Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Estudo Exploratório do Uso de Algoritmos Genéticos no Gerenciamento de Tarefas e Recursos em Cloud Computing

Arthur J. R. Farias

Monografia apresentada como requisito parcial
para conclusão do Curso de Engenharia da Computação

Orientador
Prof.ª Dr.ª Genaína Nunes Rodrigues

Brasília
2015

Universidade de Brasília — UnB
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Curso de Engenharia da Computação

Coordenador: Prof. Dr. Ricardo Zelenovsky

Banca examinadora composta por:

Prof.ª Dr.ª Genaína Nunes Rodrigues (Orientador) — CIC/UnB
Prof.ª Dr.ª Alba Cristina M. A. de Melo — CIC/UnB
Prof.ª Dr.ª Célia Ghedini Ralha — CIC/UnB

CIP — Catalogação Internacional na Publicação

Farias, Arthur J. R..

Estudo Exploratório do Uso de Algoritmos Genéticos no Gerenciamento de Tarefas e Recursos em Cloud Computing / Arthur J. R. Farias. Brasília : UnB, 2015.
78 p. : il. ; 29,5 cm.

Monografia (Graduação) — Universidade de Brasília, Brasília, 2015.

1. alocação de recursos, 2. computação em nuvem, 3. algoritmo genético

CDU 004

Endereço: Universidade de Brasília
Campus Universitário Darcy Ribeiro — Asa Norte
CEP 70910-900
Brasília–DF — Brasil

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Estudo Exploratório do Uso de Algoritmos Genéticos no Gerenciamento de Tarefas e Recursos em Cloud Computing

Arthur J. R. Farias

Monografia apresentada como requisito parcial
para conclusão do Curso de Engenharia da Computação

Prof.ª Dr.ª Genaína Nunes Rodrigues (Orientador)
CIC/UnB

Prof.ª Dr.ª Alba Cristina M. A. de Melo
CIC/UnB

Prof.ª Dr.ª Célia Ghedini Ralha
CIC/UnB

Prof. Dr. Ricardo Zelenovsky
Coordenador do Curso de Engenharia da Computação

Brasília, 21 de agosto de 2015

Dedicatória

Dedico este trabalho a você, que toma o conhecimento como ferramenta na construção de uma sociedade mais justa.

Agradecimentos

Quero agradecer, em primeiro lugar, a Deus, pela força, coragem e paciência concedida durante toda esta longa caminhada. Agradeço aos meus pais e minha irmã, que, com carinho e apoio incondicional, não pouparam esforços para que eu chegasse até aqui. Aos meus amigos, pelo incentivo e pelo companheirismo ao longo desses anos. Agradeço também a todos os professores que me acompanharam durante a graduação, em especial a minha professora orientadora, cujo auxílio foi indispensável para a realização deste trabalho.

Resumo

Com a notável expansão dos serviços em nuvem, a introdução de módulos de alocação mais eficientes se faz cada vez mais necessária. O gasto energético é a principal fonte de custo para as empresas provedoras, diretamente causado pela má utilização de recursos de hardware. A fim de prover maior flexibilidade na alocação de tarefas e um gerenciamento de recursos mais efetivo, analisaremos a viabilidade de se utilizar uma abordagem evolutiva para obter soluções com maior qualidade, assim como garantir maior lucro para a companhia provedora, satisfazendo ambos os lados da negociação. O uso de heurísticas evolutivas como otimizadores vem ganhando notoriedade no meio acadêmico, da mesma forma, deixaremos a nossa contribuição com a implementação e análise de um algoritmo genético básico funcionando como alocador, explicitando também as vantagens e desvantagens encontradas com o uso dessa abordagem. Veremos que os resultados encontrados são bastante animadores, entretanto, muito deve ser feito para que o módulo proposto possa ser aplicado em um sistema real. Uma ideia de trabalho futuro é combinar abordagens evolutivas com métodos de busca convencionais, assim como otimizar alguns parâmetros da própria implementação do algoritmo genético em busca de melhores resultados de otimização.

Palavras-chave: alocação de recursos, computação em nuvem, algoritmo genético

Abstract

With the undeniable ascension of cloud services, the introduction of efficient scheduling modules in this kind of system is more necessary than ever. Energy consumption is the main source of cost to the provider companies, partially caused by the poor administration of hardware resources. In order to offer greater flexibility to the task scheduling process and more effectiveness in resource management, we will analyze the feasibility of using an evolutionary approach to obtain solutions with greater quality, increasing the provider company's profit, satisfying the interests of both sides. Evolutionary heuristics are gaining notoriety in the academic field as alternative solutions to optimization problems. Therefore, this work presents our contribution, which consists of the implementation and analysis of a basic genetic algorithm that works as a scheduler, also explaining the advantages and disadvantages encountered with the use of this approach. Later on, we show that the results of this research are quite encouraging, however, there is still much to be done as the main objective of almost every research is to apply the proposed method in a real system. A possible subject for a future work could be the hybridization of the genetic approach with a conventional search algorithm. Optimizing the genetic operators and some other implementation parameters are also in the plans. These actions shall improve the overall performance of the algorithm as well, consequently, returning better solutions and making the method more dependable.

Keywords: resource scheduling, cloud computing, genetic algorithm

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

ACO Ant Colony Optimization. 14

AG Algoritmo Genético. xii, 14, 16, 20, 24, 29, 33, 34, 41, 44, 46–49, 59, 62

DaaS Data as a Service. 8

EAR Estratégia de Alocação de Recursos. 12

EC2 Amazon Elastic Compute Cloud. xii, 9, 31, 32

FF Fitness Function. 17

FS Fitness Score. 17, 37, 43, 48, 53, 54, 60

GCE Google Compute Engine. xii, 31, 32

IaaS Infraestructure as a Service. 8–10, 29, 32

MOGA Multi Objective Genetic Algorithm. x, 26, 27

MTTF Mean Time To Failure. 53, 59

PaaS Platform as a Service. 8

PSO Particle Swarm Optimization. 14

QoS Quality-of-Service. vi, vii, 1, 4, 5, 8–10, 29, 36, 62

SA Simulated Annealing. 14

SaaS Software as a Service. 7, 8, 10

SLA Service Level Agreement. 1, 3, 10, 13, 26, 29, 33

XaaS Everything as a Service. 8

Capítulo 1

Introdução

Nas últimas décadas, a computação em nuvem deixou de ser uma tecnologia que visava meramente praticidade para o usuário e se firmou como um paradigma necessário para o desenvolvimento de diversas áreas da sociedade. A notoriedade desse tipo de virtualização, seja de serviço ou plataforma, atrai interesse tanto do meio acadêmico, que geralmente precisa da robustez e potencial das *Clouds* para processar grandes quantidades de dados para suas pesquisas, quanto do segmento industrial e comercial, objetos de estudo deste trabalho, os quais são os maiores interessados no uso de infraestruturas prontas para hospedagem de serviços, como *e-commerce*, processamento ou armazenamento de seus dados.

A escalabilidade unida à facilidade de se prover dinamicamente recursos sob demanda, são dois fatores do modelo de *Cloud Computing* que indicam a permanência desse paradigma no mercado por ainda bastante tempo. Apesar disso, o estado da arte de tal tecnologia ainda tem pontos a serem estudados e desenvolvidos com maior atenção, entre eles podemos incluir o gerenciamento de recursos e a alocação de tarefas. Ambos os tópicos estão plenamente relacionados ao lucro da companhia provedora e impactam diretamente a *Quality-of-Service (QoS)*, qualidade do serviço prestado, dessa forma, é essencial focar no desenvolvimento de ferramentas que otimizem essas funções.

Assim como qualquer serviço prestado, a utilização de *Clouds* dispõe de uma forma de contrato entre usuário e provedor da nuvem, o chamado *Service Level Agreement (SLA)*. O acordo de nível de serviço basicamente estabelece uma lista de recursos tecnológicos de que o usuário precisa (memória RAM, disco, processamento, etc.) e um valor a ser pago, geralmente pela hora de serviço. O provedor recebe uma solicitação e designa a configuração mais adequada de computadores ou máquinas virtuais para processar as tarefas, e a partir daí retirar o seu lucro.

A partir dessas ideias, alguns questionamentos surgem e as intenções deste trabalho começam a ser delineadas: O que significa para um provedor designar a melhor forma-

ção para uma tarefa? Em qual parâmetro ele estaria interessado? Qual o reflexo dessa escolha para o usuário final? Esse usuário está buscando custo-benefício, disponibilidade, desempenho? É óbvio que o gasto energético é um fator preponderante para o provedor do serviço retirar seu lucro, mas como a economia desse recurso influencia na qualidade final do serviço?

Com a clara expansão da *Cloud Computing*, esse tipo de serviço deverá ser cada vez mais acessível para o usuário, que poderá ou não saber dos requisitos técnicos necessários para a realização do seu objetivo. Por outro lado, quem provê a nuvem precisa ser flexível o suficiente para atender diferentes tipos de perfis, sem ter seus lucros negativamente afetados. A abordagem utilizada atualmente não parece ser eficaz frente a esse tipo de variação de contexto, assim, o presente trabalho propõe solucionar algumas destas questões, realizando um estudo exploratório de heurísticas evolutivas, como algoritmos genéticos, a fim de decidir se é uma alternativa válida no aperfeiçoamento do gerenciamento de recursos e alocação de tarefas.

1.1 Definição do Problema

Um bom alocador de tarefas em um sistema de *cloud computing* visa o melhor *trade-off* entre tempo de execução e um bom gerenciamento dos recursos. Nas arquiteturas atuais, vemos que os escalonadores de recursos tem uma latência bem aceitável em relação a designação de uma tarefa para um conjunto de máquinas, entretanto, o desperdício de recursos ainda é um grande problema para as companhias provedoras. O método de busca baseado em *constraints* [29] é um bom exemplo desse paradoxo, uma vez que ao se estabelecer uma configuração mínima, o sistema pode retornar uma lista de possíveis soluções, sem a mínima preocupação com os parâmetros que melhor atenderão o cumprimento do objetivo proposto pelo usuário.

Custo-benefício, desempenho e disponibilidade, são apenas alguns dos requisitos que o usuário pode estar interessado ao submeter uma aplicação para uma *cloud*. A viabilidade do cumprimento de uma tarefa é a questão mais básica que deve ser resolvida. Contudo, para maior qualidade de serviço prestado, requeremos também que os parâmetros secundários sejam bem analisados na hora da alocação de recursos. Os modelos atuais não estão preparados para promover esse tipo de flexibilidade, principalmente em tempo de execução. A verificação estática e a intolerância à variação de contexto são comportamentos que devem ser vencidos para que os sistemas em nuvem atinjam um novo nível de prestação de serviço e interação com o usuário.

A área de algoritmos evolucionários vem ganhando atenção crescente nos últimos anos, apresentando soluções para os mais diversos tipos de problemas computacionais [34]. A

flexibilidade em relação ao contexto e o alto custo-benefício do tempo de execução relacionado às soluções encontradas fazem dessa abordagem, uma ferramenta poderosa e com bastante potencial para substituir o processo de alocação de recursos em nuvem. Como citado anteriormente, dada uma configuração válida pelo usuário, o principal objetivo do provedor é alocar recursos de uma máquina para a execução de uma tarefa, de forma que tanto requisitos quantitativos quanto qualitativos sejam atendidos da melhor maneira possível, garantindo assim, o maior lucro da provedora e satisfação do cliente com o cumprimento do SLA e com alta qualidade de serviço. Partindo desse princípio, chegamos a nossa primeira questão de pesquisa:

Questão de pesquisa 1 (QP1): Dada uma configuração mínima definida por um usuário, a ser processada por uma nuvem, é vantajoso utilizar uma abordagem evolucionária na alocação de tarefas e gerenciamento de recursos frente à busca baseada em *constraints*?

Abordagens evolucionárias, como todas as outras, têm pontos positivos e negativos, um dos fenômenos que estudaremos neste trabalho é a sua característica não determinística. O objetivo de se utilizar um algoritmo genético em *Cloud Computing* é oferecer uma solução ótima, ou ao menos próxima à ótima em um tempo aceitável. Entretanto, o não determinismo pode ser um grande inconveniente para provedores e usuários. No modo de alocação usual, a probabilidade de se retornar uma solução se a mesma existir é de 100%, diferentemente do que se obtêm quando utilizamos técnicas evolucionarias. Este fato nos leva ao segundo questionamento:

Questão de pesquisa 2 (QP2): Em que grau o comportamento não determinístico de um algoritmo evolucionário pode afetar o desempenho de um alocador de tarefas ou gerenciador de recursos em um sistema de *Cloud Computing*?

1.2 Solução Proposta

Buscando uma alternativa para os modelos atuais de alocação de tarefas e gerenciamento de recursos, este trabalho propõe avaliar a utilização de uma abordagem evolucionária no contexto de qualidade de serviço em nuvens computacionais. No lugar de uma busca definida por *constraints*, desenvolveremos um módulo de planejamento baseado em algoritmo genético para alocar tarefas, oferecendo mais versatilidade aos modos de operação do sistema.

Algoritmos genéticos possuem um bom *trade-off* entre qualidade de solução apresentada e tempo de execução, dessa forma, tentaremos criar um gerenciador de recursos mais

"inteligente" e mais ciente dos parâmetros a serem analisados. A proposta aqui é buscar uma intersecção entre o objetivo do usuário e os recursos disponíveis para cumprir esse objetivo, combinando-os de tal forma a satisfazer o cliente, fazendo o mesmo pagar menos por um serviço de mais qualidade, assim como evitar que a provedora desperdice recursos e obtenha maior lucro. De maneira geral, a alocação de recursos e gerenciamento de tarefas se dará da seguinte forma:

Dados:

- Um modo de operação no qual o algoritmo irá se basear para realizar a alocação (Custo-benefício / Disponibilidade / Desempenho);
- Uma ou mais tarefas a serem executadas contendo os requisitos mínimos para seu processamento;
- Um conjunto de instâncias de máquinas, cada uma com sua configuração de recursos específica;
- Um nível exigido de qualidade para as soluções (Alta / Média / Baixa).

O sistema deverá encontrar uma combinação tarefa-instância, respeitando as restrições de recursos, o modo de operação e o nível de QoS escolhidos.

O comportamento probabilístico da abordagem adotada é uma das questões que serão estudadas nesta monografia. Métricas serão definidas através de análises quantitativas dos resultados, para se extrair um padrão de confiabilidade para as soluções. Caso os resultados sejam insatisfatórios, uma solução será proposta para minimizar essa deficiência.

Finalmente, o módulo de planejamento criado será aplicado em diferentes situações, preferencialmente respeitando os ambientes reais. O objetivo é variar o número de instâncias de nuvens disponíveis, número e conteúdo de aplicações submetidas, parâmetros do próprio algoritmo, tal como taxa de *crossover*, taxa de mutação e outros conceitos que serão explicados mais adiante.

1.3 Avaliação

Para as arquiteturas atuais, o conceito de eficiência está ligado diretamente ao tempo de resposta do escalonador e ao lucro obtido pelas provedoras baseado no custo energético, entretanto, ao se ignorar os parâmetros mais básicos, não podemos esperar uma solução adequada em relação à qualidade. A ideia principal é avaliar o nosso alocador com base em três diferentes modos de operação:

- **Custo-benefício**: Como gerenciador de recursos, qual instância de uma nuvem deverá ser escolhida para que a aplicação seja realizada com menor desperdício de recurso e menor custo para o usuário.
- **Disponibilidade**: Como gerenciador de recursos, qual instância de uma nuvem deverá ser escolhida de modo que a tarefa seja realizada de modo confiável dentro de uma taxa de disponibilidade pré-estabelecida.
- **Desempenho**: Como gerenciador de recursos, qual instância de uma nuvem deverá ser escolhida para que a tarefa do usuário seja executada de modo mais rápido, respeitando um orçamento pré-estabelecido.

Para cada modo listado acima, serão realizados testes de qualidade de solução e testes de desempenho (tempo de execução), extraindo análises estatísticas para modelagem de comportamento. Contudo, o foco desta avaliação está mais voltado para a obtenção de um gerenciador mais flexível em relação ao contexto, ou seja, respostas de maior qualidade dependentes da preferência do usuário.

1.4 Contribuições

Esta seção apresenta as contribuições práticas deste trabalho.

1. Módulo para alocação de tarefas e gerenciamento de recursos baseado em algoritmo genético.
    - Definição dos recursos de hardware a serem analisados;
    - Explicitação das métricas que formam o parâmetro de QoS;
    - Implementação em C++ do módulo gerenciador de recursos.
2. Análise quantitativa dos resultados.
    - Vantagens e desvantagens da utilização de uma abordagem evolucionária;
    - Estudo de viabilidade.

1.5 Organização do Trabalho

Esta monografia é organizada da seguinte maneira:

- **Capítulo 2**: Base teórica que fundamenta o entendimento da proposta e das metodologias apresentadas neste trabalho;

- **Capítulo 3**: Referência e breve descrição de alguns trabalhos relevantes na área, e como suas contribuições se aplicam neste projeto;
- **Capítulo 4**: Descrição da metodologia adotada no desenvolvimento, assim como suas métricas e conceitos relacionados;
- **Capítulo 5**: Apresentação e discussão dos resultados obtidos;
- **Capítulo 6**: Considerações finais sobre a proposta apresentada e trabalhos futuros.

Capítulo 2

Referencial Teórico

Veremos neste capítulo uma breve ambientação em determinados temas, as quais servirão de base para o entendimento do trabalho e suas particularidades teóricas.

2.1 Computação em Nuvem

Seguindo a definição de *Cloud Computing* proposta por Foster, dizemos que Computação em Nuvem é um paradigma computacional amplamente distribuído, onde armazenamento, plataformas e serviços, virtualizados e dinamicamente escaláveis são ofertados sob demanda para consumidores externos através da internet [16]. Os serviços mais comumente oferecidos são divididos em três diferentes níveis: IaaS, PaaS e SaaS, ainda assim, algumas provedoras oferecem mais de um nível. *Infraestructure as a Service (IaaS)* visa o provisionamento de hardware, software e equipamentos para diferentes ambientes, onde a cobrança é baseada na utilização do recurso. *Platform as a Service (PaaS)* oferece um ambiente com alto nível de integração para construir, testar e implantar aplicações personalizadas. *Software as a Service (SaaS)* entregam softwares de propósitos específicos para serem acessados remotamente pelos usuários através da internet, também regulado pelo modelo de cobrança baseado em utilização do recurso [16].

Subindo um nível de abstração, a utilização de serviços em nuvem pelo público é feita da maneira conhecida como pagamento por utilização, ou seja, o usuário define em contrato quais os recursos deseja utilizar e paga por aqueles recursos por um determinado tempo, chamamos isso de *Nuvem Pública*, pois o serviço está disponível para uso público. Encontramos atualmente diversas provedoras de *Clouds* Públicas em ascensão e muitas outras já consolidadas no mercado, Google Apps [4], Azure [6] e Amazon Web Services [2] são exemplos desse modelo de negócio que obtiveram bastante sucesso nos últimos anos. Por outro lado, chamamos de *Nuvem Privada* aquela cujo serviço não é disponível ao público, mas sim utilizados internamente nas organizações. Preocupações com segu-

rança da informação e questões legais geralmente são menores devido à característica mais confinada desse tipo de nuvem. Temos ainda um terceiro tipo de ambiente, as *Nuvens Híbridas*, as quais são formadas tanto por provedores internos quanto externos.

Alguns atributos de computação em nuvem tornam esse modelo extremamente atrativo ao público. Primeiramente, podemos citar a característica de serviço por demanda, onde as empresas que tem necessidade por algum recurso de TI não precisam mais se preocupar com a logística e custo da posse de um centro de dados próprio, uma vez que podem optar por um serviço virtual barato e que provê a infraestrutura necessária. Ligado a isso, a terceirização e virtualização da parte informatizada da empresa facilita grandemente a vida dos gestores, tirando dos mesmos o fardo de saber os detalhes técnicos atrelados ao sistema que se deseja implementar. Tradicionalmente, além de requerer um extenso trabalho de pesquisa, implantar um servidor próprio consome muito tempo entre escolha de hardware, aprovação de orçamento e instalação da plataforma, enquanto em serviços em nuvem, o usuário interage com o sistema com tranquilidade via interfaces amigáveis para escolher uma configuração adequada para a realização de sua tarefa. Um outro aspecto crucial é a escalabilidade aliada a um nível de qualidade preestabelecido, queremos dizer com isso que o usuário do serviço não está limitado com uma quantidade fixa de recursos, podendo aumentar ou diminuir o *workload* dependendo de suas necessidades, tudo isso sem diminuir a qualidade do serviço prestado, já que o mesmo é acordado entre cliente e provedor no contrato. Veremos mais sobre as definições de qualidade de serviço e contrato formal nas Seções 2.1.1 e 2.1.2.

Com o passar dos anos, o avanço de novas tecnologias permitiu a concepção e o desenvolvimento de ofertas alternativas em *Cloud Computing*. Largura da banda, poder de processamento e capacidade de armazenamento são exemplos de recursos que deixaram de ser tão limitadores na oferta de serviços, dessa maneira, a gama de opções que uma nuvem poderia prover foi extrapolada, surgindo campos como *Platform as a Service (PaaS)*, *Data as a Service (DaaS)*, *Infraestructure as a Service (IaaS)*, e mais tarde a generalização *Everything as a Service (XaaS)*.

