UNIVERSIDADE DO ESTADO DO RIO GRANDE DO NORTE
UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

JOÃO PHELLIPE DE FREITAS PINTO

USO DE ALGORITMO GENÉTICO PARA ESCALONAMENTO DE TAREFAS EM GRADES COMPUTACIONAIS NO SIMULADOR GRIDSIM.

MOSSORÓ – RN

2012

JOÃO PHELLIPE DE FREITAS PINTO

USO DE ALGORITMO GENÉTICO PARA ESCALONAMENTO DE TAREFAS EM GRADES COMPUTACIONAIS NO SIMULADOR GRIDSIM.

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação – associação ampla entre a Universidade do Estado do Rio Grande do Norte e a Universidade Federal Rural do Semi-Árido, para a obtenção do título de Mestre em Ciência da Computação.

Orientadora: Profª. Drª. Carla Katarina Monteiro Marques – UERN.

Co-Orientador: Prof Dr. Francisco Chagas de Lima Júnior – UERN.

MOSSORÓ – RN

2012

Ficha catalográfica preparada pelo setor de classificação e catalogação da Biblioteca "Orlando Teixeira" da UFERSA

P659u Pinto, João Phellipe de Freitas.
Uso de algoritmo genético para escalonamento de tarefas em grades computacionais no simulador gridsim. / João Phellipe de Freitas Pinto. -- Mossoró, 2013.
54 f.: il.

Dissertação (Mestrado em Ciência da Computação) – Universidade Federal Rural do Semi-Árido.

Orientadora: Drª. Carla Katarina Monteiro Marques.
Co-orientador: Dr. Francisco Chagas de Lima Júnior.

1. Grades computacionais. 2. Algoritmos evolutivos. 3. Alocação de recursos. 4. Escalonamento. I. Título.

CDD:

Bibliotecária: Vanessa de Oliveira Pessoa
CRB15/453

JOÃO PHELLIPE DE FREITAS PINTO

USO DE ALGORITMO GENÉTICO PARA ESCALONAMENTO DE TAREFAS EM GRADES COMPUTACIONAIS NO SIMULADOR GRIDSIM.

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação – associação ampla entre a Universidade do Estado do Rio Grande do Norte e a Universidade Federal Rural do Semi-Árido, para a obtenção do título de Mestre em Ciência da Computação.

APROVADA EM: 31 / 01 / 2013.

BANCA EXAMINADORA

Profª. Drª. Carla Katarina de Monteiro Marques - UERN
Presidenta

Prof. Dr. Francisco Chagas de Lima Júnior – UERN
Co-Orientador

Prof. Dr. Antonio de Barros Serra – IFCE
Membro Externo

Prof. Dr. Carlos Heitor Pereira Liberalino – UERN
Membro Interno

AGRADECIMENTOS

Aos meus pais, que me propiciaram uma vida digna onde eu pudesse crescer, acreditando que tudo é possível, desde que sejamos honestos, íntegros de caráter e tendo a convicção de que desistir nunca seja uma ação contínua em nossas vidas; que sonhar e concretizar os sonhos só dependerá de nossa vontade.

A minha esposa e ao meu filho, que sempre acreditaram em mim e apoiaram meus sonhos, minhas ideias e até as maluquices.

Aos meus amigos, orientadores e parceiros Profª. Drª. Carla Katarina, Prof. Dr. Dario e Prof. Dr. Lima que incondicionalmente me ajudaram na concretização desse sonho com suas experiências, dedicação, disposição e carinho.

Aos Professores da banca, os quais arrumaram um tempo precioso para apreciar o nosso trabalho e contribuir para melhorá-lo.

EPÍGRAFE

"O único lugar aonde o sucesso vem antes do trabalho é o dicionário"
(Albert Einstein)

RESUMO

A capacidade de processamento dos computadores evoluiu bastante nos últimos anos. Porém ainda existem problemas computacionais que requerem muitas máquinas para realizar uma grande quantidade de processamento de forma paralela e distribuída. A computação em grades está surgindo como uma nova forma de computação distribuída, permitindo a agregação de recursos distribuídos geograficamente em locais diferentes. No entanto esses recursos são independentes e administrados separadamente sobre várias políticas administrativas diferentes. Nesse contexto, o escalonamento de tarefas em um sistema distribuído apresentam muitos problemas a serem solucionados. As técnicas existentes de escalonamento de tarefas não garantem os melhores desempenhos para a distribuição dos conjuntos de tarefas entre os recursos da grade de forma eficiente. Nesta dissertação é apresentada uma solução para o problema de escalonamento de tarefas baseado em algoritmos genéticos para distribuir tarefas de maneira mais eficiente em uma grade computacional. Esse algoritmo foi implementado no GRIDSIM, um simulador de grades computacionais com características e atributos de uma grade real. Em problemas da natureza de alocação de tarefas, os algoritmos evolutivos, de uma forma geral, têm-se mostrados muito eficientes, apresentando soluções melhores, em tempo hábil, do que os algoritmos exatos.

Palavras-Chave: Grades computacionais, algoritmos evolutivos, alocação de recursos, escalonamento.

ABSTRACT

The computer processing power of computers has evolved considerably in recent years. However, there are problems that still require many machines to perform a large amount of processing in parallel and distributed way. Grid technologies are emerging as the next generation of distributed computing, allowing the aggregation of resources that are geographically distributed across different locations. However, these resources are independent and managed separately by various organizations with different policies. In this context, the tasks scheduling in a distributed system present many algorithms. Those algorithms for task scheduling does not guarantee the best performance for the distribution of tasks between the sets of grid resources efficiently In this study, we present a scheduler based on genetic algorithms in order to distribute tasks more efficiently in a computational grid, it has been implemented in GRIDSIM, a computational grid simulator with features and attributes of a real grid. Problems in the nature of task allocation evolutionary algorithms, in general, have shown very efficient, presenting the best solutions in a timely manner, than the exact algorithms.

Keywords: Computational grids, evolutive algorithms, resource alocation, scheduling

LISTA DE TABELAS

LISTA DE FIGURAS

LISTA DE SIGLAS

AG – Algoritmo Genético;

DRM – *Domain Rights Management*;

GRIDSIM – *Grid Simulator* (Simulador de Grade)

GRASP – *Greedy Randomized Adaptative Search Procedure* (Procedimento de busca aleatória gulosa adaptativa);

MIPS – Milhões de Instruções por Segundo;

NP – *Non-deterministic Polynomial-time* (Não determinístico em tempo polinomial);

P – *Deterministic Polynomial-time* (Determinístico em tempo polinomial);

PE – *Processing Element* (Elemento Processante);

SA – *Simulated Annealing* (Têmpera simulada);

UERN – Universidade Estadual do Rio Grande do Norte;

UFERSA – Universidade Federal Rural do Semi-Árido.

Sumário

Capítulo 1 - INTRODUÇÃO

Desde o princípio da computação, um dos maiores interesses na área de pesquisa tem sido o aumento do poder de processamento dos computadores (XHAFA e ABRAHAM 2009). À medida que o tempo vai passando, novas tecnologias de *software* exigem máquinas cada vez mais potentes (FOSTER e KESSELMAN 2004). Essa tendência da tecnologia tem exigido dos distribuidores de processadores uma rápida evolução tecnológica. No entanto, muitas vezes, a tecnologia empregada em um processador não consegue sozinha atender os casos onde a demanda de recursos de processamento é grande. Neste contexto, visando contornar esta deficiência, foram concebidas máquinas paralelas dotadas de um número maior de processadores.

A computação em grade tornou-se uma solução viável para os casos que apresentam uma grande demanda de processamento (FOSTER, KESSELMAN *et al.* 2002).

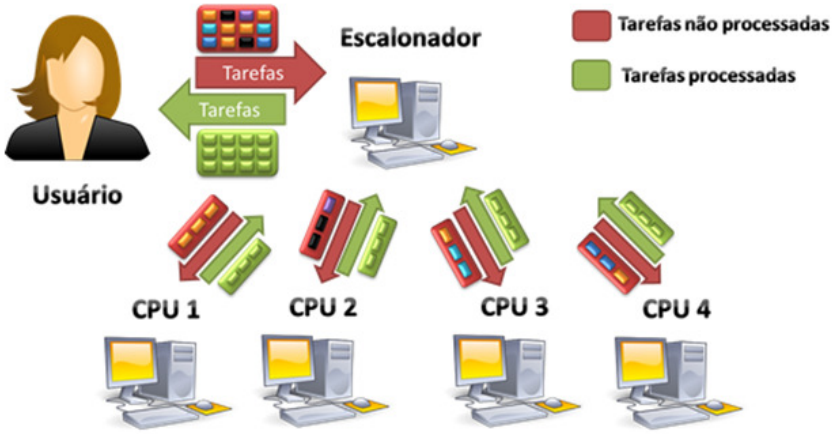

Figura 1 - Exemplo de funcionamento de uma grade computacional.

De acordo com (FOSTER, KESSELMAN *et al.* 2002, BERMAN, FOX *et al.* 2003, CHEDE 2004, FOSTER e KESSELMAN 2004) a computação em grade é um modelo de computação distribuída que usa geograficamente e administrativamente os recursos disponíveis. Usuários individuais podem acessar os computadores e os dados de forma transparente, sem ter que considerar a posição geográfica, sistema operacional, administração de contas e outros detalhes (BUCKLEY 2010). Na Figura 1 temos um exemplo de uma grade computacional, onde um usuário envia um bloco de tarefas para

o escalonador e este divide as tarefas entre os computadores disponíveis para o processamento, recebendo as tarefas processadas e enviando-as de volta para o usuário.

Na computação em grade, os detalhes são abstraídos e os recursos são virtualizados (GENTZSCH, GRANDINETTI *et al.* 2009), ou seja, toda a grade computacional aparece para o usuário como um recurso único. O uso da grade pode ser feito em várias áreas que requerem grande quantidade de processamento, tais como, cálculo de otimizações, indústria farmacêutica, simulações de desastres naturais, processamento de imagens, etc.

Para algumas aplicações, um grande número de processadores torna a solução inadequada, devido à inviabilidade econômica do projeto na construção ou aquisição de máquinas paralelas do porte necessário.

Um grande problema enfrentado nas grades computacionais é o escalonamento de tarefas, e embora seja conhecido como um problema de alocação (FOSTER, KESSELMAN *et al.* 2002), o mesmo difere dos problemas de alocação nos sistemas distribuídos convencionais. Esse problema nas grades computacionais é muito mais complexo, dada a natureza dinâmica (nós que entram e saem da grade, restrições de uso, etc.) e o alto grau de heterogeneidade dos recursos (arquiteturas, sistemas operacionais, localização diferentes) e tarefas que devem ser monitoradas.

Segundo (GAREY e JOHNSON 1979) o escalonamento de tarefas é um problema NP-completo, e de acordo com (LUKE 2011) o uso de heurísticas é de fato a abordagem mais usada a fim de lidar na prática com a dificuldade de problemas com essas características. (LINDEN 2012) afirma que os algoritmos genéticos geralmente apresentam desempenhos melhores do que heurísticas específicas, tais como as apresentadas no simulador *GRIDSIM*[1].

Em problemas de otimização, uma meta ou um valor, mesmo que seja em um intervalo, é chamado de "objetivo" ou "função objetivo" (ABRAHAM, BUYYA *et al.* 2000). Os objetivos podem ser maximizados ou minimizados, buscando os valores máximos ou mínimos para as funções dos problemas, obedecendo às restrições apresentadas para o modelo.

[1] www.gridsim.com/buya

O problema de alocação das tarefas apresenta vários objetivos a serem alcançados de uma forma geral, como diminuir o tempo total de execução das tarefas, diminuir o tempo total de processadores ociosos e aumentar a eficiência do processamento, por exemplo. Os dois objetivos considerados nesse artigo são o tempo de execução de cada tarefa e o tempo total de execução do sistema.

