela 3.1, tem-se a operação ping (responsável por verificar a disponibilidade do sistema), as operações gets (responsáveis por retornar algo para o usuário, desde os algoritmos disponíveis até os resultados gerados pelas aplicações do openModeller) e, principalmente, as operações createModel, testModel e projectModel (responsáveis por receber os dados enviados pelo usuário para criar, testar e projetar um modelo de dispersão de uma espécie, respectivamente). Após o OMWS 1 receber as requisições, um gerenciador (*script*)

é responsável por coletar o XML recebido, via SOAP, armazenado no sistema de arquivos, para executar as operações modelagem, teste e projeção, conforme a requisição do usuário.

A principal característica a se destacar nessa versão, é a forma de submissão de requisições ao servidor, onde o usuário pode solicitar cada operação individualmente, ficando com a responsabilidade de verificar e controlar as dependências envolvidas entre as aplicações.

3.2.2 oM Server 2.0

A versão OMWS 2, surgiu durante o Projeto EUBrazilOpenBio, e, além das operações apresentadas na Tabela 3.1, essa versão disponibiliza as operações descritas na Tabela 3.2.

Tabela 3.2: Comportamento do servidor para cada operação do OWMS. Fonte: [60].

Operação	Comportamento do servidor
evaluateModel samplePoints	Extrai o XML da mensagem SOAP, coloca-o no sistema de arquivos e retorna o respectivo *ticket*.
runExperiment	Extrai o XML da mensagem SOAP, coloca-o no sistema de arquivos, cria todas as tarefas individualmente e retorna todos os *tickets* associados.
getModelE valuation getSamplingEvaluation getResults cancel	Requisita acesso ao sistema de arquivos para ler as informações do *ticket*.

Com essa nova versão, criou-se, explicitamente, o termo "experimento". Um experimento é formado por um conjunto de tarefas menores (createModel, testModel e projectModel) que representam as tarefas necessárias para criar um modelo de nicho ecológico de uma ou mais espécies. Esse experimento é um *workflow* científico e pode ser representado por um DAG (capaz de indicar a dependência dos dados entre as tarefas).

A adição da operação que envolve esse novo termo no openModeller, muda completamente a forma de gerenciar as tarefas do *workflow*. Na versão anterior, o usuário era responsável por realizar esse gerenciamento, visto que só era permitido enviar requisições de tarefas independentes. Essa nova versão, OMWS 2, permite que o usuário envie em uma única requisição de um experimento inteiro (formado por diversas tarefas), além de agregar a informação relacionada a dependência de dados existentes entre as tarefas. Desta forma, o gerenciador dessas dependências deixa de ser o usuário e passa a ser um gerenciador de *workflow* local ao servidor OMWS 2. O usuário envia uma única requisição para a operação runExperiment, com um arquivo XML que contém as informações sobre as configuração das tarefas envolvidas e a indicação das dependências entre as tarefas.

As duas versões (OMWS 1 e OMWS 2) permitem a execução no próprio servidor do open-

Modeller, inicialmente, em uma única máquina. Ao longo do projeto EUBrazilOpenBio, o script relacionado com a execução (gerenciamento) das requisições, foi atualizado para permitir a submissão de requisições para *clusters* que utilizam o gerenciador de *workflows* do HTCondor, o DAGMan (Sistema de gerenciamento discutido na Seção 2.3.1.2).

3.3 Descrição de um experimento científico para criação de um modelo de nicho ecológico

Foi criado, pelos desenvolvedores do openModeller, um *script* implementado em Python, capaz de submeter e gerenciar um típico experimento *e-science* utilizado por pesquisadores da biodiversidade do INCT HVB - Herbário Virtual Brasileiro. As instâncias utilizadas foram produzidas a partir de requisições reais do HBV. O *script* suporta o uso do padrão OMWS 1 e OMWS 2, permitindo enviar várias requisições independentes ou uma única requisição de um *workflow* científico (formado por várias tarefas), respectivamente. Todos os testes foram baseados em 2 experimentos distintos, aqui denominados Experimento 1E e Experimento 5E, contendo 1 espécie e 5 espécies, respectivamente. São eles:

1. **Experimento 1E (formado por uma única espécie):**

   Esse experimento é utilizado pelos biólogos para criar um modelo de nicho ecológico de uma única espécie da flora brasileira, utilizando o openModeller. O experimento relaciona os pontos de ocorrência da espécie Passiflora luetzelburgii, de um gênero botânico da família Passifloraceae. Os dados dos pontos de ocorrência foram obtidos no portal speciesLink [80].

   O experimento gera modelos através de cinco algoritmos diferentes (MAXENT, ENFA, SVM, ENVDIST e GARP_BS). Os modelos são gerados e testados a partir de um algoritmo, posteriormente, realiza-se uma validação cruzada com o método *10-fold*, com diferentes combinações de pontos de ocorrência (esse método de validação cruzada consiste em dividir o conjunto total de dados em 5 subconjuntos mutuamente exclusivos, que são utilizados no cálculo da acurácia do modelo).

   Em seguida, um modelo final é gerado com todos os pontos de provável ocorrência, e por fim, são projetados. Assim, tem-se um experimento formado por um grupo de 115 tarefas que podem ser representadas como o DAG da Figura 3.4, onde, a criação dos modelos não possui nenhuma dependência (M1 até M55), todos os testes dependem dos modelos gerados (T1 até T55) e 5 projeções dependem exclusivamente dos modelos criados (P51 até P55). O conjunto de tarefas (M1, M2, ..., M50; T1, T2, ..., T50) representam a validação cruzada 10 vezes.

   O *workflow* desse experimento é formado por 115 tarefas originais do openModeller (criação de modelo, teste e projeção, que corresponde as seguintes aplicações: om_model,

om_test e om_project), porém, além dessas, foi desenvolvido uma aplicação, denominada build_request que precisa ser usada, sempre, entre as sequências (om_model e om_test) e (om_model e om_project). Essa aplicação buid_request é uma tarefa que precisa ser considerada no *workflow* e é responsável por coletar o modelo gerado pelo om_model e adicionar dentro do arquivo que contém as configurações necessárias para a execução do om_test e om_project. Portanto, o *workflow* realmente é formado por 175 tarefas. Destaca-se que o build_request é uma tarefa tão rápida que nem é considerada nos gráficos, mas é essencial na execução do *workflow*.

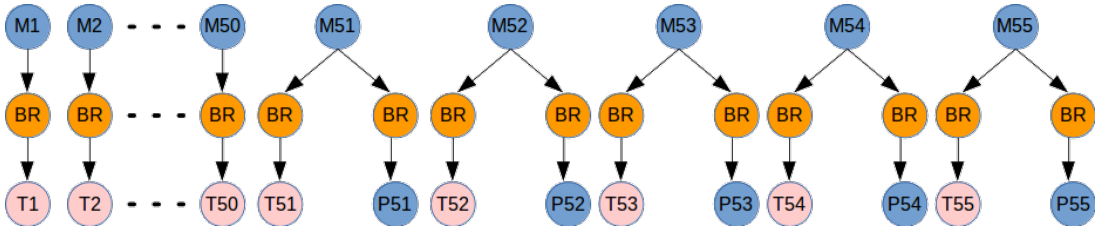


Figura 3.4: Visão geral do experimento.

2. **Experimento 5E (formado por cinco espécies):**

   Esse segundo *workflows* científico foi criado para explorar um experimento mais robusto, formado por cinco diferentes espécies da flora brasileira, com uma diversidade de parâmetros de configuração associados. Esse experimento cria modelos com 5 diferentes algoritmos disponíveis no openModeller, são eles: MAXENT (*Maximum Entropy*), ENFA (*Ecological-Niche Factor Analysis*), SVM (*Support Vector Machines*), ENVDIST (*Environmental Distance*), GARP_BS (*GARP with best subsets*).

   Nesse experimento gera-se modelos individuais com o uso de cada um dos algoritmos citados e, realiza-se uma validação cruzada através do método *5-fold*, com os pontos de ocorrência e ausência das espécies. Portanto, para cada espécie, 5 modelos são criados e testados com cada um dos algoritmos. O modelo final é gerado com todos os pontos e então é projetado. Todos os modelos criados são independentes, inclusive o modelo final.

   No total, o experimento possui 150 criações de modelos (5x25 para cada validação cruzada + 25 para a criação do modelo final), 150 testes de modelos (5x25 para cada validação cruzada + 25 para o teste do modelo final) e 25 projeções dos modelos. Totalizando 325 tarefas individuais do openModeller. Considerando as tarefas do build_request, o *workflow* realmente é formado por 500 tarefas.

3.4 Análise da execução do openModeller

Essa seção apresenta um breve resumo de alguns dos principais experimentos que foram utilizados na compreensão do funcionamento e características do openModeller (um relatório mais completo será disponibilizado num anexo da versão final da tese). Destaca-se as duas principais características que podem afetar o escalonamento e o desempenho das aplicações do openModeller: algoritmo utilizado na modelagem e; a quantidade de *layers*. A relevância desses experimentos, está na compreensão do impacto dos dados de entrada sob o desempenho total de cada tarefa do *workflow*.

3.4.1 Comparação dos algoritmos

O openModeller oferece diversos algoritmos para suas modelagens. Essa seção analisa apenas os algoritmos que compõem os experimentos descritos na seção anterior.

(1) Testes com o Experimento 1E (formado por uma única espécie):

O Experimento 1E utiliza 5 algoritmos na criação dos seus modelos: MAXENT, ENFA, SVM, ENVDIST e GARP_BS. A Figura 3.5 apresenta os tempos obtidos ao executar o experimento de forma exclusiva, ou seja, sem interferência de outros usuários ou aplicações (exceto as aplicações do próprio Sistema Operacional), obtendo, desta forma, o melhor tempo de cada tarefa do *workflow*. Os resultados foram agrupados por aplicação (om_model, om_test e om_project), onde cada barra vertical representa o tempo, em segundos, da execução de cada algoritmo.

Com esse experimento foi possível observar o comportamento de cada aplicação, com cada algoritmo. Nota-se, nitidamente, que o algoritmo GARP_BS necessita de um tempo maior de execução no OM_MODEL e OM_PROJECT do que os outros algoritmos e que o tempo de execução do OM_TEST é (praticamente) o mesmo, independentemente do algoritmo selecionado. Os tempos de execução das aplicações que utilizam os demais algoritmos são bem próximos, entre eles.

As aplicações foram executadas individualmente, de forma sequencial, sem concorrência por recursos, em uma máquina com 12 núcleos físicos de processamento e 24GB RAM (denotado como uma máquina do tipo HT2 nesta tese). Porém, nem sempre as aplicações comportam-se da mesma forma quando existe concorrência por recursos (CPU, I/O, entre outros). Por isso, para avaliar o comportamento dessas aplicações, executou-se as mesmas aplicações anteriores com até 24 processos concorrentes conforme as figuras a seguir.

A Figura 3.6 mostra a execução concorrente do OM_MODEL com os cinco algoritmos avaliados anteriormente. Os resultados mostram claramente exemplos de interferência entre instâncias das aplicações e a degradação em desempenho causada, ainda quando núcleos suficientes para

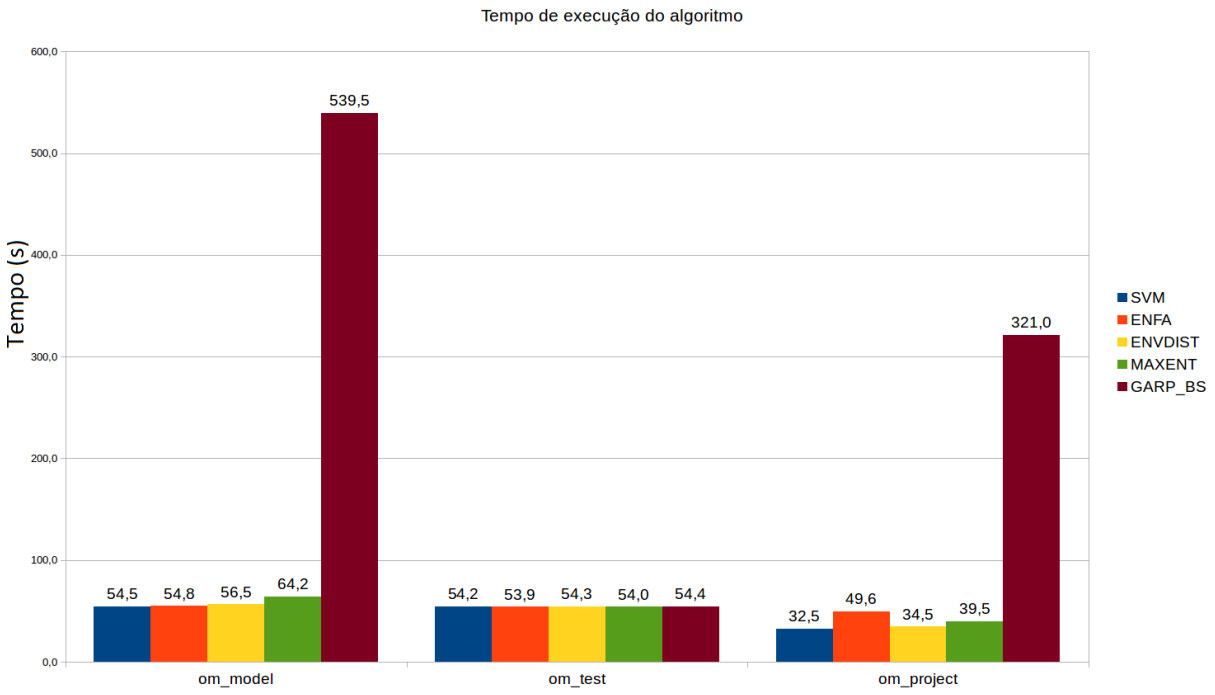
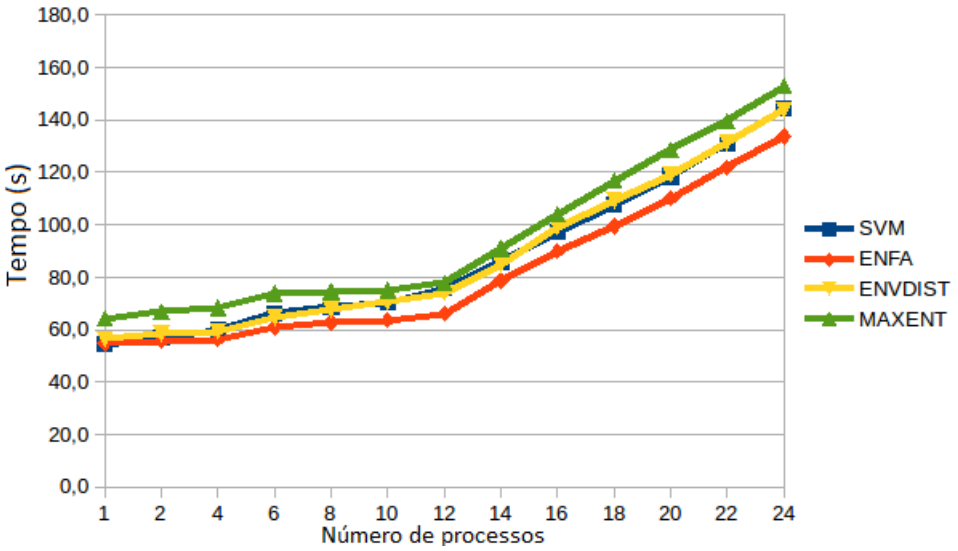


Figura 3.5: Tempo de execução dos algoritmos do openModeller.

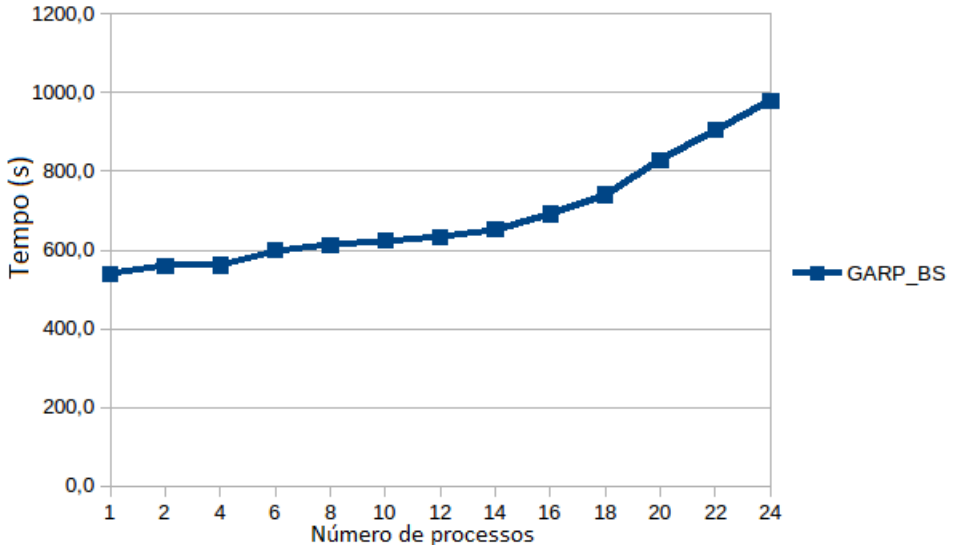
uma execução paralela. Dado a configuração do servidor, pode-se esperar que, para até 12 núcleos, não teria um aumento nos tempos de execução das instâncias. Porém, a degradação varia entre 17% para GARP_BS e 39% para SVM.

Por outro lado, também mostra os benefício de uma execução concorrente. Como já era esperado, o GARP_BS possui um comportamento diferente dos demais. Além do deslocamento da linha do GARP_BS, que indica que necessita de um tempo maior de execução em relação aos demais, é possível observar que a inclinação muda ao executar mais de 12 tarefas concorrentes (mesma quantidade de núcleos de processamentos físicos do *host*) e a partir de 18 tarefas, a inclinação torna-se ainda maior. Rodando 18 instâncias em 12 núcleos só aumenta o tempo de execução em relação a execução de 12 instâncias (50% mais) em 17%. Neste caso, os outros algoritmos têm um aumento quase proporcional (entre 40% e 51%), indicando que seus desempenhos estão piorando linearmente com o aumento da quantidade de processos concorrentes. Quando consideramos 24 instâncias, as degradações em relação da execução com 12 ficam 90% para SVM, 103% para ENFA, 94% para ENVDIST, 96% para MAXENT e 55% para GARP_BS. Para executar OM_MODEL, em um servidor do tipo HT2, os resultados indicam que com a execução de 18 instâncias concorrentes, os processos individuais necessitam de um tempo maior de execução, mas o tempo total da execução de um conjunto de processos é reduzido.

A Figura 3.7 mostra que o tempo de execução de todas as tarefas são equivalentes, independente do algoritmo utilizado, e o tempo de execução das aplicações é ligeiramente aumentado (20%) até a quantidade de núcleos de processamento disponíveis na máquina (12 núcleos), logo



(a)



(b)

Figura 3.6: Execução concorrente da aplicação OM_MODEL em uma máquina do tipo HT2.

em seguida, a partir de 12 processos concorrentes, o gráfico possui um outro comportamento, afetando significativamente o tempo de execução das tarefas como esperado. Com isso, existe uma degradação leve até a quantidade de núcleos de processamento (12 processos concorrentes), devido ao fato de ter recursos sendo subutilizados. Mas, ao extrapolar a quantidade de núcleos, a degradação aumenta linearmente, impactando no desempenho das tarefas.

A execução de aplicações do tipo om_project visto na Figura 3.8, variando somente o algoritmo, mantém um comportamento bem semelhante para quatro dos algoritmos (ENFA, ENVDIST, MAXENT, SVM), tendo pouca degradação (de 14% a 21%) até 12 instâncias, após sobrecarregar os recursos (número de processos maior que a quantidade de núcleos de processamento),

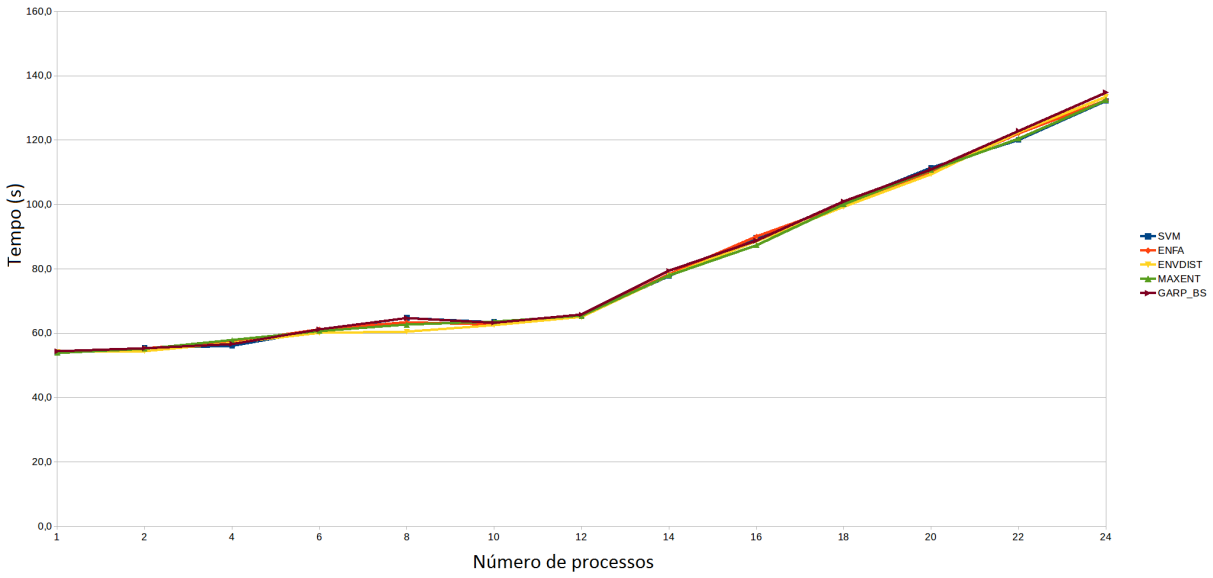


Figura 3.7: Execução concorrente da aplicação OM_TEST em uma máquina do tipo HT2.

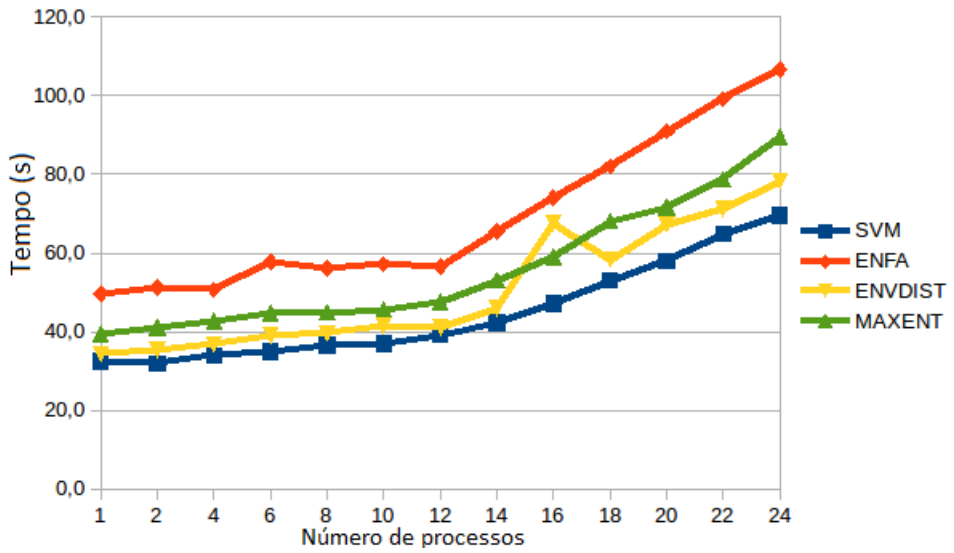
o tempo de execução da projeção apresenta ter um crescimento linear semelhante a execução do om_model. O algoritmo que possui o comportamento totalmente diferente é o GARP_BS, que mostra ter seu tempo bem impactado com uma quantidade de tarefas concorrentes maior que a quantidade de núcleos disponíveis.

Com esses experimentos foi possível verificar o comportamento das aplicações do openModeller com cinco diferentes algoritmos e o comportamento dessas aplicações com uma super-utilização de um *host*.

