Sistemas Dinâmicos e Técnicas Inteligentes para a Predição de Comportamento de Processos: Uma Abordagem para Otimização de Escalonamento em Grades Computacionais

Rodrigo Fernandes de Mello

Tese submetida ao Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo para cumprimento parcial dos requisitos para obtenção do título de Livre Docente em Ciências de Computação

USP - São Carlos
Novembro de 2010

À minha família, especialmente à minha esposa.

Agradecimentos

Agradeço primeiramente a Deus, em seguida a meus pais pela oportunidade de realizar meus estudos. À minha esposa pelo carinho, incentivo e compreensão.

Aos meus amigos que diretamente auxiliaram no desenvolvimento de pesquisas e na revisão desta tese: André Eberle Mantini, Eduardo Ferreira Alves, Evgueni Dodonov, José Augusto Andrade Filho, Luciano José Senger, Marcelo Keese Albertini, Matheus Lorenzo dos Santos, Paulo Henrique Ribeiro Gabriel, Renato Porfírio Ishii e Rodrigo Lopes.

Ao apoio de Antonio Castelo Filho em temas relativos a sistemas dinâmicos.

Ao amigo Rafael Gigena Cuenca por auxiliar na elaboração da figura sobre variedades.

Ao apoio e revisão de Maria Cristina Ferreira de Oliveira, André Carlos Ponce de Leon Ferreira de Carvalho, Zhao Liang, João Paulo Gois e Ricardo Rios.

Ao suporte de Gláucia Maria Saia Cristianini, diretora técnica de serviço da biblioteca Prof. Achille Bassi, em questões relativas ao formato deste documento.

Aos amigos de departamento e demais pessoas que auxiliaram direta ou indiretamente nesta tese.

Agradecimentos especiais ao CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico) e à FAPESP (Fundação de Amparo à Pesquisa do Estado de São Paulo) por sucessivos apoios por meio de projetos de pesquisa.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Lista de Siglas

AFS	*Andrew File System*
AMI	*Auto-Mutual Information*
ANN	*Artificial Neural Network*
AR	*Auto-Regressivo*
ART	*Adaptive Resonance Theory*
ART-2A	*Adaptive Resonance Theory, arquitetura 2A*
ATNN	*Adaptive Time-Delay Neural Network*
BG	*Broker Global*
BMU	*Best-Matching Unit*
CODA	*Sistema de arquivos de rede baseado no AFS versão 2*
CPU	*Central Processing Unit*
DFS	*Distributed Filesystem*
DNS	*Domain Name Service*
DSM	*Distributed Shared Memory*
EWMA	*Exponentially Weighted Moving Average*
FNN	*False Nearest Neighbors*
GAS	*Genetic Algorithm for Scheduling*
GTC	*Software utilizado para estudar microturbulências em plasmas de fusão toroidal magneticamente confinados por meio do método de partícula em célula*
GWR	*Grow When Required*

IBL	*Instance-Based Learning*
ICMP	*Internet Control Message Protocol*
IP	*Internet Protocol*
J2EE	*Java2 Platform Enterprise Edition*
LAN	*Local Area Network*
LSTM	*Long Short-Term Memory*
MAN	*Metropolitan Area Network*
MIPS	*Million of Instructions per Second*
MPI	*Message Passing Interface*
MSE	*Mean Squared Error*
MidHPC	*Middleware for High Performance Computing*
NERSC	*National Energy Research Scientific Computing Center*
PIC	*Particle-In-Cell*
PMEMD	*Software que inclui funções de dinâmica molecular, refinamentos e minimizações*
POSIX	*Portable Operating System Interface*
PredRoute	*Política de escalonamento que estende a RouteGA adicionando predição de eventos de comunicação entre processos*
R/S	*Rescaled Range*
RBF	*Radial Basis Function*
RRBF	*Recurrent Radial Basis Function*
RTT	*Round-Trip Time*
Route	*Política de escalonamento*
RouteGA	*Route with Genetic Algorithm Support*
SA	*Simulated Annealing*
SOAP	*Simple Object Access Protocol*
SOM	*Self-Organizing Maps*
SONDE	*Self-Organizing Novelty Detection*
SSHFS	*Secure SHell FileSystem*

SVM	*Support Vector Machine*
SchedSim	*Simulador de políticas de escalonamento de processos*
TDNN	*Time-Delay Neural Network*
TISEAN	*Software para análise de séries temporais não lineares*
UniMPP	*Unified Modeling for Predicting Performance*
WAN	*Wide Area Network*
XML	*eXtended Markup Language*

Resumo

Computação Autônoma é uma área emergente que visa capacitar sistemas a gerenciar suas tarefas de maneira inteligente, dinâmica e automática. Essa área aborda a grande complexidade dos sistemas computacionais atuais, os quais têm se tornado cada vez mais difíceis de integrar e manter, devido, principalmente, ao aumento de escala e à necessidade de mão-de-obra especializada. Computação em grade é uma das áreas com maior potencial para o projeto de novas técnicas autônomas, em vista da sua complexidade proveniente da dinâmica do comportamento de tarefas, compartilhamento de informações, variação do perfil de usuários, recursos computacionais geograficamente distribuídos e heterogêneos. Esse potencial motivou diferentes frentes de pesquisa, a maioria visando a proposta de arquiteturas autônomas (estruturação de componentes necessários para prover serviços) e a integração de ferramentas e serviços. Poucas dessas frentes são voltadas para o estudo, projeto e avaliação de técnicas e modelos com o objetivo de apoiar e melhorar funcionalidades autônomas de grades. Esses fatores motivaram o projeto *Middleware for High Performance Computing* (`MidHPC`), que investiga formas de auto-gerenciar suportes para distribuição de dados, tolerância a falhas, segurança e escalonamento de processos. Essas investigações obtiveram resultados promissores por meio do emprego da ocupação histórica média de recursos por processos. Esses resultados motivaram pesquisas adicionais com o objetivo de avaliar e compreender variações no comportamento de processos e empregá-las na melhoria de aspectos autônomos em grades computacionais. A compreensão dessas variações motivou esta tese de livre docência, que emprega conceitos sobre sistemas dinâmicos e técnicas inteligentes para modelar e predizer o comportamento de processos com o objetivo de otimizar operações de escalonamento em ambientes de grade computacional. Tais conceitos foram aplicados na predição de operações de comunicação, as quais foram utilizadas para parametrizar uma nova política de escalonamento que visa minimizar custos de comunicação e processamento. Essa política foi comparada a outras abordagens da literatura e seus resultados confirmam contribuições significativas decorrentes da predição de comportamento de processos.

Abstract

Autonomic Computing is an emerging research area which deals with systems capable of self-managing their tasks in an intelligent, dynamic and automatic manner. It addresses the increasing complexity of current computing systems, which become more and more difficult to integrate and maintain, mainly due to their scale growth and the need for specialized labor. Grid computing requires self-manageable features, essentially due to the complexity associated to its task dynamical behavior, information sharing, different user profiles, geographically distributed resources and heterogeneity. The need for those features has encouraged many research efforts, most of them focused on proposing autonomic architectures (structuring components to provide services) and the integration of tools and services. Few research initiatives address the study, design and evaluation of techniques and models to support and improve grid autonomic aspects. This scenario has motivated the *Middleware for High Performance Computing* (`MidHPC`) project which investigates self-manageable supports for data distribution, fault tolerance, security and process scheduling. Promising results were obtained by considering historical resource consumption averages. Results encouraged additional efforts to assess process behavior variations and their application to improve autonomic aspects of grid computing. Comprehension of those variations in behavior motivated this thesis, which employs dynamical system concepts and intelligent techniques to model and predict process behavior, aiming at optimizing scheduling operations in grid computing environments. Process communication operations were predicted and used to parameterize a new process scheduling policy which attempts to minimize communication and processing costs. This policy was compared to others from the literature. Results confirm relevant contributions and motivate the adoption of the proposed approach to improve other grid autonomic aspects.

Capítulo

1

Introdução

Computação Autônoma é uma área emergente que visa capacitar sistemas a gerenciar suas tarefas de maneira inteligente, dinâmica e automática (Murch, 2004). Essa área surgiu em conseqüência da grande complexidade dos sistemas computacionais atuais, os quais têm se tornado cada vez mais difíceis de integrar e manter devido ao aumento de escala e à necessidade de mão-de-obra especializada. Segundo pesquisadores da Universidade de Berkeley (Gibbs, 2002), custos de manutenção tendem a ser seis vezes maiores que os de aquisição da infra-estrutura empregada. Estudos similares, realizados pela IBM (Murch, 2004), comprovam que para cada dólar aplicado em infra-estrutura, dez outros são necessários para gerenciá-la. Esses fatores de custo aliados à dificuldade em gerenciar operações, integrar, oferecer manutenção corretiva, instalar e atualizar sistemas são alguns dos principais motivadores da Computação Autônoma.

Diversas iniciativas têm sido propostas para prover funcionalidades autônomas a sistemas computacionais, as quais consideram, basicamente, quatro aspectos: auto-otimização, auto-cura, auto-proteção e auto-configuração (Murch, 2004; Parashar & Hariri, 2006). A auto-otimização monitora, constantemente, o sistema a fim de explorar suas alterações comportamentais e aplicá-las na manutenção ou melhora do desempenho de execução de tarefas. Esse aspecto considera conceitos, tais como os aplicados em teoria de controle, para avaliar a eficiência do sistema durante sua execução. Resultados provenientes de tal avaliação são utilizados para otimizar operações futuras. Como exemplo, considere um ambiente computacional e um conjunto de tarefas a ser escalonado. A distribuição otimizada de tais tarefas pode ser realizada, e periodicamente reavaliada, com base no comportamento de utilização de recursos (e.g. processamento, comunicação, etc.), com o objetivo de manter bons níveis de desempenho para o sistema.

O segundo aspecto é denominado auto-cura, segundo o qual um sistema deve ser capaz

de detectar comportamentos incorretos de execução, falhas e outras anormalidades a fim de se recuperar com o mínimo de interferência em sua execução normal. Por exemplo, considere um sistema que salva, periodicamente, contextos de execução de seus processos (*checkpoints*) em meio de armazenamento secundário. Caso um processo falhe, pode-se retomar sua execução a partir do último contexto armazenado, sem perda de transparência para o usuário final e minimizando prejuízos de execução. Um sistema sem tal suporte precisaria reiniciar sua execução a partir do princípio, o que incorre em tempo de espera e alocação adicional de recursos.

O terceiro aspecto, denominado auto-proteção, visa identificar, detectar e proteger o sistema contra invasões e demais formas intrusivas capazes de alterar seu funcionamento normal. Assim, o sistema deve manter a integridade de processamento, de dados e ainda garantir a segurança de suas operações. Nesse sentido, se um programa ou um usuário começa a atuar de forma inesperada e intrusiva, pode-se detectar tal variação comportamental e anular suas atividades.

O aspecto final, ou de auto-configuração, visa garantir que um sistema seja capaz de instalar-se e configurar-se automaticamente. Esse aspecto considera algoritmos adaptativos capazes de cumprir atualizações sem prejudicar a execução normal do sistema. Por exemplo, considere uma aplicação em execução que realiza modificações em banco de dados. Ao atualizá-la, o banco é recriado. Nesse caso, deve-se garantir que a execução atual não tenha problemas em função de futuras atualizações. Exemplificando outra circunstância de uso da auto-configuração, considere um serviço que recebe requisições de usuários. Caso haja aumento expressivo no volume de requisições, pode-se reconfigurar o sistema para replicar tal serviço, permitindo, assim, o atendimento de um maior número de requisições.