A simulação apresenta-se como sendo uma forma rápida de análise de algoritmos em larga escala em sistemas distribuídos e de recursos heterogêneos, uma vez que o custo de materiais ficaria proibitivo para a realização deste trabalho. Outra vantagem de sua utilização é a de que diminui a complexidade e o trabalho extra que seria necessário para gerenciar máquinas reais, adicionar ou remover novas máquinas, configurar rede, comunicação e segurança (XHAFA, KOŁODZIEJ *et al.* 2010).

A simulação também se torna eficiente quando se trabalha com problemas hipotéticos muito grandes, que de outro modo iria requerer uma grande quantidade de usuários ativos e recursos. Portanto é muito caro coordenar e construir um ambiente físico real em larga escala para propósitos de investigação.

Nesta dissertação é utilizada a ferramenta *GRIDSIM* (BUYYA 2012). Esta permite modelar e simular entidades em paralelo e computação distribuída, usuários, aplicações, recursos e escalonadores, para desenvolvimento e avaliação de algoritmos. Provê tambem uma facilidade para criar diferentes classes de recursos heterogêneos que podem ser agregados usando escalonadores para resolver aplicações de computação e dados intensivos.

Um recurso pode ser um único processador ou multiprocessador com memória compartilhada ou distribuída e gerenciado por escalonadores de tempo e espaço compartilhado.

Por fim esta dissertação apresenta uma política para a alocação de tarefas baseada em algoritmo genético, implementada no simulador *GRIDSIM*, procurando reduzir o tempo total de execução de tarefas.

Outro diferencial em relação aos trabalhos relacionados é que nessa dissertação são apresentados resultados voltados para a área de otimização, mostrando que os valores obtidos para as variáveis indicam que os algoritmos convergem para resultados considerados aceitáveis. Outra vantagem foi o uso do simulador GRIDSIM, pois

simulação possibilita mudar o cenário de forma rápida e dinâmica e diminui a complexidade de operação e criação do ambiente.

Esta dissertação organiza-se da seguinte forma, no capítulo 2 introduzimos uma revisão de literatura com os trabalhos relacionados, no capítulo 3 é abordado o escalonamento em grades e o problema de escalonamento, no capítulo 4 são apresentados os algoritmos genéticos e suas características, no capítulo 5 é apresentada uma nova política de escalonamento, o algoritmo utilizado e as equações de desempenho do algoritmo. No capítulo 6 apresenta os resultados das simulações e uma discussão sobre os mesmos e, por fim, o capítulo 7 apresenta a conclusão e os trabalhos futuros.

Capítulo 2 - TRABALHOS RELACIONADOS

Neste capítulo serão apresentados os trabalhos relacionados com o objetivo desta dissertação, os seus pontos abordados e as suas diferenças em relação a este trabalho.

Escalonadores de tarefas em grades computacionais e aspectos inerentes, como balanceamento de carga, tem sido objeto de muitos estudos em computação. Os trabalhos apresentados a seguir descrevem o escalonamento de tarefas em grids usando heurísticas e metaheurísticas, e apresentam propostas de modelos computacionais. No entanto, não são apresentadas comparações entre os algoritmos aqui descritos.

Em (XHAFA e ABRAHAM 2009) apresenta-se como heurísticas e metaheurísticas podem ser abordadas para o problema de escalonamento de tarefas. O trabalho apresenta uma revisão detalhada do problema de escalonamento, mostrando abordagens feitas com metaheurísticas híbridas para o escalonamento de tarefas.

Simulated Annealing e outras abordagens são encontradas em (ABRAHAM, BUYYA *et al.* 2000, BLUM e ROLI 2003). Nestes trabalhos são apresentadas técnicas de busca local probabilística, que toma como base uma analogia com a termodinâmica, para a alocação de tarefas.

A Busca Tabu é explorada em (XHAFA, KOŁODZIEJ *et al.* 2010, AZIZ e EL-REWINI 2011). Na abordagem foi usada uma busca local para, iterativamente, mover-se de uma solução para uma solução melhor usando como critério de parada um limite de tentativas.

Os próximos trabalhos utilizam AG´s (Algoritmos Genéticos) para realizar escalonamento de tarefas, com características semelhantes a esta dissertação. O que diferencia este trabalho dos demais, é que foi realizada uma comparação com um número variável de máquinas para descobrir uma relação de custo/benefício entre máquinas/tarefas, e também foi feito um comparativo com uma quantidade variável de tarefas para tentar descobrir a eficiência do escalonador baseado em AG em relação aos algoritmos convencionais de escalonamento.

Escalonadores baseados em AG são citados em vários trabalhos. (ABRAHAM, BUYYA *et al.* 2000) apresenta uma proposta para uma estratégia de alocação de tarefas.

O problema não considera pesos para priorizar as alocações, e sua proposta visa apenas um objetivo.

Em (WAGNER e KRONBERGER 2011) é apresentado um modelo de escalonador usando AG, assim como o trabalho anterior, seu modelo é fixo e trabalha apenas uma característica de escalonamento.

(XHAFA e ABRAHAM 2009) apresenta um escalonador usando AG apresentando resultados de desempenho com poucos nós e apresenta também um modelo fixo.

Em (PAPAZACHOS e KARATZA 2010, PAPAZACHOS e KARATZA 2011) são mostrados simulações de escalonamento, elaboradas através de heurísticas fixas.

Outro diferencial desta dissertação em relação aos trabalhos relacionados é a que são apresentados resultados voltados para a área de otimização combinatória, mostrando que os valores obtidos para as variáveis convergem para um resultado considerado aceitável. O uso do simulador GRIDSIM agrega a vantagem de mudar o cenário de forma rápida e dinâmica.

Além do mais, foram realizados comparativos com outros algoritmos disponibilizados no simulador GRIDSIM e foram comparados os resultados entre cargas de trabalho diferentes, mostrando as vantagens obtidas com a utilização dos AGs.

O estudo apresentado nesta dissertação visa realizar um escalonamento de tarefas em grades computacionais usando AG para reduzir o tempo total de execução das tarefas. Isso reduz especificamente o tempo de processamento e aumenta a eficiência da grade. Outro fator que estimulou esse trabalho foi a escassez bibliográfica no que concerne as metaheurísticas para o GRIDSIM, abrindo assim um vasto espaço para pesquisa.

Capítulo 3 - REFERENCIAL TEÓRICO

Neste capítulo será feito uma apresentação sobre algoritmos genéticos e como podemos utilizá-los para solucionar problemas de escalonamento. Será apresentado também o conceito de grades computacionais e seus problemas de alocação, assim como o simulador GRIDSIM e os algoritmos que o acompanham.

3.1 Algoritmos Genéticos (AG´s)

Algoritmos evolucionários usam modelos computacionais dos processos naturais de evolução como uma ferramenta para resolver problemas. Apesar de haver uma grande variedade de modelos computacionais propostos (CAMPELLO e MACULAN 1994, RIEL 1996, ABRAHAM, BUYYA *et al.* 2000, BLUM e ROLI 2003, MICHALEWICZ e FOGEL 2004, VENUGOPAL e BUYYA 2006, DOERNER, GENDREAU *et al.* 2007, DONOSO e FABREGAT 2007, GONZALEZ 2007, COTTA, SEVAUX *et al.* 2008, AZIZ e EL-REWINI 2011), todos eles têm em comum o conceito de simulação da evolução das espécies através de seleção, mutação e reprodução, processos estes que dependem do desempenho dos indivíduos desta espécie dentro do ambiente.

Basicamente os algoritmos evolucionários funcionam mantendo uma população de estruturas que evoluem de forma semelhante à evolução das espécies. A estas estruturas são aplicados os chamados operadores genéticos, como recombinação e mutação, entre outros (LINDEN 2012).

Cada indivíduo recebe uma avaliação que é uma quantificação da sua qualidade como solução do problema em questão. Baseado nesta avaliação é que serão aplicados os operadores genéticos de forma a simular a sobrevivência do mais apto (LUKE 2011).

Os operadores genéticos consistem em aproximações computacionais de fenômenos vistos na natureza, como a reprodução sexuada, a mutação genética, entre outros (WOODWARD e SWAN 2011).

Algoritmos genéticos são um ramo dos algoritmos evolucionários e como tal podem ser definidos como uma técnica de busca baseada numa metáfora do processo biológico de evolução natural.

Das diversas definições apresentadas na literatura, a que mais aproxima os AG´s do contexto desta proposta é que eles são heurísticas de busca que imitam o processo da evolução natural e a constante descrição de sua aplicabilidade para encontrar soluções com valores satisfatórios para resolver problemas do tipo NP-Completo.

Os algoritmos genéticos são técnicas heurísticas de otimização global (RUSSELL, NORVIG *et al.* 2010). A questão da otimização global opõe os AG´s aos métodos como *hill climbing*, que seguem a derivada de uma função de forma a encontrar o máximo de uma função, ficando facilmente retidos em máximos locais, como apresentado na Figura 2.

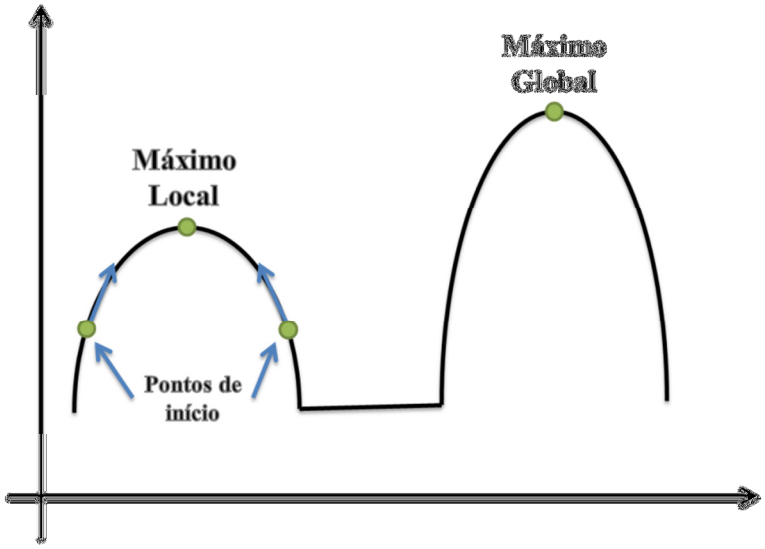

Figura 2 - Função hipotética com um máximo local e outro global. Fonte: R. Lindem 2012

Os pontos de início da figura 2 foram escolhidos aleatoriamente próximos a um máximo local. Fazer a busca pelo método de *hill climbing* fará com que o maior valor encontrado para essa função seja um máximo local, enquanto existe um valor maior na função.

Nos algoritmos genéticos populações de indivíduos são criadas e submetidas aos operadores genéticos conhecidos por seleção, *crossover* e mutação.

Estes operadores avaliam a qualidade de cada indivíduo como critério para encontrar a solução do problema. É gerado um processo de evolução natural destes indivíduos, que eventualmente gerará um indivíduo que caracterizará uma boa solução (talvez até a melhor possível) para o problema (LUKE 2011).

É possível afirmar que os algoritmos genéticos são algoritmos de busca baseados nos mecanismos de seleção natural e genética (COTTA, SEVAUX *et al.* 2008). Eles combinam a sobrevivência entre os melhores indivíduos com uma forma estruturada de troca de informação genética entre dois indivíduos para formar uma estrutura heurística de busca.

Como dito anteriormente, os AG´s não são métodos de *hill climbing*, logo eles não ficarão estagnados simplesmente pelo fato de terem encontrado um máximo local. Neste ponto, a sua abordagem se parece com a evolução natural, que não para de procurar outros indivíduos ainda melhores, mesmo que estes se destaquem no grupo (COTTA, SEVAUX *et al.* 2008).