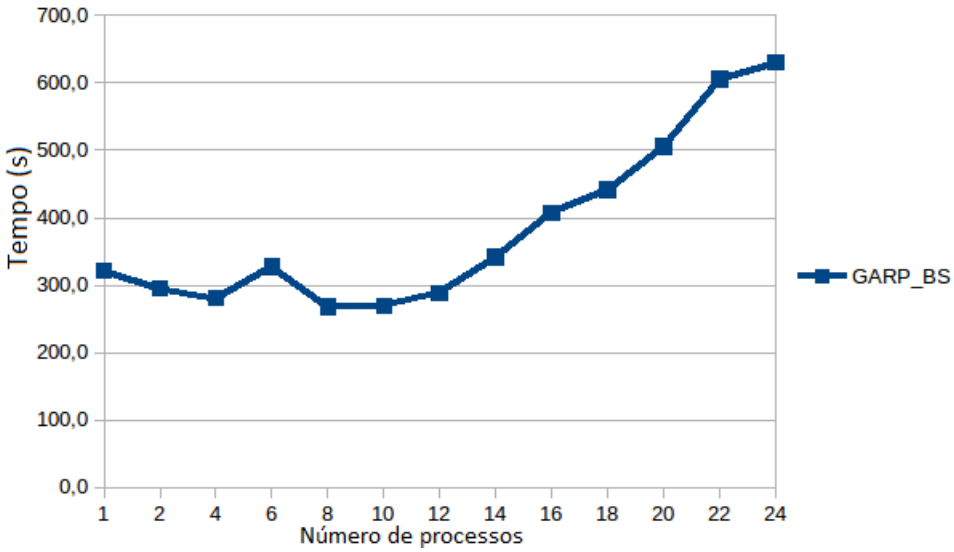
(2) Testes com o experimento 5E (formado por cinco espécies):

O Experimento 5E utiliza os mesmos 5 algoritmos (MAXENT, ENFA, SVM, ENVDIST e GARP_BS), mas, diferentemente do Experimento 1E, tem o objetivo de criar modelos para as 5 diferentes espécies, além disso, sua validação cruzada também é diferente, pois utiliza o método de $5 - fold$. A Tabela 3.3 destaca as características envolvidas com cada espécie, onde, a quantidade de camadas ambientais e pontos de ausência são os mesmos, porém, esses dados variam conforme a espécie (em outras palavras, as espécies 1 e 2 possuem a mesma quantidade de pontos de ausência, mas esses pontos são totalmente diferentes).

Após executar sequencialmente todas as tarefas do experimento, obteve-se a Tabela 3.4. Nessa tabela tem-se o tempo de execução de cada aplicação, sem sofrer perda de desempenho devido a concorrência de CPU e I/O. Pode-se considerar como o melhor tempo possível, dado o ambiente computacional disponível. O tempo total da execução sequencial desse experimento é de 29596 segundos (mais de 8 horas e 13 minutos).



(a)



(b)

Figura 3.8: Execução concorrente da aplicação OM_PROJECT em uma máquina do tipo HT2.

Tabela 3.3: Características de cada espécie.

	camadas ambientais	pontos de ocorrência	pontos de ausência
espécie 1	8	32	10000
espécie 2	8	106	10000
espécie 3	8	25	10000
espécie 4	8	24	10000
espécie 5	8	22	10000

3.4.2 Impacto da variação dos *Layers*

A quantidade de *layers* (camadas ambientais) utilizada em cada modelagem de nicho ecológico está diretamente relacionada com a quantidade de dados que será lido/escrito em cada aplicação.

Tabela 3.4: Média do tempo (em segundos) da execução sequencial de cada espécie com 5 algoritmos diferentes do openModeller.

Fase	Algoritmo	Espécie 1	Espécie 2	Espécie 3	Espécie 4	Espécie 5
Model	ENFA	52,26	52,79	52,01	51,6	52,2
	ENVDIST	51,62	51,73	51,53	51,05	51,81
	GARP BS	446,9	460,54	442,65	453,08	453,22
	MAXENT	61,4	61,41	60,31	58,41	61,27
	SVM	51,64	51,82	51,37	51,26	51,37
Test	ENFA	51,89	52,38	51,78	51,24	51,92
	ENVDIST	51,84	51,94	51,77	51,32	51,74
	GARP BS	52,11	52,48	52,13	51,69	52,91
	MAXENT	51,99	52,1	51,87	51,29	51,76
	SVM	51,91	52	51,62	51,04	51,76
Project	ENFA	58,2	153,29	57,52	27,57	48,13
	ENVDIST	32,01	32,99	32,2	18,12	32,68
	GARP BS	105,84	257,26	154,98	163,13	296,07
	MAXENT	40,85	51,32	38,15	17,91	42,92
	SVM	32,57	35,81	29,95	15,9	29,27

A Tabela 3.5 apresenta o tempo de execução, a quantidade de caracteres lidos (Rchar) e escritos (Wchar) da aplicação om_model, utilizando o algoritmo MAXENT e variando a quantidade de camadas ambientais que, inicialmente, cada modelagem contou com 8. Observa-se que cada instância dessa aplicação escreve pouco no disco, mas acessa muito o disco para ler informações, com oito *layers* a aplicação lê mais de 5 GB de dados ambientais.

Tabela 3.5: Variação da quantidade de *Layers* no om_model

	8 layers	**7 layers**	**5 layers**	**3 layers**	**1 layers**
Rchar (MB)	5436,9	5199,9	4457,6	3745,3	2849,2
Wchar (KB)	0,2	0,2	0,2	0,2	0,1
Tempo (s)	75,6	64,3	46,09	30,05	9,05

Tabela 3.6: Variação da quantidade de *Layers* no om_test

	8 layers	**7 layers**	**5 layers**	**3 layers**	**1 layer**
Rchar (MB)	5324,4	5034,8	4228,3	3567,0	2159,4
Wchar (KB)	0,0	0,0	0,0	0,0	0,0
Tempo (s)	58,05	50,07	40,02	22,5	7,02

A Tabela 3.6 mostra que a aplicação om_test (responsável por testar o modelo criado pelo om_model) possui um alto acesso a disco para leitura (mais de 5GB para testar um único modelo) mas não escreve nada em disco. Na Tabela 3.7, a aplicação om_project realiza pouco leitura em disco (Rchar) comparando com as aplicações om_model e om_test, porém é a aplicação que possui o maior acesso a disco efetuando escrita (Wchar). Além disso, evidencia-se que o tempo

Tabela 3.7: Variação da quantidade de *Layers* no om_project

	8 layers	**7 layers**	**5 layers**	**3 layers**	**1 layers**
Rchar (MB)	264,6	244,9	182,6	126,1	65,5
Wchar (KB)	1838,6	1838,6	1838,6	2606,6	2862,6
Tempo (s)	43,04	42,03	40,05	29,03	17,04

de execução não está associado exclusivamente ao tempo de leitura e escrita de dados. Portanto, o tempo de uso de CPU deve ser considerado no tempo total de cada etapa do *workflow*.

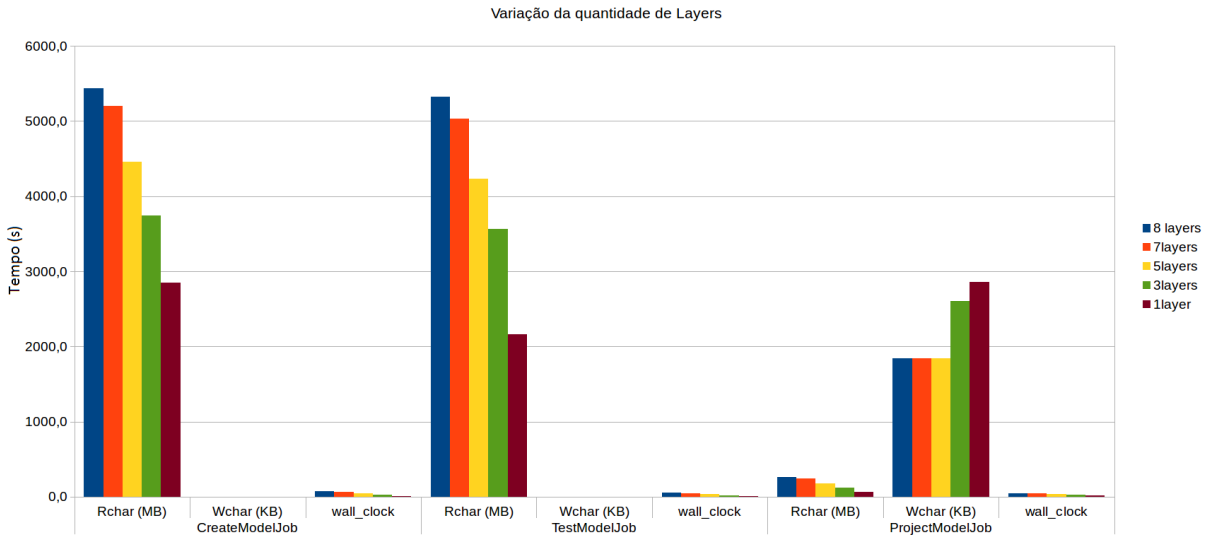


Figura 3.9: Variação da quantidade de layers.

As informações das três tabelas anteriores foram condensadas na Figura 3.9, mostrando que os dados lidos nas aplicações om_model e om_test são reduzidos linearmente com a diminuição da quantidade de *layers*, porém, a quantidade de dados escritos aumenta com a diminuição da quantidade de *layers* na aplicação om_project.

Uma observação sobre o motivo dos tempos apresentados nas Tabelas 3.5, 3.6 e 3.7, ser diferente dos tempos da Tabela 3.4, deve-se ao fato desses tempos terem sido coletados com a execução concorrente das respectivas aplicações - uma forma de execução mais realística. Destaca-se que somente o tempo é afetado (que leva em conta a degradação devida a concorrência de acessos aos discos mais os benefícios promovidos pela hierarquia da cache), a quantidade de dados necessária independe dessa concorrência.

3.5 Conclusões

Esse capítulo serviu para compreender o processo de modelagem de nichos ecológicos, estudo alvo desse trabalho. Logo em seguida, apresentou-se o openModeller, ferramenta desenvolvida pelo

Centro de Referência em Informação Ambiental - CRIA, largamente utilizada pela comunidade científica na geração de modelos de distribuição potencial de espécies.

Realizou-se uma análise extensiva das características envolvidas em todo esse processo e resumiu-se na Seção 3.4 os principais testes realizados. Esse capítulo serviu, ainda, para descrever os dois experimentos científicos reais, cedidos pelo CRIA, que foram utilizados ao londo de todo esse trabalho, demonstrando o *workflow* científico associado ao processo de modelagem com o uso do openModeller.

Capítulo 4

Uma Arquitetura para Integrar Ambientes Computacionais para Executar *Workflows* Científicos

Há bastante tempo, os movimentos de *software* de código aberto (*open source software*) e de acesso livre aos dados (*open data*) tem democratizado pesquisa científica, reduzindo custos e aumentando qualidade. Na última década, isso tem envolvido algo maior - *Open Access*. Órgãos financiadores estão obrigando que todos os produtos (por exemplo, *softwares* e conjuntos de dados) e resultados (artigos científicos) de pesquisas financiadas com dinheiro público sejam disponibilizados livremente. Mais recentemente, estas agências têm também promovido a implantação de e-infraestruturas de acesso aberto para facilitar progresso científico.

Neste contexto, este capítulo apresenta uma visão global do problema de disponibilizar uma e-infraestrutura de acesso aberto para a comunidade de biodiversidade. A solução proposta se baseia no aproveitamento de ambientes computacionais existentes para serem configurados sob-demanda para rodar *workflows* de ENM. Na Seção 4.2, a abordagem adotada pela solução proposta é descrita de forma genérica, identificando as três principais etapas de gerenciamento envolvidas.

4.1 Descrição do Problema de Escalonamento

Com o crescimento do uso de *workflows* científicos, que demandam muito processamento computacional para cálculo e processamento de dados, tornou-se cada vez mais necessário a utilização de ambientes computacionais de alto desempenho para tratar toda a complexidade envolvida nesse grupo de aplicações. Essa necessidade de um maior poder computacional, na maioria das vezes não é saciada apenas com um conjunto de computadores, pois o tempo necessário para executar um *workflow* e obter um resultado, pode ser inviável para os pesquisadores, ou a capacidade

computacional pode não ser suficiente para processar a carga de trabalho do *workflow*. Comprar e manter um conjunto maior de computadores de alto desempenho é um processo oneroso. Portanto, uma solução para tratar uma grande demanda de *workflows* científicos complexos é integrar diferentes ambientes computacionais, normalmente heterogêneos, mesmo que possuam sistemas de gerenciamento diferentes. Essa integração de ambientes compartilhados deve afetar o mínimo possível os ambientes envolvidos e respeitar todas as regras locais, pois uma mudança na forma de submeter as requisições poderia afetar a produção dos cientistas que utilizavam o ambiente anteriormente.

Por fim, os escalonadores desses ambientes locais nem sempre estão preparados para tratar *workflows* científicos (como é o caso dos gerenciadores de fila do tipo PBS - *Portable Batch System*, normalmente encontrado nos *clusters* de alto desempenho). Além disso, frequentemente, os ambientes computacionais não são exclusivos e trabalham apenas com escalonadores estáticos que não tiram o máximo de proveito dos recursos disponíveis (a natureza dinâmica dos ambientes de alto desempenho requer escalonadores também dinâmicos para evitar subutilização ou sobrecarga dos recursos). Outra característica a se considerar é a forma do gerenciamento, a maioria dos servidores utilizam um único gerenciador central, e essa abordagem não é uma boa solução porque se houver muitos ambientes envolvidos ou uma grande quantidade de *workflows* a serem executados, um único escalonador centralizado certamente terá dificuldade em gerenciar todas as informações envolvidas sem perda de desempenho.

4.2 Uma Arquitetura Multinível

Essa arquitetura surgiu da necessidade de um problema real de integrar vários ambientes computacionais distintos para executar *workflows* científicos. Apesar de existirem ambientes que gerenciam a execução de *workflows* científicos, conforme apresentado na Seção 2.3.3, a integração desses ambientes para trabalhar de forma cooperativa não é explorada. Essa seção descreve uma abordagem de escalonamento hierárquico, dinâmico e não centralizado para uma e-infraestrutura (infraestrutura computacional para *workflows* científicos) composta por ambientes compartilhados e heterogêneos, como *clusters* e grades computacionais, que possuem gerenciadores de tarefas distintos. Essa abordagem visa tirar proveito das peculiaridades de cada ambiente para otimizar o uso dos recursos e reduzir o tempo necessário para a execução de *workflows* científicos.

A Figura 4.1 ilustra a arquitetura que é composta por três grandes etapas e dois componentes que podem auxiliar nas decisões. Segue a descrição de cada uma das três etapas, nas Seções 4.2.2, 4.2.3 e 4.2.4.

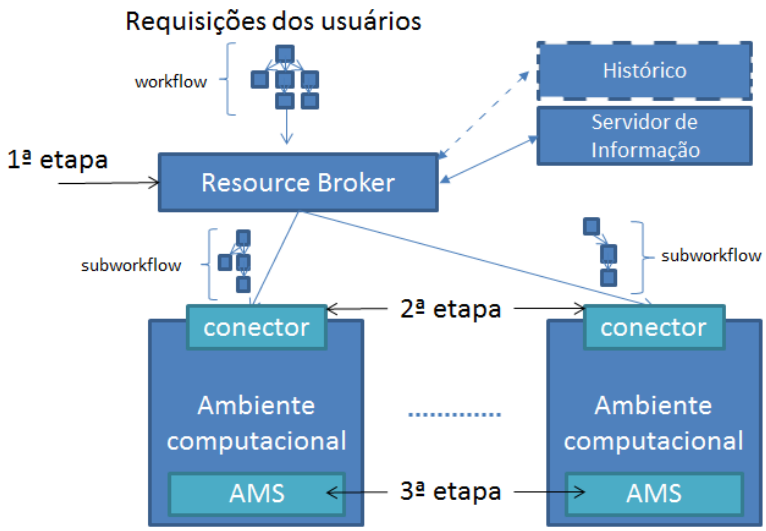


Figura 4.1: Visão geral da arquitetura de gerenciadores proposta.

4.2.1 Pré-requisitos

Para utilizar a arquitetura ilustrada na Figura 4.1, necessita-se definir o formato das requisições ENM - *Ecological Niche Modelling* (requisições enviadas pelos usuários) para que o *Resource Broker* possa processa-lás. Esse *workflow* pode ser representado por qualquer estrutura que represente todas as tarefas envolvidas, além das dependências entre as tarefas e seus dados de entrada. Para essa descrição, pode-se utilizar um arquivo XML para estruturar todas as informações relacionadas. Por exemplo, para a modelagem de nichos ecológicos do estudo de caso adotado nesse trabalho, a requisição que o usuário envia para o *Resource Broker* é feita através de um arquivo XML que possui o formato do quadro abaixo, onde destaca-se as seguintes informações:

- Dados de entrada das tarefas (Nesse exemplo específico, o arquivo XML estrutura os dados de entrada necessários para a execução das aplicações do openModeller)
  - As camadas ambientais são descritas nas linhas 3 até a linha 7 e receberam a identificação "environment0". A linha 4 indica uma camada ambiental padrão (armazenada localmente, anteriormente, em todos ambientes computacionais) e a linha 6 indica a necessidade de uma camada ambiental externa (que precisa ser obtida). Essa indicação de camada ambiental externa, exige que exista algum mecanismo para controlar essa dependência, obtendo essa camada ambiental externa antes de começar a execução das tarefas.
  - Os pontos de ocorrência são identificados por "presence0" e, nesse exemplo, foram reduzidos para apenas 2 pontos de ocorrência da espécie que deseja-se criar o modelo (essa informação encontra-se nas linhas 8 até a linha 13).

  - O algoritmo que será utilizado na modelagem está descrito entre as linhas 14 e 20 e é identificado por “algorithm0”.

- O arquivo XML pode trazer a descrição de inúmeros *workflows* que formam um Experimento Ecológico, todos eles devem ser descritos em “Jobs”. Especificamente, nesse exemplo, existe a descrição de apenas um *workflow* formado por 3 tarefas (descrito entre as linhas 21 até a linha 46), são elas:

  - Criação de um modelo de nicho ecológico, identificado na linha 22 com o id "job00". Essa tarefa utiliza as informações ambientais do id “environment0”, os pontos de ocorrência do id “presence0” e o algoritmo do id “algorithm0” (todas as informações foram descritas anteriormente).
  - A tarefa de testar um modelo ecológico está descrita entre as linhas 27 e 35, onde indica-se que o teste utiliza-rá as camadas ambientais de “environment0”, os pontos de ocorrência de “presence0” e utilizará o modelo gerado após a execução da tarefa “job0” (indicando que essa tarefa depende da execução de “job0”).
  - A tarefa de projeção é identificada como “job02” (Descrita entre as linhas 36 e 45). Essa tarefa utiliza o modelo gerado em “job00”, conforme a linha 38.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <ExperimentParameters xmlns="http://openmodeller.cria.org.br/
    xml/2.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
    instance" xsi:schemaLocation="http://openmodeller.cria.org.
    br/xml/2.0 ../../schemas/openModeller.xsd ">
3   <Environment NumLayers="2" id="environment0">
4     <Map Id="/layers/en/terrestrial/climate/global/worldclim/
        present/bioclim/10arc-minutes/14aecd40-04d9-11e1-be50
        -0800200c9a66" IsCategorical="0" />
5     <Map Id="/layers/en/terrestrial/climate/global/worldclim/
        present/bioclim/10arc-minutes/bf441530-04d9-11e1-be50
        -0800200c9a66" IsCategorical="0" />
6     <Mask Id="http://modeller.cria.org.br/exp/omwsintro/
        species1_mask.tif" />
7   </Environment>
8   <Presence Count="2" Label="Test species" id="presence0">
9     <CoordinateSystem>GEOGCS["WGS 84",DATUM["WGS_1984",
        SPHEROID["WGS 84",6378137,298.257223563, AUTHORITY["
        EPSG","7030"]],TOWGS84[0,0,0,0,0,0,0],AUTHORITY["EPSG",
        "6326"]],PRIMEM["Greenwich",0,AUTHORITY["EPSG","8901"
```

```
      ]],UNIT["degree",0.0174532925199433,AUTHORITY["EPSG","
      9108"]],AXIS["Lat",NORTH],AXIS["Long",EAST],AUTHORITY["
      EPSG","4326"]]
10    </CoordinateSystem>
11    <Point Id="1" X="-67.845739" Y="-11.327340" />
12    <Point Id="2" X="-69.549969" Y="-12.350801" />
13  </Presence>
14  <AlgorithmSettings id="algorithm0">
15    <Algorithm Id="BIOCLIM" Version="0.2">
16      <Parameters>
17        <Parameter Id="StandardDeviationCutoff" Value="0.8" />
18      </Parameters>
19    </Algorithm>
20  </AlgorithmSettings>
21  <Jobs id="jobs">
22    <CreateModelJob id="job00">
23      <EnvironmentRef idref="environment0" />
24      <PresenceRef idref="presence0" />
25      <AlgorithmRef idref="algorithm0" />
26    </CreateModelJob>
27    <TestModelJob id="job01">
28      <EnvironmentRef idref="environment0" />
29      <PresenceRef idref="presence0" />
30      <ModelRef idref="job00" />
31      <Statistics>
32        <ConfusionMatrix Threshold="0.5" IgnoreAbsences="0" />
33        <RocCurve Resolution="15" MaxOmission="1.0"
            UseAbsencesAsBackground="0" />
34      </Statistics>
35    </TestModelJob>
36    <ProjectModelJob id="job02">
37      <EnvironmentRef idref="environment0" />
38      <ModelRef idref="job00" />
39      <OutputParameters FileType="ByteHFA">
40      <TemplateLayer Id="/layers/en/terrestrial/climate/global
          /worldclim/present/bioclim/10arc-minutes/14aecd40-04
          d9-11e1-be50-0800200c9a66" />
41      </OutputParameters>
```