Espera-se, com a evolução dos quatro aspectos apresentados, que sistemas computacionais tenham capacidade de auto-gerenciamento, adaptando-se conforme necessidades do meio, tais como número de usuários, ocorrência de falhas ou alarmes falsos, volume de transações, comunicação entre tarefas, disponibilidade de recursos, etc. Conseqüentemente, diferentes áreas de pesquisa podem se beneficiar do auto-gerenciamento. Segundo pesquisadores (Murch, 2004; Parashar & Hariri, 2006), computação em grade, ou *grid computing*, é uma das áreas com maior potencial para aplicação e desenvolvimento de novas pesquisas sobre autonomia, dada sua inerente complexidade de manutenção e gerenciamento provenientes do comportamento dinâmico de tarefas, compartilhamento de informações, grande número de usuários, recursos computacionais geograficamente distribuídos e heterogêneos.

Essa complexidade de manutenção e gerenciamento têm motivado diferentes pesquisas por soluções capazes de prover autonomia a ambientes de grade (Parashar & Hariri, 2006; Othman *et al.*, 2003; Cheng *et al.*, 2002; Shirose *et al.*, 2004; Champrasert *et al.*, 2005). Othman *et al.* (2003) propõem um mecanismo centralizado para monitorar a dis-

ponibilidade de computadores e utilizar essas informações na alocação de recursos. Cheng *et al.* (2002) e Shirose *et al.* (2004) propõem arcabouços que permitem a reconfiguração de serviços em grades. Componentes centralizados são utilizados para armazenar informações sobre condições de execução da grade e tomar decisões de adaptação do sistema. Parashar & Hariri (2006) propõem o projeto `AutoMate` que investiga soluções autônomas, baseadas em sistemas biológicos, com o intuito de abordar questões relativas à complexidade, dinamismo, heterogeneidade e incerteza (relativa a variações de comportamento de processos, ocupação de recursos, volume de transações, etc.) de grades computacionais. As principais frentes de pesquisa do projeto `AutoMate` compreendem a proposta de modelos conceituais e a implementação de arquiteturas para suportar autonomia em grades. Champrasert *et al.* (2005) propõem o projeto `SymbioticSphere` que visa aplicar conceitos e mecanismos biológicos no desenvolvimento de sistemas para grades.

Outras abordagens, empregando computação autônoma em grades, têm sido propostas por empresas tais como Think Dynamics, Sun Microsystems, Microsoft, HP e Intel. Think Dynamics (Dubie, 2003), recentemente adquirida pela IBM, projeta softwares para suportar computação autônoma em grades. Esses softwares utilizam tecnologias abertas tais como `XML`, `J2EE` e `SOAP`. Sun Microsystems (Sun Microsystems, Inc., 2008) propõe o projeto `N1` voltado para o gerenciamento de recursos distribuídos, considerando virtualização, provisionamento de serviços e automação de políticas de negócio. Esse projeto integra ferramentas existentes a fim de distribuir processos e gerenciar recursos computacionais. Microsoft e HP (Meyler, 2008) propõem o projeto `Dynamic Data Center` que provisiona e gerencia, de forma centralizada, recursos computacionais. A Intel (Intel, 2008) propõe o projeto `Proactive Computing` que busca aumento de desempenho por meio da antecipação de necessidades de usuários em grades.

Essas frentes de trabalho confirmam as várias iniciativas desenvolvidas buscando prover autonomia para grades computacionais. No entanto, a maior parte das pesquisas é voltada para a proposta de arquiteturas (estruturação de componentes necessários para prover serviços), integração de ferramentas e serviços a fim de prover suporte autônomo para grades. Poucas visam o estudo, projeto e avaliação de técnicas e modelos para ampliar e fortalecer aspectos autônomos em grades (Lee & Suzuki, 2006; Suzuki & Suda, 2005; Champrasert & Suzuki, 2006a,b).

1.1 Computação Autônoma em Grades

A escassez de pesquisas relacionadas ao estudo, projeto e avaliação de técnicas e modelos a fim de prover funcionalidades autônomas para grades motivou o projeto *Middleware for High Performance Computing* (`MidHPC`)[1] (Andrade Filho *et al.*, 2008). Esse projeto

[1] Projeto `MidHPC` foi submetido para o processo seletivo de Prof. Doutor da Universidade de São Paulo e apoiado pela Fundação de Amparo à Pesquisa do Estado de São Paulo – processo número 2004/002411-9.

aborda autonomia em grades computacionais por meio da investigação e emprego de estratégias de otimização de escalonamento, distribuição de dados, rebalanceamento de cargas, configuração de serviços, segurança e tolerância a falhas.

As primeiras atividades desse projeto buscaram, por meio de técnicas de otimização, reduzir o tempo de execução de tarefas. Isso, contudo, motivou pesquisas para melhor compreender características do ambiente computacional a ser otimizado. Tais pesquisas mediram atrasos e desempenho de protocolos em redes de computadores (Ferreira & Mello, 2004; Dodonov *et al.*, 2005). Informações capturadas foram utilizadas para, por meio de modelos estocásticos, caracterizar atrasos de comunicação entre computadores da grade. Esses modelos foram aplicados em simulações de escalonamento de processos, as quais auxiliaram na compreensão de respostas do ambiente real. Modelos de simulação seriam, dessa forma, capazes de representar, com boa aproximação, atrasos de comunicação em ambientes reais. Essas simulações foram motivadas, essencialmente, pela necessidade de acesso e controle dos recursos computacionais, o que permite avaliações de ambientes de diferentes escalas e características.

Em seguida, foi proposto um conjunto de ferramentas de *benchmark* para extrair capacidades de processamento, latência e atrasos gerados pela ocupação de memória principal e virtual, e vazão de leitura e escrita em discos rígidos. Essas ferramentas complementaram a pesquisa anterior no que se refere à extração de capacidades dos recursos computacionais do ambiente. Posteriormente, um vasto conjunto de computadores foi submetido a essas ferramentas de *benchmark* e obtidas as capacidades de seus recursos. Essas capacidades foram, também, resumidas em modelos estocásticos que permitiram caracterizar recursos presentes em ambientes reais (Mello & Senger, 2006).

As capacidades de recursos computacionais extraídas permitiram projetar um modelo de caracterização de cargas em ambientes distribuídos, denominado *Unified Modeling for Predicting Performance* (`UniMPP`) (Mello & Senger, 2006). Esse modelo formaliza aspectos relativos à ocupação de recursos gerada por tarefas, tais como processamento, memória, acesso a disco rígido e rede de computadores. A partir de tal modelo, pode-se simular a execução de aplicações compostas por múltiplas tarefas em ambientes de diferentes escalas. Esse modelo foi, posteriormente, implementado em um simulador, denominado `SchedSim` (Mello & Senger, 2006), o qual simplifica o projeto e permite avaliar técnicas de escalonamento de tarefas em ambientes distribuídos.

A fim de parametrizar o modelo `UniMPP`, a próxima etapa consistiu em obter a ocupação de tarefas em ambientes reais. Para isso, métodos foram estudados, incluindo alguns baseados em inteligência artificial (Aha *et al.*, 1991; Meng, 2003; Senger *et al.*, 2005a). A partir desses estudos selecionou-se o método de aprendizado baseado em instâncias (*Instance-Based Learning* – `IBL`) (Aha *et al.*, 1991), que analisa similaridades de uma nova aplicação lançada no ambiente em relação ao histórico de execuções prévias. Essa análise resulta em ocupações médias esperadas de processamento, acesso a memória, disco

e rede. A técnica IBL foi implementada e avaliada no trabalho de Senger *et al.* (2007), o qual comprovou seus resultados promissores e motivou pesquisas adicionais em otimização de escalonamento de tarefas empregando conhecimento sobre aplicações (Mello *et al.*, 2006a,b, 2007c; Senger *et al.*, 2005b).

Essas pesquisas foram, posteriormente, integradas a um protótipo do MidHPC (Andrade Filho *et al.*, 2008) desenvolvido como uma camada de software sobre GNU/Linux. Esse protótipo distribui tarefas sobre grades computacionais de forma automática, transparente e otimizada.

Estudos empregando esse protótipo (Andrade Filho *et al.*, 2008; Mello *et al.*, 2007b) confirmaram bons resultados na otimização do escalonamento de processos, advindos do conhecimento sobre o comportamento de aplicações e recursos computacionais. O protótipo atualmente extrai esses conhecimentos de duas fontes: recursos e processos. Monitora-se, periodicamente, recursos computacionais a fim de obter medidas de ocupação de processamento, memória, disco rígido e rede. Utiliza-se uma média móvel exponencial ponderada (Achelis, 2000) de tais informações com o objetivo de caracterizar a ocupação instantânea de recursos. Quanto aos processos, suas chamadas de sistema são interceptadas e armazenadas em banco de dados. Essas chamadas auxiliam na obtenção de informações tais como a quantidade de bytes enviada e recebida, processamento em espaço de endereçamento de usuário e *kernel*.

Dados de utilização de recursos são diretamente obtidos com a atualização de médias móveis exponenciais, enquanto os de processos passam por um algoritmo IBL. Esse algoritmo extrai o comportamento médio esperado de utilização de recursos de uma nova aplicação lançada ao sistema. Assim, mesmo antes da execução de uma aplicação, pode-se estimar seu comportamento aproximado com base em similaridade de aplicações anteriormente executadas.

Informações sobre médias de ocupação de recursos e comportamento esperado de novas aplicações são submetidas a um otimizador, responsável por encontrar boas soluções de escalonamento. Essas médias não evidenciam variações na utilização de recursos no tempo, conseqüentemente, o otimizador escolhe soluções que, aproximadamente, melhoram o desempenho global do sistema. Essas soluções podem privilegiar, por exemplo, a ocupação de processamento em detrimento de comunicação via rede, o que tende a aumentar os tempos de execução de aplicações. Melhores soluções seriam encontradas ao compreender variações comportamentais de processos e adaptar decisões de escalonamento, antecipando operações futuras. Para isso deve-se detectar e compreender a dinâmica envolvida em comportamentos de processos.

1.2 Objetivo

A compreensão da dinâmica do comportamento de processos permite evoluir o projeto `MidHPC` nos quatro principais aspectos da computação autônoma. Quanto ao aspecto de auto-otimização, pode-se utilizar modelos para prever eventos de ocupação de recursos e otimizar a alocação de tarefas. Quanto à auto-configuração, pode-se indicar momentos para a replicação de serviços, visando melhor atender requisições de usuários. Quanto à auto-cura, pode-se avaliar mudanças em comportamentos de processos e identificar momentos de falha. Essa mesma avaliação pode ser empregada para detectar comportamentos intrusivos de processos e usuários, os quais auxiliam no aspecto de auto-proteção.

Para compreender essa dinâmica, modelou-se, primeiramente, comportamentos de processos como séries temporais e, em seguida, iniciou-se estudos exploratórios em busca de técnicas que permitissem avaliar mudanças, tendências e formas de predizê-las. As primeiras técnicas investigadas foram as redes neurais artificiais (*Artificial Neural Networks* – `ANN`) (Freeman & Skapura, 1991; Haykin, 2008).

Determinados tipos de redes neurais são voltados para a representação da dinâmica presente em séries temporais. Dentre tais redes pode-se mencionar a *Time-Delay Neural Network* (`TDNN`) (Waibel *et al.*, 1989), *Adaptive Time-Delay Neural Network* (`ATNN`) (Lin *et al.*, 1995) e *Recurrent Radial Basis Function Networks* (`RRBF`) (Ryad *et al.*, 2001). Essas redes são providas de etapa de treinamento que visa modelar a dinâmica de séries para aplicá-la, essencialmente, em predição. Algumas dessas redes, como a `ATNN`, permitem a alteração *on-line* de alguns parâmetros de aprendizado, adaptando-se, a fim de melhor representar a dinâmica de séries temporais. Contudo, essas redes necessitam de topologias específicas (número de neurônios e camadas) para resolver diferentes problemas. Essas topologias influenciam na extração da dinâmica. A escolha de uma topologia é, em geral, feita com base em estudos sobre conjuntos de treinamento e validação, os quais permitem avaliar a qualidade dos resultados obtidos.