A Figura 2.1 mostra a arquitetura básica de *cloud computing*. Neste trabalho, o foco será na base dessa arquitetura, ou seja, na IaaS. Como o nome sugere, o modelo baseia-se na disposição de infraestrutura como um serviço, assim, recursos de hardware como CPUs, memória, rede serão alocados por essa camada quando determinada aplicação submetida pelo usuário requerer recursos para cumprimento de uma tarefa. Uma grande vantagem na utilização desse esquema está na flexibilidade, seja em relação a configurações mais modernas ou simplesmente em relação à forma de pagamento baseada em uso, tal facilidade de utilização torna a escalabilidade do sistema rápida e natural para o consumidor [8]

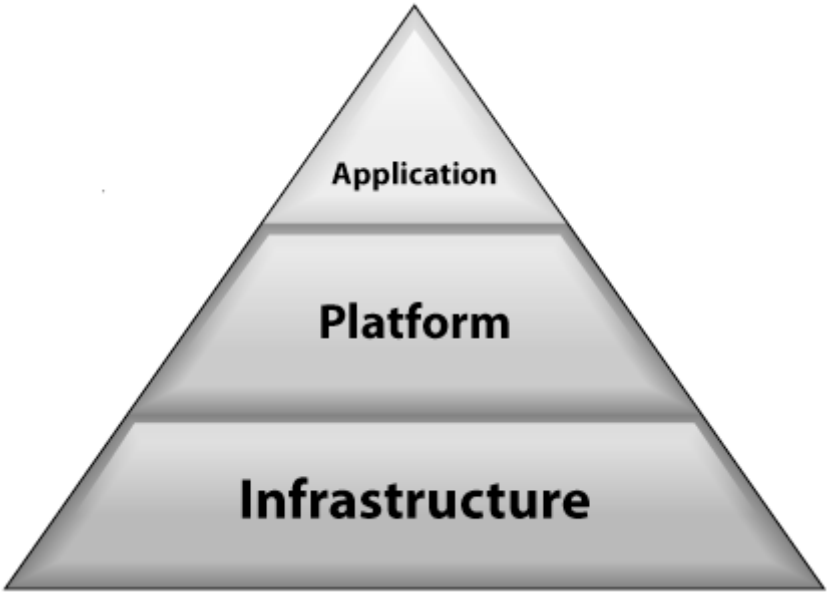

Figura 2.1: Arquitetura básica de *Cloud Computing*

2.1.1 *Quality-of-Service*

A evolução da computação em nuvem só foi possível devido ao desenvolvimento dos diversos módulos básicos que os compõem. Ainda assim, algumas áreas em *cloud computing* ainda oferecem bastante desafio para a comunidade, como é o caso do gerenciamento de *QoS*. Para fins deste trabalho, na computação em nuvem essa terminologia está ligada aos níveis de desempenho, confiabilidade e disponibilidade oferecida pelo sistema que provê o serviço. QoS é um dos parâmetro disponível para metrificar qualidade de uma solução, e é imprescindível para direcionar o usuário a fazer uma boa escolha de provedor. Por outro lado, como operador do sistema, espera-se encontrar o *trade-off* perfeito entre qualidade de serviço e os custos relacionados. Entretanto, a dualidade entre os níveis de QoS a serem atingidos e as penalidades relacionadas ao não cumprimento dos mesmos, resumem bem a dificuldade de se encontrar esse *trade-off*, como geralmente podemos ver nos contratos de serviços [9].

Qualidade é um termo genérico que pode ser aplicado a qualquer tipo de serviço prestado, seja virtual ou não. É importante ressaltar que a qualidade de serviço não está sempre relacionada ao baixo custo, impedindo assim que os usuários simplesmente escolham os provedores mais baratos. Sabe-se que alguns provedores de serviços na internet cobram até 10 vezes mais que outras em troca de um serviço com mais dependabilidade e outras vantagens adicionais, como uma melhor usabilidade [25].

Sabendo disso, traçamos um paralelo com o objetivo do presente trabalho, o qual está intimamente ligado ao conceito de QoS. A fim de prover uma melhor qualidade de serviço,

um método mais eficaz de alocação tarefa-recurso é crucial, principalmente num tipo de serviço baseado na oferta de infraestrutura. Durante o trabalho de pesquisa, encontramos na literatura alguns trabalhos focados em sistemas IaaS, geralmente baseados na Amazon Elastic Compute Cloud (EC2) [9].

Como dito anteriormente, os métodos de gerenciamento de recursos em *clouds* públicas são, em termos computacionais, bem primitivos. Basicamente, são sistemas baseados em *constraints* para controle de QoS, ou seja, sistemas que mapeiam alguns parâmetros básicos de hardware e reagem quando esses parâmetros violam algum limite pré-estabelecido. Assim sendo, essa área de pesquisa se mostra ainda inexplorada e com bastante abertura para se adotar abordagens mais eficientes. Agora que temos noção dessas limitações, buscaremos através desta monografia apresentar fundamentos para se chegar a resultados mais flexíveis e efetivos.

2.1.2 *Service Level Agreement*

Um SLA, ou Acordo de Nível de Serviço, nada mais é que um contrato estabelecido entre cliente e provedor, especificando os detalhes técnicos do serviço oferecido, limitações acordadas de desempenho, disponibilidade e confiabilidade, assim como as responsabilidades de ambas as partes e como violações devem ser tratadas [13]. Ainda que se trate de um acordo comercial, o contrato precisa explicitar todas as questões técnicas relacionadas ao modelo de negócio. Podemos citar como exemplo, um contrato de adesão de serviço de banda larga, o qual conta com o tempo que o serviço deve ficar disponível para uso, velocidade média de conexão, custo monetário por unidade de tempo, tempo médio para restabelecimento do sinal em caso de falha, etc. Entretanto, é muito raro definir uma cobertura de 100% de todos os cenários que podem ocorrer durante a utilização do serviço. Essa tarefa se torna ainda mais difícil em sistemas complexos como os de *cloud computing*, e sabendo que o cumprimento do SLA está diretamente ligado à qualidade de serviço, devemos dar atenção extra à sua formulação.

Com o objetivo de validar a metodologia adotada neste trabalho, um SLA para teste foi criado com bastante simplicidade, uma vez que o foco atual é otimizar a alocação de recursos para execução de uma tarefa, em um modelo de nuvem IaaS, priorizamos apenas alguns parâmetros de hardware para o cálculo de QoS. Entre eles estão parâmetros de processamento, memória, custo e nível de disponibilidade.

2.1.3 Tarefas e Recursos

Caracterização de Tarefas e Recursos

Como forma de generalizar nossa pesquisa, adotaremos os conceitos de **tarefas** e **recursos** como pilares do trabalho. Veremos abaixo como são definidos esses dois parâmetros e como serão aplicados nesta dissertação.

a) Tarefas

Em sistemas SaaS, o termo "aplicação" é mais comumente encontrado.Apesar de uma aplicação poder ser dividida em múltiplas partes, a fim de facilitar o entendimento, definiremos aqui que uma tarefa ou *job* é um bloco indivisível submetido à nuvem pelo usuário. As tarefas submetidas pelo usuário, no entanto, podem ser independentes ou comunicantes entre si. No caso de tarefas independentes, as mesmas podem ser alocadas individualmente pelo planejador, em forma de fila, sem causar nenhuma queda de eficiência no sistema. Por outro lado, em tarefas comunicantes, assumimos que seus processamentos devem ser paralelizados, havendo algum tipo de dependência entre elas, seja de tempo ou dados, que as forçam a serem alocadas simultaneamente, e são nessas situações que a proposta de alocação multiobjetivo usando algoritmos genéticos será empregada. Na Figura 2.2, por exemplo, percebemos que a tarefa 1 e 5 podem ser alocadas separadamente. Entretanto, as tarefas 2, 3 e 4 podem ser alocadas simultaneamente, assim como as tarefas 6 e 7.

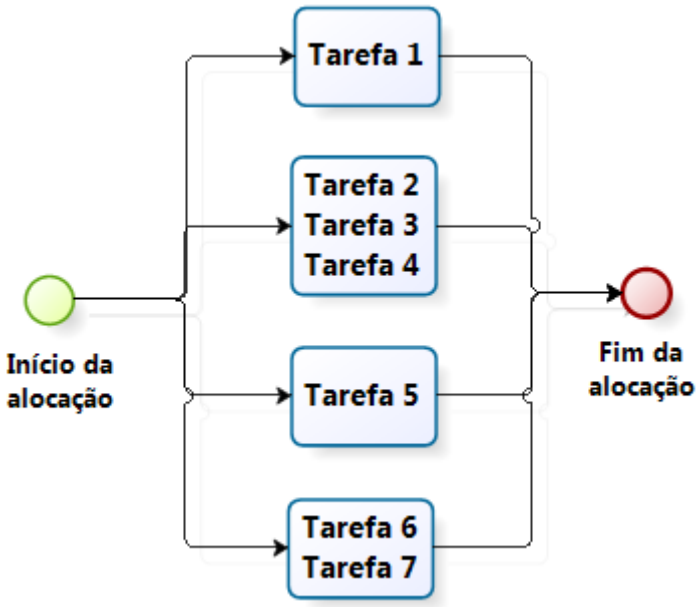

Figura 2.2: Exemplo de um conjunto de tarefas a serem alocadas respeitando suas dependências

Em sistemas de *cloud* que oferecem infraestrutura como serviço, as tarefas têm um objetivo pré definido, e para cumpri-lo, consomem um ou mais recursos em um determinado intervalo. O tempo e custo necessários para que esta tarefa seja processada dependem principalmente dos recursos alocados para sua execução [33].

As tarefas podem ser mensuradas com base em inúmeros parâmetros tais como duração, custo e consumo de recursos. Para facilitar essas medidas e torná-las aplicáveis em um alocador, assumimos que uma tarefa é dada por um conjunto de requisitos de hardware necessários para o seu processamento (núcleos de processamento, memória RAM, banda, etc.) e um custo máximo que o usuário está disposto a pagar. Além disso, podemos classificar tarefas de acordo com os seus objetivos como armazenamento ou processamento, por exemplo.

b) Recursos

O segundo conceito fundamental com o qual trabalharemos é o recurso. Os recursos geralmente são divididos em renováveis, os quais estão disponíveis a cada período sem se esgotarem, e não-renováveis, que se extinguem a medida que são utilizados. Como estamos falando de recursos computacionais, em sua maioria nos referimos aos recursos que estão disponíveis ao fim de cada período [33].

Assim como as tarefas, os recursos podem variar em capacidade, custo e outros parâmetros, e é essa plasticidade que torna possível a oferta de infraestrutura por demanda. Dessa maneira, o usuário pode definir sua tarefa de acordo com as diferentes configurações disponíveis. A disponibilidade é outro fator que pode variar, por isso é imprescindível que se faça algum tipo de monitoramento, determinando um grau de disponibilidade e confiabilidade para os recursos. Restrições de tempo e de requisitos ainda podem ser estabelecidos. Veremos como essas restrições influenciarão o desempenho da solução proposta no Capítulo 4.

Gerenciamento de Recursos

O gerenciamento de recursos e alocação de tarefas por parte do provedor da nuvem são operações fundamentais na garantia de um serviço de qualidade. Deve-se adotar uma estratégia de alocação eficiente o bastante para disponibilizar os recursos que a tarefa necessita e evitar *starvation*. Por outro lado, tais recursos não podem ser ofertados a revelia, uma vez que o desperdício dos mesmos é um problema ainda mais grave, impactando diretamente no lucro da provedora. Em suma, uma boa Estratégia de Alocação de Recursos (EAR) deve analisar o tipo e a quantidade de recursos requeridos para a realização de cada tarefa, assim como a ordem e o tempo de alocação, os quais são muito

importantes. Veremos mais adiante alguns dos algoritmos mais utilizados para solucionar essas questões.

Segundo Vinothina et al. [32], um bom alocador deveria evitar os seguintes comportamentos:

- **Contenção de recursos:** Situação em que duas tarefas tentam acessar o mesmo recurso ao mesmo tempo;
- **Escassez de recursos:** Situação em que existem recursos limitados;
- **Fragmentação de recursos:** Situação em que os recursos estão isolados, ou seja, existem recursos suficientes, entretanto não é possível alocá-los para a tarefa necessitada;
- **Sobreprovisionamento de recursos:** Situação em que a tarefa recebe mais recursos do que o requisitado;
- **Sub-provisionamento de recursos:** Situação em que a tarefa recebe menos recursos do que o requisitado.

Notamos que a questão de alocação de tarefas e gerenciamento de recursos resulta em um paradoxo interessante. Basicamente, o usuário do recurso (cliente da nuvem) tende a estimar a demanda de recursos de tal maneira a completar a tarefa o mais rapidamente possível, ocasionando um sobreprovisionamento de recursos. Por outro lado, os provedores de recursos, buscando minimizar o gasto energético e maximizar seus lucros, tendem a limitar a oferta de recursos sub-provisionando para as tarefas, podendo até causar *starvation*. Para solucionar essa questão, além de ressaltar a importância de um acordo formal (SLA), veremos no capítulo de metodologia como diferentes modos de operação podem afetar positivamente essa situação.

Métodos de Alocação de Tarefas

Assim como uma boa estratégia de gerenciamento de recursos é importante para clientes e usuários, bons algoritmos que lidem com a alocação de tarefas em *Cloud Computing* também têm papel fundamental para o cumprimento de um SLA com alto nível de qualidade. Veremos a seguir uma breve descrição de algumas abordagens já consolidadas na literatura, conteúdo necessário para fazermos uma validação ou um contraponto embasado com o algoritmo genético proposto.

- **Round Robin:** O mais simples algoritmo de alocação de tarefas. Baseia-se em reservar "fatias de tempo" de mesmo tamanho para cada tarefa e prover os recursos

nesse intervalo, quando o tempo expira, a tarefa alocada vai para o fim de uma fila circular e a próxima toma o seu lugar e começa a ser processada. É importante ressaltar que pode ou não haver algum tipo de prioridade na alocação das tarefas e o tamanho da fatia de tempo é fundamental para o funcionamento do algoritmo, uma vez que ser for muito grande ou muito pequena, assumiria a postura de outros algoritmos que serão citados mais adiante [22].

- **Min-Min:** O processo de tomada de decisão da heurística min-min baseia-se em um conjunto U de tarefas paralelas e independentes. O algoritmo forma um conjunto M de tarefas que levam, estimadamente, menos tempo para serem executadas em qualquer uma das máquinas disponíveis. Após isso, a tarefa com menor tempo de realização dentro do conjunto M de tarefas é alocada em uma máquina correspondente e o *workload* dessa máquina é atualizado. O processo é repetido até que todas as tarefas tenham sido mapeadas [20], [14].

- **Max-Min:** Como o nome sugere, esse algoritmo é bastante parecido com a heurística min-min apresentada anteriormente, diferindo apenas na segunda parte, onde a tarefa com **maior tempo estimado de resolução** será escolhida no conjunto M, conjunto esse formado por tarefas com menor tempo estimado de execução [14].

- **LJFP e SJPF:** *Longest Job to Fastest Processor* (LJFP) e *Shortest Job to Fastest Processor* (SJFP) são heurísticas geralmente implementadas em ambientes onde ha limitação de recursos, como é o caso de clouds privadas. Ambos seguem a premissa de analisar a complexidade computacional de cada tarefa no seu processo de alocação. Seguindo o mesmo processo dos algoritmos max-min e min-min, o LJFP forma um conjunto M de tarefas com maior tempo de completude e aloca-as para as x máquinas mais rápidas disponíveis. Por outro lado, no SJFP as tarefas de maior dificuldade dentro do conjunto M são alocadas para as máquinas mais rápidas disponíveis [14].

  a) LJFP (*Longest Job to Fastest Processor*) [19]:

  1. Elenca as tarefas em ordem **decrescente** de tamanho.
  2. Elenca as instâncias das nuvens em ordem decrescente de poder de processamento.
  3. Aloca as tarefas para as máquinas fazendo um mapeamento um para um das listas ordenadas.

  b) SJFP (*Shortest Job to Fastest Processor*) [19]:

  1. Elenca as tarefas em ordem **crescente** de tamanho.

2. Elenca as instâncias das nuvens em ordem decrescente de poder de processamento.
3. Aloca as tarefas para as máquinas fazendo um mapeamento um para um das listas ordenadas.

- **Suffrage:** Nessa abordagem, temos que, para cada tarefa, o primeiro e o segundo menor tempo de execução da tarefa são encontrados no primeiro passo. A diferença entre esses valores é definido como *suffrage value*. No segundo passo, a tarefa com maior valor atribuído é alocado para a máquina correspondente de menor tempo de finalização. O conceito por trás dessa metodologia é que um melhor mapeamento é feito quando se aloca uma máquina para uma tarefa que iria "sofrer" mais em termos de tempo de execução, quando comparado à mesma máquina alocada para outra tarefa qualquer [14].

Outras técnicas otimizadoras para alocadores ganharam muita força nos últimos anos, entre essas meta-heurísticas podemos citar, *Ant Colony Optimization (ACO)*, *Simulated Annealing (SA)* [30] e *Particle Swarm Optimization (PSO)*. Ainda nesse campo, encontramos também o Algoritmo Genético (AG), que por se tratar da heurística adotada neste trabalho será melhor explicado a seguir, em uma seção a parte (2.3).

2.2 Espaço de Soluções

Podemos definir que o caso tratado nesta monografia é um problema de otimização, e como tal, buscamos encontrar, em um conjunto de possíveis soluções, aquela que melhor atenda aos requerimentos, ou que ao menos esteja perto disso. Cada possível solução tem um valor de aptidão atribuído, dessa forma, o trabalho de busca baseia-se em simplesmente procurar pelo candidato com maior ou menor aptidão dentro do nosso universo de soluções.

Essa prática tem bastante potencial quando o espaço de soluções é bem definido, entretanto, não é isso que acontece geralmente, assim como também não é o caso estudado neste trabalho. Além de conhecermos apenas alguns pontos no espaço de soluções, a forma como essas soluções são vistas pelo buscador pode variar, tornando a busca por uma solução ainda mais complexa. Por exemplo, em um processo de alocação de tarefas, se o buscador está procurando uma instância de uma nuvem que processe uma tarefa de maneira mais confiável, provavelmente retornaremos um conjunto de soluções com diferentes configurações, umas com máquinas de alto poder de processamento, outras com uma alta taxa de disponibilidade, enfim, um parâmetro deve ser previamente definido, de modo que nos permita afirmar que uma instância é "melhor" que a outra. Devido a isso,

certas pesquisas se tornam bastante complicadas, a ponto de não se ter nem ao menos um ponto de partida para a busca.

2.2.1 Problemas NP

A escolha de algoritmos genéticos para a proposta de otimização deste trabalho foi baseada dentre outras coisas, na existência de outra gama de problemas. Os problemas NP (não polinomiais), são exemplos de problemas em que não conseguimos determinar uma solução em tempo polinomial deterministicamente, ou seja, problemas que são resolvidos em tempo polinomial por uma máquina de Turing não-determinística. Temos na literatura exemplos muito conhecidos desse tipo de problema, entre eles estão o isomorfismo de grafos, o problema do caixeiro-viajante e outros problemas de satisfabilidade booleana.

Ainda dentro dessa categoria, o meio acadêmico tem voltado atenção para os **problemas NP-difíceis**, problemas que devido à complexidade ou tamanho do espaço de busca são difíceis de se encontrar uma solução, além de não estarem restritos simplesmente a questões de satisfatibilidade dos problemas NP-completos. Como citado anteriormente, muitos dos problemas de alocação de tarefas são problemas de otimização, ou seja, busca-se nesse tipo de problema, minimizar uma determinada função de objetivo. Resumindo, o tema de alocação que abordamos neste trabalho é um problema de otimização NP-Completo [26] [12]. Mesmo que seja complicado encontrar uma resposta para esses problemas, a validade da mesma pode ser provada facilmente, uma vez que a solução seja encontrada. Embora seja imposta tanta dificuldade, heurísticas como as citadas anteriormente são alternativas que conseguem encontrar diferentes soluções em tempo aceitável, devido às suas características probabilísticas.

2.2.2 *Constraint Satisfaction Problem (CSP)*

De acordo com a natureza do problema apresentado nesta monografia, podemos definilo como um CSP, ou seja, problema representado por um conjunto de objetos, cujos estados devem satisfazer um determinado número de restrições. Embora existam métodos específicos para resolução desse tipo de problema, não estamos simplesmente preocupados com a busca por uma solução que satisfaça o problema, mas obter uma solução de boa qualidade em relação às demais alternativas possíveis.

Como afirmado na Seção 2.1.3, algumas meta-heurísticas vêm sendo bastante exploradas nos últimos anos como alternativa para resolver essa e outras gamas de problemas, entre elas podemos citar *Hill Climbing*, *Simulated Annealing* e Algoritmos Genéticos. Esses métodos podem ser utilizados para encontrar uma solução que satisfaça o problema, mas não necessariamente tal solução será a ótima, até porque nem em todas as ocasiões

pode-se provar qual é a melhor solução. Ainda assim, o *trade-off* entre a qualidade das respostas e o tempo, custo ou complexidade dessas heurísticas é bastante atrativo. A escolha do Algoritmo Genético entre as diferentes opções de inteligencia artificial, é influenciada pelos bons resultados que a abordagem vem apresentando nos diversos trabalhos de pesquisa realizados nos últimos anos.

2.3 Algoritmos Genéticos

Como introduzido anteriormente, um Algoritmo Genético (AG) é uma heurística exploratória bastante utilizada para encontrar soluções próximas a ótima em problemas complexos de otimização, mais precisamente problemas NP-difíceis. Da mesma forma que Darwin propôs em sua teoria evolucionista, temos aqui o conceito de um conjunto de soluções, chamados de *indivíduos* ou *cromossomos*, os quais passam por uma série de transformações em direção a uma solução aceitável. Na forma padrão, o primeiro passo é definir como uma solução será representada pelo AG, após isso, uma população de n indivíduos é gerada randomicamente e a partir dela empregamos a ideia da "sobrevivência do mais apto" . Um modelo de um algoritmo genético padrão pode ser analisado na Figura 2.3. Veremos a seguir alguns termos conhecidos da área biológica, como seleção, recombinação, reprodução, e outros conceitos que foram herdados da biologia e são aplicados de forma análoga em algoritmos genéticos. Com o auxílio dos operadores genéticos, vide Subseção 2.3.1, as características das soluções mais adequadas serão passadas para as próximas gerações [11].

```
Randomly generate or seed initial population P
Repeat
    Evaluate fitness of each individual in P
    Select parents from P according to selection mechanism
    Recombine parents to form new offspring
    Construct new population P' from parents and offspring
    Mutate P'
    P ← P'
Until Stopping Condition Reached
```

Figura 2.3: Código em alto nível de um Algoritmo Genético [23]

2.3.1 Operadores Genéticos

a) Codificação

Tomamos por codificação ou *encoding* em algoritmos genéticos, a forma que escolhemos para representar o nosso problema a nível de código. Falamos mais precisamente da

codificação dos cromossomos, representantes das possíveis soluções. Geralmente, uma sequência binária é utilizada para fazer esse mapeamento, onde partes da string conhecidas como *genes* são parâmetros de uma solução. Entretanto, esse tipo de representação não escala muito bem em problemas mais complexos, logo, cabe ao projetista do sistema encontrar uma forma de codificação que facilite o entendimento em alto nível de abstração, e ao mesmo tempo seja eficiente dentro do algoritmo proposto. Sequências de caracteres e árvores binárias são outras formas comuns de codificação [11], [23]. Veremos no capítulo de metodologia, qual abordagem foi usada neste trabalho para representar o problema de alocação de tarefas em *cloud computing*.

b) Fitness Function (FF)

Juntamente com a forma de representação do problema, o segundo aspecto crucial de qualquer algoritmo genético de otimização é a avaliação das possíveis soluções. A avaliação de cada cromossomo é baseada no valor da pontuação de aptidão, ou Fitness Score (FS), como é mais conhecida. Em linhas bem básicas, essa pontuação definirá o nível de aptidão de cada indivíduo como candidato à solução, ou seja, apontará dentre todos os indivíduos de uma população aqueles com maior qualidade de solução, os quais devem ter seus genes passados para as próximas gerações. A FS é gerada pela Função de Aptidão, ou Fitness Function (FF), que fará uma análise "semântica" de cada cromossomo e avaliará quão bom ele é como resposta para o problema proposto. Geralmente, problemas de otimização carregam métricas associadas que podem ser utilizadas como parâmetro de qualidade da solução, ou até mesmo servir de medida de comparação com outras abordagens, são nesses valores que a FF geralmente se baseia para realizar seus cálculos de aptidão.

c) Seleção

A seleção é o primeiro passo para se criar uma nova geração. Uma vez que cada cromossomo de uma população tenha um nível de aptidão anexado, é o momento de selecionar os mais aptos, ou seja, os indivíduos que passarão suas características adiante. Pensando logicamente, escolheríamos aqueles com as maiores *fitness scores*, entretanto, essa abordagem pode fazer com que o algoritmo convirja para um ótimo local muito rapidamente, deixando de explorar outras inúmeras possíveis soluções. O primeiro algoritmo genético proposto [15] nos trouxe o conceito de "aptidão proporcional", o qual é um modelo de seleção aceito e utilizado até hoje. A técnica de *"Seleção por Roleta"*, representada na Figura 2.4, é um exemplo da aplicação desse conceito, onde a probabilidade de um cromossomo ser escolhido para reprodução é proporcional à sua pontuação, deixando portanto uma chance de indivíduos pouco aptos serem escolhidos, aumentando a variabilidade genética. Ainda assim, essa técnica ainda mantém indivíduos de alta pontuação dominando o pro-

cesso nas rodadas iniciais, por isso, novas metodologias foram desenvolvidas e melhoradas para otimizar essa seleção [23]. Ranqueamento Linear [35], Elitismo [7] e Seleção por Torneio [10] são tipos de modelo que tentam elevar a eficácia do estágio de seleção em um AG.