```
42        <Statistics>
43          <AreaStatistics PredictionThreshold="0.5" />
44        </Statistics>
45      </ProjectModelJob>
46    </Jobs>
47  </ExperimentParameters>
```

Mudanças no formato de descrição do *workflow* afetam diretamente o *Resource Broker* (Etapa 1) e os *Conectors* (Etapa 2).

4.2.2 Etapa 1: *Resource Broker*

A primeira etapa, denominada *Resource Broker*, é responsável por realizar a integração de diferentes ambientes computacionais e orquestrar os *workflows* (ou *sub-workflows*) entre eles. Essa camada intermediária, entre usuários e ambientes computacionais, recebe as requisições dos usuários (através de arquivos XML) e avalia se possuem alguma dependência ou se podem ser submetidas, para os ambientes de execução, imediatamente. Caso exista dependência, esta deve ser resolvida antes de se designar um destino (por exemplo, caso falte algum arquivo que precise ser recuperado para ser utilizado como dado de entrada do *workflow*, nessa situação, o *Resource Broker* deverá resolver a pendência antes de processar as necessidades computacionais e carga de trabalho do *workflow*).

Após estar livre de pendências (com todas as camadas ambientais disponíveis localmente nos ambientes de execução) e pronto para ser submetido para os ambientes computacionais, o *Resource Broker* analisa as tarefas do *workflow* (ou *workflows*) buscando prever a carga de processamento necessária para executar todas as tarefas envolvidas. Para realizar essa predição é necessário conhecer as demandas de cada tarefa, analisando os fatores que podem influenciar no desempenho, tais como: quantidade de dados de entrada e tipo da tarefa. Desta forma, é possível tentar associar essas características com outras tarefas que tenham sido executadas anteriormente (consultando um servidor de histórico), para tentar estimar uma carga de processamento e memória. De posse de estimativa, o *Resource Broker* precisa consultar o Servidor de Informação para avaliar a capacidade de processamento dos ambientes e decidir como as tarefas serão enviadas: de forma completa (todo *workflow*) para um único ambiente ou em partes (*sub-workflows*) para ambientes diferentes, buscando minimizar o tempo total de execução de cada *workflow*.

Por tratar de ambientes computacionais heterogêneos, o *Resource Broker* precisa conhecer o grau de heterogeneidade de cada ambiente para tentar realizar uma alocação balanceada e adequada. Trata-se de uma tentativa, pois utiliza-se apenas uma estimativa de carga de cada tarefa,

e, além disso, por trabalhar com ambientes computacionais distintos (em domínios administrativos), sem garantia de prioridade ou exclusividade no uso de recursos, não é possível estimar com precisão os tempos de execução das tarefas. Destaca-se ainda que existe outros fatores que dificultam o balanceamento, por exemplo: o tempo de espera na fila local e a concorrência que pode existir com outras solicitações/*jobs*, submetidas por outros usuários localmente aos ambientes.

O *Resource Broker*, por estar entre os usuários e os ambientes, é responsável por fazer a intermediação entre eles, portanto, além de receber as requisições, precisa disponibilizar os resultados obtidos pela execução das aplicações aos usuários, independente do ambiente de execução utilizado.

É importante salientar que, por definição, o *Resource Broker* não gera um escalonamento, pois essa etapa não fixa os processadores a serem usados para execução de cada tarefa. Essa etapa resolve a dependência de "layers" externos, analisa a carga do *workflow* recebido e avalia a possibilidade de balancear a carga entre diferentes ambientes computacionais, separando as tarefas em diferentes grupos (respeitando as dependências dos *workflows*), buscando uma execução mais eficiente, em um menor tempo. A etapa que realmente aloca uma tarefa em um processador é a segunda, que define o escalonamento inicial estaticamente, e, a terceira etapa, realiza o escalonamento dinâmico entre os *hosts* durante a execução do *workflow*, com, por exemplo, o uso do EasyGrid AMS.

4.2.3 Etapa 2: Conector de cada ambiente computacional

Cada ambiente computacional possui uma aplicação denominada *conector*, como visto na Figura 4.1, responsável por receber os *jobs* ou *(sub-) workflows* que foram enviados pelo *Resource Broker* e submeter cada um para execução de acordo com as politicas de gerenciamento locais do site. No caso deste trabalho, é criada (e submetida) uma aplicação EasyGrid AMS específica que será capaz de realizar o trabalho do *job* (*i.e.*, todo o *workflow* ou *sub-workflow*).

O *conector* porém não faz qualquer tratamento no *workflow* (dividindo ou agrupando requisições). A tarefa principal dele é criar o arquivo de submissão apropriado para o sistema de gerenciamento local e identifica a largura do *workflow* para definir uma quantidade máxima de máquinas necessária para a execução do *workflow*. No caso do uso da EasyGrid AMS, serão gerados os arquivos de configuração para criar uma aplicação autônoma (que é responsável por tomar suas próprias decisões de gerência) capaz de executar as tarefas do *workflow* solicitado.

Nesse momento de criação da aplicação EasyGrid AMS, deve-se indicar como será o escalonamento estático inicial das tarefas no novo ambiente e, qual será a estratégia de escalonamento ao longo da execução da aplicação EasyGrid AMS, pode-se: manter o escalonamento estático inicial; permitir um balanceamento dinâmico durante a execução; ou executar as tarefas de forma

altruísta, provendo condições da aplicação EasyGrid AMS ceder seus recursos temporariamente. Antes de submeter o *job* ao ambiente computacional local, o *conector* deve garantir que todos os arquivos necessários para o inicio das tarefas estejam disponíveis localmente nas máquinas de execução do ambiente.

4.2.4 Etapa 3: Gerência autônoma

Enquanto as etapas 1 e 2 já existem em algumas e-infraestruturas, a etapa 3 representa a contribuição principal desta tese. Essa etapa ocorre após a alocação dos recursos computacionais e está relacionada com a execução da aplicação EasyGrid AMS que irá realizar o gerenciamento de todas as tarefas que formam o *(sub-)workflow*.

Inicialmente as tarefas são escalonadas nos recursos conforme o escalonamento estático realizado na segunda etapa, pelo *conector*. A seguir, como o recurso não é exclusivo, pode ocorrer sobrecarga nos recursos mesmo que todos usuários utilizem o mesmo gerenciador de recursos. Por exemplo, com o uso do PBS, a reserva de recursos pode ser feita por núcleo de processamento, portanto, um usuário pode requisitar um núcleo de processamento e outro usuário pode reservar outro núcleo do mesmo *host* (nesse cenário, ambas tarefas irão compartilhar memória, disco, entre outros recursos do *host*). Nessa situação, a aplicação EasyGrid AMS poderá se adequar ao ambiente submetido conforme a carga de trabalho dos recursos alocados, realizando um balanceamento dinâmico das tarefas sobre todos os recursos alocados a ela, com a finalidade de tirar o máximo de proveito destes.

Na etapa anterior, o *conector* define como a aplicação EasyGrid AMS deverá gerenciar seus processos. Existem, basicamente, três formas:

1. Escalonamento Estático: Nesse caso, o escalonamento estático será respeitado, mantendo as tarefas alocadas nas máquinas definidas pelas Etapa 2. O gerenciador poderá apenas reordenar os processos, conforme o peso associado a cada tarefa.

2. Escalonamento Dinâmico: Essa estratégia realiza um balanceamento dinâmico das tarefas entre as máquinas alocadas. Inicialmente, as tarefas iniciais, indicadas pelo escalonador estático da Etapa 2, são colocadas em execução nas máquinas alocadas, as demais tarefas prontas podem ser transferidas para outras máquinas, de acordo com o tempo previsto de término das aplicações associadas a cada máquina.

3. Escalonamento Altruísta: Essa estratégia de gerenciamento necessita de uma meta (previsão de término). Com isso, caso a aplicação EasyGrid AMS identifique que suas tarefas estão disputando o acesso aos recursos com outras aplicações, a aplicação EasyGrid AMS

pode verificar se está adiantada, em relação a sua meta, e pode deixar executar suas tarefas no recurso que está compartilhando, cedendo o acesso ao recurso para a outra aplicação que está concorrendo aos recursos. Enquanto isso, a aplicação EasyGrid AMS pode tomar três posturas: as tarefas desse recurso compartilhado podem ser transferidas para outro recurso, pode-se reduzir dinamicamente a carga de trabalho exigida do recurso ou pode-se suspender, temporariamente, as execuções nesse recurso. A aplicação EasyGrid volta a utilizar o recurso quando ele estiver ocioso ou quando a meta estiver próxima de não ser cumprida.

4.3 Conclusão

No inicio do capítulo, o problema é evidenciado pela sua complexidade e necessidade de um ambiente de processamento de alto desempenho. Para tentar suprir a falta de um único ambiente não ter a capacidade computacional esperada para executar vários *workflows* simultaneamente, buscou-se uma integração que fosse realizada de forma transparente para os usuários pré-existentes de cada ambiente computacional de alto desempenho.

A abordagem proposta mostra-se capaz de integrar diferentes ambientes computacionais (inclusive ambientes que utilizam sistemas de gerenciamento de recursos que não suportam o uso de *workflows* científicos), e mesmo assim, formarão uma e-infraestrutura computacional capaz de tratar aplicações científicas que possam ser representadas por *workflows* ou DAGs. Além disso, com o uso do EasyGrid AMS, um gerenciador de tarefas que fica acoplado com a aplicação, cria-se a ideia de um *workflow* autônomo, capaz de se autogerenciar em ambientes que não oferecem suporte a *workflows*, oferece flexibilidade no gerenciamento das tarefas, tornando-se uma poderosa ferramenta para explorar ao máximo os recursos. Diferente das implementações anteriores do EasyGrid AMS, as "tarefas" podem, agora, ser aplicações executáveis ou aplicações externas. Anteriormente, toda aplicação que fosse fazer uso do EasyGrid AMS, precisava, obrigatoriamente, ser compilada com suas bibliotecas.

Capítulo 5

Avaliação das Abordagens de Gerenciamento para *Workflows ENM*

Conforme apresentado no Capítulo 2, o gerenciamento de *workflows* consiste, além da alocação das tarefas em recursos, o gerenciamento das dependências entre as tarefas (ordem de execução), responsável por permitir que somente as tarefas prontas (sem dependência) para execução sejam escalonadas (definindo um recurso para sua execução). Como esse gerenciamento das dependências pode ser desassociado do escalonamento, torna-se possível ter as três diferentes abordagens (orientações) discutidas, conceitualmente, na Seção 2.3.4 e aplicadas nesse capítulo sobre a infraestrutura apresentada no Capítulo 4.

Essa capítulo está dividido da seguinte forma: A primeira seção aplica a abordagem apresentada no Capítulo 4, para integrar diferentes ambientes e torná-los mais robustos para a execução de *workflows* do openModeller. Analisa-se o desempenho da execução de um experimento openModeller, com as três diferentes orientações (visões) de gerenciamento, são elas: UMS, RMS e AMS. Um *script* meta-escalonador é utilizado para representar o UMS, o RMS é representado pelo HTCondor com o uso do DAGMan; e o EasyGrid é utilizado para representar o AMS, Seções 5.2, 5.3 e 5.4, respectivamente. Por fim, apresenta-se uma comparação entre as três abordagens na Seção 5.5 e a última seção trás algumas conclusões obtidas após a execução dos testes abordados neste capítulo.

5.1 Aplicação da abordagem proposta no estudo de caso

O openModeller possui duas versões para *download*, uma delas é a "oM Desktop" que fornece uma interface gráfica para o usuário. O ponto negativo dessa versão é não ser distribuída, executando apenas no computador local e, dependendo dos dados utilizados, pode necessitar de muitas horas de processamento para criar um único modelo de nicho ecológico. A outra versão disponível para

download é a "*oM Server*", exclusiva para servidores e conhecida como OMWS - *OpenModeller Web Service*. A diferença é que essa versão não possui interface gráfica, mas permite que vários clientes de diferentes locais submetam solicitações de modelagem de nichos ecológicos para esse servidor. Porém, por padrão, essa versão para servidor também não permite a execução em *clusters* ou grades computacionais [60].

5.1.1 Descrição da e-infraestrutura

Na Seção 4.2 foi realizada uma descrição genérica da solução proposta. Nessa seção, cada componente será renomeado e utilizado para executar o openModeller na criação de modelos de nichos ecológicos em ambientes de alto desempenho. O OMWS, versão para servidor do openModeller, utiliza o protocolo SOAP (*Simple Object Access Protocol*) para troca de informações estruturadas entre os usuários e o servidor, criando um ponto único de acesso do sistema e permitindo múltiplas requisições de diferentes clientes.

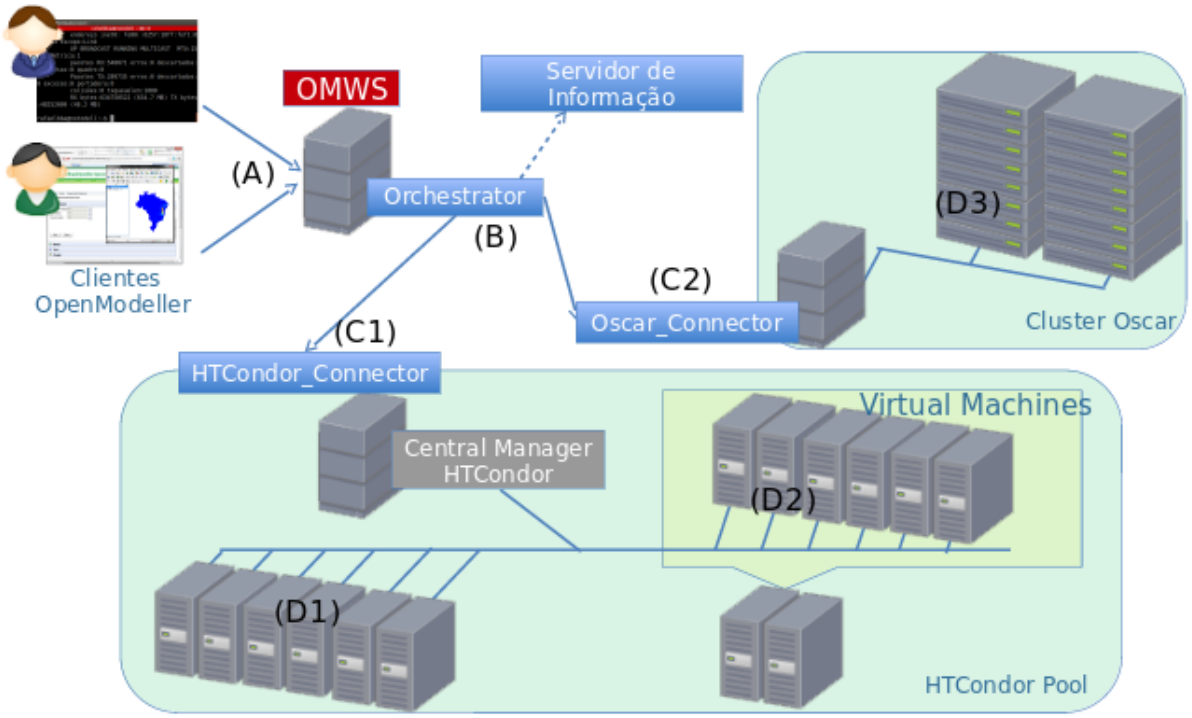


Figura 5.1: Visão geral da abordagem proposta com o estudo de caso.

O ambiente real da abordagem proposta é ilustrado na Figura 5.1. As três etapas apresentadas na Seção 4.2, foram subdivididas em 4 principais fases, a primeira etapa da abordagem está dividida em fases (A) e (B), a etapa 2 é representada por (C1) e/ou (C2), e a terceira etapa é apresentada como a fase (D1), (D2) e/ou (D3). As fases especificas de (C) e de (D) utilizadas

dependem da disponibilidade de recursos daquele tipo.

Fase (A) - Essa é a primeira etapa do processo completo de executar *workflows* científicos utilizando a abordagem proposta. A requisição enviada pelo usuário, através de um arquivo XML, é recebida pelo servidor OMWS que retira os dados associados ao SOAP, mantendo apenas as informações referentes as requisições do OpenModeller. Cada requisição recebe uma identificação única que é denominada como *ticket*.

A versão OMWS 1 aceita apenas uma única requisição por arquivo XML, em contra-partida, a versão OMWS 2 permite a descrição de um *workflow*, composto por inúmeras requisições, no qual, cada requisição (ou tarefa do *workflow*) receberá um *ticket* para identificá-lo no servidor. As requisição do tipo OMWS 2, usualmente referenciadas apenas pelo seu respectivo *ticket*, contém: tipo de tarefa (por exemplo: criação, teste ou projeção de um modelo ecológico), características (pontos de ocorrência e ausência, camadas ambientais e algoritmo para modelagem), além da indicação das dependências entre todas as tarefas envolvidas.

Fase (B) - O *Resource Broker*, denominado na Figura 5.1 como *Orchestrator*, está localizado na mesma máquina que possui o OMWS (oferece-se apoio as duas versões existentes do open-Modeller para servidor, OMWS 1 e OMWS 2). O objetivo inicial do *Orchestrator* é identificar a existência de novos *tickets* e analisar detalhadamente todas as características das tarefas, assim como suas respectivas dependências (essa análise é fundamental para conhecer as características de cada *ticket* e, posteriormente, direcioná-los para o ambiente que atenda as necessidades envolvidas de forma mais adequada). Por exemplo, uma das dependências que o *Orchestrator* precisa verificar é se todas as camadas ambientais necessárias para a modelagem estão presentes no ambiente computacional, portanto, caso não exista alguma camada ambiental, o *Orchestrator* assegura-se de obtê-la e de disponibilizá-la ao ambiente que irá executar o *ticket*.

O *Orchestrator* precisa identificar o peso (fator que representa a complexidade de uma tarefa) associado a execução de cada tarefa do *workflow*, para isso, analisa-se as informações descritas nos *tickets* (tipo de tarefa, camadas ambientais, pontos de ocorrência e ausência da espécie e algoritmo que será utilizado na modelagem) e consulta-se o "Servidor de Informação" apresentado na Figura 4.1, do Capítulo 4 para obter o peso associado a cada *ticket*.

Para definir o peso associados as tarefas do OpenModeller, utiliza a média dos tempos obtidos com a execução individual, em um ambiente dedicado, de cada tarefa do Experimento 5E, gerando pesos proporcionais entre suas tarefas, mas que não representam o tempo real de execução que será necessário (não é possível garantir o tempo de execução, pois o ambiente possui máquinas heterogêneas e a concorrência por recursos afeta diretamente o tempo de execução das tarefas). Essa predição, imprecisa, é corrigida pelo EasyGrid AMS na terceira etapa, portanto o desempenho do *workflow* não é afetado pela falta de precisão desses pesos.