Exemplos da dificuldade em extrair a dinâmica de séries são apresentados em (Wan, 1994; Zemouri *et al.*, 2003). No primeiro trabalho, (Wan, 1994), observa-se a necessidade de definir diferentes *time delays*, número de neurônios e conexões, de acordo com o problema abordado pela `TDNN`. O segundo trabalho (Zemouri *et al.*, 2003) sugere variar o número de neurônios da camada de entrada da rede neural `RRBF`, os quais influenciam na extração da dinâmica de séries. Esses aspectos dificultam a modelagem de séries, principalmente em casos em que não há um longo conjunto de validação, que permita analisar a topologia de rede neural a ser adotada para determinado problema.

Outra questão relevante é que, no ambiente de produção do `MidHPC`, processos, geralmente, apresentam comportamentos distintos de utilização de recursos no tempo, o que influencia na dinâmica e, portanto, na topologia das redes neurais adotadas. Sendo assim, um processo pode executar melhor com x neurônios interconectados em y camadas por

meio de z ligações, enquanto outro requer uma arquitetura de rede neural distinta. Esses problemas tornam complexa a adoção automática e transparente desses modelos conexionistas, que necessitam de intervenção para parametrização. Essa intervenção evidencia a necessidade de descoberta da topologia adequada para extração da dinâmica de séries temporais. Além dos problemas apresentados, as redes neurais artificiais armazenam conhecimento sobre séries temporais em parâmetros neuronais e nos pesos associados a conexões entre neurônios. Dessa forma, o conhecimento adquirido na fase de treinamento encontra-se implícito no modelo.

Esses problemas motivaram estudos na área de sistemas dinâmicos com o objetivo de compreender, modelar e predizer estados comportamentais de processos. Foram estudadas ferramentas de análise de tendências, estabilidade e correlação de histórico com eventos futuros. Essas ferramentas resultam em medidas descritivas, ao contrário das redes neurais, explícitas, que auxiliam a compreender o comportamento de séries temporais e reconstruí-las em espaços multidimensionais. Tais reconstruções evidenciam relações temporais entre estados de um sistema e auxiliam na obtenção de modelos de predição.

Esses estudos sobre sistemas dinâmicos motivaram esta tese de livre docência que emprega tais conceitos, em conjunto com técnicas inteligentes, a fim de modelar e predizer o comportamento de processos com o objetivo de otimizar operações de escalonamento em ambientes de grade computacional. A estabilidade assintótica desses comportamentos é avaliada por meio do cálculo do expoente de Lyapunov (Edmonds, 1996), a correlação de histórico e seu auxílio na predição de eventos é realizada pelo cálculo do expoente de Hurst (Kaplan, 2003). Essas duas ferramentas permitem mensurar o quão complexos são os comportamentos em estudo. Em seguida, aplicam-se técnicas de reconstrução de séries, baseadas no teorema de imersão de Whitney (1936), as quais auxiliam a desdobrar estados de um sistema em estudo e permitem observar suas relações. Essas relações são, posteriormente, modeladas por meio de uma rede neural artificial, responsável por predizer próximos estados.

Experimentos consideraram a modelagem e predição do comportamento de comunicação entre processos. Eventos de comunicação previstos foram empregados por uma nova política de escalonamento de processos, denominada `PredRoute`, que utiliza uma abordagem baseada em algoritmos genéticos para minimizar o tempo consumido em trocas de mensagens. Os bons resultados obtidos com essa política confirmam as contribuições advindas da compreensão da dinâmica de comportamento de processos.

Esta tese de livre docência está organizada da seguinte maneira:

- Capítulo 2 – apresenta as pesquisas desenvolvidas no contexto do projeto `MidHPC`, suas relações e resultados. Destacam-se a evolução de pesquisas em avaliação de desempenho de rede de computadores, na proposta de modelos estocásticos para caracterizar capacidade e carga de recursos computacionais, em modelos para analisar influências que cargas de trabalho causam em ambientes reais, e em investigações

sobre novas políticas de escalonamento de processos. Em seguida, apresenta-se a relação dessas pesquisas com orientações de mestrado e doutorado;

- Capítulo 3 – apresenta conceitos sobre sistemas dinâmicos e ferramentas para análise de estabilidade assintótica, correlação de eventos históricos, e técnicas de reconstrução de séries em espaços multidimensionais, as quais permitem estudar e observar relações entre estados de um sistema. Esse capítulo também apresenta técnicas de aproximação de funções, utilizadas para modelar o comportamento reconstruído de um sistema dinâmico e predizer próximos estados.

- Capítulo 4 – descreve o objetivo, metodologia e resultados obtidos com a aplicação de conceitos sobre sistemas dinâmicos e técnicas inteligentes na otimização de escalonamento de processos em grades computacionais;

- Capítulo 5 – destaca as conclusões e trabalhos futuros.

Capítulo

2

Projeto MidHPC

2.1 Considerações iniciais

As últimas décadas têm sido decisivas para a indústria da tecnologia de informação. Tecnologias tais como redes de computadores, serviços Web, dispositivos móveis e celulares foram propostas e têm sido adotadas em diferentes segmentos do mercado. Elas modificaram o cotidiano de grande parte da população, influenciando, decisivamente, áreas como venda de produtos, setor bancário e, principalmente, comunicação. O potencial dessas tecnologias motivou novos estudos relativos à escalabilidade de sistemas quanto ao volume de processamento, transações e dados, como também em aspectos de segurança e tolerância a falhas. Essa evolução auxiliou no aumento da produtividade de instituições, contudo, criou novos desafios para equipes de tecnologia da informação, que devem ser capazes de gerenciar, manter e integrar tecnologias resultantes (Parashar & Hariri, 2006).

Esses desafios motivaram Paul Horn, vice-presidente da divisão de pesquisas da `IBM`, a propor, em 2001, o conceito de computação autônoma, para abordar a complexidade envolvida nas tarefas de gerenciamento e manutenção de sistemas computacionais. Segundo Horn (Murch, 2004), esses desafios têm impossibilitado o crescimento de empresas, pois muito tempo e recursos são despendidos em tarefas de manutenção e gerenciamento de infra-estruturas de tecnologia da informação (Parashar & Hariri, 2006).

O custo de manutenção e gerenciamento de tais infra-estruturas não é o único problema relacionado ao novo mercado de tecnologia. A dificuldade em encontrar recursos humanos capacitados agrava, ainda mais, esse cenário. Outro aspecto negativo é que, em geral, as tarefas de manutenção e gerenciamento são repetitivas, o que tende ao desgaste da equipe (Murch, 2004). A área de computação autônoma visa prover capacidade de auto-gerenciamento para infra-estruturas tecnológicas. Conseqüentemente, sistemas seriam

capazes de observar e analisar informações, assim como responder, automaticamente, a variações do meio. Com tal mudança no paradigma de gerenciamento, equipes de tecnologia da informação poderiam focar no desenvolvimento do negócio, livrando-se de tarefas repetitivas. Para prover tal suporte de gerenciamento, a área de computação autônoma trata quatro principais aspectos:

1. Auto-otimização – capacita o sistema a otimizar-se automática e proativamente. Por exemplo, um sistema seria capaz de realocar processos, com base em modificações de comportamento, ou de distribuir dados a fim de aumentar a vazão de leitura e escrita;

2. Auto-cura – capacita o sistema a prevenir-se e recuperar-se, automaticamente, de falhas por meio da descoberta, diagnóstico e busca de soluções alternativas. Por exemplo, considere o salvamento periódico de contextos (*checkpoints*) de processos. Na queda de um processo, pode-se recuperar seu estado a partir do último contexto salvo, evitando que seja reiniciado do zero, o que gera consumo de recursos e tempo adicional de espera para usuários;

3. Auto-proteção – capacita o sistema a detectar, identificar e defender-se de intrusos, vírus, ataques externos, etc. Por exemplo, considere uma operação de *denial-of-service* (Hussain *et al.*, 2003) oriunda de múltiplos computadores externos. Uma possível solução seria bloquear essas requisições. Em outro cenário, considere um processo requisitando, sem permissão, a leitura de dados do disco rígido. Esse acesso seria considerado inesperado, e bloqueado pelo sistema.

4. Auto-configuração – capacita o sistema a reconfigurar seus serviços em função de alterações no meio (tal como a replicação de um serviço para atender mais requisições ou para aumentar sua disponibilidade) ou devido a mudanças solicitadas pelos usuários (tais como a instalação de novos softwares ou atualização dos existentes sem, contudo, prejudicar a execução do ambiente);

A mesma evolução tecnológica, que acarretou o surgimento da computação autônoma, motivou estudos para empregar recursos heterogêneos e geograficamente distribuídos, a fim de prover infra-estruturas de processamento, distribuição de dados, alta disponibilidade, etc. Essas infra-estruturas, denominadas grades computacionais, apresentam alta complexidade de gerenciamento, dada a inerente integração de diversas plataformas, serviços, formas de comunicação e usuários. Essa complexidade fez com que a área de grades computacionais fosse um dos principais focos de pesquisa em computação autônoma como observado a seguir (Parashar & Hariri, 2006; Champrasert *et al.*, 2005; Abawajy, 2005; Oprescu *et al.*, 2008; Omar *et al.*, 2006).

Parashar & Hariri (2006) propõem o projeto `AutoMate`, que visa investigar soluções autônomas para grades computacionais, baseadas em sistemas biológicos, para abordar

questões relativas à complexidade, dinamismo, heterogeneidade e incerteza decorrente de variações de comportamento de processos, ocupação de recursos, volume de transações, etc. Os autores propõem modelos conceituais e arquiteturas para apoiar o desenvolvimento e execução de aplicações auto-gerenciáveis. Pesquisas decorrentes têm investigado modelos de programação, arcabouços e serviços de *middleware* para grades (Champrasert & Suzuki, 2007, 2006a,b).

O projeto `SymbioticSphere` (Champrasert *et al.*, 2005) aplica conceitos e mecanismos biológicos no desenvolvimento de sistemas para grades. No `SymbioticSphere`, serviços são implementados como agentes de software distribuídos e autônomos, contendo comportamentos similares a entidades biológicas, tais como replicação, morte, migração, emissão de feromônio, sensibilidade ao meio, armazenamento e consumo de energia. Um *middleware* executa em múltiplos computadores e oferece suporte para esses serviços. Serviços recebem energia conforme atendem requisições de usuários ou de outros serviços e a consomem ao solicitar operações. A abundância de energia indica alta demanda pelo serviço, o que favorece, por exemplo, seu consumo em replicação, permitindo o atendimento de um maior volume de requisições. Por outro lado, a escassez indica falta de demanda, levando, eventualmente, à morte do serviço.

Abawajy (2005) propõe um arcabouço escalável que aborda o problema de escalonamento de processos em grades computacionais. Essa proposta replica aplicações em execução sobre diversos computadores, aumentando a disponibilidade do sistema.

Oprescu *et al.* (2008) concluem que o gerenciamento de recursos heterogêneos motiva pesquisas a fim de propor soluções autônomas para grades computacionais. No mesmo trabalho os autores avaliam a integração de *toolkits*, tais como o `JavaGAT` (van Nieuwpoort *et al.*, 2007), e ambientes de programação, tais como o `Satin` (Nieuwpoort *et al.*, 2005) e `Muskel` (Danelutto & Dazzi, 2006), a fim de prover autonomia ao projeto `CoreGrid`.

Omar *et al.* (2006) apresentam um modelo para prover serviços autônomos para grades computacionais. Esse modelo emprega reserva de recursos e escalonamento de tarefas a fim de estimar necessidades futuras de serviços. Reservas e escalonamentos são conduzidos com base em informações extraídas do meio e classificadas por meio de mapas auto-organizáveis (*Self-Organizing Maps* – `SOM`).