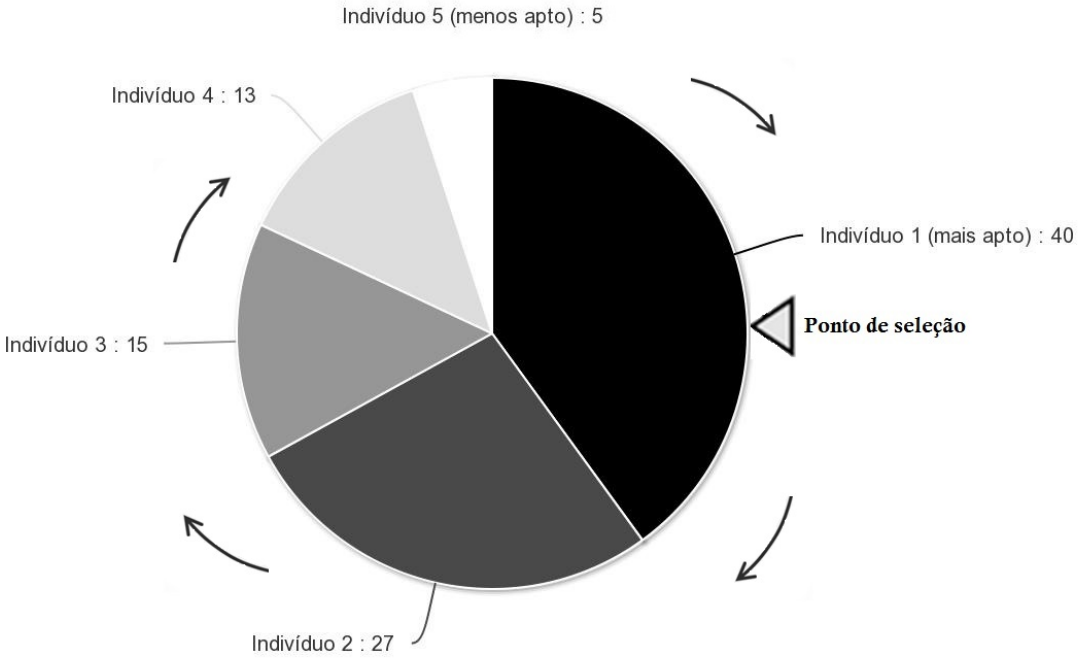

Figura 2.4: Representação visual da técnica de *Seleção por Roleta*.

d) *Crossover*

O estágio de *crossover* é responsável pela recombinação de dois indivíduos para gerar n novos descendentes. A técnica mais simples é o *crossover* de um ponto, pregando que um ponto de divisão ao longo do cromossomo deve ser escolhido randomicamente e os descendentes são gerados através da recombinação das duas partes. Por exemplo, duas strings '000000' e '111111' ao serem divididas e recombinadas na posição 4, gerarão os descendentes '111100' e '000011'. Técnicas de múltiplos pontos de *crossover* e múltiplos filhos também podem ser utilizadas, dependendo da modelagem do problema [23]. A Figura 2.5 nos dá uma visão mais clara de como o processo de recombinação ocorre. Um detalhe importante é que esse operador não é determinístico, ou seja, a probabilidade de se ocorrer uma recombinação genética em uma rodada é determinada pelo parâmetro genético *Taxa de Crossover*. Caso não haja *crossover*, os cromossomos escolhidos como "pais" são simplesmente reproduzidos na próxima geração. A ideia do cruzamento é

simplesmente passar a melhor parte dos indivíduos antigos para os descendentes, gerando novos indivíduos melhores.

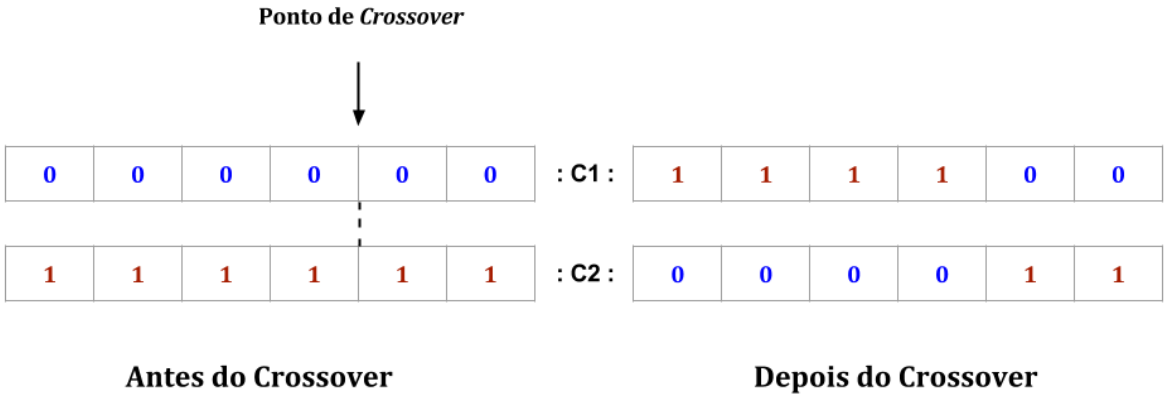

Figura 2.5: Representação visual da operação de *crossover* de um ponto entre dois cromossomos.

e) Mutação

O operador de mutação é o último ator na criação de uma descendência. Assim como na teoria da evolução, a mutação genética tem papel fundamental na busca por uma solução de alto nível. O operador tem seu funcionamento dependente da forma de codificação do cromossomo. Em sequências binárias por exemplo, a mutação resume-se a trocar randomicamente um bit pelo seu oposto. A teoria por trás desse operador é amenizar o fato de que a população inicial, gerada randomicamente, não possui toda a informação necessária para a resolução do problema. A mutação também é útil nos casos onde os cromossomos de menor *fitness score*, os quais não produzem descendência, podem conter informações fundamentais para a solução do problema. Da mesma forma que o *crossover*, a mutação ocorrerá ou não dependendo de uma *Taxa de Mutação*. Assim como na evolução das espécies, a mutação em algoritmos genéticos tem uma probabilidade muito baixa de ocorrer, uma vez que se essa taxa for alta, o algoritmo assume o comportamento de uma busca aleatória.

f) Laço e Condições de Parada

Ao observarmos o pseudocódigo mostrado na Figura 2.3, notamos que todo o processo é regulado por um *loop* condicional. Em geral, definimos no início do algoritmo um valor desejado para a *fitness score*, ao fim da etapa de mutação, se não houver nenhum indivíduo com a aptidão mínima estabelecida, quer dizer que a condição de parada não foi atingida

e a heurística começará uma nova iteração. Esse *loop* permite que o algoritmo explore um maior espaço de soluções, não ficando preso em um ótimo local [31]. Dependendo dos requisitos do problema, um *timeout* pode ser estabelecido para limitar o tempo de execução. Esse tempo limite é baseado no número de iterações do laço, ou seja, o número de gerações que o algoritmo poderá alcançar.

g) Tamanho da População

Ainda na parte inicial, onde definimos os parâmetros do algoritmo genético, notamos a presença de outro fator que tem grande impacto no processo de busca por soluções, o tamanho da população. Esse valor indica a quantidade de indivíduos que teremos em uma única geração. Por se tratar de uma variável bem sensível ao algoritmo, a escolha desse valor não é uma tarefa das mais simples. Se definirmos uma população muito pequena, há menos possibilidades de *crossover*, gerando uma menor variabilidade genética e por consequência explorando apenas uma pequena parte do espectro de soluções possíveis. Por outro lado, ao se escolher um valor muito elevado para tamanho da população, estaremos adicionando um *overhead computacional*, aumentando o esforço computacional e comprometendo em grande escala a performance do AG. Resumindo, há um ponto ótimo para o tamanho da população, o qual é dependente da modelagem e codificação do problema. Além disso, é muito inefetivo aumentar a população em função de se obter um melhor desempenho do algoritmo.

2.3.2 Aplicação e Características

Como dito anteriormente, algoritmos genéticos são geralmente empregados na resolução de problemas complexos (NP-difíceis) e sistemas de aprendizado de máquina. A facilidade de implementação pode ser explicada pelo fato do corpo do algoritmo já estar pronto, assim como seus operadores, um dos motivos da larga aplicação dessa abordagem atualmente. A dificuldade no desenvolvimento dessa heurística está na representação do problema, (*encoding*), e na definição de uma *Fitness Function* eficaz, tarefas que podem ser bastante desafiadoras dependendo do problema proposto. Além disso, os algoritmos genéticos se sobressaem entre outras abordagens devido a sua capacidade de encontrar soluções próximas a ótima global, enquanto outras heurísticas como *Hill Climbing* e *Simulated Annealing* ficam presas rapidamente em um ótimo local, como podemos notar na Figura 2.6.

Em contrapartida, o tempo de processamento ainda é um ponto fraco do algoritmo. Em um espaço de soluções pequeno, a heurística pode ter um desempenho pior do que outras técnicas. Em algumas situações, o desempenho é o principal requisito do sistema,

entretanto, problemas complexos geralmente não têm restrições de tempo tão fortes, priorizando a qualidade das soluções. Uma razão para esse desempenho variável é a sua característica não determinística, ou seja, a randomicidade na geração da população inicial, taxa de *crossover* e mutação podem aumentar o tempo de encontro de uma solução ou até mesmo atingir o tempo limite e não retornar solução alguma.

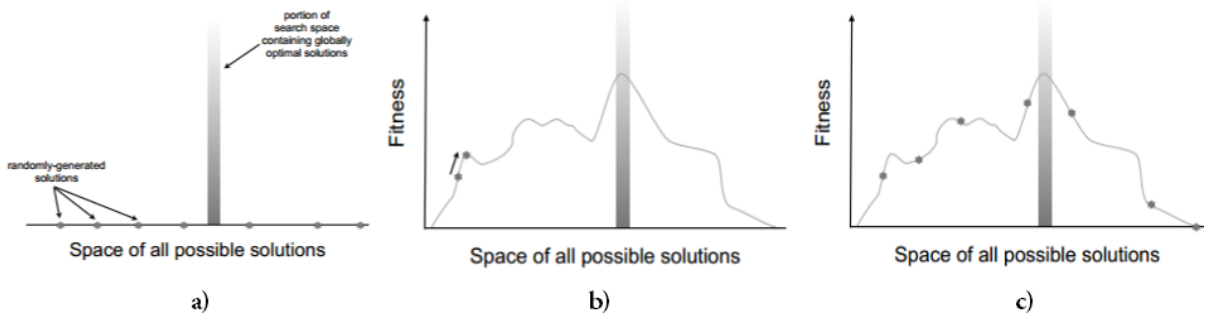

Figura 2.6: Comparativo de espectros de soluções encontradas nas técnicas de (a) *Random Search*, (b) *Hill Cimbing* e (c) Algoritmo Genético [24]

Capítulo 3

Trabalhos Relacionados

Neste capítulo veremos alguns dos trabalhos relevantes realizados na área, descrevendo brevemente o objetivo, metodologia e resultados obtidos. Entenderemos também como esses trabalhos contribuem para o desenvolvimento do presente projeto.

3.1 Trabalho Relacionado 1

M. B. Wall nos traz na sua tese de doutorado, "*A Genetic Algorithm for Resource-Constrained Scheduling*" [33], um estudo acerca do escalonamento de tarefas baseado em recursos. Embora a referência seja relativamente antiga (1996), os fundamentos estudados e qualidade do conteúdo, cuja motivação é muito parecida com a do presente trabalho, são muito relevantes dentro do escopo de estudo desta monografia, principalmente no aspecto de alocação de recursos limitados. Primeiramente, o autor faz questão de ressaltar que planejamento e escalonamento, embora sejam conceitos bem próximos em diversas áreas da engenharia, têm significados diferentes para o propósito do trabalho. No processo de alocação, podemos nos deparar com diversos tipos de limitações, sejam elas de tempo, disponibilidade, arquitetura, ou até mesmo a combinação delas. Fica claro que os planejamentos realizados hoje em dia para alocação de recursos criam diversos gargalos e deixam pouca possibilidade para adaptação.

Pensando no desafio de alocação limitada por recurso e no desempenho promissor de algoritmos genéticos, já comprovado em outros problemas similares, Wall propôs uma abordagem mais generalizada, baseando-se em um representação específica ao problema, e operadores de *crossover* e mutação especializados. De forma geral, o problema proposto nessa tese é: Dados um conjunto de atividades a serem executadas, um conjunto de recursos que serão usados na execução dessas atividades, um conjunto de *constraints* a serem satisfeitas e um conjunto de objetivos, os quais servirão de parâmetro para o julgamento da performance do escalonador, deseja-se saber qual a melhor forma de alocar

os recursos para as atividades em tempos específicos, respeitando as *constraints* e níveis de qualidade.

O algoritmo desenvolvido foi testado em um conjunto de problemas:

- *Patterson's project scheduling problems (PAT)*
- *Single mode project scheduling set by Kolisch et al.(SMCP)*
- *Single-mode full-factorial set by Kolisch et al.(SMFF)*
- *Multi-mode full-factorial set by Kolisch et al.(MMFF)*
- *Job-shop problems from the operations research "warehouse" (JS)*
- *The benchmarx problems by Fox and Ringer (BMRX)*

Todos os testes realizados foram baseados no mesmo algoritmo genético, salvo algumas pequenas variações na entrada de dados e objetivos, que não influenciaram no genoma nem na aplicação da heurística. O autor desenvolveu o sistema em C++ usando a biblioteca GAlib. Os testes rodaram em diversas *workstations* com MIPSR4x00 rodando de 100 a 150MHz.

De modo geral, o algoritmo genético demorou mais tempo do que a sua busca enumerativa equivalente ou um escalonador heurístico demoraria para executar os casos de teste. Entretanto, o autor aponta que nenhuma tentativa de afinar os parâmetros genéticos foi feita, ou seja, o conjunto de testes foi criado simplesmente para proporcionar um ambiente básico para comparação. Os resultados transmitem os dados de pior caso para o algoritmo genético e modo de representação.

Ainda assim, o algoritmo genético superou o método de solução exata [28] em problemas multi-modais. O algoritmo se saiu bem em alguns problemas que seriam muito complexos para se resolver com técnicas *branch and bound*, por exemplo. Em média, uma execução de 100 gerações para um problema simples de Patterson dura alguns segundos, podendo passar de uma hora em um problema mais complexo de 5000 gerações.

Em linhas gerais, o autor ressalta que um conjunto de problemas mais realista seria ideal para se chegar a resultados mais concretos. Apesar disso, como citado anteriormente, o algoritmo genético se saiu muito bem em atividades multi-modais e as combinações extras adicionadas por esse tipo de problema não afetaram o desempenho da heurística. Isso prova que um algoritmo genético modificado, ou um híbrido seria um bom candidato a resolver problemas de maior complexidade.

Por outro lado, a implementação não mostrou bom desempenho em problemas onde os recursos são muito limitados. Ainda que a representação do problema tenha sido modelada para facilitar o processo de encontrar soluções que satisfaçam as limitações, de acordo com

o autor, a inclusão de muitas *constraints* só torna a resolução do problema mais custosa. O algoritmo genético também não foi muito satisfatório em problemas do tipo *jobshop*, ou seja, problemas em que as tarefas ideais são alocadas aos recursos em determinados tempos. Essa gama de problemas geralmente tem características de paralelismo, quando uma pequena modificação é feita na alocação de uma tarefa, um grande impacto é sentido pelo sistema, afetando negativamente o desempenho do alocador.

Wall afirma que algoritmos genéticos têm um conceito muito simples e são ótimos candidatos para resolver problemas que mesclam variáveis discretas e contínuas. Entretanto, a implementação desses sistemas esta longe de ser trivial. Ainda assim, as ideias são bem diretas, exigindo grande trabalho futuro na implementação dessa heurística em um sistema real e com grande espaço de soluções.

3.2 Trabalho Relacionado 2

O objetivo principal do *paper The Study of Genetic Algorithm-based Task Scheduling for Cloud Computing* [31] é apresentar um modelo de alocação de tarefas baseado em algoritmos genético. Nesse modelo, o escalonador de tarefas executa o algoritmo genético a cada ciclo de alocação. Basicamente, a função cria um conjunto de esquemas de alocação e avalia cada um, tomando a satisfação do usuário e a disponibilidade de recursos como parâmetros. Segundo Jang *et al.*, os diversos alocadores de tarefas existentes atualmente não se aplicam de maneira eficiente no contexto de *Cloud Computing*. Essa deficiência é fundamentada no fato de que os modelos atuais possuem foco unicamente na melhoria de desempenho do sistema, negligenciando a qualidade do serviço.

A validação do trabalho baseou-se na comparação de desempenho obtido na implementação de AG proposta em relação a outras técnicas vigentes para alocação de tarefas, como *Round Robin Task Scheduling Model* (RRTSM) [21], *Load Index-based Task Scheduling Model* (LITSM) [18] e *Activity Based Costing-based Task Scheduling Model* (ABCTSM) [27]. Resumidamente, RRTSM aloca tarefas para as maquinas virtuais em sequência, desconsiderando as informações das tarefas ou recursos disponíveis. O (LITSM) utiliza um índice de carga das máquinas virtuais, alocando as tarefas para as máquinas menos carregadas. Por fim, o ABCTSM usa o custo requerido para o processamento das tarefas em cada máquina virtual como parâmetro de alocação. As tarefas propostas pelos autores são classificadas em *Very High Priority*, *High Priority*, *Mid Priority* e *Low Priority* (Figura 3.1).

Tendo em vista o processo de validação definido anteriormente, a comparação de desempenho foi baseada em fatores de vazão, tempo de resposta, utilização de recursos, custo de processamento e satisfação do usuário. Para os testes, 12 diferentes maquinas

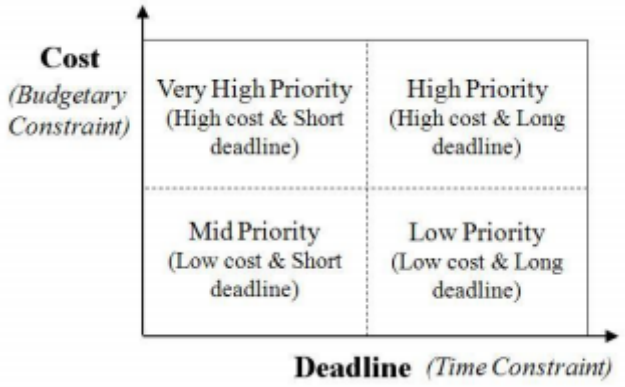

Figura 3.1: Classificação de Tarefas por Tempo e *Budgetary Constraints* [31]

virtuais com diferentes configurações foram simuladas. 2000 tarefas foram geradas pelo usuário das *clouds* em cada simulação. Vemos a seguir um breve resultado da simulação.

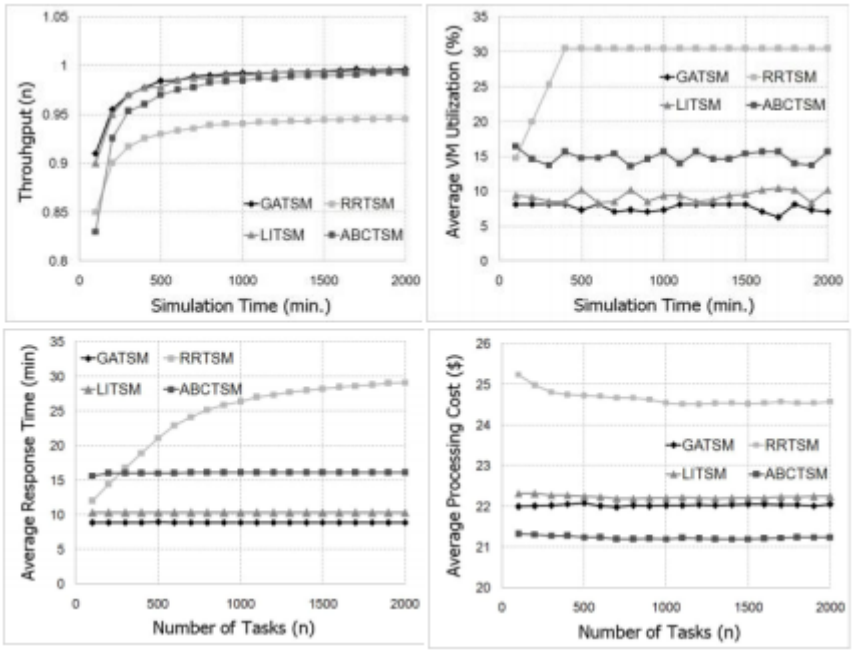

Figura 3.2: Resultados da Simulação [31]

Para avaliação de desempenho, a implementação proposta foi simulada e passou por diversos experimentos. Baseando-se na Figura 3.2 e em outros resultados empíricos, os autores mostram que o escalonador proposto supera em performance os métodos de alocação *Round Robin Task Scheduling Model* (RRTSM), *Load Index-based Task Scheduling Model* (LITSM) e *Activity Based Costing-based Task Scheduling Model* (ABCTSM).