Ainda nessa etapa, o *Orchestrator* acessa o "servidor de informação" para ver a disponibilidade dos recursos em cada ambiente computacional. Com base nesses recursos disponíveis e nos pesos associados a cada tarefa do *workflow*, o *Orchestrator* faz uso de todas essas informações para decidir qual o ambiente mais adequado para executar esse *ticket* (todas as tarefas do *ticket* podem ser enviadas para o mesmo ambiente ou podem ser divididas entre ambientes diferentes, mas mantendo suas dependências originais). Logo após essas análises, antes de enviar o *ticket* (arquivo XML que descreve o *workflow* com todas as suas tarefas) para os recursos disponíveis, o *Orchestrator* pode dividir o *ticket* (*workflow*), de acordo com os recursos disponíveis, criando *tickets* com *sub-workflows* (pode ser necessário duplicar alguma tarefa para respeitar as dependências associadas ao DAG do *workflow*). Por exemplo, a Figura 5.2, apresenta em (a) um *workflow* que pode ser dividido em duas partes, essa divisão é ilustrada em (b) e (c), onde a tarefa inicial A1 foi duplicada para criar uma independência entre os dois *sub-workflows*. A criação de *sub-workflows* é importante para balancear a carga do *workflow* entre todos os recursos disponíveis.

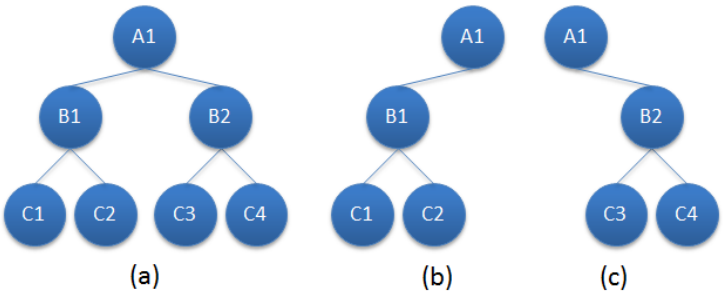


Figura 5.2: Exemplo de divisão de *workflow*. (a)*workflow* original (formado por A1, B1, B2, C1, C2, C3 e C4), (b) *subworkflow* (formado por A1, B1, C1 e C2), (c) *subworkflow* (formado por A1, B2, C3 e C4).

Posteriormente, conforme a Figura 5.1, cada e-infraestrutura possui um componente, nesse caso, denominada HTCondor_Connector e Oscar_Connector (localizados no ambiente HTCondor e no *cluster* Oscar, respectivamente, por exemplo), responsáveis por receber os *workflows* que foram enviados pelo *Orchestrator*, analisar e criar um arquivo de submissão de acordo com o sistema de gerenciamento local. O conector não irá realizar qualquer tratamento no *workflow* (dividindo ou agrupando tarefas), o seu objetivo, além de criar o arquivo de submissão para o sistema de gerenciamento do ambiente local, é criar, com o uso de uma aplicação EasyGrid AMS, a ideia de um *workflow* autônomo. A aplicação EasyGrid AMS será responsável por gerenciar todas as dependências das tarefas envolvidas e, irá se adequar aos recursos submetidos conforme a carga de uso, realizando um autobalanceamento dinâmico das tarefas sobre todos os recursos envolvidos, com a finalidade explorar ao máximo a natureza dinâmica dos recursos alocados em busca de uma execução eficiente.

Fase (C1) - O *conector* utilizado no ambiente específico do HTCondor, denominado de *HTCondor_ conector*, é responsável por receber e analisar o *ticket*, criar uma aplicação EasyGrid AMS para executar o *workflow* representado no *ticket* e criar um arquivo de submissão que utilize o universo *parallel* do HTCondor para submeter a aplicação EasyGrid AMS. Essa aplicação EasyGrid AMS é formada por todas as tarefas que compõem o *workflow* e é responsável por escalonar e gerenciar suas próprias tarefas nos recursos alocados pelo HTCondor. Ainda nessa etapa, precisa-se ter uma ideia do peso total de processamento do *workflow*, para decidir sobre a quantidade de recursos computacionais (*hosts*) que serão requisitados. O peso de cada tarefa também é importante para o escalonamento inicial (estático) e gerenciamento das tarefas pelo EasyGrid AMS na próxima etapa.

Nesse ambiente computacional existe o DAGMan, que é o gerenciador de *workflows* próprio do HTCondor [37], porém ele não explora a natureza dinâmica do uso dos recursos existentes. Em outras palavras, a forma de alocação do HTCondor é através de uma lista de *slots* de processamento e, a quantidade de *slots* oferecida por cada recurso está associada com o número de processos que poderão ser executados concorrentemente em cada máquina. Logo, como não existe um isolamento da capacidade computacional de cada slot, pode ocorrer, quando existe mais de um slot por recurso, interferência onde um processo pode influenciar na execução do outro. Além disso, a quantidade de *slots* é determinado por máquina e não por aplicação.

Como esse ambiente oferece uma forma de gerenciamento de *workflows*, pode-se deixar de usar o EasyGrid AMS e utilizar o meta-escalonador DAGMan. Para isso, precisa-se criar um arquivo de submissão para cada tarefa do *workflow* e um arquivo com o formato do DAGMan indicando as dependências entre cada tarefa do *workflow*.

Fase (C2) - Essa fase representa o *Oscar_ conector*, desenvolvido especificamente para o *cluster* compartilhado denominado OSCAR (um cluster HPC do IC-UFF), esse *cluster* possui um sistema gerenciador de recursos distribuídos semelhante ao TORQUE e, atualmente, esse gerenciador não tem suporte ao uso de *workflow*. Porém, essa limitação pode ser contornada com o uso da abordagem proposta nesse trabalho, que utiliza um sistema gerenciador de aplicação (EasyGrid AMS). O sistema de gerenciamento local submeterá, como se fosse qualquer outra requisição, uma aplicação do EasyGrid AMS que será responsável pelo seu próprio escalonamento, ou seja, cada aplicação EasyGrid está associada a um *workflow* e irá executar o escalonamento das suas próprias tarefas (as tarefas que formam o *workflow*).

Esse gerenciador local não exige a criação de um arquivo de submissão, basta que seja definido quantidade de *hosts* e a linha de execução da aplicação AMS. Portanto, necessita-se ter conhecimento da carga de trabalho associada ao *workflow* para requisitar uma quantidade adequada de *hosts*.

Fase (D1, D2 e D3) - Com essa abordagem de utilizar uma aplicação AMS, pode-se

observar dois diferentes pontos de vista: Na visão dos ambientes envolvidos, é submetido um *job* que consiste de uma aplicação que utiliza vários núcleos de processamento (inclusive em diferentes *hosts*). Sendo um ambiente compartilhado, terão vários *jobs* em execução concorrentemente. Porém, do ponto de vista da aplicação autônoma submetida, será executado um Sistema de Gerenciamento de Aplicação capaz de controlar seu *workflow*, respeitando a dependência entre as tarefas, capaz de realizar dinamicamente um balanceamento de carga entre os recursos alocados (característica relevante por se tratar de um ambiente compartilhado) e que possui ferramentas de tolerância a falhas associadas. Apesar de todas essas características e ferramentas, o EasyGrid AMS não causa um *overhead* significativo, conforme observado nas próximas seções.

5.2 *Script* meta-escalonador

O Sistema de Gerenciamento do Usuário (UMS - *User Management System*) permite que usuários gerenciem suas próprias aplicações individualmente, provendo um gerenciamento independente dos outros. O UMS tem uma visão centrada no usuário, onde a aplicação é gerenciada de acordo com suas características e requisitos especificados pelo usuário, como, por exemplo, a execução com tempo limitado ou utilizando recursos específicos. O UMS é descentralizado, entre vários usuários coletivamente, onde cada usuário é responsável pela gestão de suas aplicações. Neste caso, existe um UMS para cada usuário.

Um meta-escalonador pode ser utilizado para controlar o gerenciamento de *workflow*, como um UMS, porém, necessita utilizar um escalonador de tarefas para associar cada tarefa em uma máquina. Um *script* cliente, gerenciador de dependências, foi desenvolvido pelos pesquisadores do CRIA, fazendo uso da linguagem Perl, com a finalidade de enviar as requisições para o Servidor Web do openModeller (OMWS 1, descrito na Seção 3.2.1) e gerenciar o *workflows* descritos na Seção 3.3.

O *script* precisa controlar as dependências entre as tarefas do *workflow*, para isso, é necessário verificar constantemente o progresso das tarefas que antecede outras tarefas, para submeter as tarefas dependentes, logo que possível. Toda a manipulação do DAG deve ser feita antes do servidor, no lado do cliente, pelo meta-escalonador. No lado do servidor, o HTCondor é utilizado meramente para selecionar um recurso disponível para executar a tarefa. No servidor, para cada requisição é criado um arquivo de submissão do HTCondor.

A Figura 5.3 mostra como é realizado o processo de requisição do UMS. Um script meta-escalonador é executado no lado do cliente e é responsável por criar e submeter as requisições ao servidor. Após a submissão, é necessário ficar continuamente solicitando ao servidor o progressos das tarefas submetidas para, posteriormente, coletar os dados de saída das tarefas que foram concluídas e avaliar quais dados são necessários na entrada de tarefas que possuíam al-

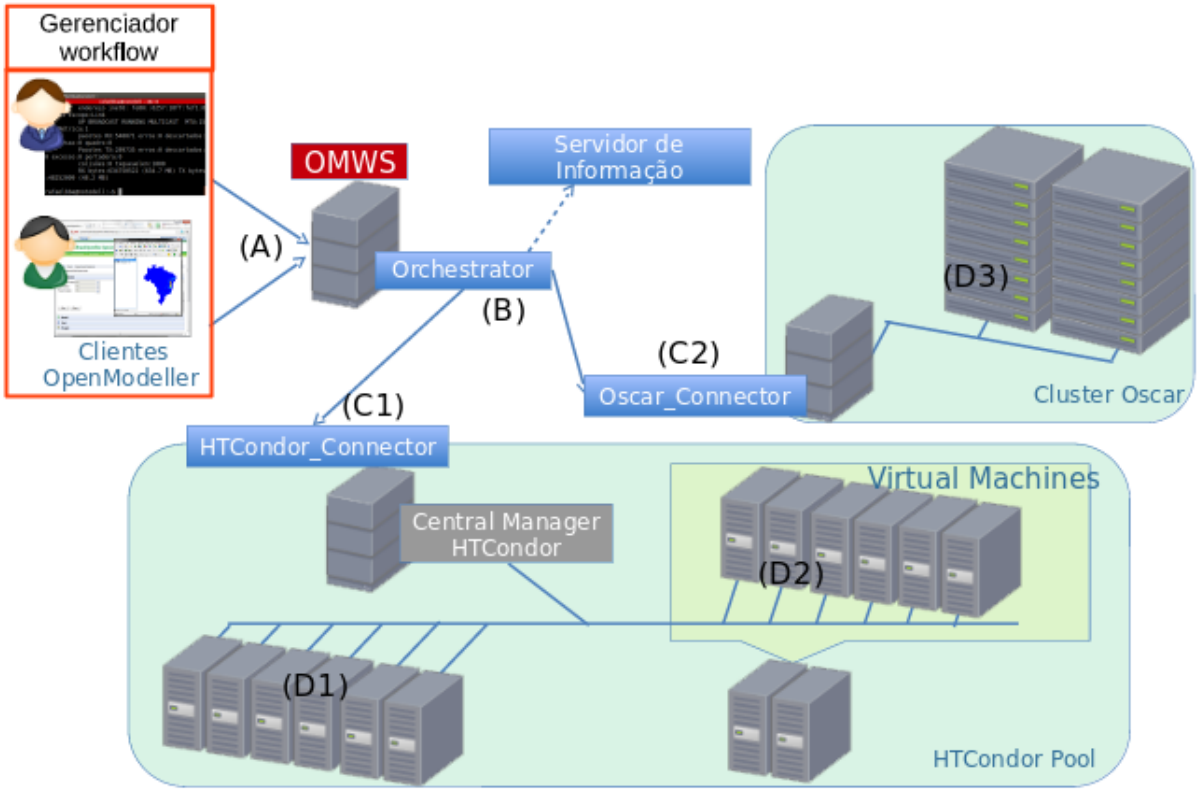


Figura 5.3: Sistema de Gerenciamento do Usuário.

guma dependência e aguardavam para serem submetidas. Coletar um dado gerado por uma etapa (transferindo dados do servidor para o cliente) e depois reenviar esses dados como entrada para outra etapa (transferindo dados do cliente para o servidor), gera um tráfego de rede que pode impactar no desempenho da execução do *workflow*. Nesse caso, o meta-escalonador não possui nenhum controle sobre os recursos existentes e não interfere na forma que as tarefas serão escalonadas. Essas atividades são completamente transparentes para o usuário.

5.3 HTCondor e DAGMan

O RMS [45] é tipicamente um sistema centralizado responsável por associar tarefas aos recursos, somente considerando as características requisitadas pela aplicação. Os RMSs representam a maioria dos sistemas de gerenciamento desenvolvidos para computação em cluster, grades e nuvem, onde o objetivo é maximizar a utilização do sistema, independente dos requisitos e das características intrínsecas das aplicações envolvidas. RMSs são voltados para o sistema, possuem uma visão global de recursos e são centralizados. Portanto, diversas aplicações são administradas simultaneamente e serviços como escalonamento, monitoramento, gerenciamento de falhas, entre outros, devem ser instalados em um servidor central.

O Projeto Condor, atualmente denominado HTCondor (*High Throughput Condor*) [14], co-

meçou em 1988 e se concentra em fornecer um acesso confiável a computação durante longos períodos de tempo. HTCondor é um sistema de gerenciamento de carga de trabalho especializado para trabalhos de computação intensiva. Ele fornece um mecanismo voltado a fila de trabalho, monitoramento e gestão de recursos, permite política de escalonamento e esquema de prioridade. Assim, os usuários podem enviar tarefas individuais, sequencias ou paralelas, e o HTCondor irá gerenciar a execução global das tarefas. Desta forma, muitos usuários HTCondor têm utilizado-o não só na execução de tarefas de longa duração, mas também com sequências de tarefas complexas, como *workflows*. Devido a isso, foi desenvolvido o DAGMan (ou *Directed Acyclic Graph Manager*), que permite aos usuários enviar *workflows* para HTCondor. O principal objetivo do DAGMan é automatizar a submissão e gerenciamento de *workflows* complexos que envolvem muitas tarefas com dependências. DAGMan foca em confiabilidade e tolerância a falhas.

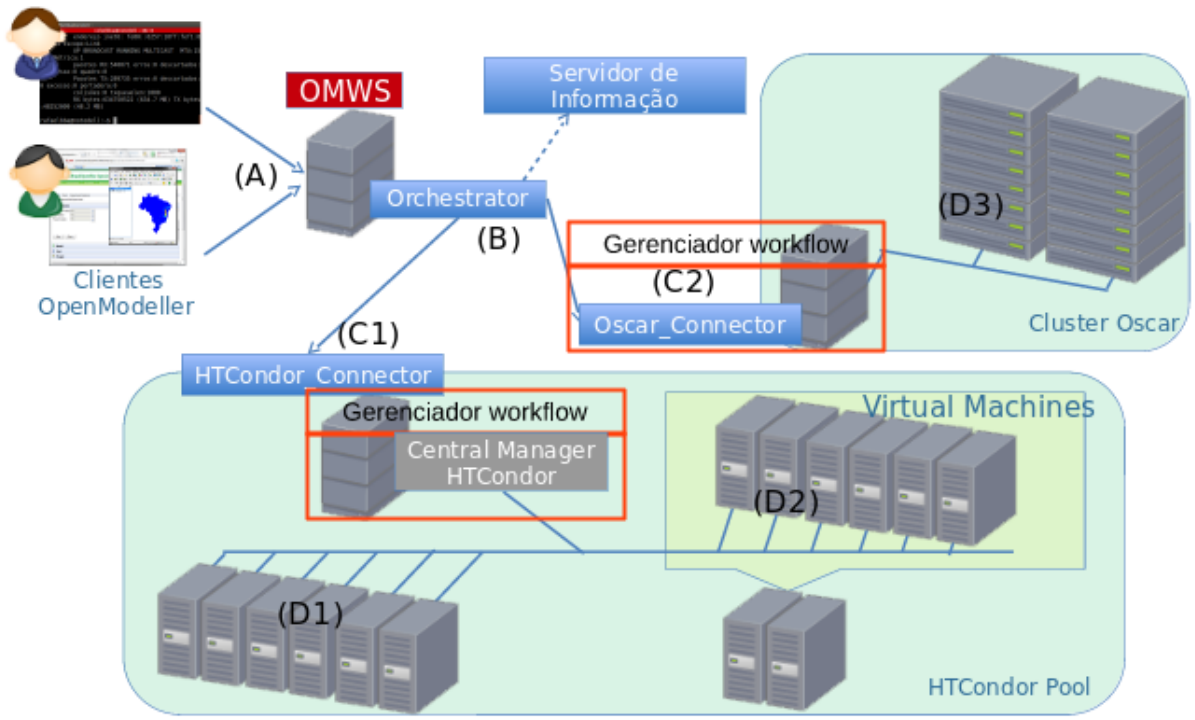


Figura 5.4: Sistema de Gerenciamento de Recursos.

O HTCondor é um sistema de gerenciamento de recursos que se concentram nas necessidades de recursos como média de carga, uso de memória e outros. Ao contrário da abordagem UMS, HTCondor agora é usado para tratar o DAG existente no *workflow* e fazer o escalonamento de cada tarefa nas máquinas disponíveis. Toda manipulação do DAG é feito no servidor, como pode ser visto na Figura 5.4. O usuário faz uma solicitação simples especificando informações sobre DAG e as tarefas envolvidas. O servidor OMWS 2 (descrito na Seção 3.2.2) recebe este pedido

e submete ao DAGMan HTCondor que irá gerenciar e executar a sequência de tarefas em cada máquina.

Para submeter um experimento para o HTCondor, é preciso conhecer previamente as tarefas envolvidas no *workflow* e as respectivas dependências do DAG. Enquanto HTCondor é responsável por realmente começar uma tarefa, o DAGMan gerencia o *workflow*, associando uma tarefa pronta para uma máquina ociosa na sequência prevista, e as submete para HTCondor. Isso nem sempre é eficiente porque faz o uso da politica FCFS *First Come First Served* - que não é um escalonador eficiente. Considerando todo o DAG, a sequência de execução pode ser uma questão importante para reduzir a execução global do experimento, o *makespan*. Para melhorar esta situação, é possível definir a prioridade das tarefas (definir os custos das tarefas que compõem o *workflow*) e essa informação pode melhorar o escalonamento e, consequentemente, o tempo total de execução do *workflow*.

5.4 EasyGrid AMS

O AMS [9] possui uma visão voltada a aplicação, onde requisitos e características intrínsecas da aplicação podem ser consideradas no gerenciamento. Nesse caso, os recursos do sistema são avaliados afim de determinar a melhor execução para as tarefas da aplicação. Isso sugere que o gerenciamento possa ser feito pela própria aplicação ou embutido nela. Possibilitando que os requisitos da aplicação e dos recursos sejam considerados no gerenciamento. EasyGrid AMS é um exemplo de Sistema de Gerenciamento de Aplicação.

Diferentemente do sistema HTCondor com DAGMan (sistemas RMS), o EasyGrid [9] é um sistema de gerenciamento de aplicação, não sendo apenas um gerenciador de *workflow*. O *middleware* EasyGrid é um sistema de gerenciamento de aplicação distribuído hierarquicamente, embutido em uma aplicação paralela MPI para facilitar a execução eficiente em ambientes computacionais distribuídos.

As aplicações podem apresentar diferentes tipos topológicos como *bag of tasks* e DAG. Acoplando aplicações com o EasyGrid AMS, estas podem ser transformadas em versões autonômicas que gerenciam sua própria execução. Os benefícios desta abordagem incluem políticas para escalonamento, comunicação e tolerância a falhas, adaptando-as de acordo com às necessidades específicas de cada aplicação, conduzindo assim a um melhor desempenho [18, 52, 53, 55, 78].

O escalonador do EasyGrid AMS pode ser estático ou dinâmico. O escalonador estático sempre é necessário para iniciar o *workflow* e o dinâmico, se for necessário, é utilizado para detectar dinamicamente a carga de trabalho desequilibrada entre os nós, redistribuindo as tarefas que ainda não começaram a ser executadas. O gerenciador dinâmico do EasyGrid segue um modelo de tarefas de granularidade fina [54] e, embora ele não tenha sido elaborado especificamente para

o modelo de *workflow* (com granularidade grossa), este trabalho pretende investigar se pode ser utilizado para esse propósito. Além do DAG, o EasyGrid AMS pode conter com informações sobre os pesos das tarefas, melhorando o escalonamento.

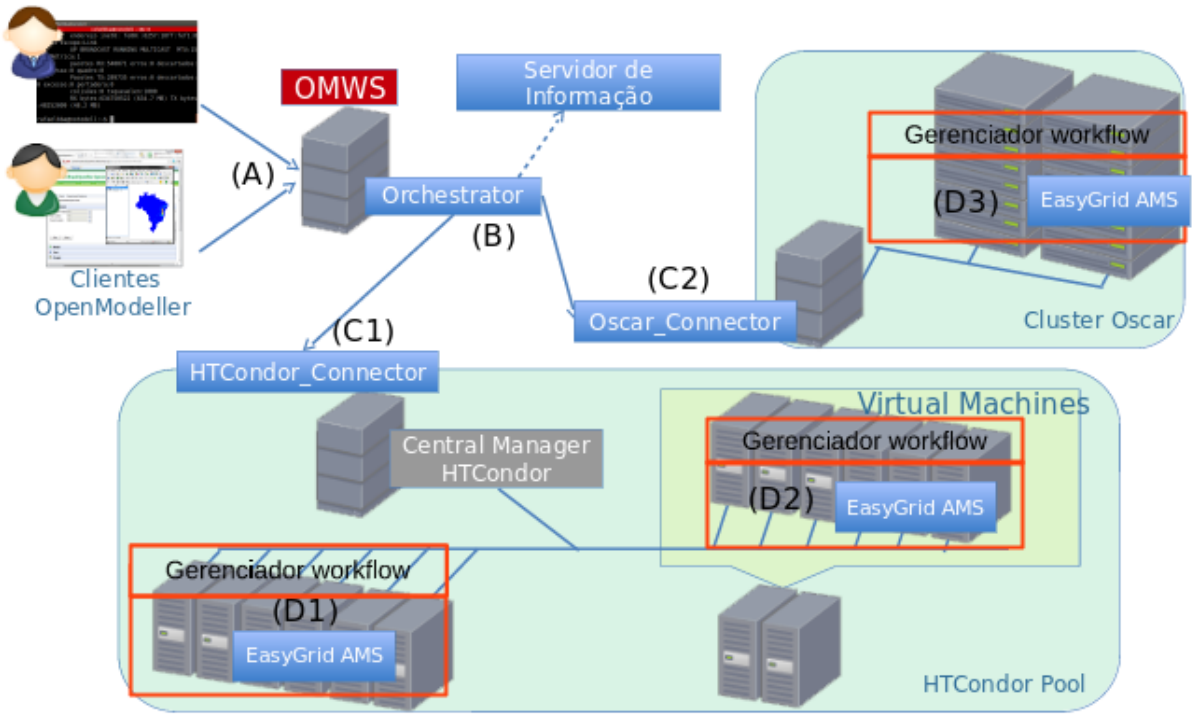


Figura 5.5: Sistema de Gerenciamento da Aplicação.

A Figura 5.5 apresenta um resumo da requisição utilizando o EasyGrid AMS. Um *workflow* AMS pode rodar em um cluster oportunístico (D1), um cluster clássico (D3) ou num ambiente de nuvem (D2) em harmonia com o sistema de gerenciamento local usado para executar *jobs* ou aplicações convencionais concorrentemente. Similarmente ao uso do HTCondor, uma simples requisição é submetida ao servidor OMWS e o servidor é quem chama o AMS para executar os processos sobre seus *hosts*. O AMS irá gerar o escalonamento inicial para iniciar as tarefas nas máquinas envolvidas.

A principal diferença entre os gerenciadores DAGMan e EasyGrid está associado a forma de escalonamento de cada um. O HTCondor cria uma lista de tarefas prontas, de acordo com suas prioridades, e começa a submetê-las em suas máquinas ociosas. No AMS é possível adicionar tomadas de decisões durante seu escalonamento dinâmico em tempo real, de acordo com a variação dos recursos. Utilizando esse tipo de gerenciamento, a decisão está próxima da aplicação, fornecendo certa autonomia a esta.

5.5 Uma Comparação UMS x RMS x AMS

Essa seção pretente fazer uma comparação conceitual das três diferentes orientações de gerenciamento para *workflows* ENM, com as duas versões disponíveis do openModeller (OMWS 1 e OMWS 2).

5.5.1 Avaliação da primeira versão do openModeller - OMWS 1

O ambiente computacional utilizado nas comparações dessa seção é formado por 8 máquinas, de uso dedicado, do tipo HT1 (compostos por: 2 processadores Intel Xeon 3GHz, 2.5GB RAM e disco SCSI). Para avaliar o OMWS 1, utilizou-se o Experimento 1E, descrito na Seção 3.3, que cria um modelo de dispersão de uma única espécie. O *workflow* relacionado ao Experimento 1E é composto por 115 tarefas. Foram realizados dois tipos diferentes de testes, o primeiro utiliza as aplicações originais do openModeller (om_model para criar um modelo, om_test para testar o modelo criado e om_project para projetar o modelo criado), o segundo teste utiliza as mesmas aplicações om_model e om_test originais, mas explora o uso de uma versão paralela do om_project [63].

a) Testes com as aplicações originais do openModeller

Para executar o *workflow* com o gerenciamento realizado pelo usuário (UMS), utilizou-se um *script* Python, conforme descrito na Seção 2.3.4.1, esse *script* envia 115 requisições compostas apenas por uma tarefa e o próprio *script* realiza o gerenciamento das dependências. Além do uso do *script* meta-escalonador, foram utilizadas outras duas formas diferentes, uma com o uso do DAGMan que é um gerenciador de sistema do HTCondor (abordagem RMS, vista na Seção 2.3.4.2), e a outra com o uso do EasyGrid AMS, um sistema de gerenciamento orientado pela aplicação (descrito na Seção 2.3.4.3).