A escassez de pesquisas relacionadas ao estudo, projeto e avaliação de técnicas e modelos a fim de prover funcionalidades autônomas para grades motivou este pesquisador a propor, em 2003, o projeto `MidHPC`, que visa projetar uma camada de software transparente e capaz de minimizar o tempo de execução de processos (por meio da otimização de escalonamento, utilização de recursos, e acesso a dados distribuídos), oferecer suporte de alta disponibilidade, reconfiguração e segurança para aplicações legadas (existentes) sobre grades computacionais.

Usuários podem submeter quaisquer aplicações sobre o `MidHPC`, da mesma forma que o fazem em sistemas operacionais de propósito geral (e.g. Linux). Essas aplicações são

enviadas à grade que fica responsável por alocar, de forma otimizada, seus processos, distribuir seus dados em regiões que maximizem a vazão de leitura e escrita em meio de armazenamento secundário, gerar e replicar contextos intermediários (*checkpoints*) com o objetivo de retomar execuções em caso de falhas, observar comportamentos de interação de usuários e processos a fim de detectar e anular ações de intrusos.

2.2 Pesquisas Desenvolvidas

As primeiras pesquisas desenvolvidas no projeto `MidHPC` foram voltadas para a extração de dados de atrasos, larguras de banda e cargas geradas por tarefas sobre o meio de comunicação. Alguns modelos para extração dessas informações foram estudados (Hockney, 1996; Culler *et al.*, 1993) e implementados a fim de avaliar atrasos gerados por diferentes protocolos de comunicação em redes locais, metropolitanas e mundiais (Ferreira & Mello, 2004; Dodonov *et al.*, 2005). Resultados foram empregados na construção de modelos estocásticos para caracterizar atrasos do sistema de comunicação.

Os modelos estocásticos permitiram compreender e simular as reações, em termos de latência e largura de banda, de diferentes tipos de redes de computadores. Esses modelos foram, em seguida, empregados em diversos estudos sobre escalonamento, sendo o primeiro voltado para avaliar fatores positivos e negativos envolvidos na transferência de processos entre computadores. Esse mesmo problema foi, anteriormente, abordado por Harchol-Balter & Downey (1997), que propuseram um modelo de custo de migração baseado na idade de processos. Segundo esse estudo, há benefícios na migração de processos de longa duração, todavia esse trabalho não avalia a influência da carga imposta por processos ao sistema. Processos com longos períodos de execução, mas que apresentam pouca ocupação, quando migrados, não agregam benefícios para o computador emissor. Além disso, essas transferências geram acréscimos ao tempo total de execução, pois há custos de comunicação, salvamento e reinício de contexto.

Observando essas limitações, Mello & Senger (2004) propuseram um modelo para quantificar impactos causados na migração de processos em ambientes heterogêneos. Esse modelo propõe um conjunto de equações que permite decidir sobre a transferência de processos entre computadores. Em seguida, os autores projetaram uma heurística de migração de processos baseada no modelo proposto, a qual foi comparada ao modelo de Harchol-Balter & Downey (1997). Para avaliar o novo modelo, foi desenvolvido um simulador (que, posteriormente, deu origem ao `SchedSim` (Mello & Senger, 2006)), o qual foi submetido a cargas de trabalho de ambientes de produção (Feitelson *et al.*, 1997). Experimentos utilizando esse simulador consideraram atrasos de comunicação caracterizados pelos modelos estocásticos de redes de computadores anteriormente obtidos. Resultados comprovaram que o novo modelo supera o anterior, pois oferece maior precisão ao quantificar atrasos envolvidos na migração de processos.

Os resultados obtidos devem-se ao modelo e à boa caracterização de atrasos e largura de banda em sistema de comunicação. Esses resultados motivaram novas pesquisas para obter conhecimentos sobre aplicações e demais recursos computacionais, a fim de otimizar o escalonamento de processos e reduzir comunicação (Ishii *et al.*, 2005; Senger *et al.*, 2005b). Foi proposta uma política de escalonamento baseada no comportamento de comunicação entre processos, que considera consumos históricos de rede com o objetivo de definir melhores regiões para executar aplicações no ambiente distribuído. Essa política pode chegar ao extremo de recomendar a alocação de muitos processos sobre um único computador. Resultados experimentais permitiram comprovar que essa técnica reduz, significativamente, o tempo médio de execução de aplicações (Ishii *et al.*, 2005).

As melhorias de desempenho ao empregar modelos estocásticos e comportamento de comunicação de processos motivaram novos estudos para obter conhecimentos sobre aplicações paralelas e considerá-los na otimização de escalonamento. Esses estudos deram origem a um modelo de classificação de operações executadas por processos, que permite caracterizar estados de ocupação de recursos (tais como utilização de `CPU`, memória, disco rígido e rede).

Esses estados de ocupação são, incrementalmente, armazenados em bancos de dados. Ao receber uma requisição para execução de uma nova aplicação, o modelo extrai parâmetros dessa solicitação e os encaminha para um algoritmo de aprendizado baseado em instâncias (`IBL`). Esse algoritmo considera execuções prévias e similaridade entre aplicações para estimar o comportamento médio esperado de uma aplicação, em termos de consumo de `CPU`, memória, dados transferidos com disco rígido e rede. Os conhecimentos obtidos por tal modelo foram então empregados em uma metaheurística de otimização de escalonamento de processos baseada em algoritmos genéticos, denominada *Genetic Algorithm for Scheduling* (`GAS`)(Senger *et al.*, 2005b). Resultados comparativos comprovam que essa heurística agrega contribuições significativas a outras propostas na literatura.

Resultados do modelo de classificação e da heurística `GAS` motivaram a extração de características de processos em ambientes reais. O *kernel* do sistema operacional Linux foi modificado a fim de monitorar, periodicamente, operações de processos, tais como leituras e escritas em disco e rede, utilização de `CPU` e ocupação de memória. No mesmo *kernel* foi introduzida uma implementação da rede neural artificial *Adaptive Resonance Theory 2A* (`ART-2A`) (Carpenter & Grossberg, 1989), para classificar as operações monitoradas e caracterizar comportamentos de processos. Essa rede neural auto-organizável e não supervisionada permitiu aplicar o modelo de classificação proposto e, conseqüentemente, gerar estados de utilização de recursos para processos. Esses estados foram utilizados para estimar cadeias de Markov, as quais foram atualizadas de maneira incremental e auxiliaram na otimização de escalonamento de processos em ambientes reais (Senger *et al.*, 2005a).

No entanto, para otimizar decisões de escalonamento, seria necessária a modelagem da

carga de utilização de processos em ambientes distribuídos. A partir dessa modelagem, poder-se-ia conhecer as influências, em termos de utilização de recursos, decorrentes da distribuição de processos. Nesse contexto, propôs-se o modelo `UniMPP` (Mello & Senger, 2006), que caracteriza ocupações geradas por processos sobre recursos computacionais adotando os modelos de consumo de `CPU` de Amir (2000) e Mello *et al.* (2002), de tempo despendido na transmissão de mensagens de Culler *et al.* (1993) e Sivasubramaniam *et al.* (1994), de caracterização do volume de mensagens e modelos estocásticos de geração de mensagens de Chodnekar *et al.* (1997) e Vetter & Mueller (2003).

O `UniMPP` permite quantificar as cargas que processos causam ao ambiente distribuído. Dessa maneira, pode-se, por exemplo, avaliar variações de ocupação de `CPU`, memória, disco e rede decorrentes do escalonamento ou migração de processos. Dentre as variáveis desse modelo estão ocupações causadas por processos e capacidades de recursos computacionais. A primeira pode ser obtida pelo modelo de classificação anteriormente apresentado. Para obter a segunda, projetou-se um conjunto de ferramentas de *benchmark* (Mello & Senger, 2006). Um simulador para o modelo `UniMPP`, denominado `SchedSim`, foi desenvolvido, o qual permite estudar políticas de escalonamento, balanceamento de carga e migração de processos em ambientes distribuídos homogêneos e heterogêneos de diferentes escalas.

O modelo `UniMPP` e o simulador `SchedSim` motivaram diferentes frentes de pesquisa com o intuito de projetar heurísticas de escalonamento de processos. Resultados anteriormente obtidos com a extração de comportamento de processos também foram considerados por essas frentes. Dentre pesquisas decorrentes estão a proposta de uma política de escalonamento de processos empregando colônias de formigas (Nery *et al.*, 2006) e outra baseada em vizinhanças (Mello *et al.*, 2006a).

A primeira empregou a técnica de otimização baseada em colônias proposta por Dorigo & Di Caro (1999), onde formigas deixam feromônio no caminho que percorrem do ninho à fonte de alimento. O acúmulo de feromônio em um caminho indica que esse é o mais adequado, auxiliando na busca por soluções aproximadas para problemas de otimização. No contexto do trabalho mencionado, o feromônio foi empregado para indicar o computador mais apto para executar processos. Quanto maior o desempenho de um computador, mais rapidamente ele atende determinado processo, o que gera aumento no nível de feromônio entre o módulo responsável por escalonamentos e o computador. Esse acréscimo favorece futuras escolhas do computador. A segunda política, denominada `Route`, agrupa computadores próximos, em função de atrasos de comunicação, e utiliza essas vizinhanças para distribuir processos. Simulações conduzidas com o `SchedSim` comprovam que ambas políticas melhoram, significativamente, os resultados de abordagens existentes.

Há, contudo, situações em que uma vizinhança de computadores não tem capacidade computacional ou latência mínima capaz de atender dada aplicação. Nessas circunstâncias a política `Route` migra processos até encontrar uma região da grade com recursos

suficientes. Contudo, essa migração gera atrasos no tempo médio de execução de aplicações, o que motivou o desenvolvimento da política *Route with Genetic Algorithm Support* (`RouteGA`). Essa política emprega um algoritmo genético para realizar o primeiro escalonamento de processos, com base no comportamento de ocupação de recursos obtido por meio do algoritmo de aprendizado baseado em instâncias. Essa modificação da política `Route` original gerou melhorias significativas de desempenho (Mello *et al.*, 2007c, 2008).

As pesquisas mencionadas estudaram aspectos de extração e aplicação de comportamento de processos a fim de otimizar operações sobre ambientes de grade computacional. Os bons resultados motivaram o desenvolvimento de um protótipo do projeto `MidHPC` com o objetivo de comprovar os estudos realizados e oferecer funcionalidades autônomas para ambientes de grade reais.

2.3 Protótipo

Os resultados de pesquisa apresentados na seção anterior foram reunidos em um protótipo do projeto `MidHPC`. Esse protótipo permite a execução de aplicações desenvolvidas segundo o paradigma concorrente e que empregam o padrão de *threads* `POSIX`. Essas aplicações não necessitam ser modificadas ou recompiladas para executar na grade. *Threads* dessas aplicações são transparentemente interceptadas e suas múltiplas linhas de execução transformadas em processos.

Estimativas da ocupação de recursos causada por esses processos são obtidas por um algoritmo de aprendizado baseado em instâncias, e são utilizados pela política `Route` e `RouteGA` para escalonar processos de forma otimizada. Tais processos podem, ainda, comunicar-se, de forma transparente, utilizando um sistema de memória compartilhada distribuída (*Distributed Shared Memory* – `DSM`) introduzido por Mello *et al.* (2007b); Andrade Filho *et al.* (2008). Os módulos que compõem esse protótipo são apresentados a seguir.

2.4 Módulos

O protótipo do `MidHPC` é composto pelos seguintes módulos: `Broker Global`, `Broker Local`, `Scheduler` e `Shell`. Ambos `Brokers` são utilizados para montar uma estrutura hierárquica de gerenciamento de recursos da grade. O `Scheduler` é responsável pela alocação e execução de processos de usuário e o `Shell` por uma interface para a submissão e gerenciamento de processos. A arquitetura concebida[1] é apresentada na figura 2.1. Seções seguintes apresentam detalhes sobre esses módulos.