3.3 Trabalho Relacionado 3

O terceiro e último trabalho relacionado que apresentamos, *Job Scheduling Model for Cloud Computing Based on Multi-Objective Genetic Algorithm* [17], também foca na alocação de tarefas em *Cloud Computing*, ainda assim, as ideias trazidas por esse *paper* em conjunto com os anteriores fundamentarão o objetivo da presente monografia e deixarão mais claro o motivo de algumas escolhas terem sido feitas em detrimento de outras.

Liu *et al.* começam o artigo apontando a grande motivação por trás de sua pesquisa. Os grandes *datacenters* provedores dos serviços em nuvem consomem uma quantidade significativa e crescente de energia elétrica, um exemplo que colabora para essa afirmação é que um centro de dados médio consome energia equivalente a 25000 residências. Dessa forma, uma computação consciente dos gastos energéticos é fundamental para sistemas em nuvem. Ainda nessa linha de pensamento, os autores alertam que a demanda de recursos por diferentes tarefas variam com o tempo, dessa forma, criar um alocador de alta performance que respeite o SLA, garantindo confiabilidade, segurança, balanceamento de recursos e ainda diminua o consumo de energia é um problema bastante desafiador em um ambiente de *Cloud Computing*.

Encontramos ainda nesse artigo, trabalhos de diferentes autores introduzindo técnicas de alocação de tarefas que consideram consumo energético ou lucro, entretanto, nenhum deles oferece uma visão a respeito da relação entre os dois. Pensando nisso, os autores propuseram um escalonador baseado em um Multi Objective Genetic Algorithm (MOGA), o qual considera gasto elétrico e o lucro da provedora como parâmetros para prover um mecanismo de seleção dinâmico do melhor esquema de alocação.

Os resultados do MOGA formam um conjunto de soluções de Pareto, oferecendo uma gama de soluções possíveis, enquanto diminui a eficiência do processo de escalonamento. Os autores afirmam que na prática, os usuários ocasionalmente precisam ajustar o nível das suas preferências de objetivos submetidos. O uso de um *Pareto Front* (Figura 3.3) oferece a possibilidade de se escolher uma solução de acordo com o requerimento atual da tarefa.

Diversos experimentos foram realizados variando os parâmetros do algoritmo genético. A fim de validar a implementação, uma comparação entre três diferentes métodos de alocação foi feita, os resultados podem ser vistos na Figura 3.4. O escalonamento por *Maximum Applications* visa maximizar o número de de aplicações alocadas, enquanto o método de Alocação Randômica atribui aleatoriamente uma tarefa para uma nuvem. Os resultados do MOGA foram deduzidos de 30 execuções independentes. O tamanho da população nos testes realizados era de 20 indivíduos, além de contar com 1000 gerações, *taxa de crossover* e *taxa de mutação* de 0.95 e 0.1, respectivamente. Os usuários devem

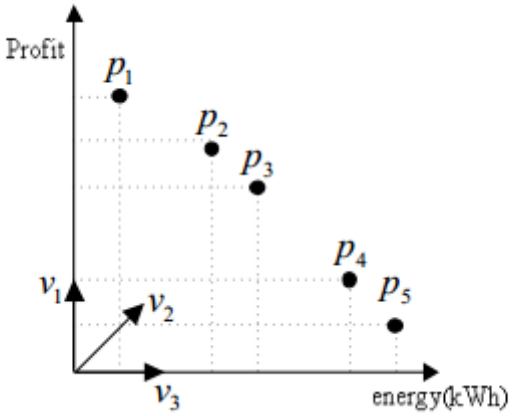

Figura 3.3: Vetores bidimensionais, onde p1 - p5 representa o arquivo externo gerado pelo MOGA [17]

pagar 2 ¥por maquina virtual a cada hora, e o provedor deve pagar 0.5 ¥por kWh de eletricidade.

Algorithm	*Arrival Rate*	*Energy (kWh)*	*Profit (¥)*	*Failed Applications*
MO-GA	Low	2340	2988	91
	Medium	2506	2927	130
	High	3875	2846.5	214
Maximum applicatio -ns	Low	4213	2826	105
	Medium	3950	2811.5	159
	High	3904	2796	281
Random	Low	1979	893	2703
	Medium	624.8	251.7	8211
	High	8.2	5.6	9635

Figura 3.4: Comparação experimental dos três algoritmos [17]

De acordo com os experimentos realizados e a tabela apresentada acima, os autores concluíram que o algoritmo genético proposto como alocador obtém maior lucro, enquanto consomem menor energia. Quando a taxa de chegada de tarefas é baixa, o método MOGA consome 44.46% menos energia, e um lucro 5.73% maior. Essas melhorias diminuem com o aumento da taxa de chegada de tarefas. A segunda conclusão é que quanto maior for o número de aplicações submetidas, maior é a taxa de falha do algoritmo. Outra implicação retirada dos experimentos é que, uma vez que o alocador aleatório não considera os requerimentos das tarefas ou os recursos das máquinas virtuais, resultados de menor qualidade foram obtidos.

Capítulo 4

Metodologia

Neste capítulo, analisaremos os detalhes de desenvolvimento da solução proposta. Primeiramente, faremos uma breve modelagem do problema, assumindo um conjunto de hipóteses que baseiam o funcionamento da heurística, assim como apresentar a estrutura básica de uma tarefa e de uma *instância de cloud*. O segundo tópico é responsável por explicar a representação de uma alocação, descrevendo o comportamento e explicando a função dos operadores genéticos no problema. Por último, mostraremos os testes realizados e as modificações feitas no ambiente para analisar o desempenho do AG em diferentes contextos.

4.1 Modelagem do Problema

O contexto de aplicação dessa abordagem resume-se, em um primeiro momento, à alocação de tarefas submetidas por um usuário e recursos computacionais ofertados por uma provedora, ou seja, serviços de IaaS. O algoritmo desenvolvido propõe realizar a alocação de uma lista de tarefas para um conjunto de instâncias de máquina, de modo um para um. Esse escalonamento deve respeitar o nível de QoS definido no SLA. Levantaremos a seguir, algumas hipóteses a respeito dos modos de operação propostos, organização das tarefas e dos recursos, necessárias para criar um bom ambiente de reprodução de experimentos e resultados. Para fins práticos, todas as instâncias de nuvens disponíveis têm características orientadas para o processamento de dados.

4.1.1 Definição dos Modos de Operação

A ideia de implementar diferentes modos de operação foi introduzida, não só para validar a solução proposta, mas também mostrar a versatilidade do algoritmo genético em diferentes tipos de contexto, simplesmente ajustando a Função de Aptidão. Diversos tipos de

abordagens podem ser aplicadas a fim de prover um serviço de melhor qualidade para o cliente. Por outro lado, o usuário deve ser munido de ferramentas que o ajudem a escolher uma instância adequada para resolver suas tarefas. Os modos de operação descritos abaixo servem para ambas as partes interessadas. Se o provedor quiser diminuir o desperdício de recursos e otimizar seus lucros, rodar o modo de operação Custo-Benefício é o ideal. No entanto, se o usuário não se preocupa muito com o custo e planeja executar sua tarefa de forma mais eficiente, talvez uma simulação no modo "Desempenho" o ajude a fazer uma escolha mais acertada ao contratar um serviço *On-Demand Cloud Computing*, por exemplo.

Tabela 4.1: Definição dos Modos de Operação

Modo de Op.	**Descrição**	**Exemplo**
Custo-Benefício	O foco do Algoritmo Genético nesse modo de operação é atribuir maior *Fitness Score* para as combinações mais justas, ou seja, idealmente o alocador deverá escolher uma instância que oferte exatamente a quantidade de recurso requerido pela tarefa, objetivando evitar o sobreprovisionamento de recursos.	Um usuário submete uma tarefa que necessita de 4GB de memória RAM para ser processada, em um caso ideal, o sistema escolherá uma instância com uma configuração que lhe proverá exatamente 4GB de memória RAM. Sendo que quanto mais recurso sobressalente, maior o decréscimo na pontuação de aptidão.
Desempenho	No modo "Desempenho", o algoritmo busca exatamente o contrário do modo anterior. Dentro de um *budget* preestabelecido na tarefa, o algoritmo melhor pontuará a solução onde a tarefa estiver alocada em uma instância com mais poder de processamento, ocasionando um sobreprovisionamento de recursos.	Se uma tarefa tem um *budget* de 2 US$ para ser executada e requer 2 núcleos de processamento, o algoritmo procurará uma instância no espaço de busca, dentro do orçamento, com maiores quantidades de núcleos. Quanto maior essa diferença maior a pontuação de aptidão.
Disponibilidade	Este último modo de operação é semelhante ao anterior, entretanto, o parâmetro fundamental é o de disponibilidade, o qual terá mais peso que os outros fatores no cálculo da *Fitness Score*. Um usuário define o nível de disponibilidade que deseja para realização da sua tarefa e o alocador procurará uma instância com maior porcentagem de tempo online e que esteja dentro do orçamento	Uma tarefa precisa processar uma longa sequência de proteína, e para que gere resultados confiáveis, é preferível que ela sofra poucas interrupções. Sabendo disso, o usuário define um valor de 80% de disponibilidade mínima. O alocador, por sua vez, procura uma instância em que a taxa de disponibilidade seja a maior possível que a requerida e que esteja dentro do orçamento da tarefa.

Alguns conceitos ainda devem ser considerados para reforçar o conteúdo apresentado

na figura acima. Com a intenção de facilitar o entendimento do modelo, o modo de operação "Custo-Benefício" foi simplificado. Alocar a quantidade de recursos exatamente igual a requerida pela tarefa, certamente não é uma abordagem interessante para quem preza por dependabilidade. É lógico que em um sistema real, uma certa redundância de recursos pode ser útil para completar uma tarefa, mas por motivo didático, assumiremos aqui que o requisito mínimo apresentado pela tarefa é suficiente para solucioná-la de forma confiável. Vale lembrar que nesse modo, além dos outros parâmetros, a instância com menor custo/hora terá uma maior pontuação.

Para o segundo modo de operação, estamos partindo do mais primitivo conceito estabelecido em *Benchmarking*. Se uma máquina tem mais recursos de hardware voltados para processamento como GPU, CPU e Memória, teoricamente, ela realizará uma tarefa em um tempo menor, e provavelmente com menor taxa de falhas do que uma máquina menos potente. Dessa forma, o algoritmo genético procurará as instâncias mais poderosas, e que ao mesmo tempo tenham um custo dentro do limite estabelecido, entretanto, como veremos mais adiante, o custo aqui tem um peso reduzido no cálculo da *Fitness Score*, funcionando mais como uma *constraint*.

4.1.2 Representação de Tarefas e Recursos

Agora que já temos uma noção de como se comporta o alocador em diferentes modos de operação, veremos como as tarefas e recursos se estruturam. Uma aplicação pode ser formada por diversos requisitos, os escolhidos nesse trabalho são: Número de núcleos necessários, memória (GB), preço/hora e nível de disponibilidade. Essa configuração mínima deve ser submetida pelo próprio usuário. Veremos mais adiante que quanto mais parâmetros estabelecidos, mais *constraints* serão criadas, e por consequência menor o desempenho do algoritmo. A Tabela 4.2 apresenta algumas tarefas submetidas por um usuário.

Tabela 4.2: Exemplo de uma sequência de tarefas submetidas pelo usuário

#Tarefa	**#Núcleos**	**Memória(GB)**	**Budget (US$) / Hora**	**Disponibilidade**
0	2	2	0.15	0.8
1	4	8	0.38	0.9
2	8	8	0.75	0.99
3	16	16	1.46	0.99
4	16	20	1.75	0.99999
5	4	6	0.23	0.9

A escolha desses requisitos, foi baseada no modelo de negócio sob demanda da Amazon Elastic Compute Cloud (EC2) [1], e da Google Compute Engine (GCE) [5]. Ambas as

provedoras de IaaS têm seus serviços fundamentados em oferta de recursos via instâncias de *cloud*, ou seja, o usuário pode escolher dentro de um banco de máquinas (instâncias), de diferentes configurações, aquela mais adequada para suprir suas necessidades, pagando pela *workstation* uma taxa por hora de utilização. O banco de dados de *clouds* utilizado para a simulação mais básica do presente trabalho contém 52 entradas, sendo 37 diferentes instâncias disponíveis na região da Virgínia pelo EC2 e 15 instâncias disponibilizadas pela GCE em toda região dos Estados Unidos. Uma parte desse banco é apresentada nas Tabelas 4.3 e 4.4.

Tabela 4.3: Parte do banco de instâncias utilizado (EC2)

#Instância	**#Núcleos**	**Memória(GB)**	**Custo/Hora**	**Disponibilidade**	**Nome**
0	2	3.75	0.105	0.7	c3.large
1	2	1.7	0.130	0.8	c1.medium
2	4	7.5	0.210	0.9	c3.xlarge
3	8	15.0	0.420	0.95	c3.2xlarge
4	8	7.0	0.520	0.98	c1.xlarge
5	16	30.0	0.840	0.99	c3.4xlarge
6	32	60.0	1.680	0.99999	c3.8xlarge
7	32	60.5	2.000	0.99999	cc2.8xlarge
8	8	15.0	0.650	0.98	g2.2xlarge
9	16	22.5	2.100	0.99999	cg1.4xlarge

Tabela 4.4: Parte do banco de instâncias utilizado (GCE)

#Instância	**#Núcleos**	**Memória(GB)**	**Custo/Hora**	**Disponibilidade**	**Nome**
37	2	1.8	0.088	0.7	n1-highcpu-2
38	4	3.6	0.176	0.8	n1-highcpu-4
39	8	7.2	0.352	0.9	n1-highcpu-8
40	16	14.4	0.704	0.98	n1-highcpu-16
41	1	3.75	0.070	0.7	n1-standard-1
42	2	7.5	0.140	0.8	n1-standard-2
43	4	15.0	0.280	0.9	n1-standard-4
44	8	30.0	0.560	0.98	n1-standard-8
45	16	60.0	1.120	0.99999	n1-standard-16
46	2	13.0	0.164	0.8	n1-highmem-2

É importante ressaltar que tanto Amazon EC2 quanto Google Compute Engine atuam com esse serviço em diversas partes do mundo, entretanto, para fim de validação deste trabalho trabalharemos apenas com as federações apresentadas acima. Mais a frente, veremos que foi necessário variar o número de instâncias a fim de verificar o comportamento do algoritmo genético. As instâncias adicionais são fictícias, entretanto, respeitamos o padrão de valores encontrado nas instâncias reais.

4.1.3 Hipóteses do Processo Experimental

Nesta subseção, serão apresentadas algumas prerrogativas necessárias para o funcionamento do AG como alocador, assim como as regras de negócios assumidas na definição do problema.

Tabela 4.5: Premissas de Tarefas e Recursos

Tópico	**Descrição**
Alimentação do Sistema	O algoritmo genético utiliza dois arquivos de texto como entrada para o processo de alocação (*ApplicationReq*.txt e *CloudRes*.txt). O primeiro arquivo é uma lista de aplicações que o usuário submete para processamento. Um exemplo de uma pequena lista pode ser observada na tabela 4.2. O segundo arquivo é o banco de *clouds* disponíveis para utilização.
Conhecimento Técnico	O ideal em um serviço de *cloud computing* é que o usuário seja poupado de saber as especificidades técnicas para cumprir seus objetivos, entretanto, para a aplicação desse alocador é necessário que o usuário saiba quais os recursos mínimos para a execução da tarefa.
Método de Alocação	Como o foco do nosso estudo é o escalonamento de recursos de infraestrutura, em especial sistemas sob demanda, assumiremos que todos os recursos são renováveis e estão disponíveis ao fim de cada ciclo, ou seja, para fins de estudo não precisamos nos preocupar com nenhuma *constraint* de tempo e o modo de alocação será realizada no sistema *First come, first served* (FCFS).
Lista de Tarefas	O Algoritmo Genético implementado foi proposto para efetuar uma alocação multi-objetivo e está preparado para processar uma lista de tarefas de tamanho indefinido, porém, se a lista for composta de uma única tarefa, o cromossomo será constituído de apenas um gene, impossibilitando a operação de *crossover*. Por outro lado, se a lista for muito grande, o cromossomo também será grande, e probabilisticamente há maior chance de violação de alguma *constraint*, diminuindo bastante a eficiência da heurística ao ponto de torná-la impraticável.
Disponibilidade	O parâmetro de disponibilidade, definido nas tarefas e recursos, são valores **fictícios** proporcionais ao custo da instância, ou seja, não representam a realidade das instâncias atreladas. Ainda assim, taxas de disponibilidade podem ser mensurados sem grandes dificuldades, por isso metas de *uptime* são estabelecidas no SLA, entretanto, valores reais não foram utilizados aqui, pois não foram disponibilizados pelas empresas provedoras.

Além das hipóteses assumidas anteriormente, apresentaremos na Tabela 4.6 algumas limitações comuns a todos os modos de operação, as quais tornarão a simulação ainda mais real, com o objetivo de obter resultados mais concretos. As restrições estão relacionadas

ao provimento de recursos, orçamento das tarefas, custo das máquinas e não paralelismo de tarefas.

Tabela 4.6: *Constraints* de Alocação

Constraint	**Descrição**	**Exemplo**
Recursos	A primeira restrição é destinada a relação tarefa-máquina, a qual proíbe o algoritmo de selecionar uma máquina com recursos inferiores ao que o usuário requisita para suas tarefas. As combinações que violam essa *constraint* recebem zero como *Fitness Score* automaticamente.	Um usuário submete uma tarefa que necessita de 2 núcleos, 4 GB de memória e 70% de taxa de disponibilidade, o alocador deverá atribuí-la a uma instância com configuração de no mínimo 2 núcleos, 4 GB de memória e 70% de disponibilidade.
Financeiro	Nesta segunda modalidade, a restrição é voltada para os parâmetros monetários. Essa *constraint* define que uma instância de *cloud* deve ser reservada para uma tarefa, de tal forma que o custo da utilização por hora da máquina seja menor do que o *budget* estabelecido pelo usuário. As combinações que violam essa *constraint* recebem zero como *Fitness Score* automaticamente.	Um usuário está disposto a pagar no máximo 1.10 US$ para que sua tarefa seja executada, logo, o alocador deve selecionar uma instância de nuvem cujo custo não ultrapasse o orçamento predefinido, ou seja, todas as máquinas retornadas pelo AG terão um custo inferior ou igual a 1.10 US$.
Não-Paralelismo	No contexto do nosso modelo de alocação, assumimos que uma tarefa é uma unidade básica e indivisível, portanto, o paralelismo para resolução de uma tarefa está fora de questão. Essa *constraint* prega que uma mesma tarefa não pode ser alocada para diferentes instâncias de *cloud*, entretanto, duas ou mais tarefas podem ser alocadas para o mesmo tipo de máquina[1]. As combinações que violam essa *constraint* recebem zero como *Fitness Score* automaticamente.	Um usuário submete uma lista de três tarefas distintas com características muito semelhantes para processamento, como resultado, o alocador retornou uma combinação com boa pontuação de aptidão, no qual um mesmo tipo de instância de nuvem foi atribuído para duas das três tarefas.

[1]Em um sistema real, é possível escolher um tipo de máquina juntamente com a quantidade de instâncias que se deseja contratar, logo, é plenamente possível que duas ou mais tarefas sejam alocadas para o mesmo tipo de instância.

4.2 Processo de Alocação

Nesta seção, veremos com mais detalhes como uma agenda de alocação é representada, assim como o passo a passo do algoritmo genético e seus operadores no processo de otimização. Para o entendimento do modelo ser mais natural, apresentaremos a seguir uma tabela de notações.

Tabela 4.7: Tabela de Notações

Notação	**Descrição**
C_i	Cloud i
T_i	Tarefa i
RT_i	Recursos requisitados pela tarefa i
RC_i	Recursos ofertados pela *Cloud* i
I_i	*Indivíduo ou cromossomo i*
FS_i	*Fitness Score* do cromossomo i
TCr_i	Tamanho do cromossomo i
Pop	Tamanho da população
$G_{j,i}$	Gene j do indivíduo i
$PFS_{j,i}$	*Fitness Score* do gene j do cromossomo i
TFS_i	*Fitness Score* total da geração i
NT_i	Número de núcleos requisitados pela tarefa i
MT_i	Memória em GB requisitada pela tarefa i
BT_i	*Budget* em US$ estipulado pela tarefa i
DT_i	Nível de disponibilidade requisitado pela tarefa i
NC_j	Número de núcleos disponíveis na *Cloud* j
MC_j	Memória em GB disponível na *Cloud* j
CC_j	Custo em US$ da nuvem i por hora de utilização
DC_j	Nível de disponibilidade provido pela *Cloud* j

4.2.1 Codificação e Inicialização

Como introduzido no Capítulo 2, a representação de um indivíduo é um ponto crucial para o bom desempenho de um algoritmo genético. Tal representação deve ser concisa, mas ao mesmo tempo deve conter informação suficiente para ser capaz de caracterizar uma solução. Informação a menos pode parecer pouco confiável, assim como um *overhead* de dados na representação de um cromossomo pode afetar negativamente a latência de resposta e o nível de qualidade das soluções retornadas. A representação ideal nessa heurística deveria conter apenas combinações possíveis, para que o problema seja meramente de otimização, e não de satisfatibilidade. Entretanto, nem toda sorte de problemas permite esse tipo de manobra, veremos que no nosso problema, as *constraints* são fortes limitadores nessa questão, eliminando boa parte das combinações geradas.

Inicialmente, o algoritmo gera aleatoriamente a primeira geração de cromossomos. A grandeza da geração é regulada pelo parâmetro genético *Tamanho da População*, que é uma variável global que define o que o próprio nome sugere. Usaremos os parâmetros de instâncias como identificadores, tanto das tarefas quanto das máquinas. Vemos na Tabela 4.8, um exemplo de um cromossomo de tamanho 6, representando a alocação de seis tarefas comunicantes para as respectivas instâncias de *cloud*. Cada coluna que relaciona uma tarefa a um tipo de instância de máquina é conhecido como um *gene*, e veremos mais adiante que tarefas que não tem dependências com outras podem ser alocadas em cromossomos de tamanho unitário.

#Tarefa	0	1	2	3	4	5
#Instância	22	31	09	45	18	16

Tabela 4.8: Representação de um cromossomo

As tarefas são apresentadas sempre em ordem crescente (dados lidos do arquivo de tarefas submetido pelo usuário), por outro lado, para cada cromossomo da primeira geração, as instâncias de máquinas são escolhidas randomicamente do arquivo de nuvens. Vale ressaltar que o tamanho do cromossomo também pode ser definido pelo usuário através da variável global *Tamanho do Cromossomo*. A única regra é que o tamanho do cromossomo deve ser igual ou INFERIOR ao número de tarefas definidas pelo usuário no arquivo de entrada. Idealmente, buscamos igualar o número de tarefas ao tamanho do cromossomo para que as combinações representem a alocação de todas as tarefas. A variação do tamanho do cromossomo, assim como do tamanho da população influenciam no desempenho do algoritmo, veremos mais a respeito na seção de avaliação.