Na evolução natural isto também decorre de circunstâncias que mudam de um momento para outro (LUKE 2011). Como exemplo, temos uma bactéria considerada como sendo a melhor em um ambiente livre de antibióticos. Entretanto, quando estes são usados, outras bactérias que antes eram menos fortes que a anterior tornam-se as únicas sobreviventes por serem as únicas adaptadas.

No caso dos Algoritmos Genéticos, o ambiente é um só (GONZALEZ 2007). Entretanto, conforme as gerações vão se passando e os operadores genéticos vão atuando, faz-se uma grande busca pelo espaço de soluções, busca esta que seria realizada pela evolução natural (das bactérias ou de qualquer outro organismo) se elas ficassem permanentemente em um ambiente imutável.

A reprodução e a mutação são aplicadas em indivíduos selecionados dentro da população (LINDEN 2012). A seleção deve ser feita de tal forma que os indivíduos mais aptos sejam selecionados mais frequentemente do que aqueles menos aptos, de forma que as boas características daqueles passem a predominar dentro da população de soluções.

Nas próximas seções será apresentada a terminologia e as características dos AGs.

3.1.1 Terminologia

Antes de apresentar as demais características dos AG´s é importante mostrar a terminologia adotada. Como os AGs são altamente inspirados na genética e na teoria da evolução das espécies, há uma analogia muito forte entre os termos da biologia e os termos usados nos AGs.

Nos sistemas naturais um ou mais cromossomos se combinam para formar as características genéticas básicas do indivíduo em questão (LUKE 2011). Na área dos AGs, os termos cromossomo e indivíduo são intercambiáveis, sendo usado de forma razoavelmente aleatória neste texto.

No campo da genética os cromossomos são formados por *gens*, que podem ter um valor entre vários possíveis, chamados de alelos. A posição do *gen* é chamada de seu *locus* (plural : *loci*). Os termos biológicos são aplicáveis também à área de AG, mas é possível o uso do termo "características" para significar *gen*, "valores" significando alelos e "posição" significando *locus*.

Outros termos importantes são "genoma", "genótipo" e "fenótipo". Genótipo é a estrutura do cromossomo, e pode ser identificada na área de AG com o termo estrutura. Fenótipo corresponde à interação do conteúdo genético com o ambiente, interação esta que se dá no nosso campo através do conjunto de parâmetros do algoritmo. Genoma é o significado do pacote genético e não possui análogo na área de AG.

Tabela 1 - Definições de termos dos algoritmos genéticos.

Linguagem natural	Algoritmo Genético
Cromossomo	Indivíduo
Gen	Característica
Alelo	Valor
Locus	Posição
Genótipo	Estrutura
Fenótipo	Conjunto de parâmetros

Um resumo dos termos e suas correspondências podem ser visualizados na Tabela 1, na qual foi incluída a nomenclatura que distingue a área de AG da área da genética. Implicitamente alguns termos da ciência natural podem ser utilizados também

no campo dos AGs, apesar dos termos listados serem os mais comuns na literatura. Neste trabalho serão usados mais os termos descritos na segunda coluna.

3.1.2 Características dos Algoritmos Genéticos

AGs são técnicas probabilísticas, e não técnicas determinísticas (LINDEN 2012). Assim sendo, iniciando um AG com a mesma população inicial e o mesmo conjunto de parâmetros podemos encontrar soluções diferentes cada vez que a aplicação for executada.

AGs são em geral programas que necessitam de informações locais ao nosso ponto, relativas à adequabilidade do ponto como solução do problema em questão, não necessitando de derivadas ou qualquer outra informação adicional (RUSSELL, NORVIG *et al.* 2010). Isto faz com que AGs sejam extremamente aplicáveis a problemas do mundo real que em geral incluem descontinuidades duras (LINDEN 2012).

Descontinuidades duras são situações onde os dados são discretos ou não possuem derivadas (CORMEN 2009). Isto é comum em situações do mundo real em que temos que alocar recursos. Não é possível alocar uma fração de um caminhão ou de uma sala de aula, logo estes problemas não admitem soluções reais, somente inteiras. Portanto, não tem como calcular derivadas ou gradientes, impossibilitando o uso de técnicas numéricas tradicionais.

AGs trabalham com uma grande população de pontos, sendo uma heurística de busca no espaço de soluções (LUKE 2011). Um AG diferencia-se dos esquemas enumerativos pelo fato de não procurar em todos os pontos possíveis, mas sim em um subconjunto destes pontos (RUSSELL, NORVIG *et al.* 2010). Um exemplo muito claro é o uso de AGs para a solução do problema do caixeiro viajante (ou outro similares) em que o número de soluções possíveis é proporcional ao fatorial do número de cidades (LINDEN 2012).

Um AG vai procurar o mesmo número de soluções que os seus parâmetros definirem, e nunca uma fração significativa das soluções possíveis, pois isto inviabilizaria a solução. Além disto, AGs diferenciam-se de esquemas aleatórios por

serem uma busca que utiliza informação pertinente ao problema e não trabalham com caminhadas aleatórias pelo espaço de soluções. Isto é garantido pelo fato de usar o valor da função de avaliação como guia na escolha dos elementos reprodutores.

É importante salientar também que AGs trabalham com uma forma codificada dos parâmetros a serem otimizados e não com os parâmetros propriamente ditos. Assim, deve ser definido um esquema de codificação e decodificação destes parâmetros. Isto equivale à representação cromossômica discutida anteriormente.

Entretanto, não importa ao AG como se codificam ou decodificam a informação dos parâmetros. Ele só se importa com a representação dos parâmetros em si. Toda a informação relativa ao problema está contida na sua função de avaliação, que embute os módulos de codificação e decodificação dos parâmetros. Assim, um mesmo AG pode ser utilizado para uma infinidade de problemas, necessitando-se apenas mudar a função de avaliação, o que obviamente gera uma grande economia de tempo e dinheiro para as organizações.

3.2 Grades computacionais

De acordo com (BERMAN, FOX *et al.* 2003, CHEDE 2004, FOSTER e KESSELMAN 2004) a computação em grade é um modelo de computação distribuída que usa geograficamente e administrativamente os recursos disponíveis. Usuários individuais podem acessar os computadores e os dados de forma transparente, sem ter que considerar a posição geográfica, sistema operacional, administração de contas, e outros detalhes (BUCKLEY 2010). Na computação em grade, os detalhes são abstraídos e os recursos são virtualizados (GENTZSCH, GRANDINETTI *et al.* 2009).

Como os computadores envolvidos no sistema em grade são de propriedade de usuários voluntários (computação filantrópica), que disponibilizam seus equipamentos para fazer parte do trabalho, o custo da computação em grade torna-se inferior ao das máquinas paralelas. Esse paradigma permite que participantes geograficamente distribuídos e sobre administrações independentes compartilhem entre si recursos computacionais ociosos que podem variar de processadores e discos rígidos até licenças de software (CHEDE 2004, FOSTER e KESSELMAN 2004). Exemplos de recursos

que podem ser utilizados são celulares, PDAs (*Personal Digital Assistants*), receptores de sinais digitais de televisão, entre outros equipamentos.

(ABRAHAM, BUYYA *et al.* 2000) retrata o quadro geral de computação em grade, focando a interação entre o escalonador e a grade de recursos, do Gerente de Recursos de Domínio (DRM) e o servidor de informações da rede. O escalonador da grade é responsável pela descoberta de recursos, decidindo alocação de uma tarefa para um determinado recurso, a conexão de aplicações do usuário a arquivos, recursos de hardware, iniciar os cálculos, se adaptar às mudanças nos recursos da rede e apresentar a grade para o usuário como um recurso único e unificado.

3.2.1 Escalonamento em grades

Devido à complexidade dos sistemas de rede e aplicações distribuídas em larga escala, diferentes versões e modos de programação podem ser consideradas (XHAFA 2009). Considera-se aqui uma versão do problema, como apresentado no Capítulo 4, que não leva em conta eventuais restrições a transmissão de tarefas interdependentes de dados, e as políticas econômicas e de custo sobre os recursos.

Preocupa-se nessa dissertação com a programação necessária para alcançar aplicações de alto desempenho e, a partir da perspectiva de usuários da grade, para oferecer *QoS* (qualidade de serviço) no sistema de grade. Este tipo de programação surge em aplicações que podem ser resolvidas por dividi-las em muitos trabalhos independentes, submetendo-as à grade e combinando os resultados parciais para obter a solução final.

Além disso, existe uma necessidade de atribuição de aplicações independentes de usuário para os recursos da grade. Considera-se, assim, o cenário em que os trabalhos enviados para a grade são independentes e não são preemptivos, ou seja, não podem mudar o recurso para qual a tarefa tenha sido atribuída uma vez que a sua execução é iniciada, a menos que o recurso seja retirado da grade.

3.3 GRIDSIM

Uma vez que o custo de materiais e mão de obra tornaria o custo muito elevado para a realização dessa dissertação, a simulação apresentou-se como uma alternativa de análise de algoritmos em larga escala em sistemas distribuídos e de recursos heterogêneos. Outra vantagem da simulação é a de que ela diminui a complexidade e o trabalho extra que seria necessário para gerenciar máquinas reais, adicionar ou remover novas máquinas, configurar rede, comunicação e segurança (XHAFA, KOŁODZIEJ *et al.* 2010). Simulação também se torna eficiente quando se trabalha com problemas hipotéticos muito grandes, que de outros modos requereriam uma grande quantidade de usuários ativos e recursos, o que é muito difícil de coordenar para construir um ambiente em larga escala com propósitos de investigação (PAPAZACHOS e KARATZA 2011).

A ferramenta GRIDSIM permite modelar e simular entidades em paralelo e computação distribuída, usuários, aplicações, recursos e escalonadores, para desenvolvimento e avaliação de algoritmos de escalonamento. Ela provê uma facilidade para criar diferentes classes de recursos heterogêneos que podem ser agregados usando escalonadores para resolver aplicações de computação e dados intensivos. Um recurso pode ser um único processador ou multiprocessador com memória compartilhada ou distribuída e gerenciado por escalonadores de tempo e espaço compartilhado (BUYYA 2012).

3.3.1 Funcionalidades

(BUYYA 2012) destaca as principais funcionalidades do GRIDSIM:

- Incorpora falhas nos recursos do Grid durante a execução;
- Uma nova política de alocação pode ser feita e integrada ao GridSim, estendendo a classe AllocPolicy que é uma classe abstrata que lida com a política de alocação interna de recursos. Novos algoritmos de escalonamento podem ser criados estendendo esta classe e implementando os métodos abstratos necessários.;

- Tem a infraestrutura para suportar a reserva avançada de um sistema em grid;
- Incorpora a funcionalidade que lê os arquivos de carga de trabalho, tiradas de supercomputadores, simulando a carga de trabalho de um ambiente de grade real;
- Incorpora o modelo de leilão. Esse modelo é uma analogia da grade com modelos econômicos, simula o processo de aquisição, oferta e reaproveitamento de recursos;
- Incorpora a extensão de rede, com isso os recursos e outras entidades podem ser ligados em uma topologia de rede;
- Incorpora a funcionalidade de tráfego da rede baseado na distribuição probabilística, podendo simular uma rede pública quando o tráfego estiver congestionado;
- Permite a simulação de múltiplas organizações virtuais, através da incorporação de uma entidade *Grid Information Service* (GIS), conectado a uma topologia de rede.