[1]Essa arquitetura foi projetada sem o intuito de oferecer a melhor estrutura de comunicação para grades, mas sim aplicar conceitos propostos em pesquisas.

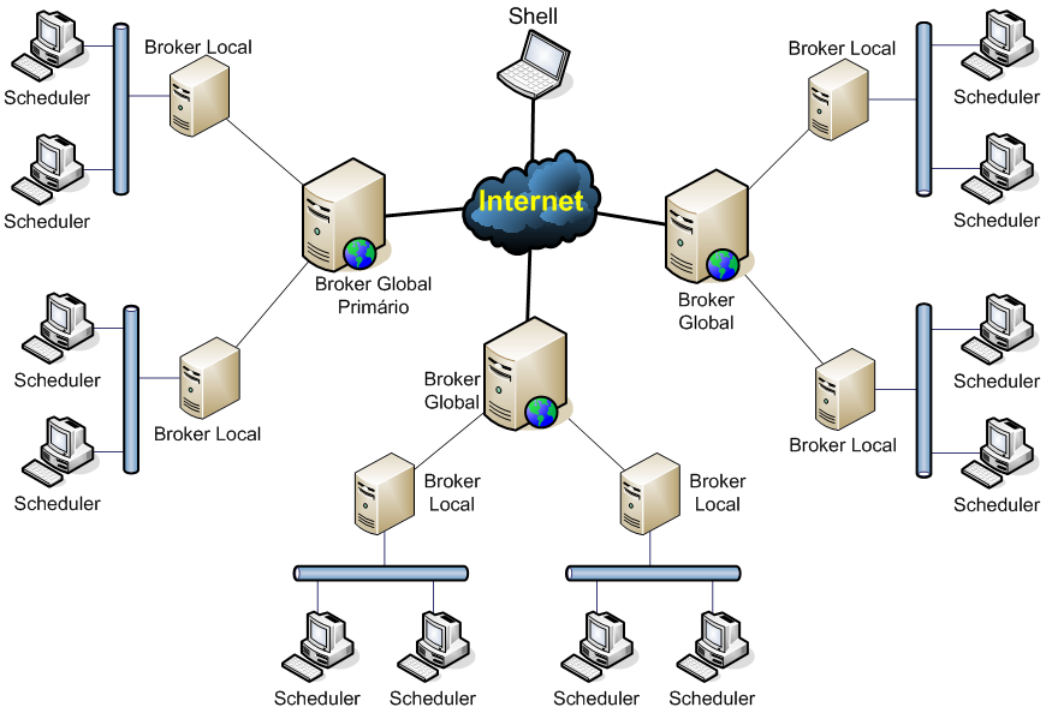


Figura 2.1: Arquitetura do Protótipo do Projeto MidHPC

2.4.1 Brokers Global e Local

O `Broker` é o módulo utilizado para conectar redes de computadores que compõem a grade. Ele é composto por dois gerenciadores distintos: `Broker Local` e `Broker Global`. O `Broker Global` gerencia conjuntos de `Brokers Locais`, os quais se comunicam diretamente com computadores que participam da grade. Esses computadores executam o módulo `Scheduler` e geram, periodicamente, informações para seus respectivos `Brokers Locais`. Essas informações incluem capacidade de processamento, latência de acesso à memória principal e virtual, vazão de leitura e escrita em disco e a carga atual de `CPU` de cada computador. O `Broker Local` resume essas informações na forma de médias e desvios-padrão que são, posteriormente, submetidos ao `Broker Global`. `Brokers Globais` armazenam tais informações com o intuito de conhecer capacidades e cargas médias dos recursos da grade, as quais são empregadas pelas políticas de escalonamento de processos `Route` e `RouteGA`.

`Brokers Globais` comunicam e sincronizam-se a fim de ter a mesma visão sobre capacidades e cargas da grade. O primeiro `Broker Global` iniciado no ambiente exerce o papel de primário. Dados do primário são replicados para os demais, com o objetivo de aumentar a disponibilidade do sistema. Requisições de usuários, oriundas do módulo `Shell`, são atendidas pelo `Broker Global` primário. Essas requisições podem iniciar aplicações ou obter informações sobre a execução de processos e ambiente.

2.4.2 Shell

Para simplificar a interação de usuários, foram desenvolvidas ferramentas de submissão e controle de execução de aplicações, as quais deram origem ao módulo `Shell`. Esse módulo executa sobre um sistema de arquivos distribuídos que contém uma imagem única do sistema operacional GNU/Linux. Para submeter uma aplicação à grade, o usuário deve apenas copiá-la para o sistema de arquivos e, após isso, iniciá-la. Caso a imagem do sistema operacional não contenha alguma das bibliotecas requeridas pela aplicação, o computador que a iniciou, instala, automaticamente, todas dependências, utilizando a ferramenta *apt-get*. As seguintes ferramentas compõem o módulo `Shell`:

1. `mkill` – solicita ao `Broker Global` o término de uma aplicação. Essa requisição é redirecionada para `Schedulers` que executam processos dessa aplicação;

2. `mps` – requisita informações sobre a topologia de interconexão dos `Brokers Globais`, `Brokers Locais` e `Schedulers`, dados de atraso entre redes de computadores envolvidas na grade e consumo de recursos por parte de processos;

3. `mstart` – solicita o início de uma aplicação ao `Broker Global`.

2.4.3 Scheduler

O módulo `Scheduler` oferece funções para a execução de processos de usuário, monitoramento de processos e recursos, troca de informações com `Broker Local`, e ativação do serviço de escalonamento de aplicações. Para monitorar processos foi desenvolvido um software que captura chamadas de sistema (orientado a eventos) utilizando a biblioteca *ptrace* do GNU/Linux. Esse software elimina aspectos intrusivos de uma versão anterior que extrai essas informações por meio de chamadas de sistema no *kernel* do Linux (Senger *et al.*, 2005a; Dodonov *et al.*, 2004). Informações monitoradas incluem leituras e escritas em meio de armazenamento secundário e redes de computadores, as quais são armazenadas em um banco de dados, que, posteriormente, é utilizado por um algoritmo de aprendizado baseado em instâncias. Para monitorar recursos computacionais foi desenvolvido um software que captura, periodicamente, ocupações de `CPU`, memória, volume de transferências para/de disco e rede. Informações sobre ocupações são resumidas em médias, encaminhadas para os `Brokers`, que criam uma imagem única e global sobre a carga do ambiente.

O `Scheduler` é também responsável por ativar o serviço de escalonamento. Ao receber uma nova aplicação, esse módulo utiliza os serviços do algoritmo de aprendizado baseado em instâncias para estimar o comportamento de processos, com base em histórico de execuções. Esse comportamento, os dados de monitoração de demais processos em execução e as informações de carga de recursos são submetidos à uma interface de escalonamento.

Qualquer escalonador pode ser conectado à essa interface. Detalhes sobre o algoritmo de aprendizado baseado em instâncias e sobre as duas políticas de escalonamento atualmente conectadas ao `MidHPC` são apresentados a seguir.

Algoritmo de Aprendizado Baseado em Instâncias

Aprendizado Baseado em Instâncias (`IBL`) é um paradigma de aprendizado que orienta a construção de algoritmos para encontrar instâncias similares em uma base de experiências por meio de funções de domínio real ou discreto (Aha *et al.*, 1991; Mitchell, 1997). O aprendizado consiste em armazenar dados de treinamento e utilizá-los para realizar buscas e aproximações.

Cada instância é composta por um conjunto de atributos, os quais podem ser classificados como entradas ou saídas: atributos de entrada descrevem as condições em que uma experiência foi observada; enquanto atributos de saída representam os resultados, de acordo com as condições descritas pelos atributos de entrada.

Os algoritmos `IBL` calculam a similaridade entre uma instância de consulta, x_q, e as instâncias da base de experiências, retornando um conjunto resposta. Nesses algoritmos, apenas as instâncias mais relevantes são utilizadas para estimar atributos de saída para o ponto de consulta.

O algoritmo `IBL` adotado no projeto `MidHPC` segue o modelo proposto por Senger *et al.* (2007), o qual é baseado no aprendizado dos k-vizinhos mais próximos (*k-nearest neighbor*). O método assume que todas as instâncias em uma base de experiências correspondem a pontos em um espaço $\mathbb{R}^n$ e que uma função de distância, nesse caso Euclidiana, é utilizada para definir pontos próximos à consulta.

Seja uma instância x descrita por um vetor de atributos $(a_1(x), a_2(x), \ldots, a_n(x))$, em que $a_r(x)$ descreve o valor do r-ésimo atributo de um instância x. A distância entre duas instâncias x_i e x_j é dada por $E(x_i, x_j) = \sqrt{\sum_{r=1}^{n} d(a_r(x_i), a_r(x_j))^2}$, em que a função distância $d(.)$ é definida pelas equações 2.1 e 2.2.

$$d(x,y) = \begin{cases} 1 & \text{se em } x \text{ ou } y \text{ faltam atributos} \\ d_n(x,y) & \text{se os atributos são nominais} \\ \|x-y\| & \text{caso contrário} \end{cases} \tag{2.1}$$

$$d_n(x,y) = \begin{cases} 0 & \text{se } x = y \\ 1 & \text{caso contrário} \end{cases} \tag{2.2}$$

Um problema recorrente da distância Euclidiana ocorre quando um dos atributos apresenta intervalo mais amplo, nesse caso ele pode dominar as saídas da equação de distância, prejudicando os resultados do algoritmo (Wilson & Martinez, 1997). Por exemplo, se um atributo apresentar valores entre 1 e 1024 e o outro estiver entre 1 e 5, o segundo será

dominado pelo primeiro. Pode-se solucionar essa questão por meio da normalização de atributos numéricos em uma mesma escala. Senger *et al.* (2007) adotam uma função de normalização linear $h(x_i)$ que resulta em valores no intervalo $[0;1]$. Seja $\min_{x_i}$ o menor valor na base de experiências para o atributo x_i e $\max_{x_i}$ o maior valor para o mesmo atributo, a função $h(x_i)$ é definida na equação 2.3.

$$h(x_i) = \frac{x_i - \min_{x_i}}{\max_{x_i} - \min_{x_i}} \tag{2.3}$$

Assim, considera-se a equação 2.4 para aproximar uma função de domínio de valores reais $f : \mathbb{R}^n \rightarrow \mathbb{R}^+$, utilizando pesos w_i para cada um dos k vizinhos mais relevantes, de acordo com a distância ao ponto de consulta x_q.

$$g(x_q) \leftarrow \frac{\sum_{i=1}^{k} w_i f(x_i)}{\sum_{i=1}^{k} w_i} \tag{2.4}$$

Os pesos w_i são calculados segundo uma função, a qual gera resultados próximos a um valor constante, quando a distância tende a 0, e aproxima de 1, à medida que a distância tende ao infinito. Embora diferentes funções de ponderação possam ser empregadas, o `MidHPC` adota a proposta de Senger *et al.* (2007), que aplica uma função Gaussiana centrada no ponto x_q com uma variância t, conforme a equação 2.5 (o termo $E(x_q, x_j)$ representa a distância Euclidiana entre o ponto de consulta x_q e outro ponto qualquer x_j).

$$w_i(x_j) = e^{\frac{-E(x_q,x_j)}{t^2}} \tag{2.5}$$

A figura 2.2 apresenta exemplos de valores para w gerados pela equação 2.5, considerando um intervalo de distâncias Euclidianas entre 0 e 10, além de $t^2 = \{0,125; 0,250; 0,500; 1,000; 2,000\}$.