4.2.2 Cálculo de Aptidão e Seleção

Uma vez que a primeira geração esteja formada, o algoritmo entra na fase de cálculo de aptidão, percorrendo cada cromossomo e atribuindo uma *Fitness Score* proporcional a qualidade da combinação como solução do problema. Como esperado, para cada modo de operação temos uma *Função de Aptidão* diferente. A função de aptidão varia para diferentes tipos de problema, e também é muito sensível ao parâmetro que será utilizado para metrificar a qualidade da solução. Adotamos uma escala bem simples para a pontuação de aptidão, sendo uma escala de zero a um, onde **0** indica que a combinação retornada não satisfaz o problema, enquanto **1** é a qualidade máxima que pode ser apresentada por uma alocação.

Para tornar o estudo mais simplificado, adotamos fórmulas simples de média ponderada para o cálculo das pontuações, mas tendo clareza que, em sistemas reais, quanto mais apurada for essa função, maior o nível de QoS oferecido pelo alocador. Veremos

a seguir qual abordagem foi adotada para metrificar a aptidão nos diferentes modos de operação. Tomaremos o indivíduo 1 abaixo como modelo para explicar os cálculos com mais facilidade. Assumiremos ainda que os dados referentes à combinação do cromossomo 1 são representados nas tabelas 4.10 (Tarefas) e 4.11 (*Workstations*).

0	1	2
61	45	11

Tabela 4.9: Cromossomo 1

Tabela 4.10: Dados referentes às tarefas do cromossomo 1 (Tabela 4.9)

#Tarefa	**#Núcleos**	**Memória(GB)**	**Budget/Hora**	**Disponibilidade**
0	0.5	1	2	0.7
1	16	50	1.2	0.7
2	0.45	0.9	2	0.7

Tabela 4.11: Dados referentes às instâncias de *cloud* do cromossomo 1 (Tabela 4.9)

#Instância	**#Núcleos**	**Memória(GB)**	**Custo/Hora**	**Disponibilidade**	**Nome**
61	1	1.0	0.013	0.7	b1.fictitious1
45	16	60.0	1.120	0.99999	n1-standard-16
11	1	1.0	0.013	0.7	t2.micro

Nível de Aptidão e Qualidade de Solução

Além dos modos de operação, estabelecemos algumas categorias para as pontuações de aptidão, a fim de classificar a qualidade das combinações oferecidas. Os valores estabelecidos podem ser modificados dentro do algoritmo para melhor atender as necessidades do usuário. Devemos lembrar que essas pontuações são limitantes inferiores das soluções, ou seja, isso implica que ao se optar por uma faixa de qualidade média, por exemplo, restringimos o algorítimo de retornar respostas de valor inferior a 0.5, entretanto, não o impedimos de retornar soluções de alta qualidade (FS $\geq$ 0.8). Para nossa validação, estabelecemos empiricamente os seguintes níveis de exigência.

- **Categoria D** (*Fitness Score* = 0): A combinação violou alguma *constraint* e não representa uma solução válida para o problema.
- **Categoria C** (0.5 > *Fitness Score* > 0): A combinação respeita todas as *constraints* de uma solução aceitável, entretanto representa uma solução pouco otimizada.
- **Categoria B** (0.8 > *Fitness Score* $\geq$ 0.5): A combinação respeita todas as *constraints* de uma solução aceitável, e representa uma solução de nível médio de otimização.

- **Categoria A** ($1 \geq$ *Fitness Score* ≥ 0.8): A combinação respeita todas as *constraints*, e representa uma solução altamente otimizada.

O usuário definirá alguma dessas faixas para ser a condição de parada do algoritmo, ou seja, ao se encontrar a primeira combinação com uma pontuação de aptidão acima da estabelecida, o algoritmo sai do laço principal e retorna esse indivíduo como solução do problema.

Custo-Benefício

Como vimos anteriormente, nesse modo de operação o foco é minimizar o desperdício de recursos. Sabendo disso, é senso comum que em um caso ideal, a divisão do valor de um recurso requerido na tarefa pelo recurso fornecido pela máquina deveria ser igual a 1. Em um gene, ou seja, em uma alocação direta de uma tarefa para uma máquina, o cálculo de aptidão envolve quatro recursos e oito variáveis: Número de núcleos(NT_i, NC_j), Memória(MT_i, MC_j), *Budget* ou Custo(BT_i, CT_j) e Disponibilidade(DT_i, DC_j).

O cálculo de aptidão de um gene nesse modo de operação é descrito pela seguinte fórmula:

$$\sum_{\substack{i=tarefa \\ j=cloud}} 0.35 \times \left(\frac{NT_i}{NC_j}\right) + 0.35 \times \left(\frac{MT_i}{MC_j}\right) + 0.15 \times \left(\frac{BT_i}{(CC_j + BT_i)}\right) + 0.15 \times \left(\frac{DT_i}{DC_j}\right)$$

Como este modo de operação é focado no mínimo desperdício de recursos, os parâmetros de hardware (Número de Núcleos e Memória) terão um papel mais importante neste cálculo de aptidão, logo, exercem um peso maior na média ponderada (70%). Assumimos a hipótese que as tarefas requerem proporcionalmente os recursos de CPU e memória, logo, excluímos os casos de tarefas *CPU intensive* e *memory intensive*. Os demais fatores têm a mesma relevância neste modo, e embora sejam menores, ainda são cruciais para a fórmula (30%). Seguindo esse pensamento, para calcular a *Fitness Score* de um indivíduo, basta repetir o procedimento acima o para todos os genes e dividir o somatório de todas essas pontuações parciais pelo tamanho do cromossomo. Tomando o cromossomo da Tabela 4.9 como exemplo, o cálculo de sua aptidão se daria da seguinte maneira:

$$\left[0.35 \times \left(\frac{0.5}{1}\right) + 0.35 \times \left(\frac{1}{1.0}\right) + 0.15 \times \left(\frac{2}{(0.013 + 2)}\right) + 0.15 \times \left(\frac{0.7}{0.7}\right) + \right.$$

$$0.35 \times \left(\frac{16}{16}\right) + 0.35 \times \left(\frac{50}{60.0}\right) + 0.15 \times \left(\frac{1.2}{(1.120 + 1.2)}\right) + 0.15 \times \left(\frac{0.7}{1}\right) +$$

$$0.35 \times \left(\frac{0.45}{1}\right) + 0.35 \times \left(\frac{0.9}{1.0}\right) + 0.15 \times \left(\frac{2}{(0.013+2)}\right) + 0.15 \times \left(\frac{0.7}{0.7}\right)\Bigg] \div 3$$

$$= \frac{(PFS_{0,1} + PFS_{1,1} + PFS_{2,1})}{TCr} = 0.806605$$

Verificamos portanto, baseado nos cálculos acima, que a combinação da tabela 4.9 representa uma solução de alto valor de qualidade, o que significa que as tarefas serão executadas nas instâncias correspondentes, com pouco desperdício de recursos, garantindo a satisfação do cliente e menor custo para a empresa provedora.

Disponibilidade

Seguindo o método, calcularemos a aptidão do cromossomo exemplo nesse modo de operação. Como esperado, o parâmetro "Disponibilidade" terá um peso maior no cálculo da *Fitness Score* (40%). O parâmetro Número de Núcleos compõe 25% do cálculo, valor idêntico ao coeficiente da Memória RAM. Os restante é multiplicador da parte monetária, que não é muito relevante aqui, uma vez que o algoritmo sempre buscará a solução de menor custo atrelado. A fórmula que descreve o cálculo de aptidão de um gene é dada a seguir:

$$\sum_{\substack{i=tarefa \\ j=cloud}} 0.25\times\left(\frac{NC_j}{NC_j + NT_i}\right) + 0.25\times\left(\frac{MC_j}{MC_j + MT_i}\right) + 0.1\times\left(\frac{BT_i}{(CC_j + BT_i)}\right) + 0.4\times\left(\frac{DC_j}{DC_j + BT_i}\right)$$

Repetindo o mesmo procedimento para encontrar o *Fitness Score* de todo o cromossomo, temos:

$$\Bigg[0.25 \times \left(\frac{1}{1+0.5}\right) + 0.25 \times \left(\frac{1.0}{1.0+1}\right) + 0.1 \times \left(\frac{2}{(0.013+2)}\right) + 0.4 \times \left(\frac{0.7}{0.7+0.7}\right) +$$

$$0.25 \times \left(\frac{16}{16+16}\right) + 0.25 \times \left(\frac{60}{60+50}\right) + 0.1 \times \left(\frac{1.2}{(1.120+1.2)}\right) + 0.4 \times \left(\frac{1}{1+0.7}\right) +$$

$$0.25 \times \left(\frac{1}{1+0.45}\right) + 0.25 \times \left(\frac{1}{1+0.9}\right) + 0.1 \times \left(\frac{2}{(0.013+2)}\right) + 0.4 \times \left(\frac{0.7}{0.7+0.7}\right)\Bigg] \div 3$$

$$= \frac{(PFS_{0,1} + PFS_{1,1} + PFS_{2,1})}{TCr} = 0.580916$$

A partir dos resultados dos cálculos acima, inferimos que o cromossomo representa uma boa solução quanto ao custo-benefício, entretanto, no modo que prioriza disponibilidade, a solução tem um nível mediano de otimização.

Desempenho

Por fim, esse modo de operação baseia-se uma abordagem de sobreprovisionamento de recursos, utilizaremos um cálculo semelhante ao modo anterior, mas atribuindo um coeficiente maior para os parâmetros de hardware (núcleos e memória). O cálculo de aptidão de um gene no modo de operação é dado pela fórmula:

$$\sum_{\substack{i=tarefa \\ j=cloud}} 0.4375\times\left(\frac{NC_j}{NC_j+NT_i}\right)+0.4375\times\left(\frac{MC_j}{MC_j+MT_i}\right)+0.0625\times\left(\frac{CC_i}{BT_i}\right)+0.0625\times\left(\frac{DC_j}{DC_j+BT_i}\right)$$

Os coeficientes foram escolhidos seguindo o mesmo pensamento do modo Custo-Benefício. Entretanto, como os parâmetros de hardware são bem mais relevantes aqui, eles compõem 87.5% do valor final de aptidão. Os 12.5% restantes são distribuídos entre os parâmetros de custo e disponibilidade, os quais servem basicamente como *constraints* e não como indicadores de qualidade. Repetindo o cálculo para todos o genes e dividindo pelo tamanho do cromossomo temos a *Fitness Score* do indivíduo:

$$\left[0.4375\times\left(\frac{1}{1+0.5}\right)+0.4375\times\left(\frac{1.0}{1.0+1}\right)+0.0625\times\left(\frac{0.013}{2}\right)+0.0625\times\left(\frac{0.7}{0.7+0.7}\right)+\right.$$

$$0.4375\times\left(\frac{16}{16+16}\right)+0.4375\times\left(\frac{60}{60+50}\right)+0.0625\times\left(\frac{1.120}{1.2}\right)+0.0625\times\left(\frac{1}{1+0.7}\right)+$$

$$\left.0.4375\times\left(\frac{1}{1+0.45}\right)+0.4375\times\left(\frac{1}{1+0.9}\right)+0.0625\times\left(\frac{0.013}{2}\right)+0.0625\times\left(\frac{0.7}{0.7+0.7}\right)\right]\div 3$$

$$=\frac{(PFS_{0,1}+PFS_{1,1}+PFS_{2,1})}{TCr}=0.552733$$

Como logicamente esperado, se um cromossomo tem uma pontuação alta no modo de operação "Custo-Benefício", o qual preza pela economia de recursos, a pontuação do mesmo cromossomo no modo de operação "Desempenho" não será muito alta, uma vez que nesse modo quanto mais recurso ofertado para a tarefa, maior a pontuação recebida.

No estudo realizado nessa subseção, percebemos também que nenhuma *constraint* apresentada na Tabela 4.6 foi violada, assim, o cálculo da pontuação de aptidão pôde proceder sem

nenhuma interrupção. Em um caso real, devido a natureza randômica da heurística, essas restrições são usualmente confrontadas, zerando a pontuação do indivíduo automaticamente. Um outro conceito que deve ficar claro é que os coeficientes escolhidos para determinar o peso dos parâmetros em cada método de operação foram determinados de uma maneira pouco refinada, e que para a utilização de todo o potencial dos modos de operação, o utilizador da abordagem genética deve usar uma metodologia criteriosa na determinação da sua Função de Aptidão, verificando as maiores necessidades e atribuindo um fator de multiplicação condizente.

Dado que todos os indivíduos tenham sido avaliados e tenham agora uma pontuação de aptidão atrelada, o algoritmo entra na fase de seleção, onde escolherá de dois em dois indivíduos pelo método de "Roleta de Seleção" para reprodução e criação de uma nova população. Como introduzido no Capítulo 2, a probabilidade de um indivíduo ser escolhido em meio a uma população é proporcional à sua pontuação de aptidão. Dessa forma, o algoritmo estará passando para as próximas gerações os genes que compõem as soluções de maior qualidade. O processo para isso dentro do programa é bem simples. O algoritmo calcula a *Fitness Score* total de uma população e gera um número aleatório entre zero e esse valor. A partir daí, a heurística percorre os indivíduos e vai somando suas pontuações, até que o valor acumulado ultrapasse o número gerado randomicamente.

4.2.3 *Crossover* e Mutação

De posse dos dois indivíduos selecionados na etapa anterior, o AG realiza uma recombinação dos dois a fim de gerar um descendente ainda mais apto, e como já introduzido anteriormente, chamamos esse processo de *crossover*. A probabilidade de que a recombinação ocorra é regulada pelo parâmetro *Taxa de Crossover*, valor sensível ao problema. A taxa escolhida para o nosso algoritmo é de 90%, esse valor é bem aceito na literatura e o algoritmo se comportou bem com essa parâmetro. Caso a heurística não realize *crossover*, os cromossomos selecionados são simplesmente copiados para a próxima geração. Uma vez que implementamos um algoritmo genético padrão, o modo de *crossover* escolhido é baseado no modelo mais simples, com apenas um ponto de divisão.

As tarefas são as unidades principais do nosso estudo, e como apresentado na seção de codificação, elas são organizadas de forma crescente em um cromossomo. Por outro lado, as instâncias de máquinas providas pelas nuvens são organizadas aleatoriamente. Fazendo uma analogia bem básica, uma prática bem comum nas situações reais é que quando uma pessoa se perde de outra, dizemos que a estratégia mais eficaz é que uma pessoa fique parada enquanto a outra procura, partindo disso, procurando o modo mais eficaz de encontrar melhores combinações, percebemos que manter as tarefas fixas e recombinarmos as instâncias de máquinas aumentou o desempenho do algoritmo e diminuiu consideravelmente o número de *constraints* violadas em comparação com o *crossover* em ambos, tarefas e máquinas. O modelo de recombinação, portanto foi modelado da seguinte maneira:

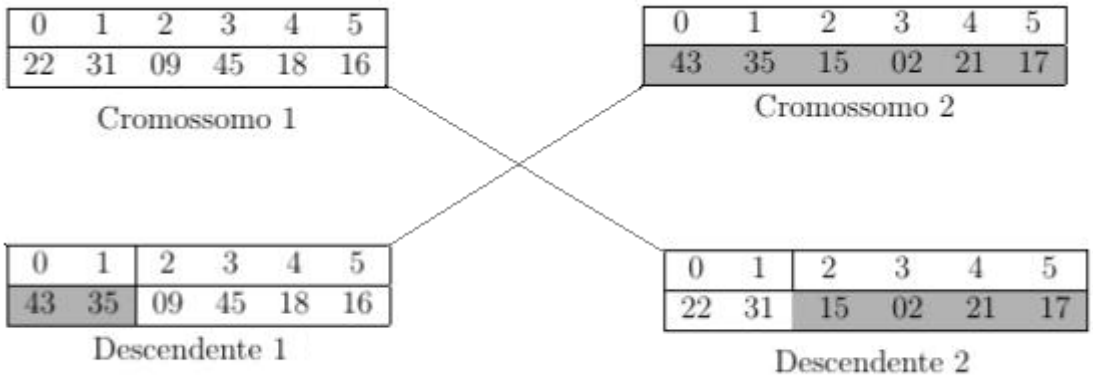

Figura 4.1: Exemplo de *crossover* no algoritmo proposto

O conceito da próxima etapa é bem simples, os dois descendentes gerados no processo de *crossover* são submetidos ao operador de *Mutação*. Essa função percorre o cromossomo gene a gene e de acordo com a *Taxa de Mutação*, a instância de máquina desse cromossomo pode ser substituída por outra instância qualquer dentro do banco de *clouds*. Assim como a taxa de crossover, o valor da taxa de mutação varia bastante dependendo do problema a ser otimizado, ainda assim, geralmente ela tem baixo valor para que a heurística seja viável, servindo para gerar variabilidade genética à solução e evitar que o algoritmo fique preso em um ótimo local. A taxa de mutação escolhida no nosso projeto é de 5%, valor baixo, como indicado por diversas literaturas [34], [23], e que obteve um comportamento satisfatório para o nosso desafio. Uma ilustração desse processo pode ser observado na Figura 4.2.

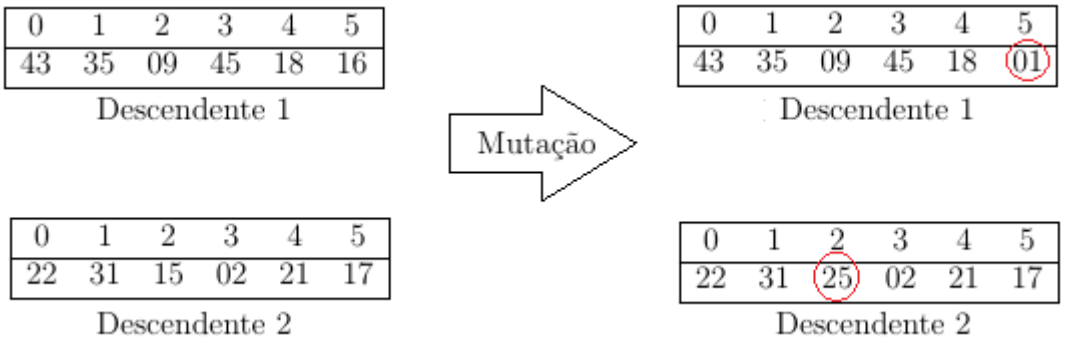

Figura 4.2: Exemplo de mutação no algoritmo proposto

4.2.4 Reinserção e Condição de Parada

Chegando na parte final do laço principal do algoritmo, temos uma nova população de descendentes guardada em uma base temporária que deve ser reinserida na estrutura principal de armazenamento de indivíduos. Uma vez que esse remanejamento seja feito, calcula-se novamente a FS de todos os indivíduos e verifica se algum deles atingiu um limite mínimo de satisfação, de-

finido pelo usuário no início do procedimento como explicado na Seção 4.2.2. Caso essa condição não tenha sido satisfeita, o algoritmo entra em uma nova iteração e repete todo o procedimento descrito até aqui. Por outro lado, se essa condição for satisfeita, esse cromossomo será retornado ao usuário como uma alocação otimizada para o problema.

No início do código, o número máximo de gerações deve ser estabelecido. Esse valor será outro modo de parada do algoritmo, uma vez que a heurística ultrapasse esse limite de gerações/iterações e um indivíduo com a pontuação requerida ainda não tenha sido encontrado. Nesse caso, a execução será caracterizada como falha, pois não obteve uma solução válida para o problema.

4.3 Ambiente e Casos de Teste

Para fins de validação, assumimos que um bom alocador deve promover melhor utilização de recursos, um tempo de execução que não comprometa o desempenho do sistema como um todo, além de ter uma boa vazão de tarefas alocadas. A fim de promover uma análise confiável do algoritmo genético, realizaremos diversas mudanças em alguns de seus parâmetros, seja de entrada ou de controle, e monitoraremos diversas medidas com o objetivo de respondermos as questões de pesquisa apresentadas no início desta dissertação. Em um primeiro momento, realizaremos os seguintes testes:

- Execução com um banco de *cloud instances* pequeno (50 entradas), médio (250 entradas) e grande ($\geq$500 entradas);
- Execução com uma lista de entradas/Cromossomo unitária (1 tarefa independente), pequena (3 tarefas comunicantes), média (6 tarefas comunicantes), grande (12 tarefas comunicantes);
- Execução de tarefas leves (*loosely-constrained*), intermediárias (*medium-constrained*) e pesadas (*tightly-constrained)*;
- Execução do alocador nas categorias C, B e A, ou seja, execuções onde o usuário exige baixo (*Fitness Score* > 0), médio (*Fitness Score* ≥ 0.5), e alto nível de otimização (*Fitness Score* entre 0.8 e 1), respectivamente.

Veremos no próximo capítulo, quais os resultados obtidos e suas análises. Tentaremos traçar um paralelo entre todos essas execuções a fim de chegarmos a uma conclusão válida sobre a utilização de algoritmos genéticos na alocação de tarefas e gerenciamento de recursos.

A plataforma em que os testes foram realizados tem a seguinte configuração: Intel Core i3 CPU M 350 @ 2.27GHz, 3.0GB RAM, Intel HD Graphics.

Capítulo 5

Avaliação

5.1 Resultados

Para validação da solução proposta, apresentamos um caso de teste a fim de simular o comportamento da heurística em uma situação real. Separamos este caso em dois ambientes distintos, objetivando descobrir qual o impacto da variação de diferentes parâmetros no desempenho e na qualidade das soluções geradas pelo AG. Para tanto, fizemos algumas alterações no algoritmo, para que o mesmo fosse capaz de executar o procedimento de alocação 10000 vezes, a fim de gerar um intervalo de confiança aceitável, suprindo as deficiências do estudo que poderiam ser causadas pelo comportamento não-determinístico da heurística proposta. Os parâmetros do AG foram escolhidos empiricamente, utilizando o método de observação de convergência para defini-los. Os parâmetros genéticos utilizados são descritos na tabela abaixo:

Parâmetro Genético	**Valor (Ambiente 1)**	**Valor(Ambiente 2)**
Número de Execuções	10000	10000
Número de Tarefas	6	1/3/6/12
Número de Instâncias	50 / 250 / 500	50
Tamanho da População	20	20
Tamanho do Cromossomo	6	1/3/6/12
Máximo de Gerações	300	300
Taxa de *Crossover*	0.9	0.9
Tipo de Crossover	1 ponto	1 ponto
Taxa de Mutação	0.05	0.05
Tipo de Mutação	Substituição Simples	Substituição Simples

Tabela 5.1: Parâmetros Genéticos para o caso de teste

Resultado Qualitativo

Para ilustrar uma saída gerada pelo algoritmo proposto, temos na Tabela 5.2 uma requisição que serviu de entrada para o sistema, e logo abaixo, na Tabela 5.3, temos as três máquinas

retornadas pelo alocador, em três diferentes modos de operação. As pontuações de aptidão obtidas foram 0.9449 no Custo-Benefício, 0.7122 no modo Disponibilidade e 0.8786 no modo Desempenho.