3.3.2 Políticas de escalonamento

De acordo com as notas de lançamento do simulador em (BUYYA 2012), o mesmo disponibiliza várias políticas de escalonamento de tarefas paralelas em clusters e supercomputadores. As políticas incluem preenchimento agressivo (MU'ALEM e FEITELSON 2001), preenchimento conservador (FEITELSON e WEIL 1998, MU'ALEM e FEITELSON 2001), preenchimento conservador com reserva avançada (FEITELSON e WEIL 1998, MU'ALEM e FEITELSON 2001), preenchimento seletivo (SRINIVASAN, KETTIMUTHU *et al.* 2002) e preenchimento agressivo com múltiplas partições de recursos (LAWSON e SMIRNI 2002).

3.3.2.1 Preenchimento agressivo

Esta politica mantém um perfil de disponibilidade, que contém informações sobre os intervalos dos elementos processantes (PEs) que serão liberados quando

completarem as tarefas em execução (MU'ALEM e FEITELSON 2001). Algumas particularidades deste algoritmo são:

- A lista de máquinas deve ser homogênea;
- O carregamento local não é considerado;
- As tarefas não podem ser paradas ou migradas.

3.3.2.2 Preenchimento conservador

Esta politica mantém um perfil de disponibilidade, que contém informações sobre os intervalos dos elementos processantes (PEs) disponíveis em tempos futuros (FEITELSON e WEIL 1998, MU'ALEM e FEITELSON 2001). O tamanho desta disponibilidade do perfil é proporcional ao número de tarefas nas filas de execução e espera. Para ilustrar como esse perfil funciona, segue o exemplo do algoritmo do simulador (BUYYA 2012), onde o início da simulação e o perfil de disponibilidade estão vazios. O recurso do grid tem 500 PEs, portanto, o intervalo dos PEs disponíveis é de [0..499]. No tempo de simulação 100, uma tarefa TA chega requerendo 100 PEs. É esperado que a tarefa seja executada por 500 segundos. Como o recurso não está executando os trabalhos, TA é aceito e dado o intervalo [0 .. 99]. Os intervalos de PEs atuais disponíveis é atualizado para [100 .. 499] e uma entrada é inserida no perfil para indicar que o intervalo [0 .. 499] estará disponível novamente no tempo de simulação de 600 segundos.

Agora, suponha que uma tarefa TB chega no tempo de simulação 200, requer 400 PEs e deverá funcionar durante 500 segundos. A política verifica os intervalos atualmente disponíveis e encontra [100 .. 499].

Em seguida, a política verifica o perfil de disponibilidade e analisa todas as entradas cujo tempo é menor do que o tempo esperado de rescisão de TB. A política encontra as interseções de PEs entre as entradas, processo semelhante ao encontrar cruzamentos de sequências. Enquanto examina o perfil, a política encontra a entrada em 600 segundos. A intersecção de [100 .. 499] e [0 .. 499] é [100 .. 499]. Isso significa que há recursos suficientes para agendar TB e então TB começa a execução. A política então atualiza os intervalos de PEs atuais para [ ] (vazio). Depois disso, as atualizações

da política acontecem, atualizando a entrada de 600 segundos para [0..99] e insere uma nova entrada com 700 segundos com o intervalo [0 .. 499].

Agora considere que um terceiro trabalho, TC, chega no tempo 250 de simulação e requer 500 PEs e deverá funcionar durante 100 segundos. Como os intervalos atuais disponíveis está vazio, a política verifica o perfil até encontrar uma entrada com suficientes PEs disponíveis. Neste caso, a entrada é encontrada em 700 segundos. A política vai continuar a varredura do perfil de encontrar a intersecção com outros intervalos se houvesse mais. Neste caso, 700 segundos é a última entrada no perfil. A tarefa é então definida para iniciar a execução no tempo 700 segundos. A política em seguida atualiza a entrada do tempo de 700 segundos para [ ] (vazio) e cria uma entrada de 800 segundos com [0 .. 499]. TC é então colocado na fila de espera.

Algumas particularidades deste algoritmo são as mesmas do algoritmo anterior.

3.3.2.3 Preenchimento conservador com reserva avançada

Baseado nos trabalhos de (FEITELSON e WEIL 1998, MU'ALEM e FEITELSON 2001), o preenchimento conservador com reserva avançada é uma política de alocação que mantém um perfil de disponibilidade, assim como o preenchimento conservador. A diferença é que, em muitos casos, uma reserva antecipada exigirá duas entradas no perfil, uma para marcar seu horário de início e outra para delimitar o seu tempo de termino.

Além disso, quando um trabalho é cancelado as reservas antecipadas não são removidas a partir do perfil de disponibilidade e, portanto, não são movidas para frente na fila de agendamento. Em outras palavras, não há compressão da fila de agendamento.

Esta política de escalonamento suporta trabalhos paralelos e algumas funcionalidades de reserva, tais como:

- Processar uma nova reserva;
- Cancelar uma reserva;
- Enviar uma reserva;
- Consultar o status de reserva;

- Listar o tempo livre durante certo período de tempo;
- Fornecer informações de disponibilidade quando uma reserva for cancelada.

3.3.2.4 Preenchimento seletivo

De acordo com esta estratégia de escalonamento, para as tarefas não são dadas uma reserva, ou seja, seus horários de início e fim não foram definidos, ou eles não são utilizados para enchimento até a sua desaceleração esperada exceder algum limite, quando então começam uma reserva. Em outras palavras, se um trabalho aguarda o suficiente, então é lhe dada uma reserva. Isto é feito quando o fator de expansão (*xfactor*) da tarefa excede algum limite "fome" (*starvation*). O *xfactor* de uma tarefa é definido como:

$$Xfactor = \frac{Tempo\ de\ espera + Tempo\ de\ execução\ estimado}{Tempo\ de\ execução\ estimado}$$

O limiar *xfactor* é inicialmente definido como 1.0, e conforme os trabalhos são concluídos ele é atualizado para a desaceleração média dos trabalhos concluídos. Alternativamente, podem-se criar categorias de trabalho, cada categoria terá seu próprio limiar da fome. Esta política mantém um perfil de disponibilidade contendo informações sobre a disponibilidade dos intervalos de elementos de processamento (PEs), que será lançado quando os trabalhos em execução finalizarem.

As particularidades deste algoritmo são as mesmas dos algoritmos anteriores.

3.3.2.5 Preenchimento agressivo com múltiplas partições de recursos

É uma política de não-FCFS (*First Come First Serve*, primeiro a chegar, primeiro a servir) para agendar trabalhos paralelos. A política se baseia em preenchimento agressivo fácil. Esta política pode usar múltiplas partições ou filas e os trabalhos podem ser direcionados para estas partições usando um predicado de partição (*PartitionPredicate*). Uma partição pode tomar emprestados recursos de outra quando ela necessita e quando estes recursos não estão sendo usados por outra partição. No entanto, você pode alterar esse comportamento chamando a função *setAllowBorrowing*

(boolean), função esta que indica quando o empréstimo de recursos entre partições é permitido ou não. Além disso, esta política suporta prioridades.

Os trabalhos são ordenados de acordo com suas prioridades. Um trabalho de alta prioridade pode tomar o lugar de um pivô com menor prioridade. Para alterar a maneira que o programador atribui prioridades para as tarefas, deve-se alterar a classe *PrioritySelector*, esta interface é utilizada pelo escalonador para obter a prioridade de um dado item escalonado (ou seja, uma tarefa ou uma reserva avançada). Esta informação é usada por políticas de alocação que usam prioridades.

Este algoritmo foi baseado no trabalho de (LAWSON e SMIRNI 2002), e nele é usado um perfil disponibilidade para armazenar a disponibilidade de elementos de processamento. Para representar os pivôs (ou seja, a primeira tarefa nas partições), são agendadas e então criadas as entradas no perfil de disponibilidade. Desta forma, não é preciso armazenar os tempos iniciais dos pivôs (ou tempos de sombra) e nós extras em variáveis diferentes. Ele também faz a busca de recursos disponíveis para um novo pivô mais fácil.

Algumas particularidades deste algoritmo são as mesmas dos algoritmos anteriores.

Nesse capítulo foram vistas as definições dos algoritmos genéticos, sua terminologia e suas características. Foi apresentado também o conceito de grades computacionais, o escalonamento de tarefas, o simulador Gridsim, assim como os algoritmos de escalonamento que o acompanham.

Capítulo 4 - FORMULAÇÃO DO PROBLEMA

Nesse capítulo será apresentado o problema de escalonamento, os objetivos do escalonador e a abstração do escalonamento real em um arquivo.

De acordo com (FOSTER, KESSELMAN *et al.* 2002, FOSTER e KESSELMAN 2004), este problema consiste em três componentes principais: consumidores, recursos e políticas de escalonamento. Os consumidores são usuários com programas a serem processados. Os recursos são as máquinas da grade disponíveis para o processamento. A política de escalonamento especifica os objetivos que um sistema de escalonamento pode satisfazer. Ele também especifica, no nível de implementação, o método de mapeamento de tarefas para os recursos. A política de escalonamento escolhida para implementar um escalonador afeta os usuários e os provedores de recursos. A política usada nessa dissertação usa um modelo baseado em algoritmo genético.

(FOSTER e KESSELMAN 2004) diz que um escalonador é projetado para satisfazer um ou mais dos seguintes objetivos comuns:

- Maximizar a vazão de saída de dados do sistema;
- Utilizar recursos de forma a se obter o melhor desempenho possível;
- Ampliar a economia de energia de processamento e de tempo;
- Diminuir o tempo de processamento de um aplicativo.

Os estudos de (ABRAHAM, BUYYA *et al.* 2000) reforçam que o escalonamento de tarefas é importante, no sentido de que quanto mais eficiente for um escalonador, mais econômico fica o processamento, uma vez que se têm menos processadores ociosos e as tarefas ficarão menos tempo nos processadores.

Neste trabalho buscamos melhorar a eficiência do escalonador reduzindo o tempo total de execução das tarefas e aproveitando melhor os processadores ociosos.

Para capturar as características mais importantes em grades computacionais, deve-se processar um modelo dessas características. Para formular o problema, é preciso uma estimativa computacional da carga de processamento de cada tarefa e a capacidade computacional de cada recurso. Nesse trabalho foi utilizado o modelo de arquivo de "carga de trabalho padrão" (*standard workload format – SWF*) (CHAPIN, CIRNE *et al.*

1999). Esta formulação é aplicável na prática, uma vez que é fácil de conhecer a capacidade de computação de cada recurso. Além disto, os requisitos de necessidade de computação das tarefas podem ser conhecidas a partir de especificações definidas pelo usuário, de dados históricos ou de previsões feitas pelas necessidades do projeto através das características que se precise na grade.

Assim, é feita uma suposição habitual que é conhecida a capacidade computacional de cada recurso, uma estimativa ou previsão das necessidades computacionais (carga) de cada trabalho, e a carga de trabalho anterior de cada recurso. O modelo *SWF* permite a introdução de possíveis inconsistências entre as tarefas e os recursos da grade. Além disso ainda é possível capturar diferentes características dos sistemas distribuídos heterogêneos, como a coerência da computação, a heterogeneidade de recursos e a heterogeneidade das tarefas. Uma definição formal da instância do problema, dentro de uma matriz *SWF*, é dada pelos seus campos de dados.

Na Figura 3 é apresentada, como exemplo, a abstração de um conjunto de tarefas, onde cada linha, representada por uma cor diferente para uma melhor visualização, corresponde a uma tarefa e os números de cada linha representa um campo de dados de cada tarefa.