O tempo de execução obtido com essas três diferentes abordagens é apresentado na Figura 5.7, onde obteve-se 3457 segundos ao executar o *script* meta-escalonador, 3037 segundos com o uso do HTCondor-DAGMan e 3002 segundos com o EasyGrid AMS, ambos sendo 88% do tempo do UMS.

A Figura 5.6 apresenta o escalonamento da execução do experimento com o uso do meta-escalonador (UMS), nessa figura, as colunas representam processos concorrentes (pode-se fazer uma alusão a núcleos de processamentos), onde cada bloco escuro representa uma tarefa de criação de modelo (om_model), as tarefas claras representam os testes (om_test) e os blocos representados por um cinza intermediário, indicam a execução de uma tarefa de projeção (om_project). Percebe-se que se não fosse a execução do último om_project, o experimento poderia ter terminado cerca de 1000 segundos antes (± 30% a menos). O

próximo teste realiza exatamente a paralelização dessa tarefa, usando a técnica proposta em [63].

Figura 5.6: Escalonamento das tarefas obtidos com o meta-escalonador (UMS).

b) Testes com o uso da projeção paralela

Nesse teste foi utilizado o mesmo *workflow* anterior, a diferença está associada à adoção de uma implementação paralela da projeção (descrito detalhadamente em [63]). Desta forma, obteve-se os seguintes resultados: a execução do *script* Meta-Escalonador precisou de 2705 segundos no total, o HTCondor-DAGMan executou em 2481 segundos e o Easy-Grid AMS executou completamente em 2355 segundos, 92% e 87% do tempo do UMS, respectivamente, conforme Figura 5.7.

Para compreender o motivo da redução do tempo total dos três testes, ilustrou-se o escalonamento efetuado em cada um dos testes na Figura 5.8, em (a) está sendo representado a execução com o UMS, (b) utilizou-se o HTCondor-DAGMan e (c) o EasyGrid AMS. Entenda-se que o bloco escuro representa uma tarefa de criação de modelo (om_model), as tarefas claras representam os testes (om_test) e os blocos representados por um cinza intermediário são as projeções paralelas (om_project).

Foram 3 instâncias de om_project que executaram em paralelo usando 4 núcleos cada. A execução do meta-escalonador com o uso da projeção paralela reduziu o tempo de execução em 22%, o HTCondor-DAGMan reduziu 19% e o EasyGrid AMS diminuiu 23%. Destaca-se a forma de gerenciamento aplicada as tarefas paralelas do HTCondor-DAGMan e do EasyGrid AMS. O

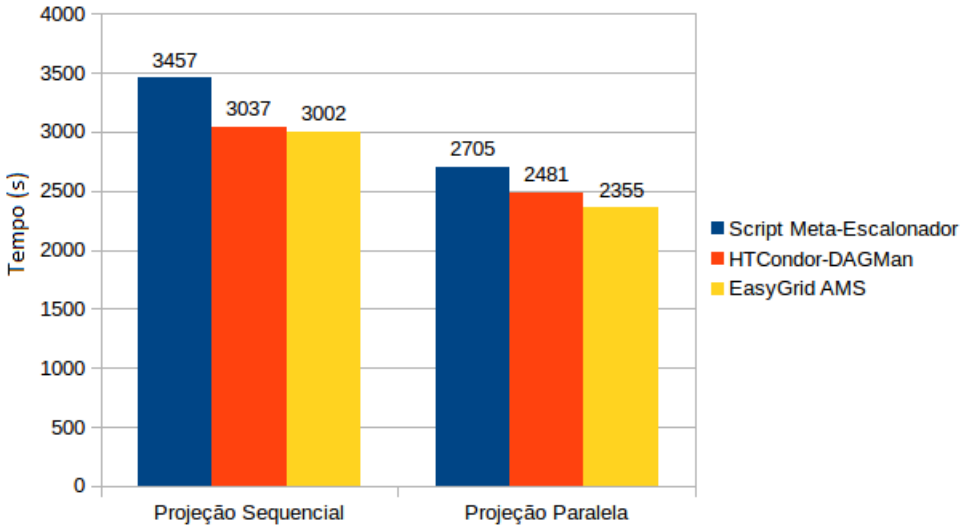


Figura 5.7: Avaliação do tempo de execução das três abordagens com a aplicação da projeção sequêncial e paralela.

RMS precisa esperar a disponibilidade de recursos suficiente para sincronizar o início de todas as tarefas que compor a aplicação paralela, como visto nos casos (a) e (b). Enquanto isso, o EasyGrid AMS é capaz de disparar um número menor de tarefas da aplicação paralela logo que possível, e durante a execução vai adaptando a disponibilidade de recursos, sem nenhum tipo de sincronização inicial. Nota-se também que as tarefas paralelas da mesma aplicação terminam no mesmo tempo independente de seus tempos de início, graças ao escalonador dinâmico do EasyGrid AMS.

5.5.2 Avaliação da segunda versão do openModeller - OMWS 2

O objetivo dessa seção é analisar o comportamento e escalabilidade das três abordagens (UMS, RMS e AMS), ao aplicar um *workflow* maior. Nesses testes foram utilizadas máquinas do tipo HT2 (12 núcleos Intel Xeon 2.6 GHz e 24 GB RAM), totalmente livres de interferência externa, tornando os resultados menos variáveis. Utilizou-se apenas as aplicações oferecidas originalmente pelo openModeller (om_model, om_test e om_project). Utilizou-se o Experimento 5E, descrito na Seção 3.3, que cria um modelo de dispersão de nicho ecológico de 5 diferentes espécies. O *workflow* desse experimento é formado por 325 tarefas originais do openModeller (om_model, om_test e om_project), mas além dessas, foi necessária desenvolver uma aplicação adicional, denominada build_request, que é usada no meio das sequência de execução (om_model e om_test) e (om_model e om_project). Essa aplicação build_request é uma tarefa que precisa ser escalonada no *workflow* e é responsável por coletar o modelo gerado pelo om_model e adicioná-lo ao arquivo que contém as configurações necessárias para a execução do om_test ou do om_project. Portanto, o *workflow* resultante é formado por 500 tarefas. Destaca-se que o build_request é

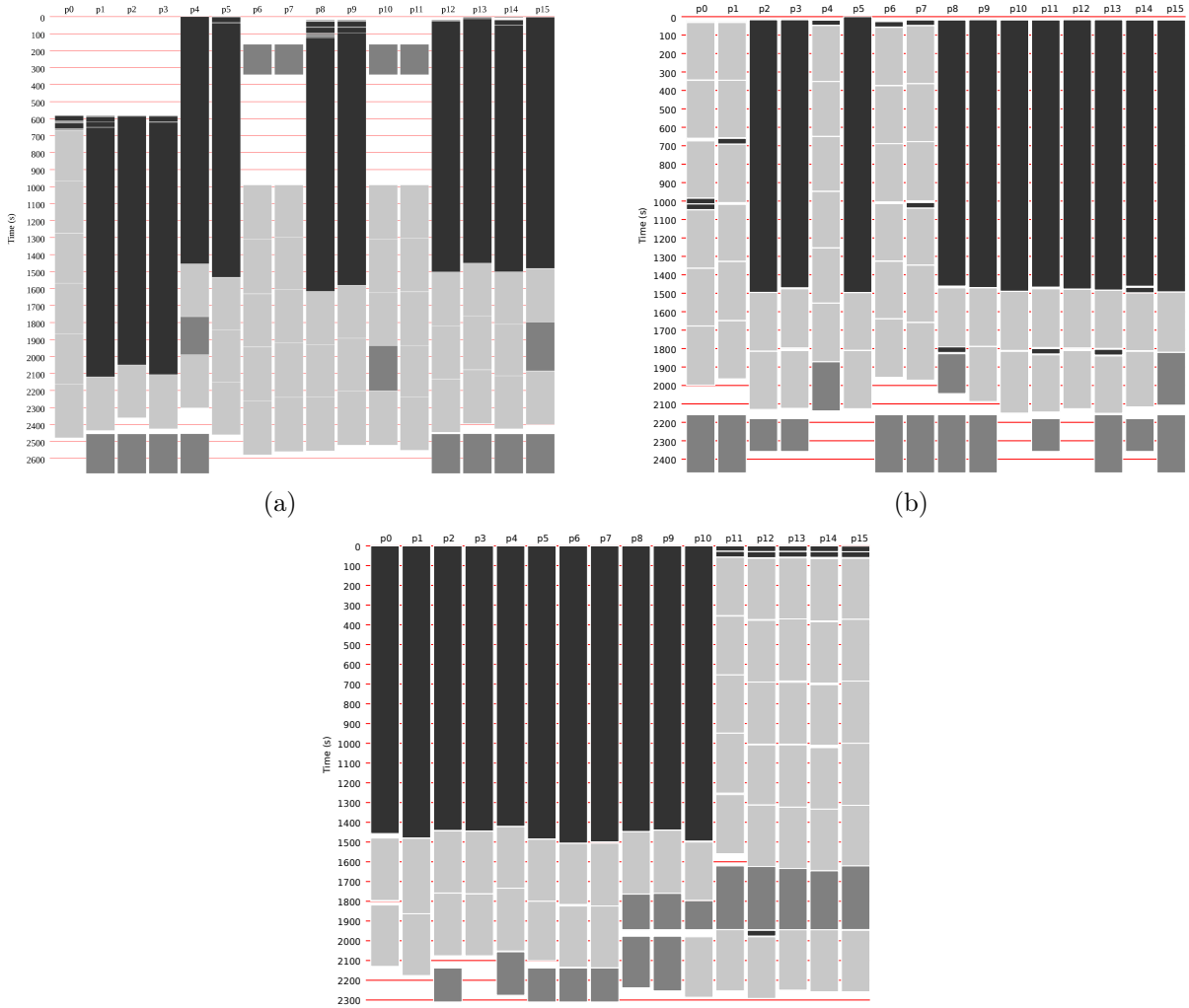


Figura 5.8: Execução concorrente da aplicação OM_PROJECT em oito máquina do tipo HT1. (a) Meta-escalonador, (b) HTCondor-DAGMan e (c) EasyGrid AMS.

uma tarefa muita rápida em relação os outras e não aparece nos gráfico de escalonamento, mas é essencial na execução do *workflow*. Os dados obtidos nesses testes concentram-se na Tabela 5.1. O tempo sequencial para este *workflow* foi de 29596 segundos.

Tabela 5.1: Média de tempo de execução de cada uma das diferentes abordagens.

Núcleos	12	24	36	48	60	72	84	96
UMS	3500	1891	1533	1272	1215	1232	1151	1138
HTCondor	3403	1936	1230	1068	1012	1015	1034	1041
AMS	3295	1715	1182	1056	967	967	936	930

As próximas figuras (5.9, 5.10, 5.11 e 5.12) apresentam a forma que as tarefas do experimento foram alocadas e executadas em máquinas (*hosts*) cada um com 12 núcleos de processamento. Nas figuras a seguir, os retângulos mais escuros representam as etapas de modelagem, os mais claros são as projeções e o intermediário são os testes.

O UMS, como já era esperado, teve um desempenho pior em todos os casos, com exceção dos teste que ele executou com 24 núcleos que ele obteve um tempo de execução menor que o HTCondor. Ao analisar o escalonamento do HTCondor, percebeu-se que foi um mal escalonamento dele que deixou uma tarefa grande para ser executada por último e isso acabou afetando o tempo total de execução do *workflow*. Destaca-se que o EasyGrid AMS demonstrou ser mais eficiente que as demais abordagens.

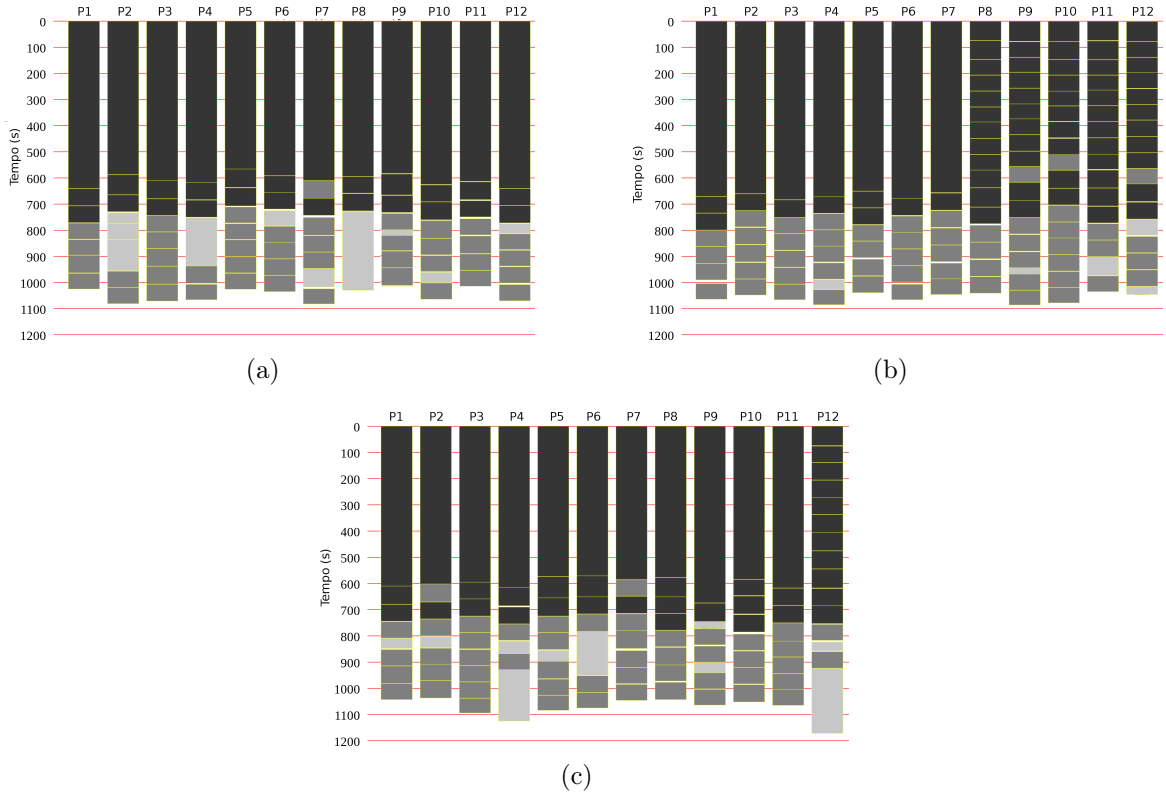


Figura 5.9: Experimento 5E com EasyGrid AMS em 3 *hosts*.

As Figuras 5.9 e 5.10, representam, respectivamente, o escalonamento em 3 *hosts* e 4 *hosts*. Destaca-se que com 3 *hosts* é impossível alcançar o menor tempo possível do *workflow*, devido a quantidade de *jobs* existente no *workflow* e os poucos recursos disponíveis. Percebe-se, com a Tabela 3.4, que respeitando as dependências impostas pelo *workflow*, a sequencia de tarefas que exige o maior tempo de execução é criação de um modelo, seguido da sua projeção. O gráfico da Figura 5.9 mostra que o escalonamento distribuiu bem os *jobs* entre os *hosts* disponíveis, mas a distribuição não foi suficiente para evitar que houvesse processamento de *jobs* além do tempo da maior sequência exigida pelo *workflow*.

Por outro lado, a Figura 5.10 possui 4 *hosts*, o que permite distribuir melhor as tarefas entre todos os recursos. O maior tempo de execução foi no *host* (a). Percebe-se que com 4 *hosts* é possível obter um escalonamento que produza o menor tempo possível de execução desse *workflow*. No caso, houve um escalonamento quase ótimo, se a maior coluna da Figura 5.10 (a) tivesse sido escalonada apenas com dois *jobs* (enviando a pequena projeção que ficou no meio para

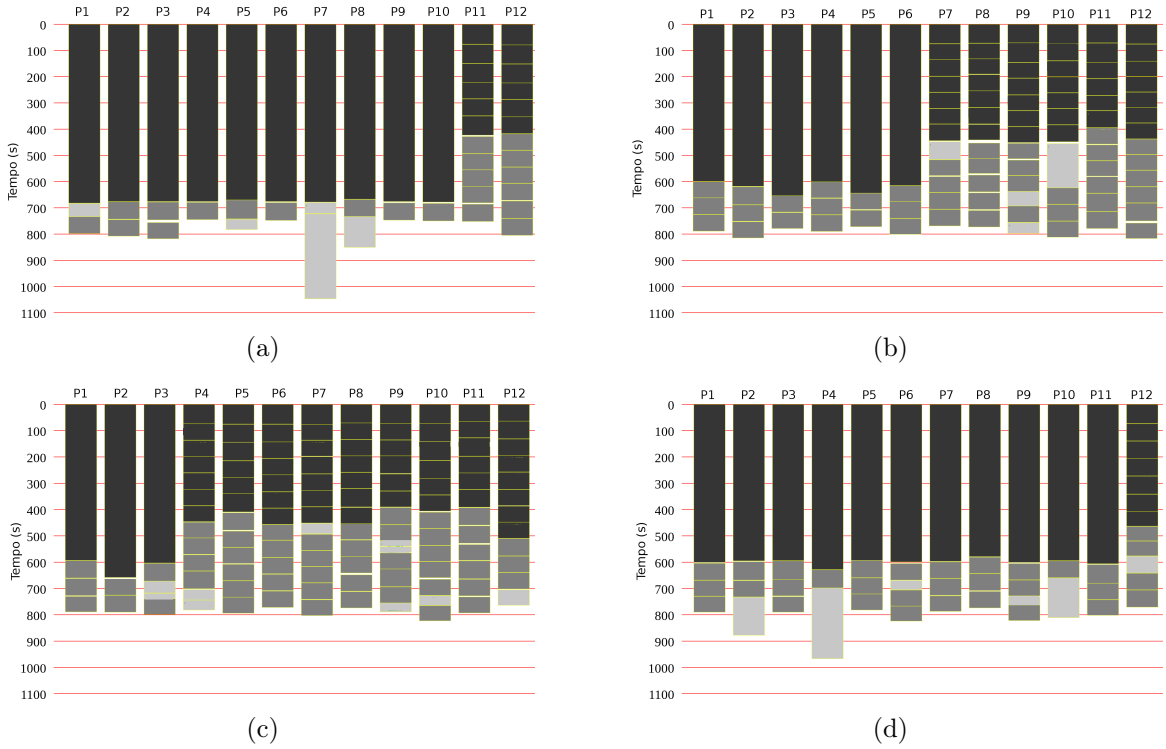


Figura 5.10: Experimento 5E com EasyGrid AMS em 4 *hosts*.

outro núcleo de processamento ou *host*), teria sido possível alcançar o menor tempo de execução desse *workflow*. Destaca-se que o EasyGrid AMS usa heurísticas simples, por serem empregadas durante a execução, e que não garantem que escalonamento ótimo será sempre alcançado.

Ao observar a Tabela 5.1, os tempos diminuem ao utilizar mais de 4 *hosts*, devido a maior capacidade de espalhamento das tarefas, o que aumenta a probabilidade de obter um escalonamento ótimo, além de reduzir a concorrência de recursos nos mesmos *hosts*, o que diminui o tempo individual das tarefas.

As Figuras 5.11 e 5.12, compara a execução dos dois melhores gerenciadores analisados, HTCondor e EasyGrid AMS, respectivamente. Pode-se ver que o escalonamento de ambos gerenciadores é eficiente, não deixando grandes lacunas desnecessárias entre os processos. Mas, comparando as tarefas individualmente, todas as tarefas do HTCondor tiveram um tempo maior, devido ao tempo gasto para gerenciamento entre o HTCondor e o DAGMan. Lembrando que DAGMan é um multi-escalonador e fica em uma camada acima do próprio escalonador do HTCondor. Ambos gerenciadores receberam as tarefas com os mesmos pesos, o que criou uma prioridade de execução da tarefa om_model em relação as outras tarefas.

Observa-se, também, a forma de escalonamento de ambos. O HTCondor, cria uma lista de *hosts* (ordenada pelo melhor desempenho) e uma lista de tarefas (ordenada pelo peso associado). Em seguida, o escalonador associa a tarefa com maior peso, no *host* de melhor desempenho (respeitando o limite de *slots* disponíveis - *slot* é o termo utilizado para definir a quantidade

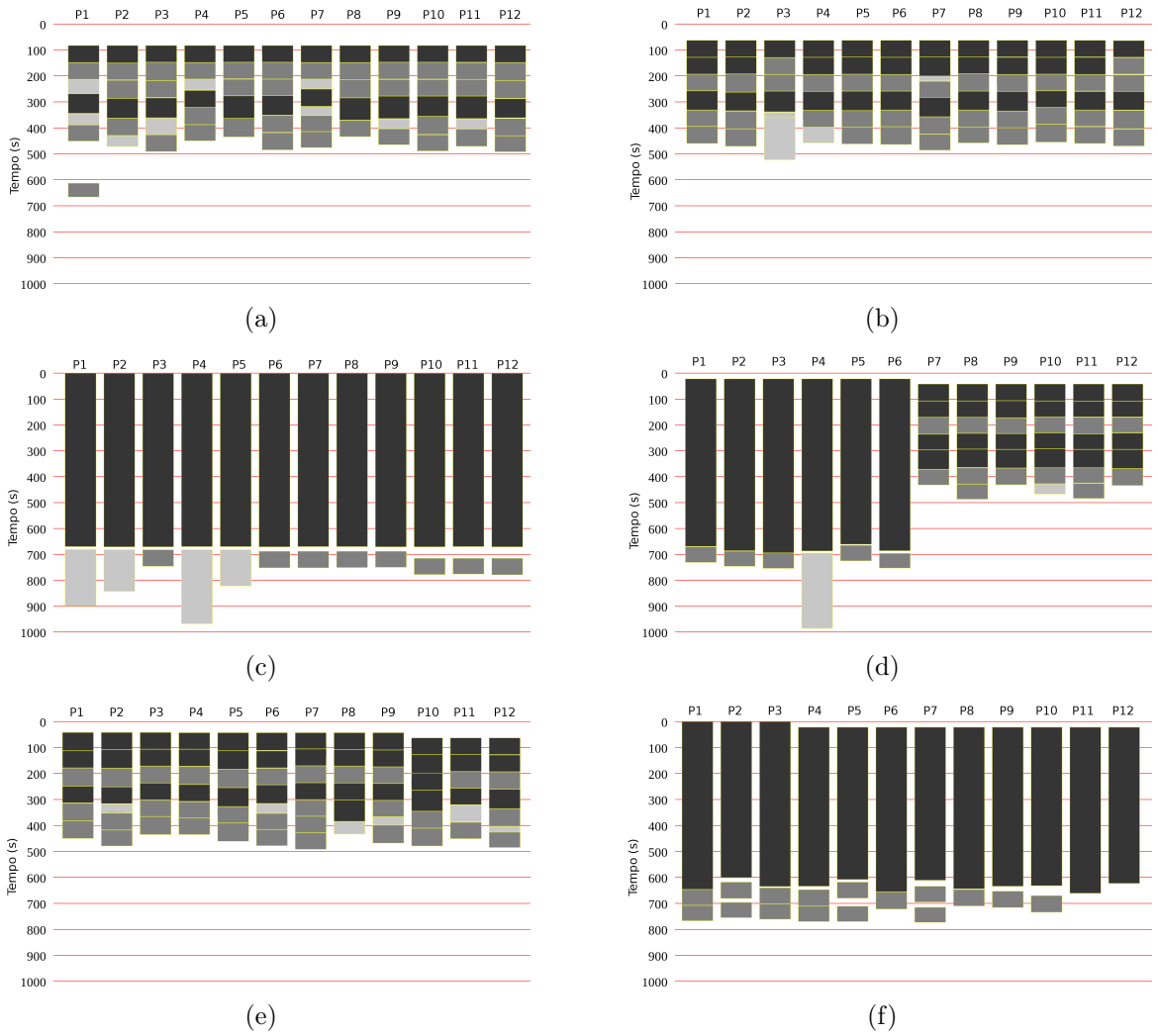


Figura 5.11: Experimento 5E com HTCondor-DAGMan em 6 *hosts*.

de tarefas concorrentes em cada *host*). Em contra-partida, o EasyGrid AMS, necessita de um escalonamento estático para iniciar (esse escalonamento é realizado pelo *conector* que cria uma lista ordenada pelo peso e distribui as tarefas mais pesadas entre todos os *hosts* que serão utilizados), dinamicamente, o EasyGrid AMS pode efetuar um balanceamento de carga entre as menores tarefas que estão para serem alocadas no *hosts* (caso, detecte desbalanceamento de carga).

Essas duas abordagens, podem ser vistas nas Figuras 5.11 e 5.12, que representam o HTCondor e EasyGrid AMS, respectivamente. Dois *hosts* do HTCondor (item (c) e (f)) estão completamente cheios com as tarefas mais pesadas, isso faz com que exista uma concorrência maior entre os recursos dessas máquinas (por exemplo: CPU, disco, memória) e consequentemente, irão demorar mais para concluir suas tarefas. Enquanto as outras máquinas irão terminar primeiro e ficarão ociosas esperando o término dessas tarefas (neste caso, considerando que o cluster inteiro foi reservado para a execução desse único *workflow*).

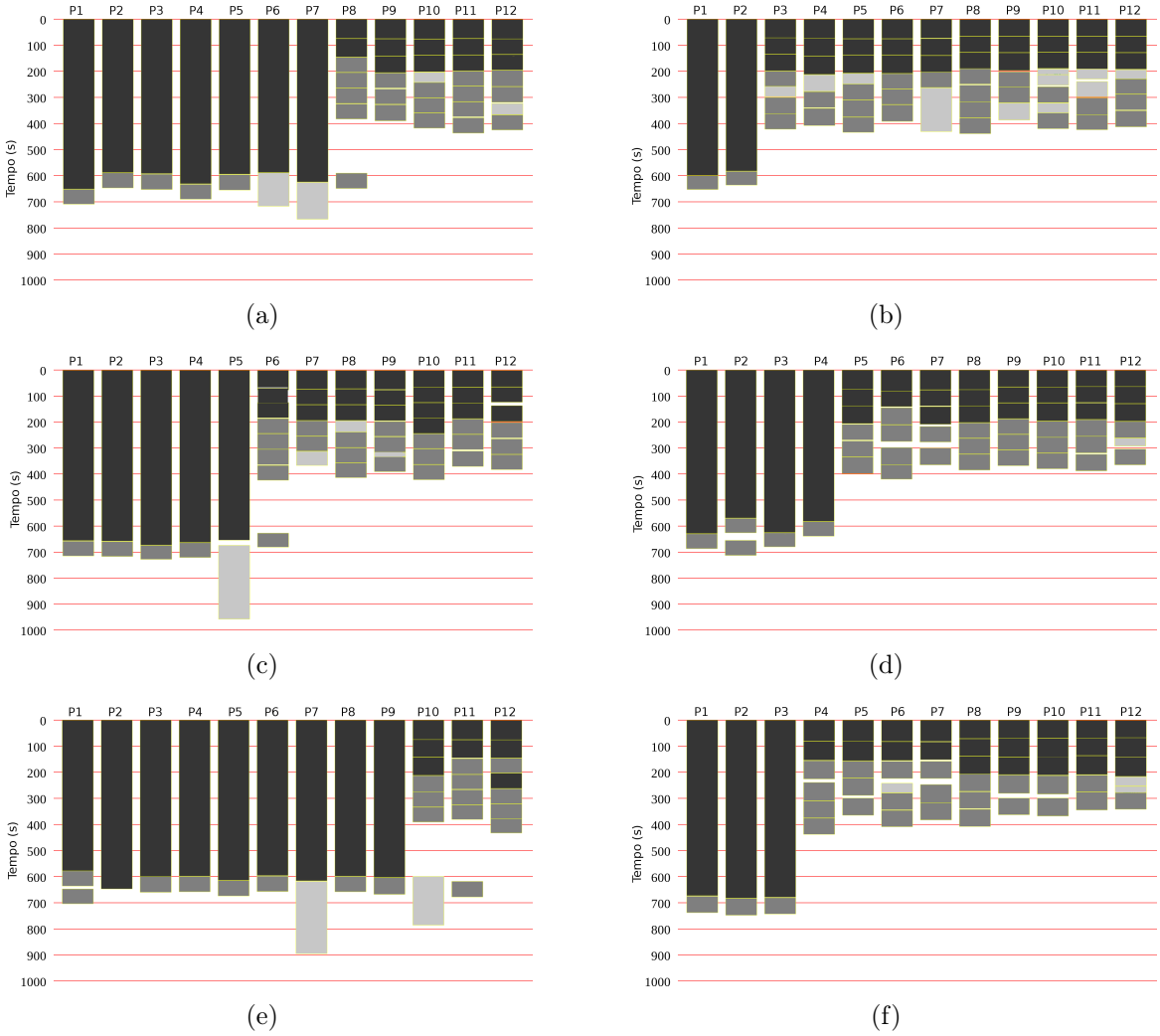


Figura 5.12: Experimento 5E com EasyGrid AMS em 6 *hosts*.

Já o EasyGrid AMS, distribuiu um pouco melhor as tarefas com peso grande, desta forma, a concorrência por recursos é reduzida, o que impacta diretamente no tempo total do *workflow*.

5.6 Conclusões

O propósito desse capítulo foi aplicar o estudo de caso (criação de um modelo de nicho ecológico, fazendo uso do openModeller) na abordagem apresentada no capítulo anterior (Capítulo 4). Para isso, analisou-se a viabilidade de aplicar os três gerenciamentos conceituais descritos no Capítulo 2 (UMS, RMS e AMS).

Os resultados demonstraram que o modelo permite o gerenciamento e execução de *workflows* do openModeller em diferentes ambientes computacionais. Por fim, realizou-se uma comparação entre as três abordagens, utilizando as duas versões existentes do openModeller (OMWS1 e OMWS2), demonstrando, na Tabela 5.1, que o uso do EasyGrid AMS é capaz de gerenciar melhor

a execução das tarefas que compôem o *workflow* associado à modelagem de nichos ecológicos, obtendo um tempo menor de execução em relação as outras formas de gerenciamento analisadas, devido a proximidade do AMS com os recursos, o que permite realizar um escalonamento dinâmico mais adequado ao estado atual do recurso.

Capítulo 6

Avaliação do Desempenho do EasyGrid AMS com *Workflows ENM*

O desenvolvimento desse trabalho exigiu uma série de experimentos que foram organizados em dois grandes grupos, são eles: na Seção 6.1 descreve-se os principais experimentos que auxiliaram na definição e configuração da infraestrutura computacional; e, na Seção 6.2, apresenta-se diferentes experimentos que abordam uma forma mais flexível de explorar a capacidade computacional dos ambientes compartilhados, concorrentemente, por *workflows* científicos distintos, mantendo um bom desempenho das aplicações.

6.1 Ambiente computacional

Esta seção apresenta a infraestrutura computacional disponível para a realização deste trabalho e traz os principais experimentos relacionados com a configuração do ambiente utilizado.

6.1.1 Infraestrutura

Atualmente, na Universidade Federal Fluminense (UFF), existem cerca de 826 núcleos de processamento que estão disponíveis para criar uma e-infraestrutura computacional que suporte as necessidades do projeto EU-Brazil OpenBio executando *workflows* científicos. Parte desses computadores são de uso dedicado ao projeto (cerca de 410 núcleos de processamento) que são gerenciados com o HTCondor, e a outra parte é o cluster Oscar, de uso compartilhado com outros grupos de pesquisa (composto por 320 núcleos de processamento). Porém, por motivos de problemas na infraestrutura física da UFF, nem todos os computadores estão disponíveis. Com a infraestrutura disponível na UFF (Tabela 6.1), tem-se três ambientes computacionais distintos (um ambiente dedicado e um ambiente compartilhado) que serão explorados para abordar o problema descrito na Seção 4.1, que trata do escalonamento de *workflows* científicos entre diferentes

ambientes computacionais e, ainda, considera-se que outras e-infraestruturas computacionais externas serão adicionadas futuramente aumentando o poder computacional dessa e-infraestrutura.

Os computadores envolvidos podem ser divididos em três tipos de arquiteturas diferentes, conforme a Tabela 6.1.

Tabela 6.1: Ambientes computacionais utilizados.

	Computadores dedicados		**Computadores compartilhados**
	HT1	HT2	HT3
Quantidade total:	205 servidores IBM	8 servidores Accept	40 servidores BULL
Quantidade disponível ou funcionando:	10	8	16
Dois processadores de tipo:	Intel Xeon CPU 3.06GHz (mono-core)	Intel Xeon hexacore X5650 2.67GHz (total de 12 cores)	Intel Xeon quad-core 5335 2.4GHz (total de 8 cores)
Memória:	2.5 GB DDR1 (266 MHz)	24 GB DDR3 (1333 MHz)	16GB DDR2 (667 MHz)
Discos:	SCSI 10K RPM	SATA II 7200 RPM	SAS 7200 RPM

Existe uma quantidade muito maior de máquinas do tipo HT1, porém, por possuírem uma capacidade menor de processamento e ter-se uma limitação de espaço físico e de energia elétrica, priorizou-se a disponibilidade das demais máquinas. Os computadores do tipo HT2 possuem o melhor hardware (maior capacidade de processamento e memória), por esse motivo, foram utilizados para virtualização e reservou-se uma dessas máquinas para armazenar todos os servidores (OMWS e SGBD). O tipo HT3, representa as máquinas disponíveis no *cluster* Oscar que são de uso compartilhado por diversos grupos de pesquisa da UFF.

6.1.2 Análise da memória RAM

O *workflow* do OpenModeller realiza muito acesso a disco, devido ao volume de dados que ele precisa para criar um modelo (a criação de um modelo que, em alguns casos, lê mais de 5 GB de dados), conforme análise apresentada nas Tabelas 3.5, 3.6 e 3.7. Desta forma, essas aplicações podem alocar uma grande quantidade de memória RAM para manipular esses dados durante a execução de uma tarefa. Caso o computador possua uma quantidade menor de memória RAM, acarretará na necessidade do sistema operacional fazer uso de *swap* em disco rígido, no qual possui um tempo de acesso muito superior ao da memória RAM, afetando, significativamente, o tempo necessário para executar uma tarefa.

Na Seção 6.1.2.1, foram realizados testes para verificar a quantidade de memória utilizada na execução das aplicações do estudo de caso escolhido, e, posteriormente, na Seção 6.1.2.2, analisou-

se o impacto da variação da memória RAM da máquina virtual (VRAM - *Virtual Random Access Memory*) no desempenho das aplicações.

6.1.2.1 Consumo de memória RAM

Para avaliar o total de memória necessário para evitar o uso de *swap*, foi utilizado uma das máquinas que possui a maior quantidade de mémoria RAM, no caso, uma máquina do grupo HT2, contendo 24 GB RAM. Foi analisada a execução de todas as aplicações do estudo de caso, com a variação dos principais algoritmos utilizados pela comunidade científica. A Figura 6.1, mostra a execução da aplicação que necessita da maior quantidade de memória ao longo do tempo, chegando a alocar 1.8GB de memória RAM, logo antes de finalizar a sua execução (próximo a 500 segundos de execução).

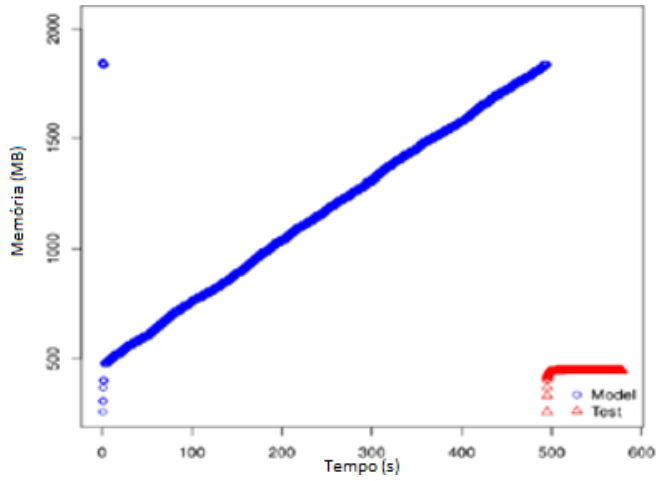


Figura 6.1: Avaliação da quantidade de memória RAM necessária.

Conclui-se que para evitar o uso de *swap* pelo sistema operacional, necessita-se de no mínimo 1.8GB de memória RAM para as aplicações avaliadas.

6.1.2.2 Variação da VRAM - sem execução concorrente de MV