A técnica `IBL` proposta em (Senger *et al.*, 2007) é empregada pelo `MidHPC` para estimar a ocupação de recursos gerada por processos. Essa técnica é acionada antes de iniciar aplicações e tem sido empregada na parametrização de algoritmos de escalonamento de processos. Para exemplificar o uso do `IBL`, considere uma aplicação α composta por três processos (α_1, α_2 e α_3). Durante a execução da aplicação, os processos α_1, α_2 e α_3 são monitorados e seus dados de ocupação de recursos são salvos em uma base de dados. Em seguida, o `IBL` estima, com base nos históricos, a ocupação de recursos para novas aplicações. Essas estimativas são utilizadas para otimizar decisões de escalonamento.

Políticas de Escalonamento

Dentre os estudos de escalonamento realizados, comprovou-se que as políticas `Route` e `RouteGA` apresentam melhores resultados (Mello *et al.*, 2006a, 2007c). O funcionamento dessas políticas, incorporadas ao protótipo do `MidHPC`, é apresentado a seguir.

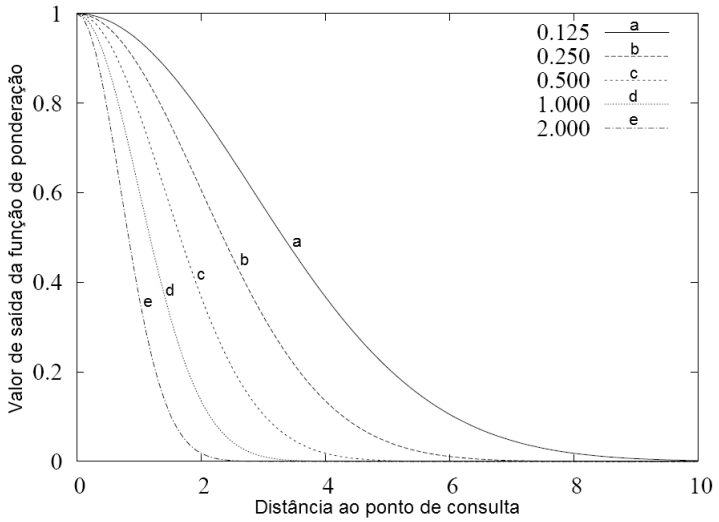


Figura 2.2: Comportamento da função de ponderação

Route

`Route`(Mello *et al.*, 2006b)[2] é uma política de escalonamento de processos que visa o balanceamento de carga em ambientes de grade, caracterizados por grande quantidade de recursos, heterogeneidade, alta latência de comunicação e um grande número de usuários. Algoritmos de balanceamento de carga utilizam-se de reatribuição dinâmica de processos para prover uma distribuição eqüitativa de utilização de recursos (Shivaratri *et al.*, 1992). A política `Route` foi desenvolvida a partir do conceito de roteamento de mensagens, o qual é utilizado para definir vizinhanças de computadores. Cada computador pode distribuir processos entre seus vizinhos.

Para entender essa política, suponha três redes distintas: α, β e γ. Cada computador, com diferentes capacidades de processamento, memória, disco e acesso a rede, deve executar um módulo capaz de tomar decisões de escalonamento. Depois de iniciar esse módulo em um computador $C_{n,\alpha}$ (computador n da rede α), calcula-se o atraso de comunicação interno dessa rede, definido como RTT_α (*Round-Trip Time* – `RTT`). O atraso (ou latência de comunicação) é o custo para transmitir e receber uma mensagem de tamanho mínimo entre dois computadores da rede α. Após essa etapa define-se o `RTT` máximo (dado pela equação 2.6, em que k é um parâmetro de atraso para comunicação entre computadores. O valor de k pode ser definido de maneira fixa ou adaptativa, baseando-se nas capacidades das redes envolvidas na grade), o qual é utilizado para definir a vizinhança de $C_{n,\alpha}$.

$$RTT_{\max} = k \cdot RTT_\alpha,\ k \geq 1 \tag{2.6}$$

Todos os computadores do ambiente que apresentarem um atraso de comunicação, em

[2] *Best paper award* na *IEEE 20th International Conference on Advanced Information Networking and Applications* – 521 submissões.

relação ao computador $C_{n,\alpha}$, menor que $RTT_{\max}$ são definidos como vizinhos. Quando um computador $C_{n,\alpha}$ inicia uma aplicação paralela, ele requisita informações sobre a carga de seus vizinhos com o objetivo de identificar os mais ociosos. Processos são distribuídos sobre esses computadores.

Como a distribuição de carga é feita entre computadores mais próximos, e tal proximidade é definida pelo atraso de comunicação entre computadores, privilegia-se a utilização de redes com maior desempenho, as quais garantem menores atrasos de sincronização entre processos. Migrações de processos são conduzidas segundo o modelo de custo de migração definido por Mello & Senger (2004), o qual considera a carga dos processos e o custo de transferí-los para outros computadores.

RouteGA

Otimizações na primeira alocação de processos da política `Route` deram origem a uma nova política de balanceamento de carga denominada `RouteGA` (Mello *et al.*, 2007c)[3]. Para ilustrar essas otimizações, suponha que um computador c_i inicie uma aplicação paralela a_j composta por n tarefas, ou processos $(p_0, ..., p_n)$, em um tempo t e que a vizinhança v desse computador, definida pelo atraso de comunicação (equação 2.6), é formada pelos recursos $vc_0, ..., vc_k$, os quais apresentam, respectivamente, capacidades de processamento $cap_0, ..., cap_k$. Suponha, também, que cada recurso da vizinhança tem uma longa fila de processos que gera alta carga por longos períodos.

No `Route`, processos são alocados sobre os v vizinhos do computador c_i, que os redistribuem, de acordo com o modelo de migração de Mello & Senger (2004), até convergir para regiões da grade capazes de atender esses processos. Essa convergência pode ser lenta, o que aumenta o tempo de resposta de aplicações.

Essa limitação motivou o desenvolvimento do `RouteGA`, o qual otimiza a primeira etapa de alocação, considerando o consumo de recursos da aplicação, a capacidade e a carga do ambiente. Informações de consumo são armazenadas em uma base de experiências. Um algoritmo `IBL` é executado sobre essa base a fim de estimar o comportamento médio esperado de novas aplicações que chegam ao ambiente. Esse comportamento é utilizado para parametrizar o algoritmo genético do `RouteGA`. Para isso, o `IBL` requer detalhes de aplicações tais como: nome do executável, argumentos e usuário. O `IBL` retorna a média e o desvio padrão de utilização esperada dos recursos, incluindo a latência e a largura de banda necessárias para comunicação entre processos, as quais são utilizadas para definir a vizinhança de computadores do `RouteGA`. Quanto maior e mais freqüente a troca de informações entre processos, menor é a latência considerada pela política ao distribuir processos entre computadores vizinhos.

O algoritmo genético gera soluções, ou cromossomos, que são avaliados de acordo com

[3] *Outstanding paper* na *IEEE 21st International Conference on Advanced Information Networking and Applications* – 444 submissões.

uma função de aptidão. Um exemplo de cromossomo é $ic = \{0, 3, 8, 5, 7\}$, em que cada índice j do vetor representa um processo da aplicação (o tamanho do cromossomo $|ic| = 5$, representa o número de processos) e cada elemento do vetor descreve onde o processo será alocado. Por exemplo, o processo de índice 0 será alocado no computador 0, o de índice 1 no computador 3 e assim sucessivamente.

O algoritmo genético do `RouteGA` considera a função de aptidão (F_{ic}) apresentada na equação 2.7, onde: MRT_{ic} é o tempo de resposta do processo com maior custo computacional de uma aplicação; η é um parâmetro para evitar divisões por zero e LV_{ic} é a variação de carga que a solução proposta causa ao ambiente (equação 2.8).

$$F_{ic} = \frac{1,0}{MRT_{ic} + \eta} + \frac{1,0}{LV_{ic} + \eta} \tag{2.7}$$

$$LV_{ic} = \sum_{j=1}^{|i|} (C_j - \frac{\sum_{j=1}^{|i|} C_j}{j})^2 j \tag{2.8}$$

O tempo de resposta MRT_{ic} é obtido pela equação 2.9, onde C_j é o custo total de execução do processo j. A equação 2.10 define C_j, onde PM_j é o custo total de processamento do processo j, CAP_d é a capacidade do computador d onde o processo j está alocado, e CC_j é o custo de comunicação do processo j com demais processos da mesma aplicação que executam em outros computadores. A equação 2.11 define o custo de comunicação para dado processo j, onde $CPS(i_w, i_k)$ é o custo de comunicação entre as tarefas de índices w e k do cromossomo i, as quais executam, respectivamente, nos computadores i_w e i_k. $CPS(i_w, i_k)$ é definida na equação 2.12, em que D_{i_w,i_k} representa a quantidade de dados a ser transmitida e LB_{i_w,i_k} a largura de banda entre os computadores que executam as tarefas de índices w e k do cromossomo i.

$$MRT_{ic} = \max(C_j), j = 0, ..., |i| - 1 \tag{2.9}$$

$$C_j = \frac{PM_j}{CAP_d} + CC_j \tag{2.10}$$

$$CC_j = \sum_{w=1}^{|i|} \sum_{k>w}^{|i|} CPS(i_w, i_k) \tag{2.11}$$

$$CPS(i_w, i_k) = \frac{D_{i_w,i_k}}{LB_{i_w,i_k}} \tag{2.12}$$

A função de aptidão visa minimizar o tempo de resposta do processo com maior custo (MRT_{ic}) e causar a menor variação de carga no ambiente (LV_{ic}), o que resulta em melhor balanceamento. Quanto maior o valor F_{ic}, melhor é a solução obtida.

Após executar o algoritmo genético por várias gerações, onde, em cada uma, n indi-

víduos são criados, cruzados e mutados, converge-se para uma solução válida. O maior valor de aptidão obtido representa a melhor solução de alocação de processos encontrada.

Esse algoritmo é utilizado na primeira distribuição de processos em grades. Em seguida, processos podem migrar para computadores vizinhos, segundo o mesmo modelo adotado pela política `Route`.

2.5 Funcionamento do Protótipo

Esta seção detalha a integração e o funcionamento dos módulos do `MidHPC`. Primeiramente, esses módulos devem ser iniciados segundo uma ordem pré-definida (figura 2.3) para constituir o ambiente de grade computacional. Por meio da figura 2.3, observa-se que, após iniciar o `Broker Local`, pode-se escolher entre iniciar o módulo `Scheduler` ou o `Shell`. Porém, todas as funcionalidades do `MidHPC` somente estarão disponíveis caso existam módulos `Scheduler` ativos, pois esses são os responsáveis pela execução de aplicações.

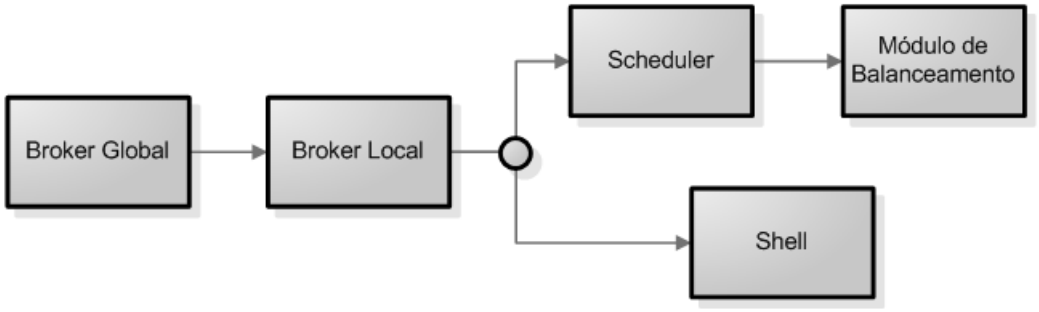


Figura 2.3: Ordem de execução dos módulos do MidHPC

O primeiro módulo a ser iniciado é o `Broker Global` (`BG`) que busca por um `Broker Global` primário. Se não houver resposta, o `Broker` ativo assume o papel de primário (figura 2.4).