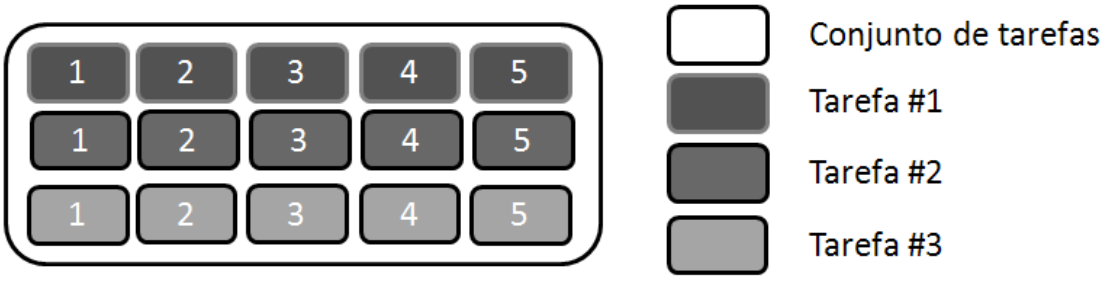

Figura 3 - Exemplo abstrato de um conjunto de tarefas. Fonte: O autor.

Como o objetivo deste trabalho é a redução do tempo total da execução, também chamado de *Makespan*, e consequentemente do tempo de ociosidade dos processadores, os campos levados em consideração nas simulações foram os 1,2,3,4 e 5. A descrição de todos os campos é dada a seguir.

1. Número da tarefa - campo de contagem das tarefas, a partir de 1.

2. Tempo de envio - em segundos. O tempo inicial ao qual o log da simulação refere-se é zero, e é também o tempo de envio da primeira tarefa. As linhas no log são ordenadas por ordem crescente do tempo de envio. As tarefas também serão numeradas nesta ordem.
3. Tempo de espera - em segundos. É a diferença entre o tempo de envio da tarefa e o momento em que a tarefa realmente começou a executar.
4. Tempo de execução - em segundos. É o tempo no qual o trabalho foi executado (tempo final menos tempo inicial).
5. Número de processadores alocados - inteiro. Na maioria dos casos este é também o número de processadores que a tarefa usa.
6. Tempo médio de CPU Usado - em segundos. Esta é a média do tempo de CPU usada sobre todos os processadores, podendo ser menor do que o tempo de execução. Se um registro contém o tempo total de CPU utilizada por todos os processadores, este então é dividido pelo número de processadores atribuídos para derivar à média.
7. Memória Utilizada - em *kilobytes*.
8. Número solicitado de processadores, número mínimo de processadores solicitados.
9. Tempo requisitado. Pode ser tanto o tempo de execução (medido em segundos), quanto o tempo médio de CPU para cada processador (também em segundos). O significado exato é determinado por um comentário do cabeçalho. Se um log contém um pedido de tempo total de CPU, este é dividido pelo número de processadores solicitados.
10. Memória solicitada.
11. Estado da tarefa – recebe o valor 1 se a tarefa foi concluída, 0 se falhou, e 5 se foi cancelada. Se a informação sobre *checkpointing* ou *swap* está inclusa, outros valores também são possíveis. Quando este campo é insignificante para os modelos recebe o valor -1.
12. ID do usuário – É um número natural, entre um e o número de usuários diferentes.
13. ID do grupo – É um número natural, entre um e o número de grupos diferentes. Alguns sistemas controlam o uso de recursos por grupos ao invés de usuários individuais.

14. Número (aplicativo) executável – É um número natural, entre um e do número de aplicações diferentes que aparecem na carga de trabalho. Em alguns logs, isso pode representar um arquivo script utilizado para executar tarefas, em vez de executá-las diretamente, o que deve ser observado em um comentário de cabeçalho.
15. Número de fila – É um número natural, entre um e o número de filas diferentes no sistema. A natureza das filas do sistema deve ser explicada em um comentário de cabeçalho. Este campo é onde o lote e as tarefas interativas devem ser diferenciados: é sugerido a convenção de denotar trabalhos interativos por 0.
16. Número de partição - um número natural, entre um e o número de diferentes partições nos sistemas. A natureza de partições do sistema deve ser explicada em um comentário de cabeçalho. Por exemplo, é possível utilizar os números de partição para identificar qual máquina foi usada em um agrupamento.
17. Número anterior de trabalho - este é o número de uma tarefa anterior na carga de trabalho, de modo que o trabalho atual só pode começar após o término do trabalho anterior.
18. Tempo de tarefa precedente - este é o número de segundos que deve decorrer entre o término do trabalho anterior e a apresentação de um presente.

O algoritmo busca reduzir o valor do tempo total de execução realocando tarefas para processadores ociosos, reduzir o tempo de espera e aumentar o número de processadores alocados.

Capítulo 5 - ESCALONADOR BASEADO EM ALGORITMO GENÉTICO

Esse capítulo apresenta a concepção e implementação da política de escalonamento usando AGs. Cada escalonador da grade é projetado para ser parte de uma arquitetura específica de gestão de recursos. O escalonador apresentado nesta dissertação foi projetado para ser compatível com outras ferramentas que façam parte de um sistema de alocação de recursos.

5.1 O algoritmo

Visto que os AGs são um ramo da computação evolucionária, seu funcionamento no escalonador de tarefas é descrito no pseudocódigo apresentado na Figura 4, onde cada iteração do loop é chamada de "geração". É possível resumir o funcionamento de um AG através da Figura 5.

```
   Entrada: População Inicial
   Saída: Melhor solução encontrada até a parada do algoritmo
1  Início Algoritmo Genético
2      Enquanto o valor de todos os indivíduos da população inicial não for obtido faça
3          Para cada indivíduo faça
4              atribua uma tarefa para cada processador de maneira tal que o tempo mais cedo de
               processamento das tarefas com o mínimo tempo de processadores ociosos possa ser
               obtido( );
5          fimPara
6          Calcule a função de avaliação( );
7      fimEnquanto
8      Selecione os melhores indivíduos da população inicial usando o método da Roleta( );
9      repita
10         Gere uma nova população( );
11         realize uma operação de crossover( );
12         realize uma operação de mutação( );
13         selecione os indivíduos remanescente da população anterior, através de elitismo( );
14         Avalie os indivíduos de acordo com a função de avaliação( )
15     até que o critério de parada seja alcançado;
16 fim
```

Figura 4 - Algoritmo Genético usado na política de escalonamento.

A Figura 6 é apresentada para facilitar o entendimento do fluxo de execução do simulador. As setas indicam o fluxo das tarefas selecionadas pelo usuário. Elas entram

no simulador e são escalonadas de acordo com a política escolhida para, em seguida, apresentar os dados da simulação.

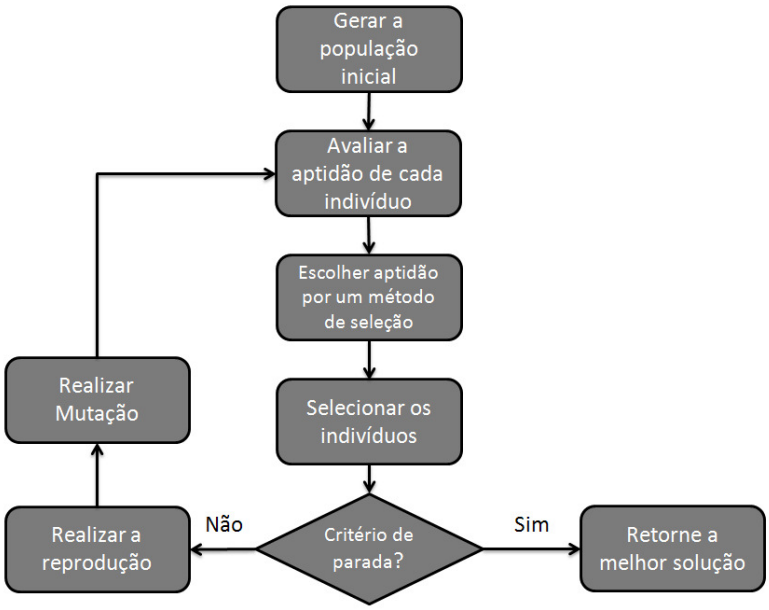

Figura 5 - Fluxograma de funcionamento do algoritmo genético.

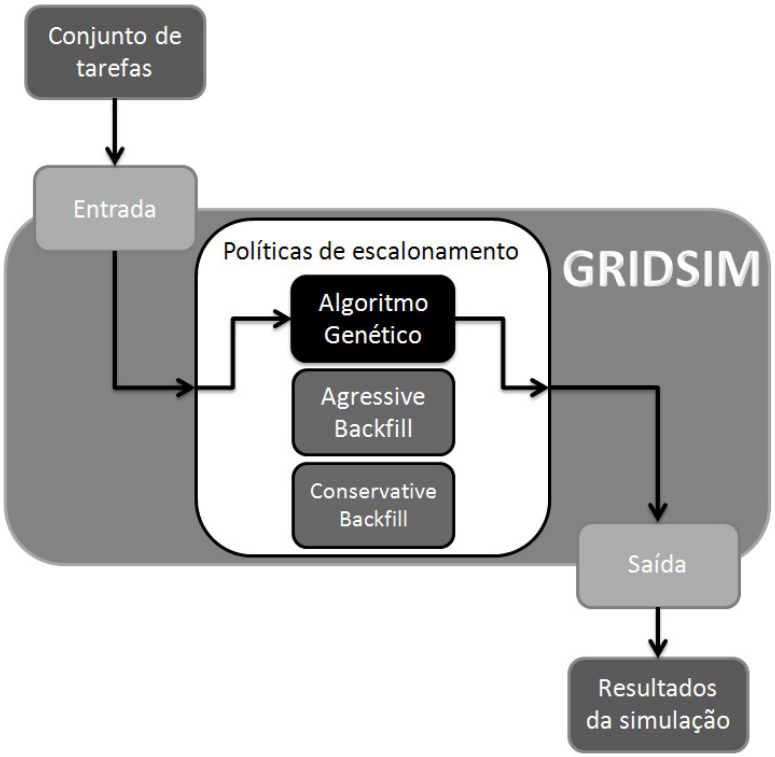

Figura 6 - Fluxograma do escalonamento de tarefas no GRIDSIM usando algoritmo genético.

5.2 A política de escalonamento baseada em Algoritmo Genético

O desempenho de diversos algoritmos de escalonamento foi avaliado neste trabalho através de simulações de alocações de tarefas, modeladas a partir de aplicações reais. Os conjuntos de tarefas utilizadas estão disponíveis em (FEITELSON 2012).

Comparações de desempenho desses algoritmos de escalonamento usando o mesmo conjunto de tarefas são possíveis. Para permitir aos pesquisadores avaliar os seus algoritmos, é proposto um conjunto de tarefas e os métodos de geração de gráficos. Cada conjunto de tarefas é representado por um único arquivo *SWF* (*Standard Workload Format* – Formato Padrão de Carga de trabalho).

Em um ambiente de grades, os objetivos do usuário e dos provedores de recursos podem ser conflitantes, isto é, quando se tenta melhorar um critério de desempenho, outro critério pode ser degradado. Uma busca baseada em algoritmo genético pode conciliar objetivos conflitantes pela tentativa de satisfazer todos os envolvidos.

Uma busca baseada em algoritmo genético pode obter um valor quase ótimo em uma quantidade razoável de tempo, mesmo quando o espaço de solução é muito grande. Escalonamento em grades é um problema de otimização (FOSTER e KESSELMAN 2004). Pesos podem ser usados com cada objetivo para mudar o comportamento do escalonador em tempo de execução, permitindo ao escalonador priorizar múltiplos objetivos.

Um escalonador baseado em algoritmo genético pode se adaptar facilmente a pequenas alterações no espaço do problema (LINDEN 2012). Nesse trabalho isso é obtido quantificando o peso de cada um dos atributos medidos. Como esses pesos são variáveis que multiplicam os fatores que se quer melhorar, pode-se aumentar essas variáveis quando for necessário obter um valor maior na função de avaliação, ou diminui-la quando o que se procura for o inverso. Assim, essa dissertação apresenta um escalonador baseado em algoritmo genético para a distribuição de tarefas em grade para o problema proposto de alocação de recursos.