Para validar o resultado do experimento da seção anterior, foi realizado um teste utilizando máquinas virtuais em *hosts* do tipo HT2 e variando a quantidade de memória disponível. Os resultados obtidos foram sumarizados na Figura 6.2, onde o gráfico mostra o tempo de execução em máquinas virtuais que foram instanciadas com a VRAM variando de 1 GB até 10GB. Observa-se que uma máquina virtual com 1GB de VRAM necessitou 24494,74 segundos para executar por completo a aplicação, enquanto, o mesmo experimento necessitou de aproximadamente a metade do tempo (12538,33 segundos) em uma máquina com uma quantidade de memória pouco acima de 1.8 GB (quantidade mínima recomendada na seção anterior). Ainda nesse gráfico,

realizou-se mais dois testes, um deles dobrando a quantidade de VCPUS da máquina virtual e outro utilizando uma máquina virtual com 10GB de VRAM (mais de 550%acima do mínimo recomendado), mas ambos os testes não reduziram o tempo total de execução.

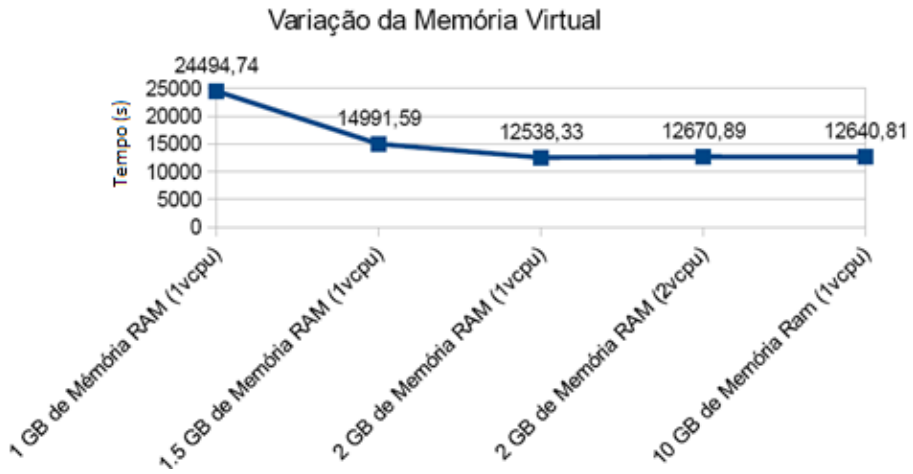


Figura 6.2: Variação da quantidade de VRAM.

Esse teste serviu para avaliar o impacto da quantidade de memória disponível para a execução das aplicações. É evidente a degradação do desempenho em computadores que não atendem a quantidade mínima de memória utilizada pelas tarefas, acarretando no uso de *swap* pelo sistema operacional.

Ainda fazendo o uso de máquinas virtuais, para dimensionar a quantidade de memória RAM total e avaliar o impacto da execução concorrente de máquinas virtuais no mesmo *host*, o próximo gráfico, Figura 6.3, mostra o tempo de execução da aplicação que possui o maior consumo de memória RAM, sendo executado primeiramente em uma única máquina virtual, e posteriormente, variando a execução de forma concorrente em 2 até 12 máquinas virtuais. Observa-se, que os dois gráficos possuem o mesmo comportamento, porém os tempos de execução apresentados em (A), são praticamente o dobro dos tempos obtidos em (B). Destaca-se que o gráfico de (B), atende o mínimo de VRAM descrito anteriormente, e o gráfico (A), que precisou do dobro do tempo de execução, possui a metade da quantidade mínima necessária de RAM.

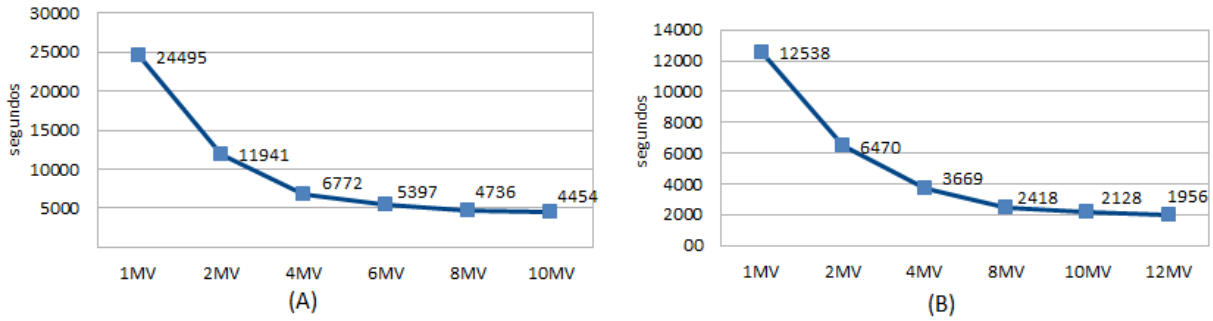


Figura 6.3: Análise do experimento com diferentes quantidade de VRAM. (A) Utilizando 1GB de VRAM; (B) Utilizando 2GB de VRAM.

Conclui-se com esse teste que a execução concorrente reduz o desempenho, visto que o tempo de execução de uma tarefa não é exatamente a metade do tempo de duas tarefas concorrentes, devido ao fato da sobrecarga ao acesso a disco. Mas, o objetivo específico desse experimento era avaliar se o comportamento da execução quando tem-se o uso de *swap* era semelhante a execução sem o uso de *swap*. Portanto, pode-se observar que os gráficos possuem o mesmo comportamento ao observar a parábola formada nos itens (A) e (B) da Figura 6.3.

6.1.3 Quantidade de processos concorrentes

Após definir a quantidade de memória RAM mais adequada para as aplicações do openModeller (>1.8GB), realizou-se um teste para avaliar a quantidade de processos concorrentes ideal em cada *host* do tipo HT2. A Figura 6.4 mostra os resultados obtidos. O teste foi realizado com o uso do gerenciador HTCondor, como gerenciador das tarefas, que utiliza o termo *slot* para definir a quantidade máxima de processos concorrentes que ele permitirá executar. Os testes foram todos executados em um único *host*, de forma exclusiva, com o *workflow* do experimento 1E.

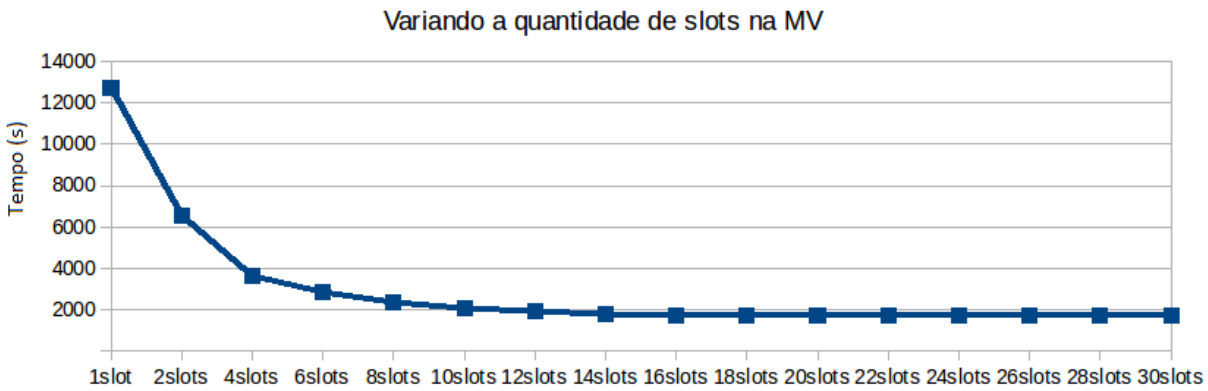


Figura 6.4: Variação da quantidade de processos concorrentes.

A Figura 6.3 mostra que o tempo de execução é reduzido consideravelmente até 12 slots (mesma quantidade de núcleos de processamento). Porém, ainda existe uma melhora do tempo até 16 slots.

6.1.4 Local de armazenamento dos dados

O estudo de caso escolhido, possui um conjunto de dados base, de aproximadamente 35GB, que representam as camadas ambientais de cada região que será analisada. O volume dessas informações que serão lidas por cada aplicação varia conforme o algoritmo e espécie selecionados. Esses dados não são fixos, pois representam as condições climáticas de cada ambiente. Logo, eventualmente, devem ser atualizados para uma melhor representação climática de cada região (fator importante para criar modelos de nichos ecológicos mais coerentes com a atualidade).

Realizou-se um teste, no qual o *workflow* do experimento E5 (descrito anteriormente) do OpenModeller foi completamente executado em único *host*, fazendo uso de diferentes formas de armazenamento dos arquivos que representam as camadas ambientais, utilizou-se o armazenamento local (LD - *Local Disk*) e NFS (*Network File System*), que é um sistema de arquivos distribuídos, que objetiva compartilhar arquivos e diretórios entre diferentes *hosts* conectados em rede, de forma transparente aos usuários.

Ao executar um *workflow* do OpenModeller com poucas tarefas paralelas em um único *host*, os tempos obtidos foram ligeiramente parecidos. Porém, no momento que aumenta-se a quantidade de *hosts* do ambiente, o que proporciona executar um número maior de tarefas, no qual todas necessitam acessar, simultaneamente, os arquivos que possuem as camadas ambientais, o NFS apresentou uma degradação significativa, demonstrando que, mesmo com um servidor dedicado de armazenamento, esses dados não podem permanecer centralizados em um único local. Logo, a distribuição local dos arquivos causa uma redundância de informação, mas, definitivamente, o armazenamento local das camadas ambientais melhora o desempenho das aplicações do openModeller.

6.1.5 Flexibilidade de infraestruturas

Muitas vezes, ambientes computacionais voltados a computação de alto desempenho (*High Performance Computing* - HPC) são compartilhados entre diferentes usuários e podem executar diferentes tipos de aplicações. Dado esse contexto, em vez de executar as aplicações diretamente nas máquinas, são utilizadas tecnologias de virtualização por diversos motivos, tais como: simplificação na instalação e manutenção; isolamento; segurança; balanceamento de carga; migração de servidores de forma transparente; dimensionamento do hardware da máquina virtual conforme a necessidade, máquinas virtuais com diferentes sistemas operacionais. Em outras palavras,

virtualização permite que ambientes computacionais de HPC fiquem mais ágeis e gerenciáveis. Entretando, o uso de virtualização possui o *overhead* associados a uma camada extra existente na tecnologia de virtualização, responsável por gerenciar o sistema operacional nativo. Essa camada extra pode ser vista no item b, da Figura 6.5, localizada entre o *Hardware* e a Máquina Virtual.

Os dois principais métodos de virtualização são: virtualização completa e paravirtualização. Na virtualização completa o *hypervisor* é executado diretamente no *hardware* (Figura 6.5 - A) e na paravirtualização o *hypervisor* é executado em um sistema operacional que permitirá o acesso ao hardware (Figura 6.5 - B).

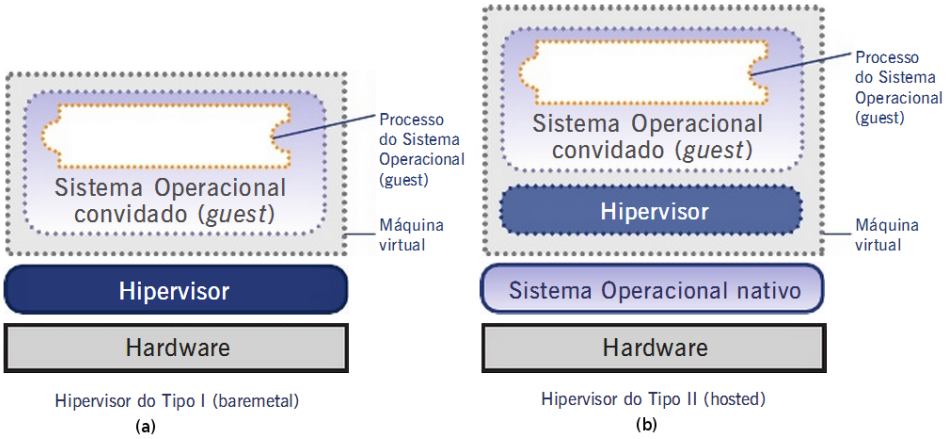


Figura 6.5: Tipos de virtualização.

Além das duas técnicas tradicionais de virtualização, apresentados na Figura 6.5, existe também a virtualização no nível de sistema operacional, conhecida como container. A tecnologia LXC (*Linux Containers*) é um ambiente de virtualização no nível de sistema operacional que utiliza o mesmo *kernel* do servidor *host* para permitir execuções isoladas de múltiplos sistemas Linux sobre o controle de uma única máquina (*host*) [31, 40, 75], Figura 6.6.

O uso de métodos de virtualização em nível de sistema operacional tem sido utilizado, principalmente, por fornecer uma camada adicional de abstração, permitindo isolamento e outras caracteríscas relevantes das máquinas virtuais, sem o *overhead* de uma camada específica para gerenciar o sistema operacional desse ambiente virtualizado.

O LXC é baseado nos grupos de controle do Linux (cgroups), onde cada grupo pode oferecer recursos de forma isolada para suas aplicações (processador, memória e acesso I/O).

Um dos principais sistemas de CaaS (*Containers as a Service*) é o Docker [24], que permite agilidade, portabilidade e controle na criação e gerenciamento de containers.

Para definir o melhor ambiente de execução de *workflows* científicos em infraestruras computacionais, foi realizado um comparativo entre o uso de máquians virtuais e containers, conforme Figura 6.7. Nesse experimento, utilizou-se um experimento típico do OpenModeller, apresentado

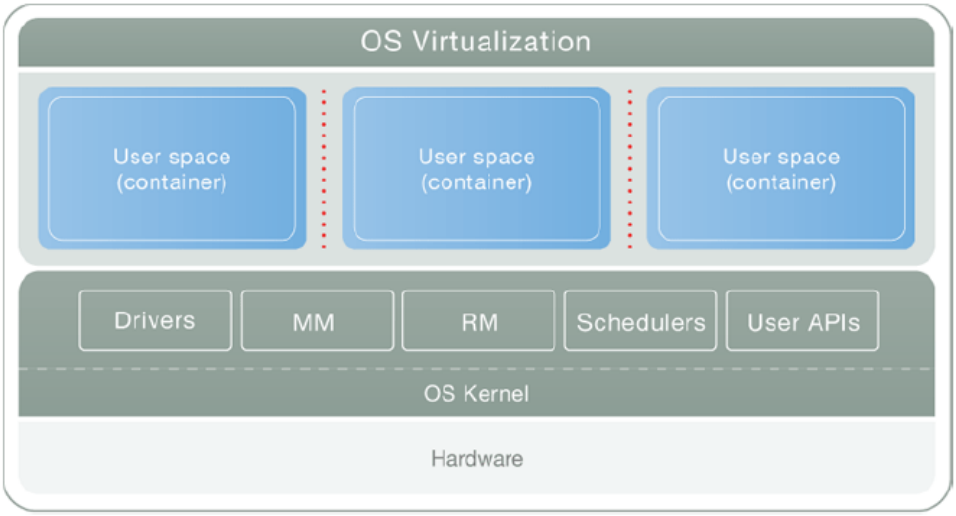


Figura 6.6: Virtualização baseada em container. Fonte:[4]

na Seção 3.3, onde o workflow é responsável por criar um modelo de nicho ecológico é formado por três etapas, são elas: modelagem, teste e projeção.

É importante destacar que não houve execução concorrente entre as tarefas do experimento e nem restrição de hardware na máquina virtual e no container.

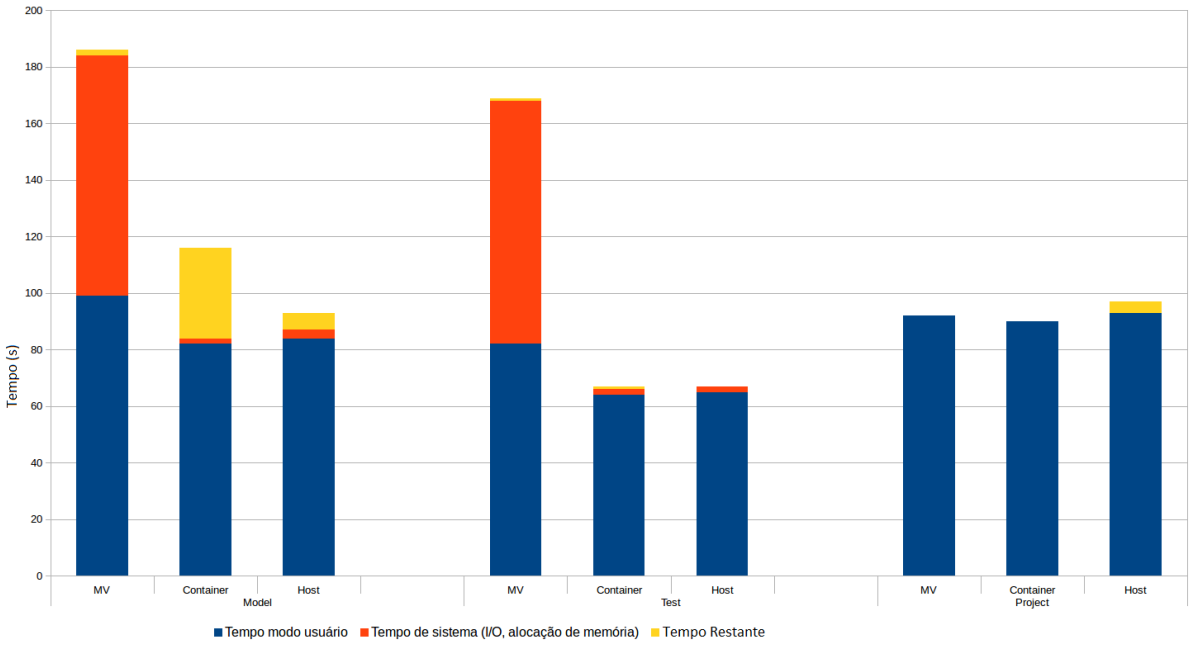


Figura 6.7: Comparação entre MV x Docker x Host

A Figura 6.7 apresenta nove barras verticais, onde a altura de cada barra indica o tempo total em segundos da execução de cada tipo de tarefa do *workflow*. As três primeiras barras da esquerda, correspondem a criação do modelo, as três do meio estão relacionadas com a etapa de

teste, as outras representam o tempo de execução da etapa de projeção. O tempo de cada tarefa analisada pode ser dividida em três partes, são elas:

- Tempo modo usuário: Representa o tempo da tarefa executando na CPU.
- Tempo de sistema: É o tempo de CPU gasto com processos do *kernel*. O modo kernel contém operações como, por exemplo, leitura e escrita em: disco, rede e memória.
- Tempo restante: Está associado ao tempo que o processo estava na fila de execução parado, devido a fatia de tempo (*time slice*) do processador. Esse tempo é formado por Tempo total - (tempo modo usuário + tempo de sistema).

Com a Figura 6.7, ficou explicito que o gerenciamento de escrita e leitura da camada de software, que existe entre o sistema operacional e a própria máquina virtual, torna o uso de máquina virtual inadequado para executar essas tarefas que possuem muito acesso a disco, pois o tempo total de execução praticamente dobrou em relação ao tempo de execução do container e *host*.

6.2 Desempenho do AMS com o OpenModeller

Os experimentos do Capítulo 5 demonstraram que o uso do AMS para resolver um *workflow* que representa um experimento ENM, é viável, além de prover um método que mostrou-se mais eficiente no gerenciamento das dependências e na alocação das tarefas, devido ao fato de estar mais próximo dos recursos.

A Tabela 6.2 possui apenas os tempos referente a execução do AMS, retirados da Tabela 5.1, e possui seus valores projetados no gráfico da Figura 6.8. Nesta figura, cada máquina é representada por um número múltiplo de 12 (quantidade de núcleos físicos existentes), observa-se que a alocação superior a 5 máquinas é totalmente desnecessária, e o ganho da utilização de 5 máquinas, em relação a 4 máquinas é inferior a 9%. Com isso, destaca-se uma das dificuldades encontradas no problema de escalonamento de tarefas, a quantidade de recursos necessária para uma execução eficiente. A alocação de poucos hosts, pode aproveitar ao máximo os recursos envolvidos, mas o tempo de execução pode ser afetado consideravelmente. Por outro lado, a utilização de muitos recursos pode resultar no melhor tempo possível para execução de uma aplicação, mas uma grande subutilização dos recursos devido a ociosidade.

Tabela 6.2: Média de tempo de execução, em segundos, do AMS.

Núcleos	12	24	36	48	60	72	84	96
AMS	3295	1715	1182	1056	967	967	936	930

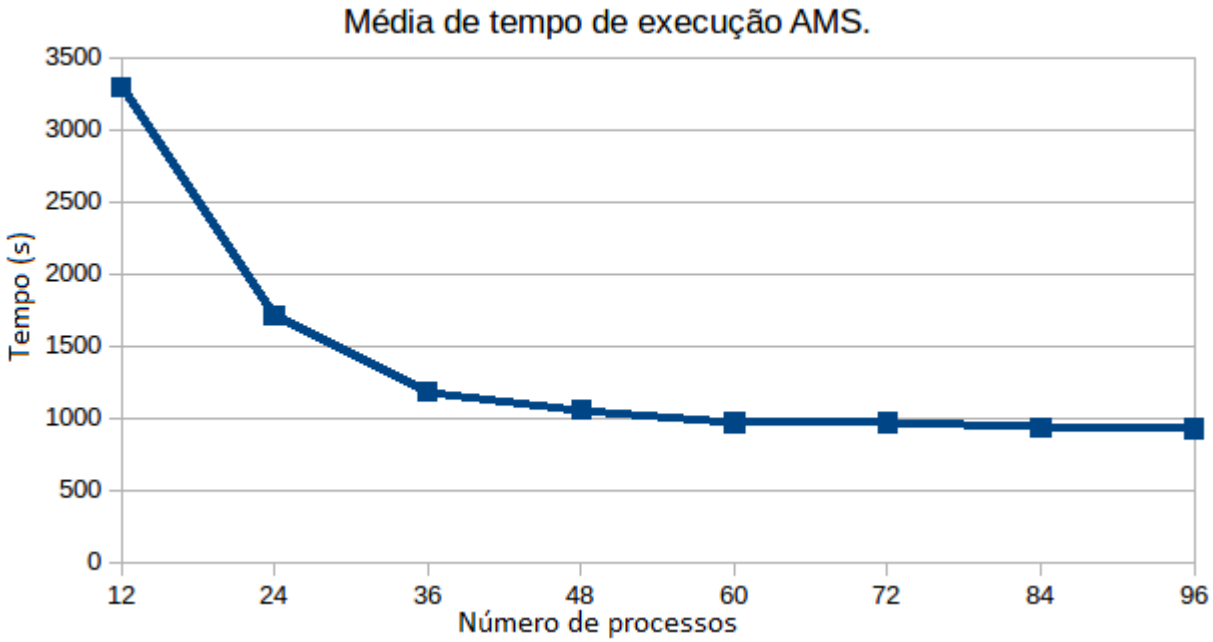


Figura 6.8: Tempo de Execução do AMS variando a quantidade de hosts

Em busca do equilíbrio entre o tempo de execução de um experimento e a utilização do recurso, analisou-se o compartilhamento de recursos por diferentes experimentos.

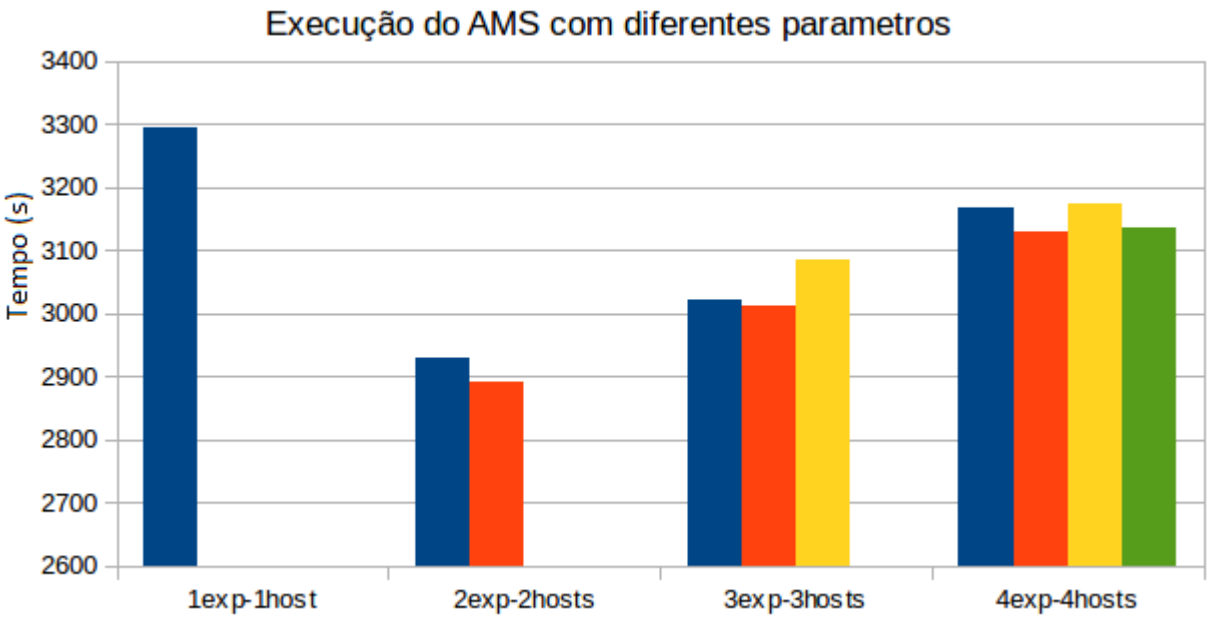


Figura 6.9: Tempo de Execução do AMS variando a quantidade de hosts

A Figura 6.9, mostra quatro diferentes testes (destaca-se que o eixo vertical começa em 2600 segundos para salientar somente a faixa de interesse), que serão descritos na Tabela 6.3, a seguir:

Tabela 6.3: Descrição dos testes com compartilhamento de recursos

Identificação do teste na Figura 6.9	Análise
1exp-1host	Esse teste foi executado com um único experimento, em um único *host*. Com um tempo total de 3295 segundos.
2exp-2hosts	Nesse teste executou-se dois experimentos, em dois *hosts* (os experimentos utilizaram os dois *hosts*, disputando recursos). Obtendo o tempo de 2931 segundos para um dos experimentos e 2890 segundos para o outro. Ao comparar esses tempos, com o tempo obtido ao executar um único experimento em um único host, observa-se que o tempo de ambas tarefas foi reduzido. O que está em conformidade com a Seção 6.1.3, que mostra aumentar o tempo individual das tarefas, mas reduz o de tempo total do *workflow* ao utilizar uma quantidade de tarefas maior que a quantidade de recursos. Enquanto um experimento executa 12 tarefas em cada recurso e essas tarefas não utilizam CPU em tempo integral, ao permitir o compartilhamento de um recurso para dois experimentos, tem-se um total de 24 tarefas em cada recurso, explorando o tempo de CPU que ficava ocioso ao executar apenas 12 tarefas por recurso.
3exp-3hosts	Esse teste explora a sobrecarga dos recursos. Executou-se, de forma compartilhada, 3 experimentos em 3 recursos, onde cada experimento executa duas tarefas em todos os recursos. Desta forma, tem-se 36 tarefas concorrentes em cada recurso. O tempo obtido (3021, 3013 e 3086 segundos) é menor que ao executar um único experimento por recurso, porém, é pior que a execução de 2 experimentos que compartilham 2 recursos, devido ao fato dos recursos já estarem sobrecarregados, afetando assim o tempo de execução de todas as tarefas compartilhadas.
4exp-4hosts	Os tempos dos experimentos são 3168, 3129, 3175, 3134 segundos. O tempo de execução obtido é maior do que o obtido no teste anterior, devido aos mesmos motivos. E mesmo assim, ainda é melhor que o tempo obtido com 1 experimento e 1 host

Outros testes foram realizados de forma que o recurso compartilhado não seja o mesmo para todos os experimentos. Por exemplo, na Tabela 6.4, considera-se três recursos (denominados h1, h2 e h3).

Tabela 6.4: Média de tempo de execução de cada uma das diferentes abordagens.

Teste	Recurso alocado	Tempo (segundos)
2 experimentos em 3 *hosts*	h1+h2 (experimento 1)	2193
	h2+h3 (experimento 2)	2218
3 experimentos em 3 *hosts*	h1+h2 (experimento 1)	3013
	h2+h3 (experimento 2)	3019
	h1+h3 (experimento 3)	2982

A Tabela 6.4 mostra o tempo de execução de experimentos com diferentes distribuições de *hosts*, obtendo-se diferentes tempos de execução. Compara-se esses resultados com os da Tabela 6.3, da sequinte forma:

1. Na Tabela 6.4, mostra-se que ao executar 3 experimentos em 3 recursos, onde cada experimento compartilha apenas parte dos recursos disponíveis, obteve-se um tempo máximo de 3019 segundos. Em contra-partida, a Tabela 6.3, mostra que ao executar 3 experimentos que utilizam completamente os recursos disponíveis, obteve-se o tempo máximo de execução de 3086 segundos.

2. Com a seguinte sequência: 1 experimento em 1 host (3295 segundos) em paralelo com 2 experimentos em 2 hosts (2931 segundos), são necessários 3295 segundos para executar completamente os 3 experimentos.

3. Ou, 2 experimentos em 3 hosts (2218 segundos) até terminar, depois, na sequência, mais 1 experimento nos 3 hosts (1182 segundos), necessita-se de 3400 segundos.

Analisando essas diferentes formas de executar 3 experimentos em 3 hosts, a melhor execução (menor tempo) foi a execução dos 3 experimentos alocados em recursos de forma diversificada. Pois, os recursos não foram sobrecarregados com uma quantidade muito grande de tarefas concorrendo pelos mesmos recursos. Além disso, ao diversificar os experimentos entre os recursos, aumenta-se a chance de executar tarefas que utilizem recursos diferentes concorrentemente (memória, I/O, CPU, disco, ...).

6.2.1 Flexibilidade na execução do OpenModeller

A característica do AMS, desenvolvido nesse trabalho, de estar junto com o *workflow* científico, permite uma gerencia mais refinada sobre as tarefas que a compõem. Dado que o AMS está junto com a tarefa no recurso computacional, isso permite que o recurso seja monitorado, mais precisamente, para realizar, dinamicamente, uma decisão sobre a quantidade de tarefas que serão executas nos *hosts*, inclusive, permitindo que cada *host* possua uma quantidade diferente.

Essa decisão pode ser tomada a partir do estado do *host*, caso esteja em uso por alguma outra aplicação (de outro usuário, por exemplo), o AMS pode decidir reduzir a carga de trabalho exigida do *host*, diminuindo a alocação de tarefas nesse *host*, podendo, inclusive, parar de utilizá-lo para ceder recursos para a outra aplicação que encontra-se no *host*.