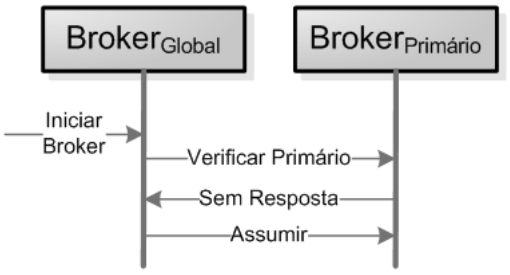


Figura 2.4: Diagrama de Seqüência: Início do Broker (Caso 1)

Caso o primário responda, o `BG` ativo registra-se junto a ele (figura 2.5). Em seguida, dois tipos de mensagens são enviados pelo `Broker Global` primário: o primeiro, para

o novo BG, contendo as localizações dos demais Globais da grade (endereços IP) e, o segundo, para os demais Globais, informando sobre a localização do novo BG.

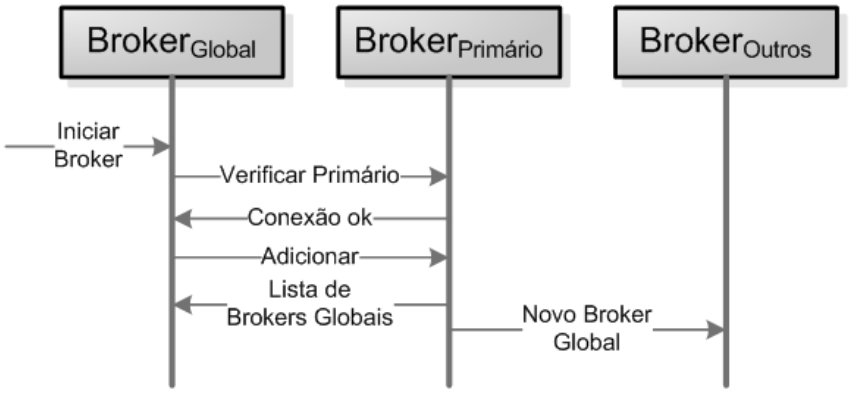


Figura 2.5: Diagrama de Seqüência: Início do Broker (Caso 2)

A busca pelo Broker Global primário utiliza uma entrada em *Domain Name System* (DNS) dinâmico, tal como *broker.midhpc.usp.br*. Para verificar se esse Broker está ativo, realiza-se uma conexão com ele. Caso não seja bem sucedida, o Broker Global que iniciou a conexão modifica a entrada no DNS e, assim, novas requisições serão a ele direcionadas. Embora o nome seja utilizado para referenciar um Broker Global como primário, não existe diferença entre os Globais, pois eles replicam todas as informações. Caso o primário fique inoperante, o primeiro BG, que a ele requisitar alguma informação, assumirá o papel de primário. Além disso, periodicamente, ocorre uma verificação para identificar Brokers ativos.

As distâncias entre os BGs são definidas na forma de latências do sistema de comunicação (RTT (Hockney, 1996)) as quais são, periodicamente, calculadas de modo a manter valores atualizados. Essas medidas são realizadas por meio de requisições ICMP e armazenadas em uma base de dados.

Após o início do Broker Global, pode-se iniciar Brokers Locais que se registram junto ao Global. Há um arquivo de configuração do Broker Local que indica qual será o Global a se conectar. A partir desse momento, a infra-estrutura básica do ambiente está criada. Com os Brokers Global e Local em execução, o próximo módulo a ser iniciado é o Scheduler (seção 2.4.3). Ao iniciar, o Scheduler verifica as capacidades do computador[4] e envia uma mensagem para o Broker Local definido em arquivo de configuração. É através dessa mensagem que o Scheduler é adicionado na lista do Broker Local. Em seguida, o Scheduler armazena as informações de capacidade coletadas, juntamente com os endereços IP dos Brokers Local e Global (utilizados para identificar a rede que o Scheduler pertence), em uma base de dados. Posteriormente, o Broker

[4]Capacidade em milhões de instruções por segundo (MIPS), *throughput* de leitura e escrita em disco (bytes por segundo), quantidade (bytes) e desempenho (bytes por segundo) de memórias principal e virtual. Essas informações são coletadas pelas ferramentas de *benchmark* propostas por Mello & Senger (2006).

`Local` atualiza suas informações de capacidade e carga, em decorrência da adição de um novo `Scheduler` no ambiente, e envia uma mensagem (com detalhes sobre todos seus recursos) para atualizar as informações do `Broker Global` (figura 2.6).

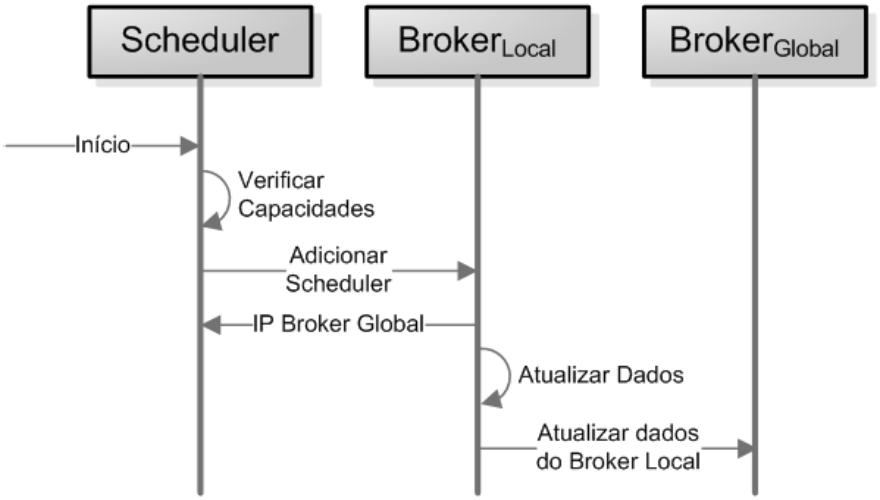


Figura 2.6: Diagrama de Seqüência: Início do Scheduler

O módulo `Shell` necessita que apenas o `Broker Global` primário esteja em execução, pois ele não se conecta diretamente a um `Scheduler`. Entretanto, nenhuma aplicação pode ser iniciada sem `Schedulers` ativos, pois eles disponibilizam recursos para a grade.

Requisições de usuário são enviadas ao `Broker Global` primário, que é responsável por respondê-las. Requisições provenientes do comando `mps` solicitam ao `Broker Global` a coleta de informações sobre aplicações e ambiente. Aquelas oriundas de um comando `mkill <gid>` solicitam o término de execução de processos da aplicação com identificador `<gid>`.

Quando uma requisição `mstart myapp` (onde `myapp` representa o nome da aplicação e seus argumentos) é recebida pelo `Broker Global` primário, o `BG` com menor índice de carga I_B é selecionado para iniciá-la. Esse índice é definido segundo a equação 2.13, onde B_{Cap} é o somatório da capacidade de todos os computadores conectados a determinado `Broker Global`, e B_{Carga} é o somatório da carga a ser executada.

$$I_B = \frac{B_{Carga}}{B_{Cap}} \tag{2.13}$$

O `Broker Global` selecionado envia as informações da aplicação para o `Broker Local` com menor carga, o qual escolhe um `Scheduler`, também com menor carga, para enviar a aplicação. Em seguida, essa aplicação é iniciada e suas *threads* são convertidas em processos. *Checkpoints* desses processos são gerados e suas execuções interrompidas. O `Scheduler` executa, então, o algoritmo `IBL` para estimar médias e desvios-padrão de consumo de `CPU`, memória, disco e rede. Com as informações do `IBL` e de capacidades e cargas de todo o ambiente, o algoritmo de balanceamento de carga é executado pelo

`Scheduler`. Esse algoritmo retorna uma distribuição de processos a ser adotada. Após migrados, os processos são restaurados e seus comportamentos, monitorados. As etapas que fazem parte da execução de uma aplicação estão demonstradas na figura 2.7.

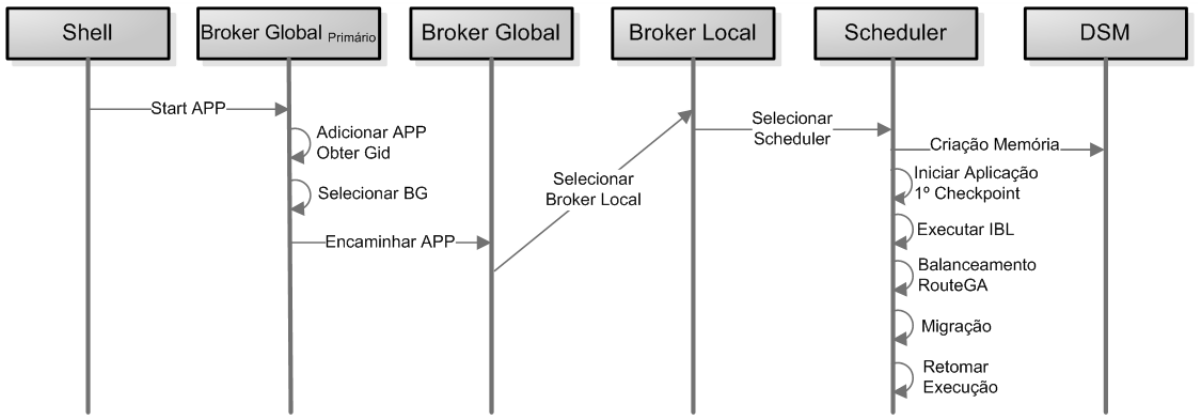


Figura 2.7: Diagrama de Seqüência: Iniciando uma aplicação

Atualmente, há dois algoritmos de escalonamento implementados: `Route` e `RouteGA`. Ambos consideram vizinhanças entre computadores para a distribuição de processos, contudo, somente o `RouteGA` analisa os resultados gerados pelo algoritmo `IBL`. As estimativas do `IBL` são utilizadas para parametrizar o algoritmo genético e obter uma boa distribuição de processos.

Para suportar a migração, utiliza-se uma ferramenta de *checkpoint* (atualmente o `CryoPID` (Blackham, 2008)), que gera o contexto de processos em sistema de arquivos. Esses contextos são utilizados para restaurar execuções em outros computadores.

2.6 Modelo de Programação

O `MidHPC` utiliza um suporte de memória compartilhada distribuída (*Distributed Shared Memory* – `DSM`) que permite a comunicação transparente entre processos. Para compreender seu funcionamento, considere uma aplicação α, composta por um grupo de processos α_{T_n}. Os processos α_{T_1} e α_{T_2} comunicam-se através da mudança do valor da variável x, que é compartilhada por ambos. Portanto, quando α_{T_1} modifica o valor de x, essa alteração também ocorre em α_{T_2}. Como α_{T_1} e α_{T_2} podem executar em computadores distintos, o `DSM` é utilizado para manter a consistência do valor em memória.

Quando a aplicação α inicia, o módulo `Scheduler` cria, no sistema de arquivos distribuídos, um arquivo[5] f que será utilizado como memória principal pelo `DSM`. A função `mmap` (GNU/Linux) é, então, executada para associar esse arquivo à aplicação α. Assim, quando a aplicação alocar espaço em memória, ela utilizará uma área no arquivo f, mape-

[5]Criado por meio do comando `dd if=/dev/zero of=MEMORY_FILE bs=1M count=32` – exemplo do comando utilizado para criar um arquivo com 32MB.

ada em sua memória principal. O tamanho do arquivo f é determinado pela informação contida na base de conhecimento IBL, sendo o padrão 32MB.