A função de seleção no algoritmo escolhe os elementos da população que participarão do processo de reprodução, isto é, seleciona os pais dos indivíduos que estarão presentes na nova população. Esta escolha deve ser feita de tal forma que os

membros da população mais adaptados ao meio ambiente, que neste caso são as tarefas que apresentem um escalonamento mais eficiente, tenham maior chance de reprodução, ou seja, àquelas tarefas que apresentam um valor da função de avaliação mais elevado. A forma que foi utilizada para fazer a seleção dos indivíduos é o método da roleta (LINDEN 2012).

No método da roleta todos os indivíduos da população ocuparão uma porção da roleta, proporcional ao seu índice de aptidão (*fitness*). Com isto os indivíduos que possuem uma alta aptidão ocuparão uma porção maior do que os indivíduos que possuem uma aptidão menor. Esta roleta é girada varias vezes, onde a quantidade de giros varia conforme o tamanho da população. Em cada giro da roleta é selecionado um individuo que participará do processo de geração da nova população (LUKE 2011).

Um cromossomo ou um indivíduo representa um agendamento de uma tarefa submetida aos recursos. Cada conjunto de tarefas representa uma população e é um arquivo *SWF*. Esse formato de arquivo pode conter muitas tarefas que podem ter restrições com vários níveis de precedência. Um novo cromossomo é gerado permutando as tarefas em um agendamento, mantendo o elemento processante e trocando a tarefa que se vai processar.

A função de avaliação analisa a qualidade do escalonamento, que é definida nos termos dos objetivos que se esperam alcançar. Assim, o usuário pode ter os objetivos de minimizar o tempo total de execução e ainda satisfazer o prazo apresentado pelo usuário. Enquanto isso, um provedor de recursos pode ter os objetivos de minimizar o tempo de conclusão da sessão, ou o tempo total de execução e minimizar o tempo de ociosidade dos nós computacionais, bastando atribuir os pesos apresentados na equação de *fitness* – ou função de avaliação, as restrições que se busca alcançar.

Um serviço de alocação de recursos tenta agendar um número de tarefas em um conjunto durante uma sessão de processamento (FOSTER, KESSELMAN *et al.* 2002). Várias tarefas alocadas ao mesmo tempo levam a uma melhor utilização dos nós. O escalonador é capaz de alocar tarefas nos espaços de tempo ociosos dos nós computacionais deixados de fora devido a restrições de precedência de uma tarefa (PAPAZACHOS e KARATZA 2011). Portanto, a função de avaliação representa cumulativamente os objetivos de um número de usuários.

Uma vez que as tarefas podem ter tamanhos diferentes, para a obtenção de um resultado cumulativo de várias tarefas e para a obtenção de uma métrica de múltiplos objetivos medidos, os valores devem ser normalizados.

A normalização de uma variável tem por objetivo colocar a magnitude dessa variável dentro de um conjunto de valores de maneira que os valores de todas as variáveis estejam dentro do mesmo conjunto. Assim é possível obter os dados de uma forma mais apropriada para a análise. Em suma, o propósito da normalização é minimizar os problemas oriundos das diferentes unidades e dispersões distintas entre as variáveis (WOODWARD e SWAN 2011).

Uma característica de uma tarefa que ajuda na normalização é o "momento crítico" de uma tarefa. Este é o tempo mínimo no qual uma tarefa pode ser processada se todos os recursos necessários estão disponíveis sem qualquer atraso. Uma sessão pode ter p tarefas independentes. Seja $\mathrm{t}_{\mathrm{tarefafinal}}(i)$ o tempo de execução total da i-ésima tarefa, e seja T_{je} o valor acumulado do tempo de execução das tarefas p. Seja t_{st} o tempo final da última tarefa para um determinado agendamento. Na equação a seguir, tem-se como calcular o tempo total da execução de todas as tarefas. Essa fórmula (1) é usada para se medir o desempenho da política de escalonamento.

$$T_{je} = \sum_{i=1}^{p} t_{tarefafinal}(i) \quad (1)$$

Para se calcular o valor normalizado do tempo acumulado de execução de cada tarefa é utilizada a seguinte equação (2):

$$T_{je_{normalizado}} = \frac{T_{je}}{t_{st} * p} \quad (2)$$

Seja $t_{deadlineTarefa}(i)$ o tempo final da i-ésima tarefa, e seja $T_{dt}(i)$ o atraso no encontro do tempo final de requerimento para a i-ésima tarefa, ou seja, o tempo final (prazo máximo) da tarefa. Fica explicito seu calculo na seguinte equação (3):

$$T_{dt}(i) = (t_{tarefafinal}(i) - t_{deadline_{tarefafinal}}(i)) \quad (3)$$

Para $\left(t_{tarefafinal}(i) > t_{deadline_{tarefafinal}}(i)\right)$, (caso contrário, recebe 0). O atraso cumulativo do prazo final é definido na equação (4):

$$T_{dt} = \sum_{i=1}^{p}\left(T_{dt}(i)\right) \quad (4)$$

Denotando t_{st} como o tempo final da última tarefa para um determinado agendamento, define-se o atraso do tempo final cumulativo normalizado na equação (5):

$$T_{dn} = \frac{T_{dt}}{t_{st}*p} \quad (5)$$

Seja $t_c(i)$ o tempo crítico da i-ésima tarefa, T_c sendo o maior valor de $t_c(i)$ sobre todas as tarefas sendo escalonadas e t_{st} o tempo final para a última tarefa para um dado escalonamento. E denotando ω como o tempo total da sessão, seu cálculo é o realizado através da equação (6):

$$\omega = \left(1 - \frac{T_c}{T_{st}}\right) \quad (6)$$

A ociosidade em um nó aparece devido aos tempos de processamentos não utilizados que precisam ser deixados para satisfazer as restrições precedentes das tarefas.

Seja m o número de espaços no j-ésimo nó e seja n o número de nós computacionais. Seja $t_{gs}(j,k)$ o tempo inicial do k-ésimo tempo ocioso do j-ésimo nó e seja $t_{ge}(j,k)$ o tempo final do k-ésimo tempo ocioso do j-ésimo nó. A equação (7) é demonstra como realizar esse calculo.

$$T_{ge} = \sum_{p=1}^{n}\left(\sum_{k=1}^{m}\left(t_{ge}(j,k) - t_{gs}(j,k)\right)\right) \quad (7)$$

Seja T_{gen} o tempo acumulado de ociosidade de todos os n nós com o seu valor normalizado, a equação (8) demonstra como se calcular esse valor.

$$T_{gen} = \frac{T_{ge}}{(t_{st}*p)} \quad (8)$$

Finalmente, os quatro componentes α, β, θ e λ, são pesos para se obter a função de avaliação (*fitness*) F (9):

$$F = 1 - \left(\frac{\alpha*\omega + \beta*T_{jen} + \theta*T_{dn} + \lambda*T_{gen}}{(\alpha*\beta*\theta*\lambda)}\right) \quad (9)$$

Os quatro pesos são usados para priorizar qualquer componente especial de acordo com as necessidades das tarefas, dos usuários ou do provedor de recursos. Basta

atribuir um valor maior para qualquer um dos pesos associados às equações de desempenho e a função de avaliação tenderá a selecionar aquele cromossomo que favorecer o cenário de busca pretendido, convergindo para a solução que se espera alcançar.

Depois que a função de aptidão para cada um dos cromossomos da população inicial for encontrada, o valor médio das funções de aptidão e o desvio padrão serão calculados. Se o desvio padrão atingir um limiar suficientemente baixo e uma solução aceitável estiver disponível, é dito que o algoritmo genético convergiu. Uma vez que o algoritmo converge para uma solução, a geração de uma nova população não é necessária (LINDEN 2012). No entanto, uma vez que a população inicial foi gerada de forma aleatória, sem qualquer consideração sobre a qualidade da solução, não é provável que o desvio padrão seja inferior ao limite aceitável da solução e os cromossomos disponíveis nesta fase não são suscetíveis de fornecer uma solução aceitável. Assim para evitar ótimos locais, o escalonador foi concebido para se mover imediatamente para a geração da primeira nova população, utilizando a população inicial como base.

Para gerar a próxima geração da população, o método do elitismo é usado. Primeiro, um conjunto de funções de aptidão é construído. Números aleatórios são gerados para selecionar, a partir da população existente, os candidatos para gerar a nova população. Os métodos de seleção, crossover, mutação e elitismo são aplicados em conjunto para gerar a nova geração da população.

A taxa de crossover r_c é escolhida para promover a rápida convergência do algoritmo. Esta taxa foi escolhida para utilização em escalonadores de grade através de um grande número de experiências. Se o tamanho da população existente é P, então os pais $r_c * P$ são escolhidos, por meio do processo de seleção, para criar um número igual de descendentes para a nova população. Para o método de seleção, foi utilizado o esquema de Roleta (RUSSELL, NORVIG *et al.* 2010, LUKE 2011, LINDEN 2012) devido a sua simplicidade.

O processo de cruzamento pode produzir cromossomos ilegais que podem conter as duplicatas da mesma tarefa. Foi usado o método do *crossover* uniforme para a permutação (LUKE 2011) quando são gerados dois cromossomos descendentes de dois cromossomos pais. Para isso, considere *P1* o primeiro indivíduo selecionado para a

reprodução, *P2* o segundo indivíduo selecionado para a reprodução, *D1* o primeiro descendente resultante do cruzamento de *P1* e *P2* e *D2* o segundo descendente resultante do cruzamento de *P1* e *P2*.

Aleatoriamente é gerado uma máscara binária (*M*) de comprimento L (igual ao do cromossomo) e um inteiro aleatório j recebendo o valor 0 ou 1. Então, em cada posição (i) de *M* tal que $M[i] = j$, copiamos *P1*[i] para *D1*[i] e copiamos P2[i] para *D2*[i], quando $M[i] \neq j$, a operação é invertida copiamos *P1*[i] para *D2*[i] e copiamos P2[i] para *D1*[i]. A Figura 7 descreve um exemplo do processo de cruzamento.

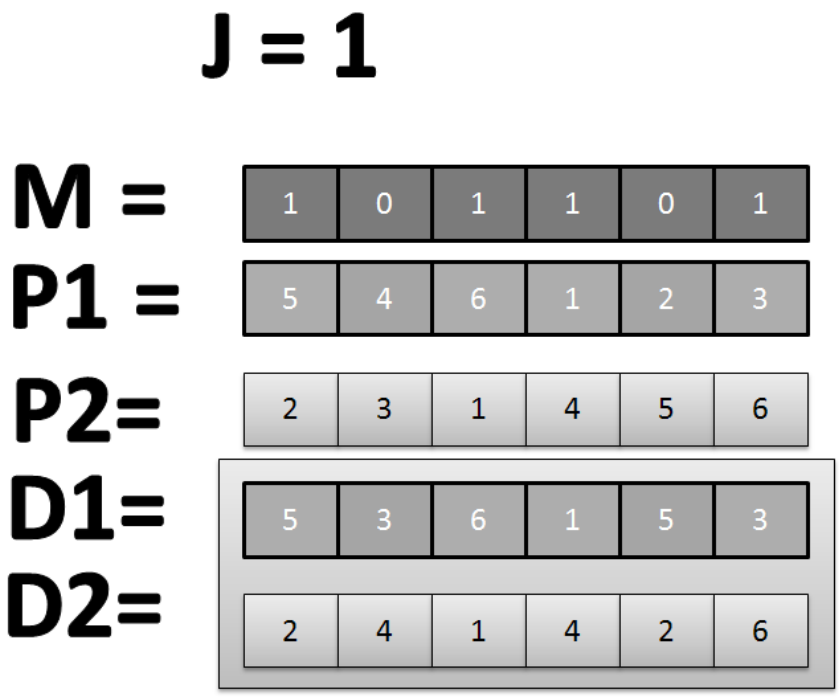

Figura 7 - Exemplo do processo de *crossover*.