Um parâmetro muito utilizado em Computação em Nuvem é o tempo total previsto para uso, dado que Nuvens computacionais, normalmente, são alugadas (reservadas) por um determinado período, portanto, é usual tentar prever o tempo máximo de execução para não alugar uma nuvem computacional e deixa-la ociosa. Em contra partida, alugar um tempo inferior ao necessário, pode ocasionar na não conclusão da aplicação e, talvez, perda dos resultados parciais do tempo alugado. Esse parâmetro pode ser considerado como uma meta, sem a necessidade de finalizar a aplicação caso a meta não seja alcançada.

Fazendo uso de uma meta, denotado aqui como tempo de execução esperado, é possível compartilhar recursos entre diferentes experimentos (*workflows*), fornecendo prioridades diferentes de execução para cada um. Por exemplo, a Figura 6.9, mostrou a execução de 2 experimentos que compartilharam 2 *hosts*, ambos concorrendo aos mesmos recursos, com mesma prioridade, com um tempo total de execução próximo de 2900 segundos. Caso um dos experimentos tivesse uma prioridade maior para execução sobre os *hosts*, o outro experimento poderia ceder parcialmente ou totalmente o recurso para que o experimento com maior prioridade pudesse terminar sua execução antes, logo em seguida, o experimento com menor recurso poderia voltar a executar suas tarefas sem concorrer aos recurso com outro experimento.

Tabela 6.5: Execução do AMS com meta.

Teste	Meta (segundos)	Tempo (segundos)
1 experimento em 1 *host*	3000	2903
2 experimentos em 3 *hosts*	1200	2103
	1200	2130
2 experimentos em 3 *hosts*	2000	2043
	2000	2160
2 experimentos em 3 *hosts*	3000	2507
	3000	2582
2 experimentos em 3 *hosts*	1200	1292
	10000	2415

A Tabela 6.5 traz alguns experimentos que abordam o uso da meta. No primeiro teste, a aplicação AMS percebe que está sozinha e executa o máximo de tarefas concorrentes permitidas, no caso, 18 tarefas concorrentes (1.5 vezes a quantidade de núcleos físicos do *host*). Com isso, conseguiu-se executar completamente o *workflow* em 2903 segundos, reduzindo-se 392 segundos do tempo obtido, sem o uso de meta e com 12 tarefas concorrentes, apresentado na Tabela 6.2.

Nos 3 próximos testes, avaliou-se o compartilhamento de 3 recursos entre 2 experimentos. O primeiro desses testes possue uma meta impossível de ser atendida, o segundo uma meta apertada e o terceiro uma meta folgada (1200, 2000 e 3000 segundos, respectivamente).

No teste que possui meta de 1200 segundos, ambos experimentos, avaliam que estão atrasados e executaram suas tarefas exigindo o máximo dos recursos, ou seja, cada aplicação executou 18 tarefas concorrentes em cada *host*, totalizando 36 tarefas concorrentes por *host*. Desta forma, ambos experimentos, possuem seus tempos degradados, devido a sobrecarga nos *hosts*, finalizando em 2103 e 2130 segundos, respectivamente.

No teste com metas fixadas em 2000 segundos, inicialmente, os experimentos avaliam que a meta é factível (estando adiantados) e estão acompanhados (existe mais alguma outra aplicação compartilhando o *host*), reduzindo carga de trabalho dos seus *hosts*. Porém, ao longo da execução, o AMS recalcula a viabilidade de cumprir a meta e descobre que devido a redução de carga, a meta tornou-se apertada (ou até mesmo, impossível de ser atendida), portanto, o AMS começa a executar novamente as suas tarefas no máximo (18 tarefas concorrentes). Essa variação da quantidade de tarefas concorrentes em cada experimento, provocou uma variação no tempo de execução deles (2043 segundos e 2160 segundos).

O teste que possui os experimentos, ambos com a meta de 3000 segundos, inicialmente descobrem que estão compartilhando recursos e reduzem a carga de processamento dos seus *hosts*, ambos reduzem na mesma proporção, o que ocasiona em um tempo total próximo (2507 e 2582 segundos).

O último teste dessa tabela, avalia metas bem distintas, uma delas é impossível de ser atendida e a outra é exageradamente grande. Nesse teste é possível verificar que a tarefa que possui a meta de 1200, percebe que está acompanhada, mas como está atrasada, executa o máximo de tarefas possíveis em seus *hosts* (18 tarefas), enquanto o outro experimento está com uma meta folgada e ao perceber que está acompanhado, reduz a execução de suas tarefas Com isso, o experimento que possuí uma meta impossível de ser cumprida, ganha prioridade na execução e executa todo o seu *workflow* em 1292 segundos, enquanto o outro experimento que possuía uma meta folgada, executa o seu *workflow* em 2415 segundos, conforme ilustrado na Figura 6.10.

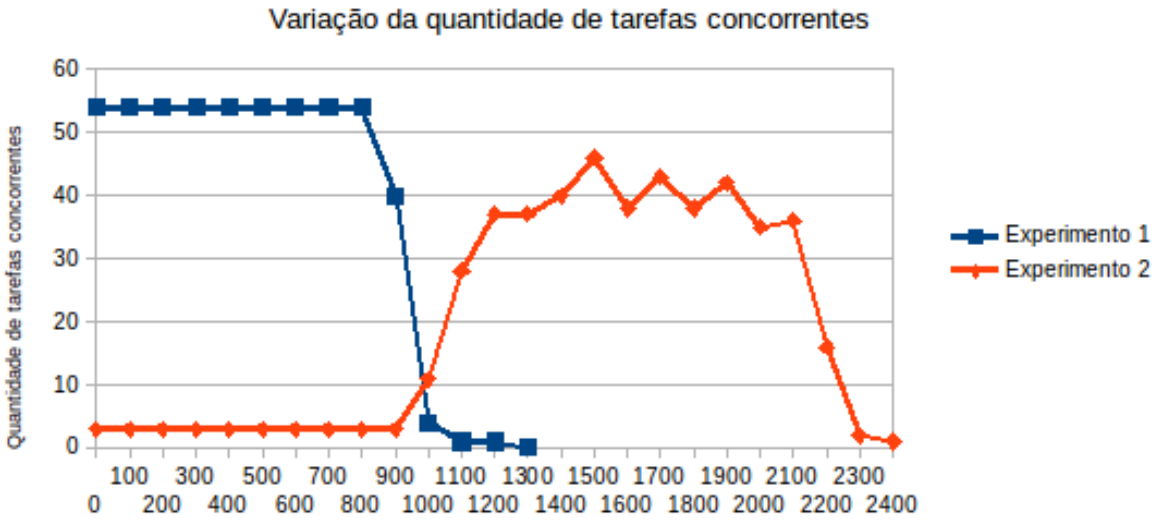


Figura 6.10: Variação da quantidade de tarefas concorrentes.

6.2.2 Estratégia Paralela para a Projeção de Modelo de Nicho Ecológico

Na Seção 2.1.4 foi apresentado a relação entre aplicação, aplicação paralela e *workflow*, onde, nas Figuras 2.8 e 2.9 mostra-se um *workflow*, formado por etapas paralelas, onde a etapa 6 é formada por etapas independentes e foi paralelizada para minimizar o tempo de execução do *workflow* científico.

Nesse contexto, buscando-se minimizar o tempo total de execução, criou-se uma nova versão da aplicação responsável por projetar os modelos criados no openModeller. Nessa versão separou-se as etapas internas da projeção, que eram independentes, em diversas pequenas aplicações com o uso do MPI. E criou-se uma aplicação AMS para gerenciar todas essas novas tarefas menores da projeção do openModeller.

Como o *workflow* do experimento é gerenciado por uma aplicação AMS, as tarefas que eram responsáveis pela projeção, inicialmente, foram substituídas por uma tarefa que é uma aplicação AMS, capaz de gerenciar a nova aplicação paralela que será responsável pela projeção dos modelos de nichos ecológicos criados. Criando uma estrutura onde uma AMS (aplicação que irá gerenciar o *workflow*) contém vários outros AMSs (aplicações que irão gerenciar a projeção do modelo ENM).

A Tabela 6.6, apresenta uma comparação entre os tempos encontrados ao executar o *workflow* contendo a projeção sequencial e o *workflow* contendo a nova projeção paralela.

Tabela 6.6: Comparação, em segundos, entre a projeção sequencial e paralela.

Quantidade de núcleos	12	24	36	48	60	72
AMS com projeção original	3295	1715	1182	1056	967	967
AMS com projeção paralela	3358	1664	1112	843	731	724

Observa-se que o tempo de execução da projeção sequencial é menor que o tempo obtido com a projeção paralela quando utiliza-se um único *host* com 12 núcleos de processamento. Esse resultado para 12 núcleos está associado com a forma de escalonamento utilizado nas duas abordagens, onde cria-se uma lista das tarefas do *workflow* que estão prontas para serem executas (não possuem dependências a serem cumpridas) e ordena-se essa lista com prioridade de execução para as tarefas que possuem um maior peso, essa abordagem tem a tendência (visto que é DAG) de executar as tarefas com menor peso por último, portanto, ao aplicar o balanceamento de carga nessas tarefas, o escalonamento tende a ser bom, em contra-partida, a criação de tarefas paralelas possuem um *overhead* envolvido, ocasionando em um maior tempo de execução, para poucos recursos.

Em todos os demais tempos de execução coletados, a versão paralela foi melhor, visto que, devido a paralelização, tarefas ainda menores são mais fáceis de serem distribuídas em quantidades maiores de recursos, suprindo o *overhead* da paralelização e ainda sim, obtendo um tempo menor que a versão sequencial. Nota-se que, como vimos anteriormente, com 4 *hosts* é possível ter o melhor escalonamento (mas isso nem sempre acontece devido as tarefas que surgem dinamicamente ao terem suas dependências resolvidas). A probabilidade de conseguir o melhor escalonamento é maior com 5 *hosts* (60 núcleos de processamentos), onde obteve-se quase o melhor tempo possível desse *workflow*. Considera-se o menor tempo possível, o tempo de uma tarefa de modelagem + uma tarefa que realiza o teste do modelo gerado, desconsiderando o tempo de execução da tarefa de projeção, pois a nova versão paralela permita que a criação de várias tarefas tão pequenas, que podem ser espalhadas pelos demais recursos a medida que forem ficando discponíveis.

6.3 Conclusões

O desempenho da execução de qualquer aplicação, depende além do escalonador, de uma infraestrutura adequada. Desta forma, a Seção 6.1.1 buscou realizar uma série de experimentos práticos para identificar a melhor arquitetura para o estudo de caso selecionado. Resume-se as conclusões dessa seção com a Tabela 6.7.

A Seção 6.2 mostrou diferentes formas de execução de experimentos em *hosts* compartilhados. Concluiu-se que simplesmente compartilhar o *host* entre diferentes aplicações, pode ocasio-

Tabela 6.7: Infraestrutura Apropriada para uso do OpenModeller.

Memória RAM	Os experimentos da Seção 6.1.2.1 mostraram a necessidade de no mínimo 1.8 GB de memória RAM por processo, para evitar o uso de *swap* e degradação do desempenho.
Quantidade de processos concorrentes	Tendo como objetivo o tempo total do experimento, recomenda-se que a quantidade de processos concorrentes seja no máximo 1.5x a quantidade de núcleos físicos de processamento do *host*. Porém, se o objetivo for redução do tempo individual dos processos, evitando a subutilização do poder computacional do *host*, indica-se no máximo a quantidade de núcleos do *host*.
Local de armazenamento dos layers	Devido a característica de *workflows* terem um acesso intensivo de I/O, os *layers* precisam ser armazenados localmente em cada host.
Arquitetura	Os experimentos demonstraram que devido ao intenso acesso de I/O o uso de máquinas virtuais possui uma significativa degradação no acesso concorrente a disco. Desta forma, recomenda-se utilizar diretamente o *host* ou o uso de *conteiners*, que demonstraram ter um desempenho equivalente ao acesso direto ao *host*, oferecendo as vantagens de manutenção, isolamento, segurança, entre outras vantagens equivalentes ao uso de máquinas virtuais.

nar uma sobrecarga nos recursos, ocasionando uma degradação do desempenho. Posteriormente, mostrou-se que devido a característica do AMS estar junto com a aplicação e adicionando-se uma meta (estimativa de tempo de execução) além da inserção de meta para as execuções, o AMS mostrou-se capaz de gerenciar as tarefas dos *workflows* de maneira mais eficiente, dando prioridade aos experimentos com metas apertadas e dando prefência para outros *workflows* quando suas metas encontram-se no estado adiantado.

Além disso, mostrou-se que com a implementação de uma versão paralela da aplicação que realiza a projeção do OpenModeller. Pode-se utilizar uma aplicação AMS para gerenciar o *workflow* e outras aplicações AMS para gerenciar apenas a versão paralela da projeção do OpenModeller. Com essa abordagem, foi possível obter quase o tempo ótimo do *workflow* avaliado.

Capítulo 7

Conclusão

Esse trabalho propôs uma abordagem de um gerenciador de *workflows* científicos que fosse capaz de utilizar ambientes computacionais pré-existentes (tipicamente heterogêneos e compartilhados), sem alterar seus respectivos sistemas de gerenciamento e respeitando suas politicas existentes. Em um ambiente computacional, principalmente em ambientes compartilhados, o ponto essencial para uma execução eficiente é utilizar um sistema de gerência que seja capaz de perceber as variações do ambiente para adaptar a execução do *workflow* de acordo com a carga de trabalho de cada recurso.

Esse trabalho teve como principal foco apresentar uma proposta conceitualmente diferente dos convencionais modelos de gerência, RMS e UMS, que são amplamente utilizados, mas que não aproveitam ao máximo os recursos envolvidos. Tipicamente, esses gerenciadores são centralizados e simplesmente criam uma lista de tarefas prontas para execução e uma lista de recursos disponíveis no ambiente computacional, posteriormente essas tarefas são distribuídas entre os recursos, conforme suas disponibilidades. Depois deste momento, não tem mais nenhuma forma de gerencia até terminar o *job*. Estes gerenciadores foram projetados supondo que os recursos serão alocados de forma exclusiva (porém, pode ocorrer uma subutilização dos recursos), ou, em caso de recursos compartilhados por tempo, tarefas de diferentes usuários podem ter seus respectivos desempenhos afetados, significativamente, caso todas utilizem de forma intensiva o mesmo recurso (CPU, memória, disco, entre outros). Esses gerenciadores não conseguem perceber ou aproveitar a subutilização de recursos enquanto tem *jobs* em execução. O equilíbrio entre subutilizar recursos (melhorar o tempo individual dos processos, mas não explorar completamente o recurso) e superutilizar recursos (degradar o tempo individual das tarefas, mas explorar ao máximo os recursos e a vazão de *jobs*) é um desafio para os gerenciadores.

O EasyGrid AMS é um sistema gerenciador de aplicações, responsável por tornar as aplicações auto-gerenciáveis, viabilizando uma execução eficiente e segura nos recursos disponíveis no ambiente computacional. Porém, o EasyGrid AMS originalmente foi desenvolvido apenas

para aplicações paralelas, com troca de mensagens, não sendo projetado para trabalhar com as peculiaridades de *workflows* científicos (uso intensivo de I/O, troca de informações por arquivos). Portanto, o EasyGrid AMS necessitou ser adaptado para prover o conceito de um *workflow* autogerenciável, com uma estrutura de escalonamento dinâmico, eficiente e flexível, capaz de lidar com aplicações científicas externas e fechadas (sem a necessidade de compilar as aplicações com as bibliotecas do EasyGrid AMS) e, ainda, adaptar-se às dificuldades impostas pelos ambientes compartilhados. Além de permitir políticas de escalonamento distintas entre os *workflows*, o uso do EasyGrid AMS proporciona maior escalabilidade ao sistema gerenciador global, pois não mantem todo o gerenciamento centralizado (cada *workflow* é responsável pelo seu próprio gerenciamento). O uso do EasyGrid AMS também permite que ambientes computacionais que não oferecem suporte para *workflows* científicos sejam considerados na execução dessa classe de aplicações.

Uma importante característica do EasyGrid AMS é manter o gerenciador próximo das suas tarefas e dos recursos computacionais, isso permite que o EasyGrid AMS tenha total conhecimento do ambiente e possa redistribuir as tarefas para balancear a carga de cada *host*. Explorando esse contexto, adicionou-se o conceito de meta (tempo de execução desejado) para a execução de um *workflow*, fornecendo flexibilidade ao executar mais de um *workflow* no mesmo recurso, pois, com a definição da meta, uma aplicação que está adiantada, pode deixar de concorrer a recursos (reduzindo a carga de processamento, sem a necessidade de parar completamente), permitindo que uma outra aplicação que possua uma meta apertada, possa utilizar os recursos com prioridade.

Os resultados obtidos revelam que o EasyGrid AMS mostrou-se competitivo as duas abordagens de gerenciamento de *workflows*, apresentando resultados melhores que o reconhecido gerenciador RMS analisado (HTCondor possui quase 30 anos de existência) e um resultado bem superior ao UMS.

Por fim, a solução desenvolvida nesta tese, foi implantada em uma e-infraestrutura, formada por ambientes computacionais distintos de recursos convencionais, de forma transparente, para oferecer um serviço de nuvem (*ENM-as-a-service*) para a comunidade da biodiversidade. Acredita-se que a solução desenvolvida nesta tese, irá contribuir para que as aplicações científicas (existentes) aproveitem de forma mais eficiente, ambientes computacionais compartilhados e heterogêneos. Os resultados até o momento de publicação mostraram um *speedup* de 40,88 para um experimento de ENM.

Como trabalhos futuros, propõem-se refinar a autonomia para que as aplicações perceberem que estão compartilhando recursos. A Seção 6.2.1 mostrou que houve ganho, mas os resultados ainda podem ser melhorados. Além disso, o *Resource Broker*, que foi o foco deste trabalho, pode ser melhorado tanto no aspecto do particionamento do workflow quanto o controle de *workflows*

já submetidos, reenviando e/ou retirando requisições de ambientes.

Enquanto, neste trabalho, o EasyGrid AMS usou uma grafo acíclico direcionado (GAD) para descrever as dependências no *workflows*, devido suas complexidades, alguns *workflows* podem ser representados por grafos cíclicos. Como foi mostrado nos trabalhos [76, 77], o EasyGrid AMS pode ser projetado para criar o grafo ao poucos dinamicamente e instanciar separadamente tarefas repetidas, convertendo um grafo cíclico em uma GAD. Isso não foi necessário para *workflows* ENM, mas se outros *workflows* científicos foram alvo de estudos futuros, seria viável adaptar o EasyGrid AMS para estes casos.

Referências

[1] Aliaga, A. H. M. Estudo comparativo de técnicas de escalonamento de tarefas dependentes para grades computacionais. Master's thesis, Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, SP, Brasil, Julho 2011.