Tabela 5.2: Requisição feita pelo usuário

#Instância	#Núcleos	Memória(GB)	Budget/Hora	Disponibilidade
0	3.8	7.0	2.000	0.9

Tabela 5.3: Máquinas retornadas pelo algoritmo proposto

Modo de operação	#Núcleos	Memória(GB)	Custo/Hora	Disponibilidade	Nome
Custo-Benefício	4.0	7.5	0.210	0.9	c3.xlarge
Disponibilidade	32.0	60.0	1.680	0.99999	c3.8xlarge
Desempenho	32.0	60.5	2.000	0.99999	cc2.8xlarge

Ambiente 1

Nas Tabelas 5.4, 5.5 e 5.6 podemos ver a quantização de alguns parâmetros quando o sistema é alimentado com tarefas leves (pouco recurso requisitado), médias (recurso moderado requisitado) e pesadas (alto nível de recurso requisitado), respectivamente. É importante ressaltar que a classificação das tarefas são baseadas no próprio banco de *clouds*, ou seja, tomamos as instâncias de máquina mais fracas e definimos uma tarefa que pode ser executada naquela máquina (tarefa leve), por outro lado, pegamos as máquinas mais poderosas e definimos uma tarefa que requisite recursos mais condizentes com tal instância (tarefas pesadas), enquanto as tarefas medianas tem configurações no meio termo das outras duas classificações de tarefas.

Para que os testes sejam viáveis, estabelecemos para cada nível da lista de tarefas, um orçamento suficiente para contratar qualquer máquina com características compatíveis, logo, ficou definido para as tarefas de baixa e média complexidade, um *budget* de 2,00US$/hora, enquanto para as tarefas pesadas, o usuário está disposto a pagar 4,00US$/hora. Para esse ambiente fixamos a lista de tarefas em 6 requisições interdependentes.

Ambiente 2

Para o segundo ambiente, mostrado na Tabela 5.1, utilizaremos uma metodologia semelhante ao ambiente 1. Há diferença apenas na alimentação do sistema e no tamanho do cromossomo. Enquanto no primeiro ambiente variávamos o número de instâncias de máquinas, no segundo caso, manteremos esse valor fixo e trocaremos o número de requisições juntamente com o tamanho do cromossomo, ou seja, o número de tarefas interdependentes submetidas pelo usuário em cada ciclo. Basicamente, nosso banco de *clouds* é fixado em 50 instâncias, enquanto o tamanho da lista de requisições assumirá os valores de uma, três, seis ou doze tarefas a serem alocadas simultaneamente. Os resultados obtidos podem ser observados na Tabela 5.7.

Número de Instâncias	Categoria de Otimização Exigida	Modo de Operação	Soluções Otimizadas(%)	*Fitness Score* Médio	Tempo Médio (ms)
50	C	Custo-Benefício	100.00	0.4216	0.0629
		Disponibilidade	100.00	0.7285	0.0474
		Desempenho	100.00	0.7975	0.0666
	B	Custo-Benefício	100.00	0.5111	0.2446
		Disponibilidade	100.00	0.7472	0.0499
		Desempenho	100.00	0.8143	0.0495
	A	Custo-Benefício	0.00	0.0000	12.1739
		Disponibilidade	0.00	0.0000	12.3521
		Desempenho	100.00	0.8323	0.0610
250	C	Custo-Benefício	100.00	0.3911	0.0759
		Disponibilidade	100.00	0.7182	0.0464
		Desempenho	100.00	0.8853	0.0471
	B	Custo-Benefício	100.00	0.5088	0.3891
		Disponibilidade	100.00	0.7612	0.0419
		Desempenho	100.00	0.7900	0.0414
	A	Custo-Benefício	0.00	0.0000	12.9708
		Disponibilidade	0.00	0.0000	10.3284
		Desempenho	100.00	0.8479	0.0492
500	C	Custo-Benefício	100.00	0.3728	0.0543
		Disponibilidade	100.00	0.7775	0.0635
		Desempenho	100.00	0.8062	0.0530
	B	Custo-Benefício	100.00	0.5145	0.3525
		Disponibilidade	100.00	0.7485	0.0527
		Desempenho	100.00	0.8681	0.0442
	A	Custo-Benefício	0.00	0.0000	12.2204
		Disponibilidade	0.00	0.0000	9.5208
		Desempenho	100.00	0.8495	0.0489

Tabela 5.4: Resultados de 10000 execuções do AG com uma lista de tarefas leves (*loosely-constrained*).

5.2 Análises

Como pudemos notar, existe uma grande variedade de parâmetros que influenciam os resultados do algoritmo genético, faremos uma análise desses resultados nesta seção, tentando extrair algumas informações a respeito do comportamento do sistema, as quais nos darão base para validar ou refutar as hipóteses levantadas no início desta monografia. Dividiremos essa análise em algumas etapas, sendo que cada uma estudará o impacto de um parâmetro diferente, tendo como base a qualidade da solução e tempo de execução.

Número de Instâncias	Categoria de Otimização Exigida	Modo de Operação	Soluções Otimizadas(%)	*Fitness Score* Médio	Tempo Médio (ms)
50	C	Custo-Benefício	80.87	0.6192	1.1906
		Disponibilidade	86.61	0.6331	0.6098
		Desempenho	90.39	0.6500	0.4573
	B	Custo-Benefício	29.16	0.6247	3.2557
		Disponibilidade	66.44	0.6233	1.5110
		Desempenho	100.00	0.6577	0.0388
	A	Custo-Benefício	16.10	0.8127	3.6101
		Disponibilidade	100.00	0.0000	2.9469
		Desempenho	100.00	0.0000	6.5552
250	C	Custo-Benefício	80.11	0.6606	1.2362
		Disponibilidade	90.95	0.6236	0.4083
		Desempenho	29.80	0.6598	3.2220
	B	Custo-Benefício	35.57	0.6540	3.7467
		Disponibilidade	94.91	0.6421	0.3106
		Desempenho	45.98	0.6229	2.6416
	A	Custo-Benefício	21.06	0.8359	2.4664
		Disponibilidade	0.00	0.0000	3.4949
		Desempenho	0.00	0.0000	4.4615
500	C	Custo-Benefício	61.57	0.6015	2.2467
		Disponibilidade	62.44	0.6155	2.1662
		Desempenho	32.28	0.6405	3.2239
	B	Custo-Benefício	59.49	0.5861	2.5329
		Disponibilidade	78.15	0.5992	1.4111
		Desempenho	28.09	0.6625	3.3335
	A	Custo-Benefício	35.94	0.8096	2.9063
		Disponibilidade	0.00	0.0000	4.9380
		Desempenho	0.00	0.0000	3.8615

Tabela 5.5: Resultados de 10000 execuções do AG com uma lista de tarefas medianas (*medium-constrained*).

Espaço de Busca (Banco de *Clouds*)

Uma das grandes qualidades do algoritmo genético é a capacidade de se adaptar e oferecer soluções de qualidade em problemas de variados espaços de busca. Na maioria dos alocadores, o método de busca utilizado é sequencial ou binário, os quais sabemos que têm uma queda considerável de desempenho caso o espaço de busca seja muito expandido.

Baseado nessa premissa, cruzamos alguns dados e chegamos a conclusão que, para o nosso caso de estudo, esse fenômeno realmente ocorre, e a heurística se mostrou robusta para maiores bancos de *clouds*. O tamanho padrão de um banco de máquinas contém 50 instâncias, quintuplicamos e decuplicamos esse banco e obtivemos resultados dentro da mesma margem de

Número de Instâncias	Categoria de Otimização Exigida	Modo de Operação	Soluções Otimizadas(%)	*Fitness Score* Médio	Tempo Médio (ms)
50	C	Custo-Benefício	47.20	0.6928	2.2730
		Disponibilidade	28.72	0.6163	3.4266
		Desempenho	23.97	0.6369	2.8329
	B	Custo-Benefício	47.47	0.5991	2.5138
		Disponibilidade	30.43	0.6191	3.7100
		Desempenho	28.52	0.6709	4.5831
	A	Custo-Benefício	4.60	0.8175	4.2808
		Disponibilidade	0.00	0.0000	4.7055
		Desempenho	0.00	0.0000	5.4767
250	C	Custo-Benefício	27.88	0.6311	3.0073
		Disponibilidade	69.59	0.6200	2.2056
		Desempenho	46.34	0.6790	2.6953
	B	Custo-Benefício	54.52	0.5666	2.9205
		Disponibilidade	61.06	0.6091	2.5371
		Desempenho	47.84	0.6799	2.5694
	A	Custo-Benefício	2.84	0.8098	4.4670
		Disponibilidade	0.00	0.0000	4.5936
		Desempenho	0.00	0.0000	4.5331
500	C	Custo-Benefício	84.40	0.6464	0.6006
		Disponibilidade	57.61	0.6169	2.3235
		Desempenho	55.77	0.6903	1.9149
	B	Custo-Benefício	59.80	0.5844	2.1986
		Disponibilidade	25.25	0.6161	3.1364
		Desempenho	55.70	0.6302	1.9349
	A	Custo-Benefício	1.51	0.8430	2.8789
		Disponibilidade	0.00	0.0000	4.4745
		Desempenho	0.00	0.0000	4.8094

Tabela 5.6: Resultados de 10000 execuções do AG com uma lista de tarefas pesadas(*tightly-constrained*).

tolerância. É importante ressaltar que a expansão da base de dados se deu de forma homogênea, ou seja, a proporção entre as diferentes configurações foi mantida, logo, se no primeiro banco temos uma instância X, no segundo temos 5 instâncias de mesma configuração de X, e no terceiro, 10 instâncias.

Para entendermos melhor, utilizamos como exemplo, o modo de **operação Custo-Benefício**, **categoria B** como nível de qualidade exigido (FS ≥ 0.5) e variando o tamanho da lista de máquinas, obtivemos os seguintes resultados para tarefas leves, médias e pesadas (Figuras 5.1, 5.2 e 5.3):

Pudemos observar na figura 5.1 que ao contrário do que se espera de um escalonador, o volume de sucessos na busca por alocações otimizadas em nossa proposta se manteve em um

Número de Tarefas	Categoria de Otimização Exigida	Modo de Operação	Soluções Otimizadas(%)	*Fitness Score* Médio	Tempo Médio (ms)
1	C	Custo-Benefício	100.00	0.3090	0.0173
		Disponibilidade	100.00	0.6044	0.0208
		Desempenho	100.00	0.7691	0.0186
	B	Custo-Benefício	100.00	0.9515	0.0165
		Disponibilidade	100.00	0.6161	0.0259
		Desempenho	100.00	0.6274	0.0205
	A	Custo-Benefício	100.00	0.9515	0.0204
		Disponibilidade	0.00	0.0000	2.8832
		Desempenho	100.00	0.8515	0.0177
3	C	Custo-Benefício	100.00	0.5387	0.3681
		Disponibilidade	100.00	0.6370	0.1186
		Desempenho	100.00	0.6816	0.5028
	B	Custo-Benefício	100.00	0.5714	0.0530
		Disponibilidade	100.00	0.5706	0.0306
		Desempenho	100.00	0.6234	0.0725
	A	Custo-Benefício	100.00	0.6582	0.0349
		Disponibilidade	0.00	0.0000	5.1331
		Desempenho	16.39	0.8018	5.1614
6	C	Custo-Benefício	80.87	0.6192	1.1906
		Disponibilidade	86.61	0.6331	0.6098
		Desempenho	90.39	0.6500	0.4573
	B	Custo-Benefício	29.16	0.6247	3.2557
		Disponibilidade	66.34	0.6233	1.5110
		Desempenho	100.00	0.6577	0.0388
	A	Custo-Benefício	16.10	0.8127	3.6101
		Disponibilidade	0.00	0.0000	2.9469
		Desempenho	0.00	0.0000	6.5552
12	C	Custo-Benefício	0.00	0.0000	5.6668
		Disponibilidade	0.00	0.0000	5.0610
		Desempenho	0.00	0.0000	4.6549
	B	Custo-Benefício	0.00	0.0000	4.7670
		Disponibilidade	0.00	0.0000	4.7276
		Desempenho	0.00	0.0000	6.1330
	A	Custo-Benefício	0.00	0.0000	4.8290
		Disponibilidade	0.00	0.0000	5.1028
		Desempenho	0.00	0.0000	6.3325

Tabela 5.7: Resultados de 10000 execuções do AG variando o tamanho da lista de tarefas interdependentes (1/3/6/12).

mesmo patamar, e neste modo de operação em particular, tivemos até um decréscimo no número de falhas em relação a expansão do espaço de busca, ou seja, queda no número de execuções que

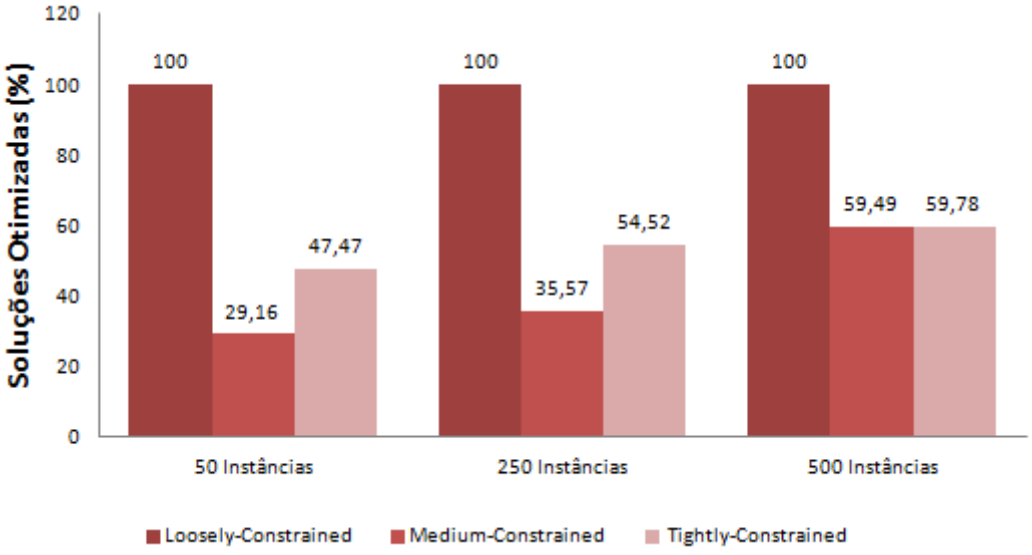

Figura 5.1: Taxa percentual de soluções otimizadas encontradas em 10000 execuções para diferentes espaços de busca

não retornaram algum resultado.

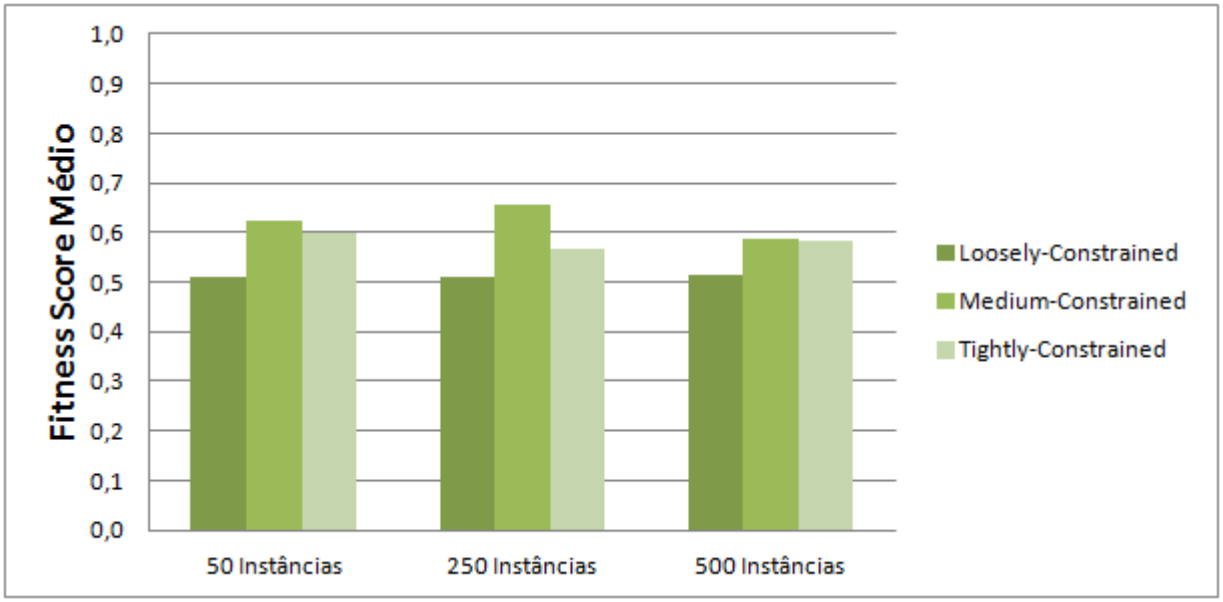

Figura 5.2: *Fitness Score* médio para diferentes espaços de busca

Como ilustrado na figura 5.2, a pontuação de aptidão média se manteve relativamente constante com a variação da base de dados. Como foi estipulada uma faixa de qualidade média para esse caso de teste, tivemos soluções com *Fitness Scores* flutuando entre 0.5 e 0.6, verificando uma média levemente maior em uma base de 50 instâncias. Mais precisamente, para uma base de **50** diferentes tipos de máquinas obtivemos uma pontuação média de **0.5783**, para uma base um pouco maior, com **250** entradas, tivemos uma pontuação média de **0.5765**, enquanto para um

banco de **500** instâncias obtivemos uma pontuação média de **0.5617**. Notamos que a diferença entre as pontuações é praticamente desprezível, tendo em vista o espectro total que essa métrica pode assumir.

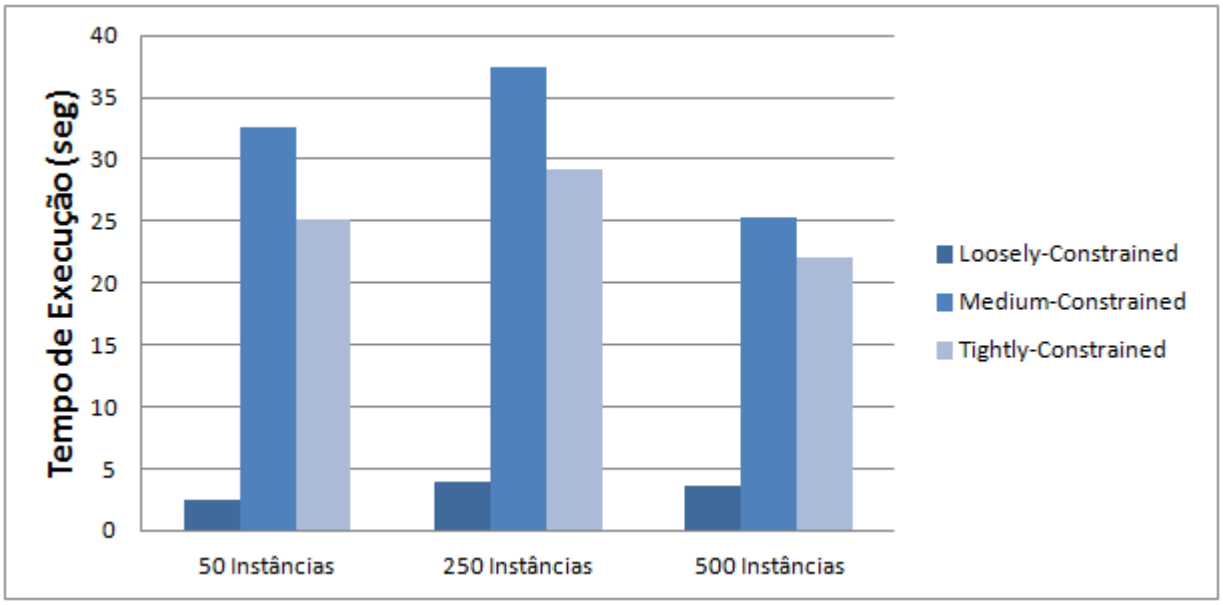

Figura 5.3: Tempo de 10000 execuções para diferentes espaços de busca

Como último parâmetro de estudo nesse caso de teste, utilizamos o tempo como comparativo. A primeira vista, já podemos notar certa relação entre número de falhas e o tempo total obtido nas 10000 execuções. Se compararmos os resultados apresentados na Figura 5.1 com os dados da Figura 5.2, essa relação fica ainda mais evidente e pode ser explicada com base no funcionamento do algoritmo. Basicamente, um caso é classificado como sucesso quando uma solução é encontrada pelo algoritmo genético dentro do máximo de gerações estabelecido, isso faz com que ele aborte as futuras iterações e retorne o resultado ao usuário. Por outro lado, atribuir fracasso à uma execução, significa dizer que a heurística precisa iterar até o limite máximo de gerações definido e notificar o usuário que nenhuma solução dentro das especificações foi encontrada, consumindo o máximo tempo possível de computabilidade limitado pelo algoritmo. Enfim, vimos que o tempo de execução varia bastante entre os diferentes tipos de tarefa (leves, intermediárias e pesadas), entretanto, isso será estudado mais adiante. Quanto ao foco dessa seção, percebemos que o tempo de execução teve uma variação razoável no banco de 250 instâncias, diretamente causada pelo número de falhas. Contudo, percebemos que o tempo médio no banco de 500 máquinas foi o menor de todos, portanto, não podemos afirmar que o tempo de execução escala com o tamanho do banco de máquinas. Da mesma forma, a diferença no padrão do tempo no segundo banco de dados pode ser atribuído às configurações e características das tarefas e das máquinas, e não ao tamanho do espaço de busca.

Configuração das Requisições

Neste tópico, tentaremos inferir qual o impacto que as diferentes configurações das tarefas podem causar no desempenho do algoritmo. Para amostrar, tomamos as execuções referentes aos três diferentes modos de operação (Custo-Benefício, Disponibilidade e Desempenho), fixando um nível mínimo de qualidade exigido (Categoria C) e uma base de *clouds* com 50 instâncias. Embora a configuração das tarefas tenha sofrido variação, o tamanho da lista adotada ficou estática em seis entradas interdependentes, as quais devem ser alocadas simultaneamente.