A taxa de mutação r_m é escolhida experimentalmente, de forma a promover a rápida convergência. Se o tamanho da população existente é *P*, $r_m * P$ cromossomos são escolhidos, por meio do processo de seleção, para criar um número igual de cromossomos para a nova população. O processo de mutação é utilizado para orientar a nova população a partir de um valor local ideal. Para um processo de mutação é gerado aleatoriamente um inteiro *j* de tamanho máximo do cromossomo, para depois trocar os elementos nas posições *j* entre dois cromossomos. Um valor elevado de r_m pode inverter o progresso no sentido da convergência, portanto, esse valor deve ser selecionado em cada caso por meio de estudo cuidadoso.

Para fazer com que o tamanho da população nova seja igual ao tamanho da população existente, após os processos de cruzamento e mutação terem sido utilizados, os cromossomos restantes são selecionados entre a população existente, tornando-os idênticos aos melhores cromossomos existentes, escolhidos através do processo de seleção.

Nesse capítulo foi apresentada a ideia de como foi feito o novo escalonador usando algoritmos genéticos, as fórmulas e as equações por trás de sua concepção, e os métodos de seleção, *crossover* e mutação.

Capítulo 6 - RESULTADOS E DISCUSSÃO

Esse capítulo apresenta os resultados e discussões da política de escalonamento usando algoritmo genético.

As máquinas utilizadas na simulação dessa dissertação apresentam as seguintes características:

1. 2 processadores por máquina;
2. 100 Mbits de *baud* na rede;
3. 377 Mips (Milhões de instruções por segundo) de poder de processamento por processador.

O escalonador baseado em algoritmo genético apresentado nesse trabalho foi projetado para agendar várias tarefas. Cada tarefa pode conter várias outras tarefas, cada uma com restrições de precedência arbitrárias e tempos de execução arbitrários. O algoritmo apresentado realiza uma otimização multiobjetivo (diminuir o tempo total da execução, aumentar a eficiência do escalonador) para o critério de concorrência.

Os primeiros conjuntos de testes utilizados para obter os resultados de escalonamento de tarefas continham entre 1950 e mais de 3000 tarefas com restrições arbitrárias de precedência e de tempos de execução. Os valores das simulações dos algoritmos de escalonamento que acompanham o *GRIDSIM* são os mesmos para a mesma quantidade de máquinas e de tarefas. Para resolver o mesmo problema usando as mesmas características do simulador, foi possível comparar os melhores resultados obtidos das simulações com os resultados do escalonador baseado em algoritmo genético. Foram feitas simulações para executar tarefas de arquivos *SWF* diferentes. Os arquivos utilizados nas experiências foram:

- sdsc_blue_01.txt - acompanha o simulador
- sdsc_blue_02.txt - acompanha o simulador
- NASA-iPSC-1993-3.1-cln - obtido de (FEITELSON 2012)
- OSC-Clust-2000-3.1-cln - obtido de (FEITELSON 2012)
- LLNL-Atlas-2006-2.1-cln - obtido de (FEITELSON 2012)
- SDSC-Par-1995-3.1-cln - obtido de (FEITELSON 2012)

A partir dessas simulações é que foram obtidos os melhores valores da taxa de cruzamento, da taxa de mutação e do critério de convergência. Após as três validações com 100 simulações cada, foram escalonados os arquivos *SWF* listados anteriormente em um número variável de nós. Os resultados mostraram que o escalonador é capaz de satisfazer múltiplos objetivos e converge para uma solução aceitável em um número razoável de gerações.

Foram comparados os valores obtidos através do algoritmo proposto, com os valores ótimos dado em (BUYYA 2012) para as tarefas que acompanham o simulador e com os valores fornecidos em (FEITELSON 2012) para as tarefas que não acompanham o simulador.

Seis arquivos *SWF* contendo de 1.950 a mais de 50.000 tarefas foram selecionados para o teste. As tarefas em cada arquivo têm diferentes limitações de precedência, em geral possuem vários pais e descendentes múltiplos. As tarefas têm vários campos e cada campo possui um tamanho arbitrário. Além disso, o tempo de execução de cada tarefa é variável. As experiências com escalonadores foram realizados com 64 a 1024 nós contidos no arquivo "sdsc_blue_02.txt" e um teste com todos os arquivos citados acima com apenas 64 nós computacionais. Foram obtidos os valores ótimos dos arquivos em algumas simulações do escalonador usando a política do algoritmo genético e, na maioria dos casos, valores aproximados aceitáveis.

Tabela 2 – Comparação de algoritmos segundo valores do tempo total da simulação do *GRIDSIM*.

Número de máquinas x algoritmos – *Timespan* em segundos						
	64	128	256	512	764	1024
Aggressive backfiling	4193086	4377605	2829396	1616401	13556503	1354434
Conservative backfiling	4201438	4370754	2829517	1631777	13556503	1354434
Aggressive Multipartitions	4193086	4377605	2829396	1616401	13556503	1354434
AR Consevative	4201438	4370754	2829517	1631777	13556503	1354434
Selective Backfill	4379843	4543232	2911274	1681533	1357309	1354434
Algoritmo Genético	4022707	4293508	2770043	1602618	1356503	1354434

Os resultados são mostrados na Tabela 2. Estes são os valores obtidos com a variabilidade da quantidade de máquinas. Nessa tabela, embora os ganhos apresentados em relação a quantidade de máquinas não seja muito grande, o desempenho do escalonador usando a política de algoritmo genético é superior, exceto quando o número de máquinas excede a quantidade de tarefas a ser escalonada.

Para um número menor de nós computacionais o esforço exigido na otimização é bastante elevado. Em geral, um algoritmo heurístico funcionará melhor para um maior número de nós do que para um número menor. O caso de dois nós pode ser considerado como um caso degenerado. Quando a escolha é limitada a apenas dois nós, o processo de otimização não pode fazer muita melhoria do que o algoritmo heurístico que acompanha o simulador. A Figura 8 apresenta os resultados da simulação de forma gráfica.

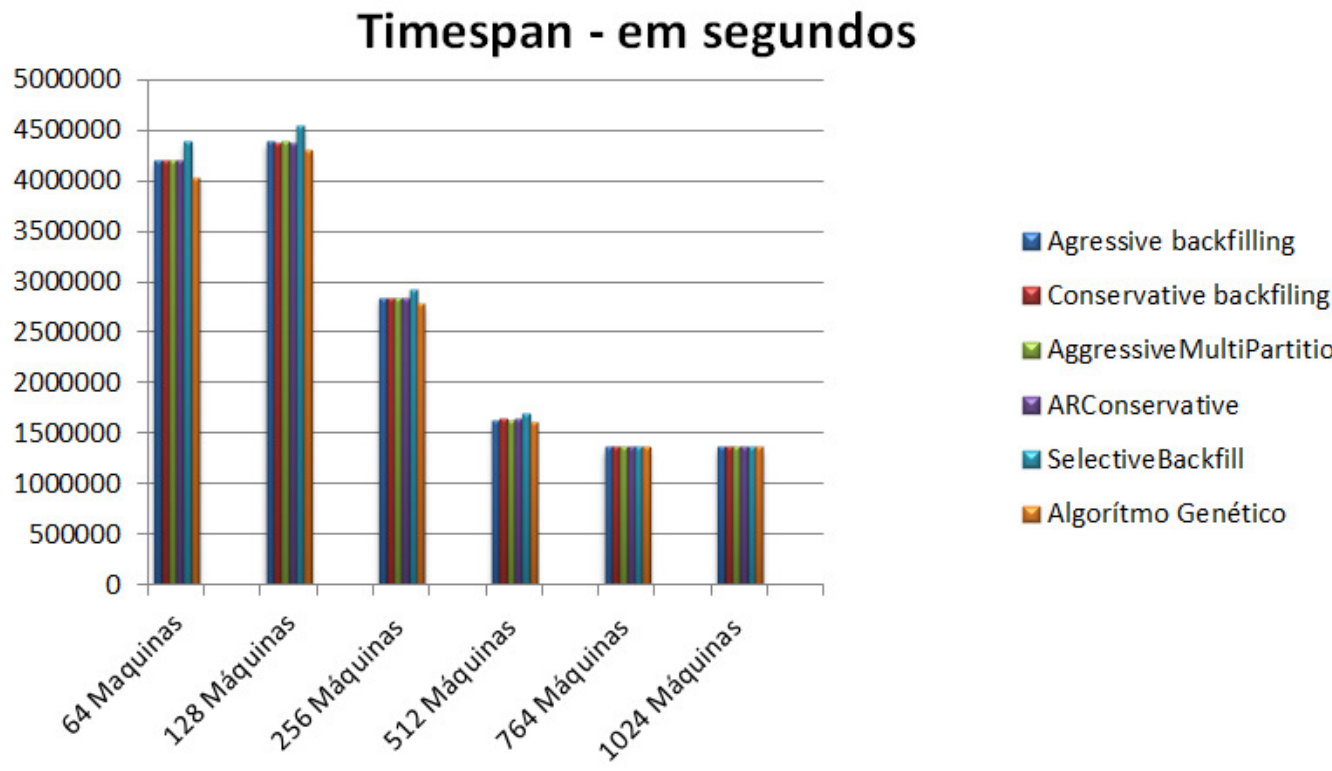

Figura 8 - Gráfico de valores do tempo total da simulação do *GRIDSIM* comparando os algoritmos.

Na Figura 9 é apresentado o tempo total da simulação de 64 maquinas variando a quantidade de tarefas.

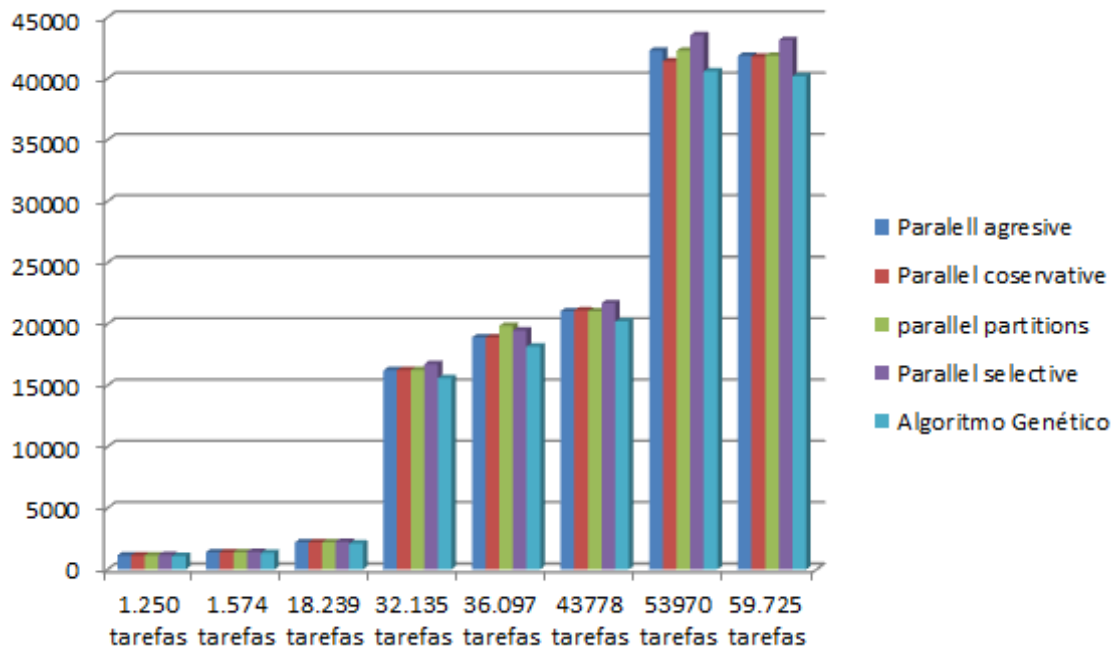

Figura 9 - Gráfico de valores do tempo total da simulação do *GRIDSIM* variando as tarefas.