[2] Amaral, R. B.; Badia, R. M.; Blanquer, I.; Braga-Neto, R.; Candela, L.; Castelli, D.; Flann, C.; Giovanni, R. D.; Gray, W. A.; Jones, A.; Lezzi, D.; Pagano, P.; Perez-Canhos, V.; Quevedo, F.; Rafanell, R.; Rebello, V.; Sousa-Baena, M.; Torres, E. Supporting biodiversity studies with the EUBrazilOpenBio hybrid data infrastructure. In *Concurrency and Computation* (2015), pp. 376–394.

[3] Amaral, R. B.; Badia, R. M.; Candela, L.; Castelli, D.; Giovanni, R. D.; Gray, W. A.; Jones, A.; Lezzi, D.; Pagano, P.; Perez-Canhos, V.; Quevedo, F.; Rafanell, R.; Rebello, V.; Torres, E. EU-Brazil open data and cloud computing e-infrastructure for biodiversity. In *IWSG 2013 International Workshop on Science Gateways* (2013), pp. 50–58.

[4] Balenzategui, R. Virtualizacion. Disponível em `http://blogs.itpro.es/problemas/tag/virtualizacion-2/`.

[5] Barker, A.; van Hemert, J. I.; Baldock, R. A.; Atkinson, M. P. An e-infrastructure to support collaborative embryo research. In *2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid* (May 2009), pp. 520–525.

[6] Belhajjame, K.; Wolstencroft, K.; Corcho, O.; Oinn, T.; Tanoh, F.; William, A.; Goble, C. Metadata management in the taverna workflow system. *CCGRID* (2008).

[7] Bernus, P.; Blazewicz, J.; Schmidt, G. J.; Shaw, M. J. *International Handbooks on Information Systems*. Springer, 2015.

[8] Blythe, J.; Jain, S.; Deelman, E.; Gil, Y.; Vahi, K.; Mandal, A.; Kennedy, K. Task scheduling strategies for workflow-based applications in grids. *CCGRID* (2005).

[9] Boeres, C.; Rebello, V. E. F. EasyGrid: Towards a framework for the automatic grid enabling of legacy MPI applications. *Concurrency and Computation: Practice and Experience 16*, 5 (2004), 425–432.

[10] Braghetto, K. R.; Cordeiro, D. *Introdução à Modelagem e Execução de Workflows Científicos*. XXXIII Jornadas de Atualização em Informática, 2014.

[11] Bux, M.; Leser, U. Parallelization in scientific workflow management systems. *CoRR* (2013).

[12] Cao, J.; Jarvis, S.; Saini, S.; Nudd, G. R. GridFlow: workflow management for grid computing. In *Cluster Computing and the Grid, 2003. Proceedings. CCGrid 2003. 3rd IEEE/ACM International Symposium on* (2003), pp. 198–205.

[13] Chen, W.; da Silva, R. F.; Deelman, E.; Sakellariou, R. Balanced task clustering in scientific workflows. *IEEE 9th International Conference on e-Science* (2013).

[14] Couvares, P.; Kosar, T.; Roy, A.; Weber, J.; Wenger, K. Workflow management in condor. In *Workflows for e-Science*, I. Taylor, E. Deelman, D. Gannon, and M. Shields, Eds. Springer London, 2007, pp. 357–375.

[15] da Costa Sena, A. *Um Modelo Alternativo para a Execução Eficiente de Aplicações Paralelas MPI nas Grades Computacionais.* Tese de Doutorado, Instituto de Computação, Universidade Federal Fluminense, Niterói, RJ, Brasil, Novembro 2008.

[16] de Giovanni, R.; Torres, E.; Amaral, R. B.; Blanquer, I.; Rebello, V.; Perez-Canhos, V. OMWS: A web service interface for ecological niche modelling. In *Biodiversity Informatics* (2015), pp. 35–44.

[17] de Oliveira, F. J. B. Escalonamento dinâmico, distribuído e colaborativo de aplicações paralelas e autônomas com prazos. Master's thesis, Instituto de Computação, Universidade Federal Fluminense, Niterói, RJ, Brasil, 2016.

[18] de Oliveira Passos, F. G. *Provendo autonomia às aplicações da e-ciência.* Tese de Doutorado, Instituto de Computação, Universidade Federal Fluminense, Niterói, RJ, Brasil, Julho 2014.

[19] de Paula Nascimento, A. *Escalonamento Dinâmico para Aplicações Autonômicas MPI em Grades Computacionais.* Tese de Doutorado, Instituto de Computação, Universidade Federal Fluminense, Niterói, RJ, Brasil, Maio 2008.

[20] Deelman, E.; Singh, G.; Su, M.-H.; Blythe, J.; Gil, Y.; Kesselman, C.; Mehta, G.; Vahi, K.; Berriman, G. B.; Good, J.; Laity, A.; Jacob, J. C.; Katz, D. S. Pegasus: a framework for mapping complex scientific workflows onto distributed systems. *Scientific Programming Journal 13*, 3 (2005), 219–237.

[21] Deelman, E.; Vahi, K.; Juve, G.; Rynge, M.; Callaghan, S.; Maechling, P. J.; Mayani, R.; Chen, W.; Ferreira da Silva, R.; Livny, M.; Wenger, K. Pegasus: a workflow management system for science automation. *Future Generation Computer Systems 46* (2015), 17–35. Funding Acknowledgements: NSF ACI SDCI 0722019, NSF ACI SI2-SSI 1148515 and NSF OCI-1053575.

[22] Dellman, E.; Gannon, D.; Shields, M.; Taylor, I. Workflows and e-science: An overview of workflow system features and capabilities. *Future Generation Computer Systems 25*, 5 (2009), 528–540.

[23] Dennis Gannon, Ewa Deelman, M. S.; Taylor, I. *Workflows for e-Science: Scientific Workflows for Grids*, 1 ed. Springer-Verlag London, 2007.

[24] Docker. Whitepaper - enabling modern microservices architectures for enterprise applications, 2016. Disponível em `https://www.docker.com/sites/default/files/WP_Modern%20App%20Architecture%20for%20Enterprise%20-%20Jan%202016.pdf`.

[25] Dongarra, J.; Lastovetsky, A. *High Performance Heterogeneous Computing.* Wiley Series on Parallel and Distributed Computing. Wiley-Interscience, 2009.

[26] dos Reis, V. Q. Escalonamento em grids computacionais: estudo de caso. Master's thesis, Instituto de Ciências Matemáticas e de Computação - ICMC-USP, São Carlos, SP, Brasil, 2005.

[27] Elith, J.; H. Graham, C.; P. Anderson, R.; Dudík, M.; Ferrier, S.; Guisan, A.; J. Hijmans, R.; Huettmann, F.; R. Leathwick, J.; Lehmann, A.; Li, J.; G. Lohmann, L.; A. Loiselle, B.; Manion, G.; Moritz, C.; Nakamura, M.; Nakazawa, Y.; McC. M. Overton, J.; Townsend Peterson, A.; J. Phillips, S.; Richardson, K.; Scachetti-Pereira, R.; E. Schapire, R.; Soberón, J.; Williams, S.; S. Wisz, M.; E. Zimmermann, N. Novel methods improve prediction of species' distributions from occurrence data. *Ecography 29*, 2 (2006), 129–151.

[28] Fahringer, T.; Jugravu, A.; Pllana, S.; Prodan, R.; Jr., C. S.; Truong, H. L. ASKALON: a tool set for cluster and grid computing. *Concurrency - Practice and Experience 17*, 2-4 (2005), 143–169.

[29] Foster, I.; Kesselman, C.; Tuecke, S. The anatomy of the grid: Enabling scalable virtual organizations. *Int. J. High Perform. Comput. Appl. 15*, 3 (Aug. 2001), 200–222.

[30] Global Biodiversity Information Facility. Disponível em `http://www.gbif.org/`.

[31] Goasguen, S. *Docker Cookbook - Solutions and Examples for Building Distributed Applications*. OReilly Media, 2015.

[32] Goble, C.; Bhagat, J.; Aleksejevs, S.; Cruickshank, D.; Michaelides, D.; Newman, D.; Borkum, M.; Bechhofer, S.; Roos, M.; Li, P.; Roure, D. myexperiment: a repository and social network for the sharing of bioinformatics workflows, 2010. Disponível em `http://www.myexperiment.org/workflows`.

[33] Gordienko, Y.; Bekenov, L.; Gatsenko, O.; Tatarenko, V. Complex workflow management and integration of distributed computing resources by science gateway portal for molecular dynamics simulations in materials science. *High Performance Computing" HPC-UA 2013 (Ukraine, Kyiv)* (2013), 148–155.

[34] Gritsenko, A. A workflow-forecast approach to the task scheduling problem in distributed computing systems. *IJASCSE* (2013).

[35] Gutschick, V. P.; Sheng, Z. Control of atmospheric fluxes from a pecan orchard by physiology, meteorology, and canopy structure: Modeling and measurement. *Agricultural Water Management 129* (2013), 200 – 211.

[36] Hollingsworth, D. Workflow management coalition the workflow reference model, 1995. Disponível em `ftp://www.ufv.br/dpi/mestrado/Wkflow-BPM/The%20Workflow%20Reference%20Model.pdf`.

[37] HTCondor - High Throughput Computing. Disponível em `http://research.cs.wisc.edu/htcondor/`.

[38] Hull, D.; Wolstencroft, K.; Stevens, R.; Goble, C. A.; Pocock, M. R.; Li, P.; Oinn, T. Taverna: A tool for building and running workflows of services.

[39] Hussain, H.; Malik, S. U. R.; Hameed, A.; Khan, S. U.; Bickler, G.; Min-Allah, N.; Qureshi, M. B.; Zhang, L.; Yongji, W.; Ghani, N.; Kolodziej, J.; Zomaya, A. Y.; Xu, C.-Z.; Balaji, P.; Vishnu, A.; Pinel, F.; Pecero, J. E.; Kliazovich, D.; Bouvry, P.; Li, H.; Wang, L.; Chen, D.; Rayes, A. A survey on resource allocation in high performance distributed computing systems. *Parallel Comput. 39*, 11 (Nov. 2013), 709–736.

[40] Johnston, J.; Batchelli, A.; Cormace, J.; Fiedler, J.; Gajdos, M. *Docker in the Trenches Early Release*. Bleeding Edge Press, 2015.

[41] Jun Qin, T. F. a. *Scientific Workflows: Programming, Optimization, and Synthesis with ASKALON and AWDL*, 1 ed. Springer-Verlag Berlin Heidelberg, 2012.

[42] The Kepler Project. Disponível em `https://kepler-project.org/`.

[43] Khan, R. Z.; Ali, J. Use of dag in distributed parallel computing. In *International Journal of Application or Innovation in Engineering* (IJAIEM, November 2013), pp. 81–85.

[44] Kothe, D. B. Science prospects and benefits with exascale computing, 2007. Disponível em `https://www.olcf.ornl.gov/wp-content/uploads/2010/03/Science-Case-_012808-v3__final.pdf`.

[45] Krauter, K.; Buyya, R.; Maheswaran, M. A taxonomy and survey of grid resource management systems for distributed computing. *Software: Practice and Experience 32*, 2 (2002), 135–164.

[46] Liu, J.; Pacitti, E.; Valduriez, P.; Mattoso, M. A survey of data-intensive scientific workflow management. *Journal of Grid Computing 13*, 4 (2015), 457–493.

[47] Malathi, G.; Sarumathi, S. Survey on grid scheduling. *Journal of Computer Applications 3*, 3 (2010), 22–29.

[48] Marco, P. D.; Sirqueira, M. Como determinar a distribuição potencial de espécies sob uma abordagem conservacionista? In *Megadiversidade* (2009).

[49] Meyer, L. A. V. C. *Estratégias para o Escalonamento Dinâmico de Workflows em Grid.* Tese de Doutorado, COPPE/UFRJ, Engenharia de Sistemas e Computação, Rio de Janeiro, RJ, Brasil, Julho 2006.

[50] MyGrid Team. Disponível em `http://www.mygrid.org.uk/`.

[51] Nandagopal, M.; Gokulnath, K.; Uthariaraj, V. R. A two-level hierarchical resource selection model in grid environment. *Journal of Computational Information Systems* (2011).

[52] Nascimento, A.; Sena, A.; Boeres, C.; Rebello, V. E. F. Distributed and dynamic self-scheduling of parallel MPI grid applications. *Concurrency and Computation: Practice and Experience 19*, 14 (Sept. 2007), 1955–1974. Published Online: 14 Nov 2006.

[53] Nascimento, A.; Sena, A.; da Silva, J.; Vianna, D. Q. C.; Boeres, C.; Rebello, V. E. F. Managing the execution of large scale MPI applications on computational grids. In *Computer Architecture and High Performance Computing. SBAC-PAD'05. 17th International Symposium on* (Oct. 2005), pp. 69–76.

[54] Nascimento, A.; Sena, A.; da Silva, J.; Vianna, D. Q. C.; Boeres, C.; Rebello, V. E. F. On the advantages of an alternative MPI execution model for grids. In *CCGRID '07: Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid* (Rio de Janeiro, Brazil, 2007), IEEE Computer Society, pp. 575–582.

[55] Nascimento, A.; Sena, A.; da Silva, J.; Vianna, D. Q. C.; Boeres, C.; Rebello, V. E. F. Autonomic application management for large scale MPI programs. *International Journal of High Performance Computing and Networking 5*, 4 (2008), 227–240.

[56] Nativi, S.; Mazzetti, P.; Saarenmaa, H.; Kerr, J.; Tuama, É. Ó. Biodiversity and climate change use scenarios framework for the geoss interoperability pilot process. *Ecological Informatics 4* (2009), 23–33.

[57] Ocaña, K. A. C. S.; de Oliveira, D.; Horta, F.; Dias, J.; Ogasawara, E.; Mattoso, M. *Exploring Molecular Evolution Reconstruction Using a Parallel Cloud Based Scientific Workflow.* Springer Berlin Heidelberg, Berlin, Heidelberg, 2012, pp. 179–191.

[58] Oda, R.; Cordeiro, D.; da Silva, R. F.; Deelman, E.; Braghetto, K. The case for resource sharing in scientific workflow executions. In *XVI Simposio em Sistemas Computacionais de Alto Desempenho (WSCAD)* (2015). Funding Acknowledgements: NSF ACI SI2-SSI 1148515.

[59] Ogasawara, E.; Dias, J.; Silva, V.; Chirigati, F.; de Oliveira, D.; Porto, F.; Valduriez, P.; Mattoso, M. Chiron: a parallel engine for algebraic scientific workflows. *Concurrency and Computation: Practice and Experience 25*, 16 (2013), 2327–2341.

[60] openModeller. Disponível em `http://openmodeller.sourceforge.net/overview.html`.

[61] Ostermann, S. *Resource Management for Scientific Application in Hybrid Cloud Computing Environments.* Tese de Doutorado, Faculty of Mathematics, Computer Science and Physics of the University of Innsbruck, Innsbruck, Austria, April 2012.

[62] Pandey, S.; Buyya, R. A survey of scheduling and management techniques for data-intensive application workflows. In *Data intensive distributed computing : challenges and solutions for large-scale information management* (Hershey, Pa : Information Science Reference, 2012).

[63] Passos, F. G. O.; Rebello, V. E. F. An autonomic parallel strategy for the projection of ecological niche models in heterogeneous computational environments. In *Euro-Par 2016: Parallel Processing: 22nd International Conference on Parallel and Distributed Computing, Grenoble, France, August 24-26, 2016, Proceedings* (2016), P.-F. Dutot and D. Trystram, Eds., Springer International Publishing, pp. 363–375.

[64] Peterson, A.; Soberón, J.; Pearson, R.; Anderson, R.; Martinez-Meyer, E.; Nakamura, M.; Araújo, M. Ecological niches and geographic distributions. *Princeton University Press* (2011).

[65] Peterson, A. T. Biogeography of diseases: a framework for analysis. *Naturwissenschaften 95*, 6 (2008), 483–491.

[66] Peterson, A. T.; Ball, L. G.; Cohoon, K. P. Predicting distributions of mexican birds using ecological niche modelling methods. *Ibis 144*, 1 (2002).

[67] Peterson, A. T.; Vieglais, D. A. Predicting species invasions using ecological niche modeling: New approaches from bioinformatics attack a pressing problem: A new approach to ecological niche modeling, based on new tools drawn from biodiversity informatics, is applied to the challenge of predicting potential species' invasions. *BioScience 51*, 5 (2001), 363–371.

[68] Pinaya, J. L. D. *Processo de Pré-análise para a Modelagem de Distribuição de Espécies.* Tese de Doutorado, Escola Politécnica da Universidade de São Paulo, São Paulo, SP, Brasil, 2013.

[69] Pinedo, M. L. *Scheduling - Theory, Algorithms, and Systems.* Springer, 2012.

[70] Projeto openModeller - CRIA. Disponível em `http://openmodeller.cria.org.br/`.

[71] Qin, J.; Fahringer, T. *Scientific Workflows - Programming, Optimization, and Synthesis with ASKALON and AWDL.* Springer, 2012.

[72] Radu Prodan, T. F. *Grid Computing: Experiment Management, Tool Integration, and Scientific Workflows*, 1 ed. Lecture Notes in Computer Science 4340. Springer-Verlag Berlin Heidelberg, 2007.

[73] Russell, N.; Hofstede, A.; van der Aalst, W.; Mulyar, N. Workflow control-flow patterns: A revised view. In *BPM Center Report - BPMcenter.org* (2006).

[74] Russell, N.; van der Aalst, W.; ter Hofstede, A. *Workflow Patterns: The Definitive Guide.* MIT Press, 2016.

[75] Sabharwal, N.; W, B. *Docker Hands on: Deploy, Administer Docker Platform.*

[76] Sena, A.; Nascimento, A.; Boeres, C.; Rebello, V. Easygrid enabling of iterative tightly-coupled parallel mpi applications. In *2008 IEEE International Symposium on Parallel and Distributed Processing with Applications* (Dec 2008), pp. 199–206.

[77] Sena, A.; Ribeiro, F.; Rebello, V.; Nascimento, A.; Boeres, C. Autonomic malleability in iterative mpi applications. In *2013 25th International Symposium on Computer Architecture and High Performance Computing* (Oct 2013), pp. 192–199.

[78] Silva, J.; Rebello, V. E. F. Low Cost Self-healing in MPI Applications. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface, 14th European PVM/MPI User's Group Meeting* (2007), Springer, pp. 144–152.

[79] Soberón, J.; Peterson, A. Interpretation of models of fundamental ecological niches and species' distributional areas. *Biodiversity Informatics* (2011).

[80] SpeciesLink. Disponível em `http://www.splink.org.br/`.

[81] Tampelini, L. G. *Técnicas Heurísticas de Escalonamento Paralelo em Workflow.* Tese de Doutorado, Instituto de Computação, Universidade Federal Fluminense, Campinas, SP, Brasil, Março 2012.

[82] Taverna Workflow Management System. Disponível em `http://www.taverna.org.uk/`.

[83] Vajteršic, M.; Zinterhof, P.; Trobec, R.; Trobec, R.; Vajteršic, M.; Zinterhof, P. *Parallel Computing: Numerics, Applications, and Trends*, 1 ed. Springer-Verlag London, 2009.

[84] Wilde, M.; Hategan, M.; Wozniak, J. M.; Clifford, B.; Katz, D. S.; Foster, I. Swift: A language for distributed parallel scripting. *Parallel Computing 37*, 9 (Sept. 2011), 633–652.

[85] Woodman, S.; Hiden, H.; Watson, P.; Cała, J. Workflows and applications in e-science central. *All Hands Meeting* (2009).

[86] Yu, J.; Buyya, R. A taxonomy of scientific workflow systems for grid computing. *SIGMOD Rec. 34*, 3 (Sept. 2005), 44–49.

[87] Yu, J.; Buyya, R.; Ramamohanarao, K. Workflow scheduling algorithms for grid computing. *Springer Berlin Heidelberg 146* (2008), 173–214.

[88] Yuan, D.; Yang, Y.; Liu, X.; Chen, J. A data placement strategy in scientific cloud workflows. *Future Gener. Comput. Syst. 26*, 8 (Oct. 2010), 1200–1214.