Utilizou-se três diferentes tipos de listas de tarefa, a primeira é classificada como *loosely-constrained*, que consiste em tarefas de requisições leves de recursos, ou seja, tarefas que poderiam ser executadas por 90% dos tipos de máquinas sem violar alguma *constraint*. O segundo tipo, *medium-constrained*, é uma lista de requisições intermediárias de recurso, ou seja, tarefas que poderiam ser executadas por aproximadamente 50% das máquinas, sem violação de *constraints*. Por último, estão as tarefas *tightly-constrained*, lista composta por tarefas que só podem ser executadas por um seleto grupo de máquinas, cerca de 10% das instâncias disponíveis, sem que uma regra de negócio seja violada. Os resultados da amostra analisada estão representados na Figura 5.4.

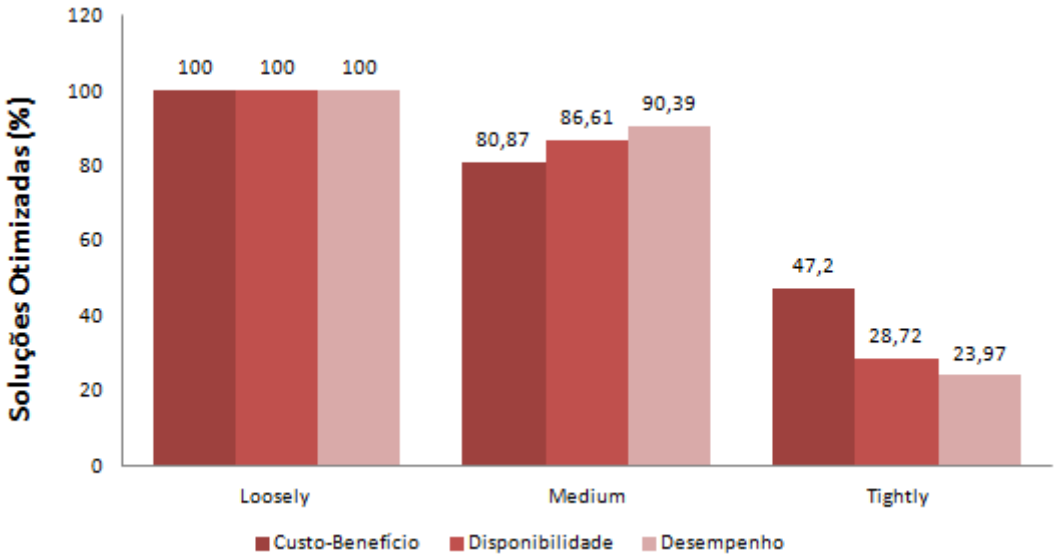

Figura 5.4: Taxa percentual de soluções otimizadas encontradas em 10000 execuções para diferentes configurações de tarefas

O resultado demonstrado na Figura 5.4 expressa o comportamento do algoritmo para todos os casos onde variamos a configuração das requisições. Sem grande esforço, notamos que as execuções das tarefas mais leves (*loosely-constrained*) são isentas de falhas, nas tarefas intermediárias (*medium-constrained*) um número razoável de erros já começa a ser notado, enquanto nas requisições mais pesadas (*tightly-constrained*) a taxa de falhas beira a extrapolação. Como vimos anteriormente, o tempo de execução segue o padrão de falhas, em tarefas pesadas, isso nos gerou execuções que demoram de 4 milissegundos em casos mais leves até 6 milissegundos em

situações mais severas. Também baseado nisso, trazemos o conceito de Mean Time To Failure (MTTF), que representa o tempo médio entre falhas do algoritmo. Observamos que o tempo médio entre falhas em requisições pesadas são de ordem 10 vezes menor do que em aplicações leves, ou seja, em um intervalo de tempo Δt, a heurística tende a falhar 10 vezes mais com tarefas *tightly-constrained* do que com tarefas *loosely-constrained*.

Esse tópico nos traz uma importante conclusão a respeito das limitações dessa abordagem. Um dos fatores que mais impactaram o bom funcionamento do algoritmo baseia-se na configuração das tarefas em relação às instâncias disponíveis. Em termos gerais, o algoritmo se vale de suas propriedades aleatórias para encontrar combinações em todo espectro de possíveis soluções. A medida que definimos *constraints* de execução e aumentamos a complexidade de nossas tarefas tornando-as (*tightly-constrained*), reduzimos o universo de soluções válidas, e por consequência, diminuímos o número de execuções bem sucedidas. Em aplicações leves, a limitação não está nas *constraints*, ou seja, qualquer combinação estabelecida em um cromossomo é computacionalmente factível, logo, o único limitador de soluções será o próprio nível de qualidade exigido pela *Fitness Score*.

Nível de Qualidade Exigido

Nesta etapa, faremos uma breve análise da heurística quando variamos o nível de qualidade exigido das alocações. No capítulo anterior, definimos que o usuário pode restringir as respostas retornadas pelo sistema em três diferentes classificações: categoria C (FS>0), categoria B (FS$\geq$0.5) e categoria A (FS$\geq$0.8). É difícil gerar uma análise confiável nesse tipo de cenário, uma vez que a performance e a qualidade das soluções são altamente sensíveis às características das tarefas e das instâncias disponíveis, ainda assim, em nosso caso de estudo, o resultado obtido seguiu um padrão intuitivamente esperado. Assumindo parâmetros idênticos ao teste anterior, escolhendo uma configuração de tarefas *medium-constrained* e contrastando diferentes modos de operação com diferentes categorias de qualidade exigidas, obtivemos os resultados explicitados na figura 5.5.

Na prática, se um nível **C** de qualidade é exigido, o espaço de possíveis soluções é o maior possível, ou seja, qualquer combinação que não viole alguma *constraint* será aceita como solução. Por outro lado, dependendo das características das tarefas e das instâncias de máquina disponíveis, apenas algumas combinações terão pontuações de alto nível, logo, o número de falhas tende a crescer quando exigimos um nível alto de qualidade das soluções. Entretanto, o objetivo do nosso trabalho é oferecer uma alternativa viável para que as alocações de recursos sejam realizadas com maiores níveis de qualidade, satisfazendo o cliente e gerando mais lucro para a provedora. Sabendo disso, devemos traçar uma estratégia diferente para contornar o padrão mostrado no gráfico 5.5.

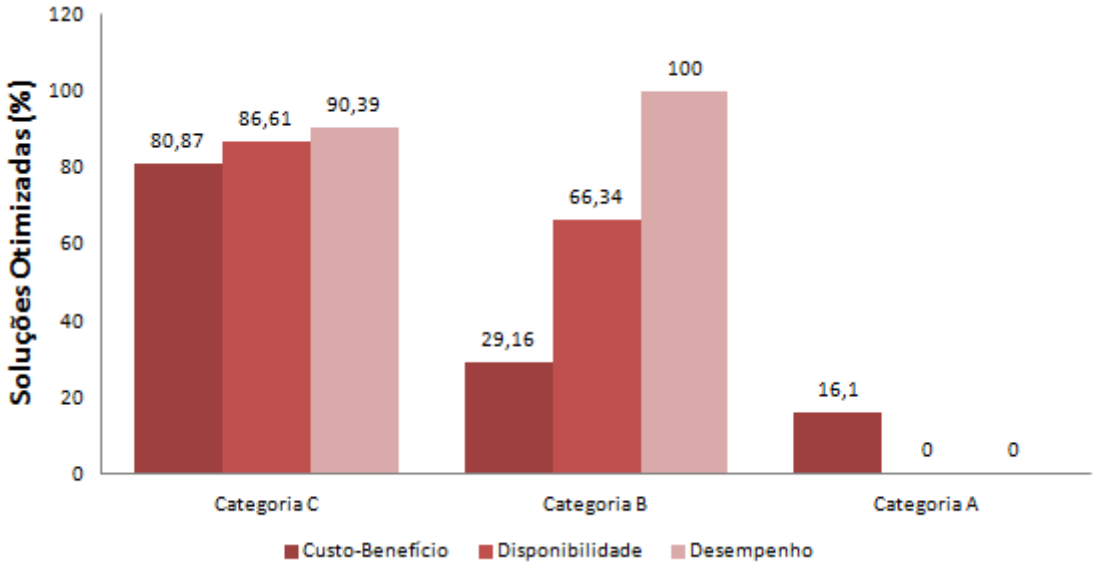

Figura 5.5: Taxa percentual de soluções otimizadas encontradas em 10000 execuções para diferentes níveis de qualidade exigidos

Tamanho da Lista de Tarefas e do Cromossomo

Começamos aqui as execuções do segundo ambiente do caso de teste. Como introduzido no início desse capítulo (subseção 5.1), buscamos saber aqui qual o tamanho ideal que nossa lista de requisições deve ter para que a abordagem de alocação multiobjetivo seja viável. Para tal, executamos o algoritmo 10000 vezes para quatro diferentes tamanhos de lista de requisições, sendo a primeira lista unitária, ou seja, com apenas uma tarefa independente, e as demais listas com três, seis e doze tarefas dependentes entre si. As características do ambiente são semelhantes as do teste anterior, tarefas intermediárias (*medium-constrained*), exigindo um nível **B** de otimização (FS $\geq$ 0.5) e utilizando 50 instâncias de máquina como base de dados das nuvens. Variando o número de tarefas em relação aos três diferentes tipos de modos de operação, obtivemos os gráficos evidenciando a taxa de sucesso, pontuação de aptidão média e tempo médio de execução 5.6, 5.7 e 5.8.

De acordo com a Figura 5.6, verificamos que existe um padrão no crescimento do número de falhas a medida que expandimos a lista de tarefas à serem alocadas simultaneamente. Notamos que para esse modelo, deveríamos optar por uma lista com tarefas independentes, ou uma lista de três tarefas quando as mesmas são interdependentes, uma vez que o algoritmo foi executado dez mil vezes e nenhuma delas foi infrutífera. Se o único parâmetro a disposição fosse o número de falhas, certamente deveríamos escolher a lista de 3 tarefas, uma vez que todas as execuções sucederam, e a heurística ainda possuiria um *throughput* de alocação 3 vezes maior do que uma lista unitária. Entretanto, como veremos nos próximos gráficos, há outros fatores que devem ser colocados na balança para definirmos uma configuração ideal para o planejador genético.

Dado que definimos um nível **B** de qualidade exigido, a média da pontuação de aptidão

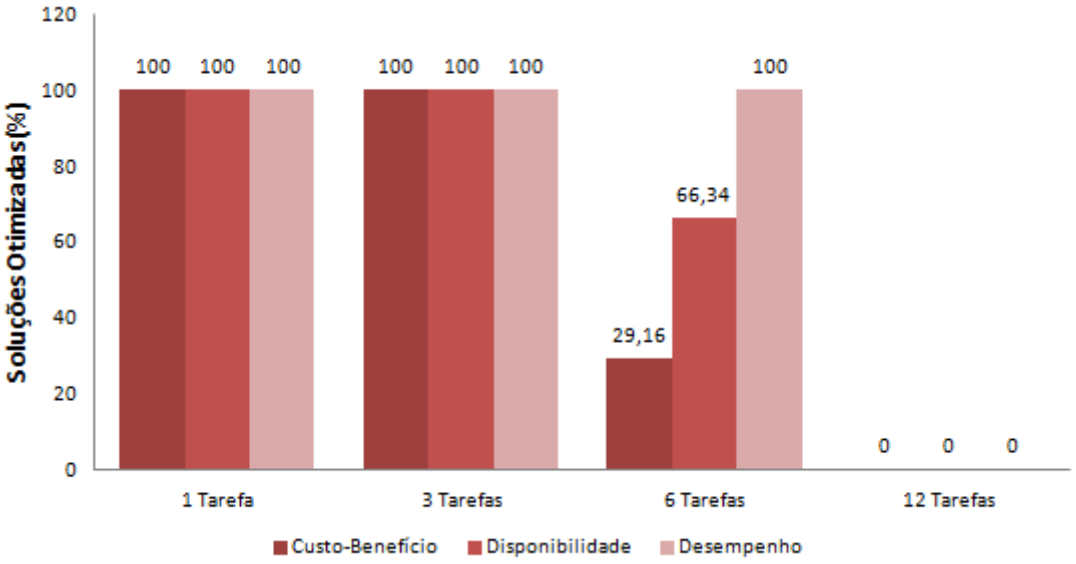

Figura 5.6: Taxa percentual de soluções otimizadas encontradas em 10000 execuções para diferentes tamanhos de lista de tarefas

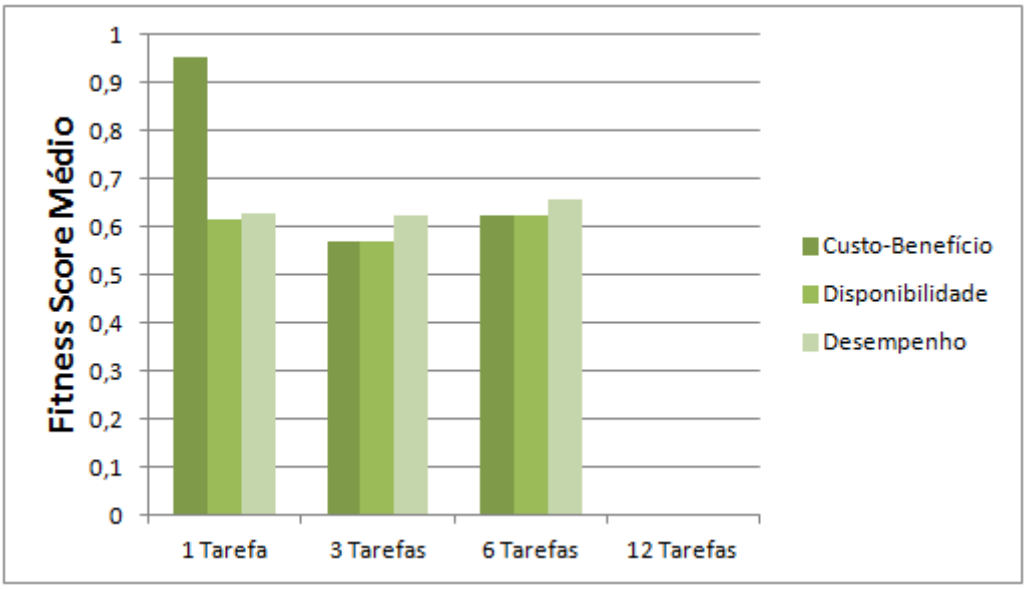

Figura 5.7: *Fitness Score* médio de 10000 execuções para diferentes tamanhos de lista de tarefas

se manteve constante entre os diferentes tamanhos de lista de requisições. Com uma lista de doze tarefas, a média obtida foi zero, pois nenhuma das dez mil execuções foi bem sucedida, em nenhum modo de operação. Na prática, esse fenômeno pode ser facilmente explicado. Sabemos que a pontuação de uma combinação é o somatório da aptidão de cada gene, ou seja, da alocação de uma tarefa para uma máquina. Sabemos também, que se qualquer um dos genes que compõem o cromossomo violar uma *constraint*, a pontuação de todo indivíduo será zerada automaticamente. Estatisticamente falando, quanto mais genes adicionarmos, ou seja, quanto

maior for o nosso cromossomo, maior a probabilidade de alguma combinação violar uma regra, logo, dificilmente um indivíduo de grande porte terá uma formação viável.

Um fenômeno interessante observado no gráfico 5.7, é que a composição da pontuação de todos os genes formará a *fitness score* do indivíduo, dessa forma, a pontuação de um gene poderá puxar a pontuação total para baixo, dificultando que um indivíduo de grande porte atinja um alto nível de qualidade. Devido a esse fato, percebemos um pico na pontuação média do modo Custo-Benefício no cromossomo unitário. Através de outros testes, percebemos que a utilização de uma lista unitária retornou menos falhas, principalmente quando exigimos um nível de qualidade alto, e as falhas apontadas estavam relacionadas às características das tarefas e instâncias, ou seja, se uma execução falhou, a probabilidade de que não exista uma combinação válida para aquela pontuação exigida é muito alta, forçando o usuário a diminuir seu critério de qualidade ou a provedora de inserir máquinas com novas configurações.

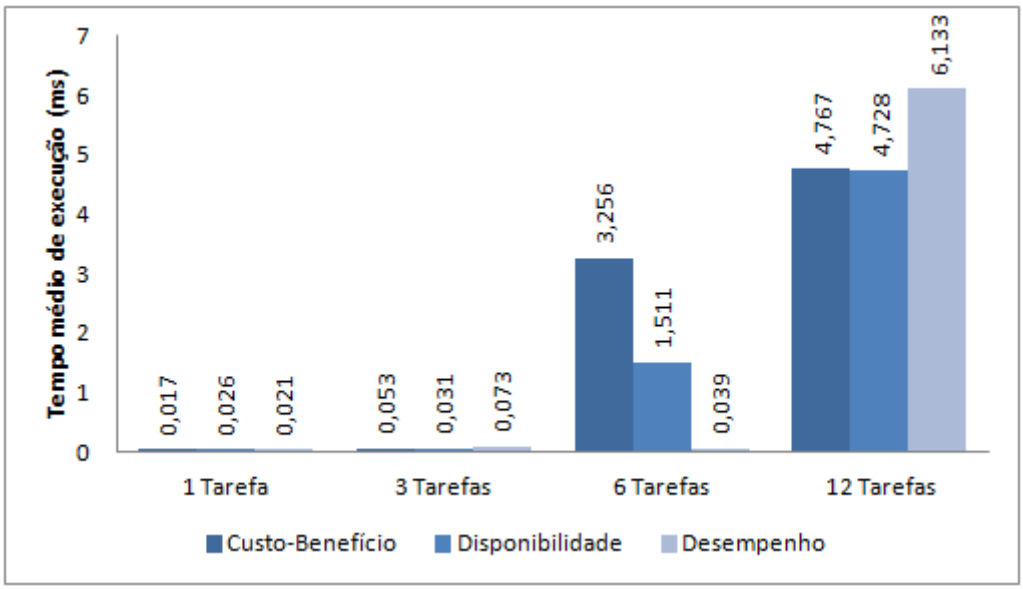

Figura 5.8: Tempo médio de execução para diferentes tamanhos de lista de tarefas

Como inferimos anteriormente, o tempo de execução está relacionado diretamente à quantidade de falhas, isso explica a grande diferença entre os tempos médios de execução com uma lista de uma e três tarefas em relação às listas de seis e doze tarefas. A explicação desse fenômeno já foi explicado anteriormente. O que nos chama atenção aqui é que na ausência de falhas, o tempo médio de execução de uma requisição é equiparável ao tempo médio de execução de uma lista de três requisições. Utilizaremos isso mais adiante, na criação de um modelo ideal para o nosso alocador.

Parâmetros Genéticos

Além dos diferentes cenários apresentados acima, podemos adicionar mais uma camada de complexidade ao sistema, simplesmente variando os parâmetros genéticos. Como explicado anteriormente, os parâmetros genéticos escolhidos, são consolidados na literatura para a resolução

de diversos tipos de problemas. Logicamente, outras combinações foram testadas no desenvolvimento deste trabalho, porém não geraram resultados satisfatórios. Um exemplo disso é o tamanho da população, ao contrário do que podemos imaginar, aumentar o tamanho de uma população não necessariamente melhorará o desempenho do algoritmo. No nosso caso de estudo, por exemplo, ao aumentar o tamanho da população para 100 indivíduos em cromossomos de médio porte, que não é um valor tão alto, o tempo total das 10000 execuções ultrapassaram cinco minutos em alguns casos, e uma situação real não pode suportar um gargalo dessa magnitude. Isso ocorre também com outros parâmetros como *taxa de crossover*, *taxa de mutação* e *número de gerações*.

5.3 Visão Geral

Baseado em todos os testes e observações apontados na seção anterior, temos agora uma base para definir quais as limitações devemos evitar, além da capacidade de criar um modelo de alocação mais robusto e eficiente. Nesta seção daremos um parecer a respeito das questões de pesquisa levantadas no capítulo de introdução.

- **QP1: Dada uma configuração mínima definida por um usuário, a ser processada por uma nuvem, é vantajoso utilizar uma abordagem evolucionária na alocação de tarefas e gerenciamento de recursos frente à busca baseada em *constraints*?**

De modo geral, percebemos que a heurística teve um bom desempenho na otimização do problema de planejamento de recursos. Embora algumas situações tenham retornado 100% de falha, podemos afirmar, com auxílio dos testes exaustivamente realizados, que nesses casos o resultado certamente não existe no nosso espaço de busca. Dessa forma, apesar da característica não determinística da heurística, é factível estabelecer um intervalo de confiança no qual uma solução tenderá a aparecer.

Provamos que o tamanho do espaço de busca tem uma influência mínima no desempenho da heurística. Influência que nem se compara ao impacto da variação da base de dados em outros métodos de alocação. Por outro lado, sabemos que esse tipo de problema geralmente conta com bases de dados pequenas, e um buscador baseado em satisfação de *constraint* como o **CHOCO Solver** [3] poderia lidar com isso sem dificuldades. Entretanto, duas vantagens são claras no uso da abordagem evolucionária: escalabilidade e flexibilidade na escolha de soluções, ou seja, nenhum outro método de alocação pode oferecer diferentes modos de busca (modos de operação), com a mesma facilidade e agilidade proporcionada pelo algoritmo genético.

Em um segundo momento, percebemos que diferentes configurações de tarefas têm impacto considerável no desempenho da heurística. Em um mundo ideal, deveríamos ter um banco personalizado para cada classe de tarefas (leves, intermediárias e pesadas), ou seja, a heurística deveria ser isenta de tratar de problemas de satisfatibilidade, e se ocupar unicamente em prover

soluções de melhor qualidade. Como isso é inviável em um caso real, a alternativa encontrada é criar uma estrutura híbrida de busca entre algoritmos genéticos e métodos de buscas baseados em *constraints*. Basicamente, em uma primeira rodada, o algoritmo convencional separaria todas as possíveis máquinas que poderiam atender a tarefa sem violação de uma regra de negócio, gerando um novo espaço de busca mais limitado. Seguido a isso, o algoritmo genético executaria suas rotinas tomando o novo banco de máquinas para selecionar a solução que melhor atenderia aos requisitos preconizados pelo modo de operação.

O nível de qualidade exigido, assim como os diferentes modos de operação são *features* que teriam espaço em um alocador ideal. Notamos que, desde que a função de aptidão seja bem definida, essas opções só acrescentam valor ao modelo proposto, fornecendo ferramental ao usuário para que o mesmo decida que tipo de solução procura e qual o nível de qualidade essa solução deve manter, se responsabilizando por um possível *trade-off* com o tempo de execução.