Existem dois parâmetros cruciais a serem selecionados, a taxa de *crossover* e a taxa de mutação. Experimentos foram realizados para selecionar os valores adequados. Dez simulações com o arquivo blue_sdsc_01.txt com 50 tarefas foram executadas. Uma população inicial de 100 cromossomos foi gerada. Um grande conjunto de valores para a taxa de *crossover* e taxa de mutação foi usado. Em cada caso, as novas gerações de população foram criadas até que a convergência fosse alcançada com um desvio padrão de $9{,}8\ x\ 10^{-7}$.

Para a taxa de mutação, experimentos mostraram que o valor 0,01 vai leva à convergência. Com qualquer valor maior, mesmo com uma taxa de *crossover* muito grande, a convergência pode não ser obtida. Logo, adotou-se a taxa de mutação de 0,01.

Com esse valor na taxa de mutação, experimentos para a taxa de *crossover* entre 0,70 a 0,80 foram realizados. Foi descoberto que com uma taxa de *crossover* entre 0,72 e 0,78 o mesmo valor para a função de avaliação foi obtido.

Para selecionar o número de gerações e o tamanho da população requerida para obtenção do escalonamento, experimentos foram conduzidos com uma mudança de população de tamanho de 10 a 200.

Até aqui é possível tirar duas conclusões a partir desse experimento. Primeiro, que os parâmetros escolhidos para a taxa de *crossover* e de mutação se mostram satisfatórios para a resolução do problema. Segundo, que o tamanho da população inicial de 100 aparenta funcionar bem para o modelo.

Nesse trabalho, em experimentações usando algoritmos genéticos, bons resultados foram obtidos usando dois pontos no operador de *crossover* e operadores de mutação com uma probabilidade de seleção de 1,0.

Por fim, um conjunto de 8 arquivos diferentes foram considerados. As tarefas foram mapeadas em um sistema de 64 máquinas. Os resultados apresentados na Figura 9 mostra que em todos os casos o escalonador baseado em AG é capaz de obter melhores resultados do que os algoritmos convencionais do simulador *GRIDSIM*. Isto acontece por causa da política de *First in First Out* (primeiro a entrar é o primeiro a sair) dos algoritmos convencionais. O escalonador baseado em AG considera todas as tarefas que estão sendo agendadas, como um conjunto único, e tenta encontrar uma configuração que satisfaça os objetivos, enquanto que os algoritmos convencionais inicialmente escalonam as tarefas que entraram primeiro, sem levar nada em consideração escalonando outras tarefas ao mesmo tempo. Isto leva a um desempenho inferior em relação ao escalonador baseado em algoritmo genético.

Capítulo 7 - CONCLUSÃO

Nesta dissertação foi apresentado um escalonador baseado em algoritmo genético para o simulador *GRIDSIM*. O algoritmo apresentado aqui teve um desempenho superior aos algoritmos convencionais, tanto em achar uma solução viável, quanto em tempo de processamento. Este simulador foi escolhido pelo seu baixo custo de implantação e pela sua facilidade de operação e reconfiguração em relação a uma grade real, além de apresentar outros algoritmos para serem realizadas comparações.

Para a validação e testes do escalonador, foram utilizados como estudo de caso, dois cenários, no primeiro cenário foi utilizado um único arquivo *SWF* variando a quantidade de máquinas a cada simulação, no outro cenário a quantidade de máquinas era fixa variando os arquivos *SWF* com a quantidade de tarefas a cada simulação.

O escalonador também é capaz de alocar múltiplas tarefas com restrições arbitrárias e tempos de execução diferentes, satisfazendo múltiplos objetivos. Ele foi testado com mais de 10.000 tarefas, o qual é bastante significativo para um escalonador realizar de uma única vez. A taxa de *crossover*, o índice de mutação e o tamanho da população que possa ser apropriado para o escalonador, foram levados em consideração nesse trabalho. A simulação de múltiplas tarefas com características realistas e com a função de avaliação apropriada para o escalonamento de grades apresentaram bons resultados.

O desempenho superior do escalonador usando a política de algoritmo genético se dá pelo fato dele agendar um recurso para uma tarefa, levando em consideração muito mais características do que os outros algoritmos comparados, o que deixa várias perspectivas para trabalhos futuros. Os resultados podem ser usados para desenvolver um grande banco de dados com os melhores tempos de conclusão das simulações e tempos mínimos e máximos de nós ociosos. Essa base de dados pode ser útil para os pesquisadores realizarem diversas comparações entre as características de novos algoritmos de escalonamento.

Sendo assim, esta é uma inciativa que serve de base para a construção de aplicações mais elaboradas que irão utilizar os recursos disponíveis na grade de forma mais eficiente, como capacidade de processamento, memória e economia de energia.

Outras características da grade podem ser apreciadas em trabalhos futuros, pretende-se desenvolver um módulo de consumo de energia para o *GRIDSIM*. Com isso, descobrir a eficiência dos algoritmos em termos de consumo de energia e comparar com o algoritmo genético para criar estratégias de economia de energia.

Com os resultados positivos obtidos através de metaheurística, pretende-se desenvolver outras metaheuristicas e como é possível a interação entre elas, combiná-las entre si, para se descobrir qual apresentará um resultado melhor.

Por fim estudar as características dos escalonamentos de forma mais profunda e para adicionar mais campos à função de avaliação tornando o algoritmo mais reativo e inteligente.

Capítulo 8 - REFERÊNCIAS

Abraham, A., Buyya, R. and Nath, B. (2000). Nature's Heuristics for Scheduling Jobs on Computational Grids. in Proc. of 8th IEEE International Conference on Advanced Computing and Communications (ADCOM 2000)**:** 45-52.

Aziz, A. and El-Rewini, H. (2011). "Power efficient scheduling heuristics for energy conservation in computational grids." The Journal of Supercomputing **57**(1): 65-80.

Berman, F., Fox, G. C. and Hey, A. J. G. (2003). Grid computing : making the global infrastructure a reality. Chichester, Wiley.

Blum, C. and Roli, A. (2003). "Metaheuristics in combinatorial optimization: Overview and conceptual comparison." ACM Comput. Surv. **35**(3): 268-308.

Buckley, P. (2010). The Rough Guide to cloud computing. London, Rough Guides : Distributed by the Penguin Group.

Buyya, R. (2012). "http://www.buyya.com/gridsim/." Retrieved 2012, 2012.

Campello, R. E. and Maculan, N. (1994). Algorítmo e Heurísticas, EDUFF - Editora Universitária.

Chapin, S. J., Cirne, W., Feitelson, D. G., Jones, J. P., Leutenegger, S. T., Schwiegelshohn, U., Smith, W. and Talby, D. (1999). Benchmarks and Standards for the Evaluation of Parallel Job Schedulers. Proceedings of the Job Scheduling Strategies for Parallel Processing, Springer-Verlag**:** 67-90.

Chede, C. T. (2004). Grid Computing: Um Novo Paradima Computacional. Brasil, Brasport.

Cormen, T. H. (2009). Introduction to algorithms. Cambridge, Mass., MIT Press.

Cotta, C., Sevaux, M. and Sörensen, K. (2008). Adaptive and Multilevel Metaheuristics, Springer.

Doerner, K. F., Gendreau, M. and Greistorfer, P. (2007). Metaheuristics : progress in complex systems optimization. New York, Springer.

Donoso, Y. and Fabregat, R. (2007). Multi-objective optimization in computer networks using metaheuristics. Boca Raton, Auerbach Publications.

Feitelson, D. G. (2012). "http://www.cs.huji.ac.il/labs/parallel/workload/swf.html."

Feitelson, D. G. and Weil, A. M. a. (1998). "Utilization and Predictability in Scheduling the IBM SP2 with Backfilling." International Parallel Processing Symposium: 4.

Foster, I. and Kesselman, C. (2004). The grid : blueprint for a new computing infrastructure. Amsterdam ; Boston, Morgan Kaufmann.

FOSTER, I., KESSELMAN, C., NICK, J. and TUECKE, S. (2002). "The physiology of the grid: An open grid services architecture for distributed systems integration." Open Grid Service Infrastructure WG, Global Grid Forum: 1-5.

Garey, M. R. and Johnson, D. S. (1979). Computers and intractability : a guide to the theory of NP-completeness. San Francisco, W. H. Freeman.

Gentzsch, W., Grandinetti, L. and Joubert, G. R. (2009). High speed and large scale scientific computing. Amsterdam ; Washington, DC, IOS Press.

Gonzalez, T. F. (2007). Handbook of approximation algorithms and metaheuristics. Boca Raton, Chapman & Hall/CRC.

Lawson, B. G. and Smirni, E. (2002). "Multiple-Queue Backfilling Scheduling with Priorities and Reservations for Parallel Systems." Workshop on Job Scheduling Strategies for Parallel Processing: 25.

LINDEN, R. (2012). Algoritmos Genéticos (2a edição), BRASPORT.

Luke, S. (2011). Essentials of Metaheuristics. Department of Computer Science, George Mason University, Lulu.

Michalewicz, Z. and Fogel, D. B. (2004). How to solve it : modern heuristics. Berlin ; New York, Springer.

Mu'alem, A. W. and Feitelson, D. G. (2001). "Utilization, Predictability, Workloads, and User Runtime Estimates in Scheduling the IBM SP2 with Backfilling." IEEE Transactions on Parallel and Distributed Systems: 14.

Papazachos, Z. C. and Karatza, H. D. (2010). "Performance evaluation of bag of gangs scheduling in a heterogeneous distributed system." The Journal of Systems and Software **83**: 1346-1354.

Papazachos, Z. C. and Karatza, H. D. (2011). "Gang scheduling in multi-core clusters implementing migrations." Future Generation Computer Systems - The International Journal of Grid Computing and eScience **27**: 13.

Riel, A. J. (1996). Object-Oriented Design Heuristics, Addison Wesley.

Russell, S. J., Norvig, P. and Davis, E. (2010). Artificial intelligence : a modern approach. Upper Saddle River, Prentice Hall.
Srinivasan, S., Kettimuthu, R., Subramani, V. and Sadayappan, P. (2002). "Selective Reservation Strategies for Backfill Job Scheduling." Workshop on Job Scheduling Strategies for Parallel Processing: 16.

Venugopal, S. and Buyya, R. (2006). A Set Coverage-based Mapping Heuristic for Scheduling Distributed Data-Intensive Applications on Global Grids. Proceedings of the 7th IEEE/ACM International Conference on Grid Computing, IEEE Computer Society**:** 238-245.

Wagner, S. and Kronberger, G. (2011). Algorithm and experiment design with heuristiclab: an open source optimization environment for research and education. Proceedings of the 13th annual conference companion on Genetic and evolutionary computation. Dublin, Ireland, ACM**:** 1411-1438.

Woodward, J. R. and Swan, J. (2011). Automatically designing selection heuristics. Proceedings of the 13th annual conference companion on Genetic and evolutionary computation. Dublin, Ireland, ACM**:** 583-590.

Xhafa, F. (2009). Parallel programming, models, and applications in grid and p2p systems. Amsterdam ; Washington, DC, IOS Press.

Xhafa, F. and Abraham, A. (2009). A Compendium of Heuristic Methods for Scheduling in Computational Grids Intelligent Data Engineering and Automated Learning - IDEAL 2009. E. Corchado and H. Yin, Springer Berlin / Heidelberg. **5788:** 751-758.

Xhafa, F., Kołodziej, J. and Bogdan´ski, M. (2010). A Web Interface for Meta-Heuristics Based Grid Schedulers. International Conference on P2P, Parallel, Grid, Cloud and Internet Computing, IEEE Computer Society**:** 6.