O tamanho do cromossomo se mostrou um real empecilho na robustez e eficiência do alocador genético. Percebemos que para cromossomos com mais de três tarefas interdependentes, a taxa de falhas e tempo de execução aumentam, e fatalmente se tornariam um gargalo em um sistema real. Como mencionado anteriormente, a pontuação de um gene interfere diretamente na pontuação total do cromossomo, nesse caso, o esquema de alocação multiobjetivo tem uma desvantagem considerável. Podemos pensar de imediato em duas medidas para contornar essa limitação. A primeira medida consiste em hibridizar o sistema genético com outro método de busca baseado em *constraint*, como já sugerido anteriormente, gerando assim um espaço de busca intermediário otimizado, e dessa maneira, como não haverá violação de *constraints*, um gene não terá a capacidade de zerar a pontuação de todo o indivíduo. O segundo método baseia-se na ideia de que cada tarefa é independente, ou seja, a alocação de um recurso para uma tarefa não se relaciona com a alocação dos outros recursos para as outras tarefas, mesmo que tenham sido requisitadas por um mesmo usuário. Sabendo disso, podemos deixar o método de alocação multiobjetivo de lado e adotarmos uma abordagem de planejamento por fila de requisição, caso as tarefas submetidas sejam independentes entre si e não necessitem de alocações simultâneas. Notamos nas Figuras 5.6, 5.7 e 5.8 que as execuções com um cromossomo unitário foi o que nos trouxe melhor qualidade média, robustez e desempenho em comparação com os outros testes. Logicamente, estaríamos adicionando outras complicações ao sistema, como gargalos relacionados a taxa de chegada, por exemplo, e um estudo mais aprofundado sobre isso seria necessário.

- **QP2: Em que grau o comportamento não determinístico de um algoritmo evolucionário pode afetar o desempenho de um alocador de tarefas ou gerenciador de recursos em um sistema de *Cloud Computing*?**

Quanto ao padrão estocástico encontrado em algumas partes da heurística, podemos afirmar que ele não interfere de maneira sensível no funcionamento da alocação de recursos. Um estudo probabilístico seria muito bem vindo nesse tipo de trabalho, entretanto, os resultados obtidos seriam bem específicos ao caso. Sabemos que as características das tarefas e das máquinas são

os grandes responsáveis pelo sucesso ou fracasso de uma execução, ainda assim, diferentes codificações, parâmetros genéticos, modos de operação, etc., terão diferentes impactos na execução do algoritmo. No caso de estudo utilizado neste trabalho, podemos usar o MTTF para estabelecer um intervalo de confiança para execução do algoritmo, ou seja, utilizamos o tempo médio estimado em que o algoritmo tem sucesso, adicionamos uma pequena margem e definimos esse tempo como um *timeout* para que o sistema retorne uma solução válida.

5.3.1 Algoritmo Alternativo (Fila de Alocação)

Considerando todas as conclusões extraídas anteriormente, percebemos que o processo de alocação multiobjetivo em um ambiente restringido por diversas *constraints* pode ser bastante desafiador. Utilizando as vantagens observadas no AG e buscando eliminar suas limitações, propomos uma variação para essa abordagem. Utilizando a mesma base de desenvolvimento do algoritmo genético, sugerimos nesse novo algoritmo abandonar a alocação simultânea de tarefas, ou seja, deixamos de lado o conceito de um cromossomo ser formado por uma combinação de diversas alocações tarefa-cloud, e assumimos que um indivíduo é composto por apenas um gene de alocação. Dessa maneira, **o processo agora é visto como uma fila de alocação, onde cada tarefa submetida pelo usuário é alocada para uma máquina de forma independente a cada ciclo**. Com essa modificação, não limitamos o tamanho da lista de tarefas que o usuário pode submeter, assim como diminuímos drasticamente o impacto da configuração dessas tarefas no desempenho do alocador, uma vez que, em um cromossomo unitário, não temos o fenômeno em que a pontuação de um gene interfere no outro, diminuindo a *fitness score* total. Como já citado anteriormente, a utilização de um cromossomo composto de um único gene inviabiliza o processo de *crossover*, fazendo com que o progresso de um indivíduo dentro do algoritmo seja regido apenas pelo operador de mutação, o que de certa forma tira a identidade do algoritmo genético. Podemos dizer que a alternativa proposta é uma hibridização do algoritmo genético com o método de busca aleatória, e, como concebido na seção anterior, obteve resultados que surpreendem positivamente.

Analisamos anteriormente que, para o nosso caso de teste, o pior cenário possível é formado por uma lista de doze tarefas *tightly constrained*, com um banco de máquinas de 250 entradas e um nível **A** de qualidade exigido (FS $\geq$ 0.8). Rodando a nossa abordagem alternativa nesse cenário, obtivemos resultados bem superiores ao método multi-objetivo em todos os aspectos.

Diferentemente da abordagem multiobjetivo, a taxa de falhas sofre bem menos influência da característica não determinística do algoritmo. Notamos que na aplicação da abordagem anterior nesse cenário, todas as execuções falhariam, enquanto que na alocação por fila, todas as execuções tiveram sucesso, exceto os modos de Disponibilidade e Desempenho com alta qualidade exigida (Figura 5.9). Através de uma verificação manual, notamos que as tarefas *tightly constrained* impossibilitam encontrar alocações altamente otimizadas nesses modos, ou seja, as falhas ocorreram devido a inexistência de uma solução possível, e não pela característica probabilística do algoritmo.

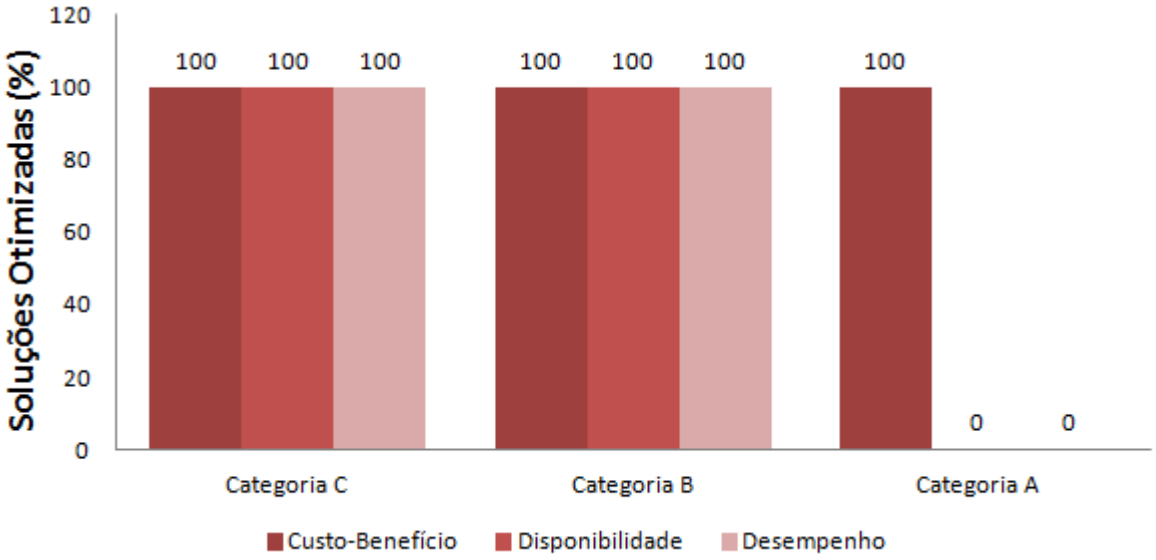

Figura 5.9: Taxa percentual de soluções otimizadas encontradas em 10000 execuções na abordagem Fila de Alocação

O segundo fato observado é que, na grande maioria das execuções, se as soluções existirem no espaço de busca, o algoritmo as retornarão, independentemente do nível de exigência de qualidade. Percebemos na Figura 5.10 que as médias de pontuação de aptidão obtidas com a abordagem de fila de alocação, respeitando os diferentes escopos analisados, são superiores às médias da abordagem multiobjetivo. Isso ocorre devido ao fato de que, agora, as alocações são independentes, ou seja, a pontuação da alocação de uma tarefa, não influencia na pontuação total da lista, dessa forma, há espaço para o algoritmo procurar as melhores combinações individualmente, diminuindo o peso da característica comum das tarefas no processo de busca.

Por fim, a Figura 5.11 nos mostra o tempo médio das dez mil execuções na abordagem alternativa. Inferimos que o número de falhas ainda é diretamente relacionado ao tempo total de execução, com isso, notamos que todas as execuções que obtiveram sucesso foram executadas em tempo desprezível, em comparação com o mesmo cenário na abordagem multiobjetivo.

Concluindo essa etapa, ressaltamos que esta seção não visa tirar os méritos de toda a pesquisa baseada em planejamento multi-objetivo, muito pelo contrário, sabemos que essa abordagem em casos onde temos tarefas interdependentes e inúmeros *interleavings* é a única alternativa viável para resolução de um problema. Queremos deixar claro com essa comparação que devemos analisar com cautela o escopo do nosso problema, e que geralmente, a utilização de algoritmos genéticos para alocação em problemas regulados por *constraints* podem não ser a melhor opção. Uma das grandes lições que tiramos desta pesquisa é que a responsabilidade do AG deve ser mantida simplesmente na otimização das soluções, deixando as questões de satisfatibilidade para outras abordagens. Apesar disso, com algumas modificações, pudemos mesclar o algoritmo genético com características de outros métodos, nos trazendo uma solução muito mais robusta e que oferece soluções de maior QoS, reforçando a ideia de hibridização como uma poderosa ação

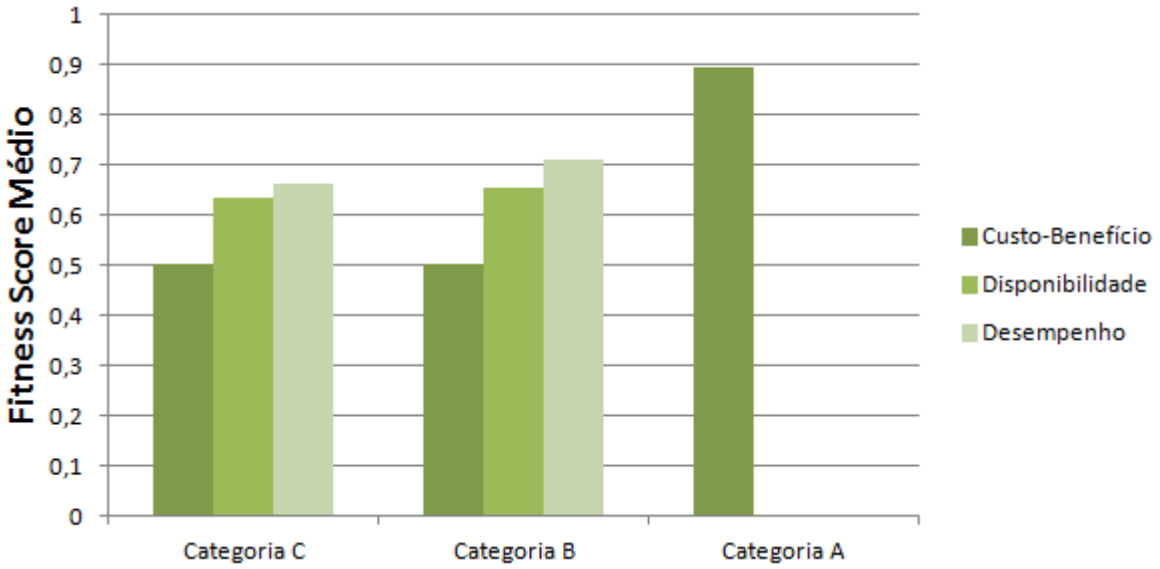

Figura 5.10: *Fitness Score* médio de 10000 execuções na abordagem Fila de Alocação

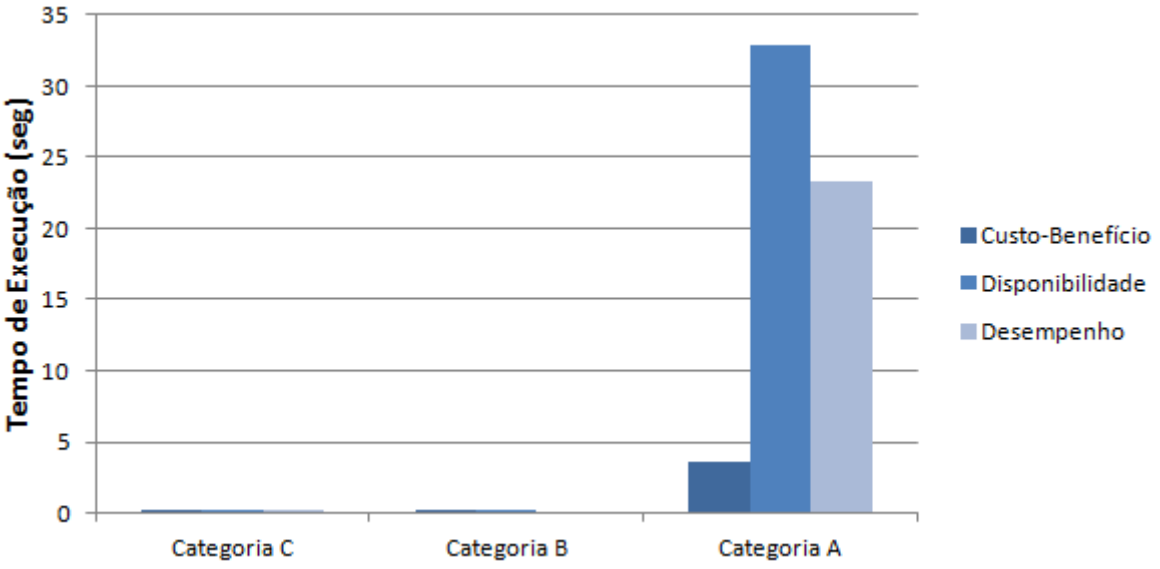

Figura 5.11: Tempo médio de 10000 execuções na abordagem Fila de Alocação

na melhoria do alocador genético.

Capítulo 6

Conclusão

Neste trabalho, propusemos a utilização de algoritmos genéticos na alocação de recursos em *Cloud Computing*. O conceito por trás do uso de uma abordagem evolucionária está no fato de promover maior flexibilidade, desempenho e qualidade na alocação de tarefas e no gerenciamento de recursos. O paradigma autoadaptativo é uma das diversas abordagens que temos hoje para criar sistemas mais inteligentes, capazes de se autorregularem e tomarem decisões que facilitem o cumprimento de um determinado objetivo, dependente ou independentemente de contexto.

Resumindo as considerações realizadas nesta monografia, e direcionados pelas questões de pesquisa levantadas, inferimos que esse estudo exploratório dá indícios de que a utilização da metodologia proposta é bastante válida na otimização do processo de alocação de tarefas e recursos em computação em nuvem. Podemos ainda extrapolar essa afirmação, dizendo que os algoritmos genéticos têm potencial para serem excelentes planejadores em um grande número de sistemas autônomos, desde que o mapeamento do problema para a heurística seja feito corretamente. Devemos ressaltar ainda que a flexibilidade atrelada a utilização de um módulo de planejamento inteligente, baseado em uma abordagem evolucionária, é de grande valor para a dependabilidade do sistema como um todo. Queremos dizer com isso, por exemplo, que um alocador de qualidade deve ser ciente do que se passa em todo o sistema no qual está inserido e dos contextos aos quais pode ser submetido, sendo capaz de distinguir quando uma *cloud* não está disponível ou simplesmente perdendo sua capacidade de processamento e criar um plano alternativo de alocação, mais do que isso, deve ser capaz de se recuperar de falhas sem interferência de um operador humano, e tudo isso é possível com sistemas de monitoramento e planejamento bem fundamentados. Este é um pequeno, mas importante passo para a criação de uma computação mais poderosa e homogênea.

Muito ainda tem a ser feito no âmbito desta pesquisa. Algumas melhorias no algoritmo proposto unidas às ideias de *context-awareness* e verificação em tempo de execução fazem parte do escopo de nossos futuros trabalhos:

- Melhorar a implementação do algoritmo proposto, baseado nas análises feitas no Capítulo 5, hibridização do algoritmo genético com um buscador baseado em *constraints* (CSP),

além de refinamentos nos operadores genéticos a fim de otimizar o processo de tomada de decisão do alocador.

- Avaliar qualitativamente os resultados alcançados em termos de QoS e o impacto dessa abordagem nos preceitos de dependabilidade.
- Aplicação do algoritmo refinado em um caso prático de estudo, considerando todas as limitações e requerimentos de um ambiente real.
- Teste com monitoramento de recursos em tempo real, a fim de verificar o comportamento do algoritmo genético em situações de tempo real e variação de contexto, utilizando um planejador automático como escalonador de tarefas com temporalidade incluída.

Referências

[1] Amazon elastic compute cloud - amazon ec2. aws.amazon.com/pt/ec2, 2015. 31

[2] Amazon web services. aws.amazon.com, 2015. 7

[3] Choco, free and open-source java library dedicated to constraint programming. choco-solver.org, 2015. 57

[4] Google cloud platform (app engine). cloud.google.com/appengine, 2015. 7

[5] Google compute engine (gce). cloud.google.com, 2015. 31

[6] Microsoft azure. azure.microsoft.com/en-us/services/cloud-services, 2015. 7

[7] Shumeet Baluja and Rich Caruana. Removing the genetics from the standard genetic algorithm. *Proceedings of the Twelfth International Conference on Machine Learning*, July 1995. 19

[8] Ian Lumb Bhaskar Prasad Rimal, Eunmi Choi. A taxonomy and survey of cloud computing systems. In *INC, IMS and IDC, 2009. NCM '09. Fifth International Joint Conference on*, pages 44–51, Aug 2009. 8

[9] Michele Ciavotta Juan Pz Weikun Wang Danilo Ardagna, Giuliano Casale. Quality-of-service in cloud computing: modeling techniques and their applications. *Journal of Internet Services and Applications*, 2014. 9, 10

[10] Kalyanmoy Deb and David Goldberg. A comparative analysis of selection schemes used in genetic algorithms. *In Foundations of Genetic Algorithms*, pages 69–93, 1991. 19

[11] J. F. Frenzel. Genetic algorithms. *Potentials, IEEE*, 12(3):21–24, Oct 1993. 17, 18

[12] M. R. Garey and D. S. Johnson. *Compute and Intractability: A guide to the theory of NP-completeness*. W. H. Freeman and co, 1979. 16

[13] Karsten Oberle Andreas Menychtas Kleopatra Konstanteli Georgina Gallizo, Roland Kuebert. Service level agreements in virtualised service platforms. In *eChallenges e-2009 Conference Proceedings*. IIMC International Information Management Corporation, 2009. 10

[14] Vaclav Snasel Hesam Izakian, Ajith Abraham. Comparison of heuristics for scheduling independent tasks on heterogeneous distributed environments. *Computational Sciences and Optimization, 2009. CSO 2009. International Joint Conference on*, 1:8–12, April 2009. 14, 15

[15] John Henry Holland. *Adaptation in Natural and Artifcial Systems*. University of Michigan Press, Ann Arbor, 1975. 18

[16] Ioan Raicu Shiyong Lu Ian Foster, Yong Zhao. Cloud computing and grid computing 360-degree compared. *Grid Computing Environments Workshop*, pages 1–10, Novembro 2008. 7

[17] Xing-Ming Zhang Fan Zhang Bai-Nan Li Jing Liu, Xing-Guo Luo. Job scheduling model for cloud computing based on multi-objective genetic algorithm. *IJCSI International Journal of Computer Science Issues*, 10(3):134–139, January 2013. x, 27, 28

[18] George Kakarontzas and Ilias Savvas. Agent-based resource discovery and selection for dynamic grids. *Proc. 15th IEEE Intl Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pages 195–200, 2006. 25

[19] Shaminder Kaur and Amandeep Verma. An efficient approach to genetic algorithm for task scheduling in cloud computing environment. *I.J. Information Technology and Computer Science*, 10:74–79, 2012. 14

[20] Hai Jin-Yun Yang Ke Liu, Jinjun Chen. Min-min average algorithm for scheduling transaction-intensive grid workflows. *Conferences in Research and Practice in Information Technology*, 2009. 14

[21] Keqin Li. Job scheduling and processor allocation for grid computing on metacomputers. *Journal of Parallel and Distributed Computing*, 65(11):1406–1418, 2005. 25

[22] Srinivas Pothapragada M. Padmavathi, Shaik Mahabbob Basha. A survey on scheduling algorithms in cloud computing. *IOSR Journal of Computer Engineering*, 16(4):27–32, Jul - Aug. 2014. 14

[23] Jerffeson Teixeira de Souza Shin Yoo Mark Harman, Phil McMinn. *Search Based Software Engineering: Techniques, Taxonomy, Tutorial.* Springer Berlin Heidelberg, 2012. x, 17, 18, 19, 42

[24] Phil McMinn. Search-based testing: Past, present and future. In *In Proceedings of the 3rd International Workshop on Search-Based Software Testing (SBST 2011)*, Berlin, Germany, 2011, To Appear. IEEE. x, 22

[25] Rean Griffith Anthony D. Joseph Randy Katz-Andy Konwinski Gunho Lee David Patterson Ariel Rabkin Ion Stoica Matei Zaharia Michael Armbrust, Armando Fox. *Above the Clouds: A Berkeley View of Cloud Computing.* Electrical Engineering and Computer Sciences University of California at Berkeley, 2009. 9

[26] Christos H. Papadimitriou. *Computational Complexity.* Addison-Wesley, 1994. 16

[27] Wen-Mao Gong Qi Cao, Zhi-Bo Wei. An optimized algorithm for task scheduling based on activity based costing in cloud computing. *Proc. 3rd Intl Conf. on Bioinformatics and Biomedical Engineering*, pages 1–3, 2009. 25

[28] Andreas Drexl Rainer Kolisch, Arno Sprecher. *Characterization and Generation of a General Class of Resource-Constrained Project Scheduling Problems.* Institut fur Betriebswirtschaftslehre, 1992. 24

[29] Stuart Jonathan Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach.* Prentice Hall, 2010. 2

[30] Mario Vecchi Scott Kirkpatrick, Daniel Gellat. Optimization by simulated annealing. *Science, 220(4598)*, 1983. 15

[31] Jae Kwon Kim Jong Sik Lee Sung Ho Jang, Tae Young Kim. The study of genetic algorithm-based task scheduling for cloud computing. *International Journal of Control and Automation*, 5(4):157 – 162, December 2012. x, 21, 25, 26

[32] Padmavathi Ganapathi V.Vinothina, R. Sridaran. A survey on resource allocation strategies in cloud computing. *International Journal of Advanced Computer Science and Application*, 3(6):97–104, 2012. 13

[33] Matthew Bartschi Wall. *A Genetic Algorithm for Resource-Constrained Scheduling*. PhD thesis, Massachusetts Institute of Technology, June 1996. 12, 23

[34] Thomas Weise. *Global Optimization Algorithms - Theory and Application*. http://www.it-weise.de, 2009. 2, 42

[35] Darrell Whitley. The genitor algorithm and selection pressure: Why rank-based allocation of reproductive trials is best. In *Proceedings of the International Conference on Genetic Algorithms*, pages 116 – 121, 1989. 19