O módulo DSM intercepta as chamadas POSIX para controle de condições de corrida, tais como mutex e semáforos, visando criar e modificar variáveis compartilhadas armazenadas no arquivo do DSM. Esse módulo utiliza o modelo de consistência seqüencial (Adve & Gharachorloo, 1996), adotado pelo padrão POSIX (IEEE, 1993). Qualquer operação de leitura ou escrita é executada no arquivo do DSM e o desenvolvedor é responsável por utilizar primitivas lock e unlock para manter a memória consistente.

Para utilizar o DSM, faz-se necessária a adoção de um sistema de arquivos distribuídos (*Distributed File System* – DFS), o qual disponibiliza o arquivo f para computadores que executam tarefas da aplicação α. Qualquer DFS pode ser utilizado desde que permita leituras distribuídas, mesmo que dados sejam centralizados. Se os dados forem distribuídos, ou replicados, o desempenho é maior.

Atualmente, emprega-se, em experimentos, o sistema de arquivos distribuídos NFS (Sun Microsystems, Inc., 1989), devido, principalmente, à sua simplicidade de instalação e configuração. Sistemas de arquivos tais como o SSHFS (Hoskins, 2006) (que provê a segurança), o AFS (Foster *et al.*, 2004) e o CODA (Satyanarayanan *et al.*, 1990) (que tendem a oferecer maior desempenho) podem ser utilizados como DFS base. Nenhuma modificação na aplicação ou no MidHPC é necessária para adotar outro sistema de arquivos distribuídos.

2.7 Evolução das Pesquisas

O protótipo do MidHPC permitiu comprovar, em ambientes reais, que conhecimentos sobre aplicações e recursos computacionais auxiliam na otimização de escalonamento de processos. Essa comprovação motivou novas frentes de pesquisa a fim de extrair e empregar esses conhecimentos em diferentes etapas envolvidas na autonomia de grades computacionais. Dodonov *et al.* (2006) propuseram uma técnica adaptativa para a descoberta de padrões de comunicação, visando assistir a pré-busca de dados. Comportamentos de comunicação de processos foram classificados com o intuito de gerar modelos Markovianos para representar transições entre estados e instantes em que dados seriam utilizados. Resultados experimentais comprovam aumento no desempenho dessa abordagem para a busca antecipada de dados. Essa pesquisa motivou um trabalho de doutorado voltado para a classificação e predição de comportamentos de processos, a fim de auxiliar na otimização de operações de escalonamento, distribuição de dados, pré-busca, etc.

Os comportamentos de processos e os modelos estocásticos de redes de computadores (Ferreira & Mello, 2004; Dodonov *et al.*, 2005; Mello & Senger, 2006), ainda foram empregados no estudo de atrasos envolvidos no acesso paralelo a dados distribuídos em grades (Mello *et al.*, 2007a). Essa frente motivou outro trabalho de doutorado que visa o estudo e avaliação de técnicas de otimização para distribuição, migração, replicação e

acesso paralelo a dados distribuídos em grades computacionais.

Além dessas frentes, o estudo de comportamentos de processos e usuários motivou a proposta de uma rede neural artificial auto-organizável e não supervisionada, denominada *Self-Organizing Novelty Detection* (`SONDE`). Essa rede é tema de um terceiro doutorado que, atualmente, investiga sistemas dinâmicos como forma de auxiliar na parametrização e controle do aprendizado (Albertini & Mello, 2007). A primeira aplicação de maior porte dessa rede neural foi voltada para a caracterização de comportamentos de usuários. Estudos utilizando assinaturas digitais permitiram comprovar bons resultados na identificação de usuários por meio de seus comportamentos de escrita (pontos traçados) (Santos *et al.*, 2007). Esse trabalho é conduzido em uma dissertação de mestrado. Espera-se empregar os resultados obtidos na detecção de interações intrusivas em grades, visando melhorar suportes de auto-proteção existentes.

Resultados dessas pesquisas devem ser integrados ao protótipo do `MidHPC`. Além disso, outros aspectos também devem ser reavaliados, tais como a arquitetura atual do protótipo, a qual deve ser adaptada para um modelo *peer-to-peer*, permitindo maior nível de descentralização, e o algoritmo de aprendizado baseado em instâncias que deve ser adaptado para utilizar bases de dados descentralizadas, a fim de obter alta disponibilidade e maior desempenho de acesso.

2.8 Considerações finais

Este capítulo apresentou as principais pesquisas relacionadas ao projeto `MidHPC`. Resultados obtidos nesses trabalhos motivaram esta tese de livre docência, que emprega conceitos sobre sistemas dinâmicos e técnicas inteligentes para modelar e predizer comportamentos de processos, visando otimizar operações de escalonamento em grades computacionais.

Capítulo

3

Sistemas Dinâmicos

3.1 Considerações iniciais

O protótipo do `MidHPC` permitiu comprovar as contribuições da utilização de conhecimentos sobre aplicações e recursos computacionais na otimização de escalonamento de processos. Esses conhecimentos são atualmente obtidos por meio de ferramentas de *benchmark*, monitores de sistema e de processos (Mello & Senger, 2006; Senger *et al.*, 2007).

Os *benchmarks* extraem capacidades máximas de recursos tais como processador, leitura e escrita em memória primária, meio de armazenamento secundário e redes. O monitor de sistemas avalia a carga instantânea de recursos. O monitor de processos intercepta eventos de utilização de recursos (tais como volume de processamento, acesso à memória principal, leitura e escrita em rede e disco rígido) e os armazena em uma base de dados. Sobre essa base é aplicado um algoritmo de aprendizado baseado em instâncias, a fim de estimar comportamentos de novas aplicações submetidas ao sistema. Esses comportamentos são representados por médias; conseqüentemente, não se observa, avalia ou emprega suas variações no tempo.

A compreensão dessas variações motivou o estudo e aplicação de conceitos sobre sistemas dinâmicos com o objetivo de melhorar os atuais suportes de otimização e demais aspectos de autonomia em grades computacionais. Conceitos sobre sistemas dinâmicos são apresentados neste capítulo.

3.2 Conceitos

Um sistema dinâmico é composto por um conjunto de estados possíveis e uma regra que determina seu estado atual em função do passado. Para exemplificar, considere a

equação $x_{n+1} = 2x_n$ como representação da taxa de crescimento de uma população de indivíduos no tempo, onde n indica o instante de tempo e x_n define, precisamente, o tamanho da população em n. Nesse caso a regra, ou equação, define o próximo estado, ou tamanho da população no próximo instante, em função do passado. Sistemas com tais tipos de regras são denominados determinísticos. Há, contudo, outra classe de sistemas dinâmicos sujeita a outros efeitos além do passado. Considere um modelo de predição de preços de ações do mercado em função do tempo. O preço atual é composto por uma equação que considera preços anteriores. Contudo, essa equação conta, também, com uma variável aleatória, que modifica seus resultados. Esse tipo de sistema é denominado dinâmico estocástico ou aleatório (Alligood *et al.*, 1997).

Para melhor compreender um sistema dinâmico considere uma seqüência de observações de uma variável aleatória X no tempo, ou série temporal, $\{x_0, x_1, ..., x_{n-1}\}$. Esses eventos podem ser monitorados de forma discreta ou contínua, oriundos de regras determinísticas ou de processos estocásticos. A fim de compreender os próximos estados desse sistema, deve-se investigar modelos matemáticos capazes de representá-lo. Seus comportamentos são simples de serem estudados, caso se conheça sua regra (conjunto de equações que define o próximo estado com base no passado) e ele seja determinístico. Conhecendo a regra e sendo ele definido por processo estocástico, pode-se, também, compreender suas tendências comportamentais. Desconhecendo a regra, deve-se encontrar um conjunto de equações capaz de representá-lo, o que se torna mais complexo para sistemas estocásticos, pois há termos aleatórios.

Uma das técnicas mais antigas e estudadas para obter as regras que governam um sistema dinâmico é o modelo auto-regressivo (`AR`) (Penny & Harrison, 2006). Esse modelo visa encontrar uma equação na forma $x_k = c + \sum_{n=0}^{T-1} a_n \cdot x_{k-n} + \epsilon_k$ que permite obter o próximo estado x_k em função de uma soma de termos passados (onde c é uma constante, $a_0, a_1, \cdots, a_{T-1}$ são parâmetros do modelo e ϵ_k é um ruído branco, sinal ou processo aleatório). Esse modelo apresenta erros médios de predição ótimos para séries lineares (Box *et al.*, 1994), contudo gera resultados insatisfatórios para séries mais complexas, as quais motivaram aproximações locais (Casdagli, 1989; Zeevi *et al.*, 1998) (que também apresentam limitações para séries caóticas), e o estudo de regularidades internas de sistemas dinâmicos (Alligood *et al.*, 1997) (pontos fixos, órbitas, etc.).

3.3 Estudo de Órbitas

Ao modelar o conhecimento embutido em um sistema dinâmico compreende-se a repetição de seus padrões a qual permite, por exemplo, conhecer suas tendências, realizar predições e classificar suas operações. Essas possibilidades motivam diferentes áreas da ciência, dentre elas a voltada para o estudo de populações (Alligood *et al.*, 1997). Considere, por exemplo, que o tamanho de uma população dobra a cada hora de observação,

segundo a equação $f(x) = 2x$. A evolução temporal da população é definida por $f^k(x)$, onde k representa o número de instantes de tempo futuros que será feita a próxima observação. Dessa forma, caso a população inicial seja igual a 10, após uma hora ter-se-á 20 e após $k = 3$, onde $f^3(x) = f(f(f(x)))$, 80. De acordo com a equação dada, a população terá fator de crescimento exponencial igual a 2.

Tabela 3.1: Tamanho da população aplicando a função $f^k(x)$

k	$f^k(x)$
0	$0,01$
1	$0,02$
2	$0,04$
3	$0,08$
4	$0,16$
5	$0,32$
6	$0,64$
7	$1,28$
8	$2,56$
9	$5,12$
10	$10,24$

A tabela 3.1 apresenta os resultados para uma população inicial igual a $0,01$ milhão. Um modelo denominado Cobweb plot (Alligood *et al.*, 1997) foi adotado a fim de estudar a variação dos valores produzidos, ou órbita, por esse sistema (considerando que uma saída no instante t torna-se entrada em $t + 1$). Segundo esse modelo, traça-se uma curva com os valores das saídas produzidas em relação às entradas e uma reta diagonal $g(x) = x$. A órbita do sistema em relação a uma condição inicial é dada como segue.

Considere uma entrada inicial, seja $x = 0,01$, calcule a saída $f(0,01) = 0,02$. Em seguida, considere $0,02$ como entrada e calcule $f(0,02)$. O Cobweb plot auxilia na representação desses cálculos por meio de segmentos de reta que tocam $f(x)$ e $g(x)$. Por exemplo, inicialmente traça-se um segmento de reta vertical a partir da entrada $0,01$ até tocar a função $f(x)$. Esse segmento indica que a saída para $0,01$ é igual a $f(0,01) = 0,02$. Em seguida, traça-se um segmento horizontal do padrão entrada-saída $(0,01; 0,02)$ até tocar na reta diagonal $g(x)$. Em seguida, traça-se outro segmento de reta vertical com origem no ponto que toca em $g(x)$ até chegar em $f(x)$, tal como apresentado na figura 3.1. Os pares de valores entrada-saída obtidos, ou seja, os pares de pontos da curva $f(x)$, representam as saídas produzidas pelo sistema, sua órbita ou, também, variação no tempo.

No exemplo anterior, $f(x)$ apresenta a órbita de um sistema com características lineares e que tende a gerar saídas infinitas. Contudo, a fim de melhor caracterizar o tamanho de populações, deve-se considerar funções que impõem limites máximos de crescimento, respeitando os recursos escassos do ambiente. Para isso, seja, por exemplo, a função $h(x) = 2x(1 - x)$ onde x representa a população de entrada em milhões. Nesse caso,

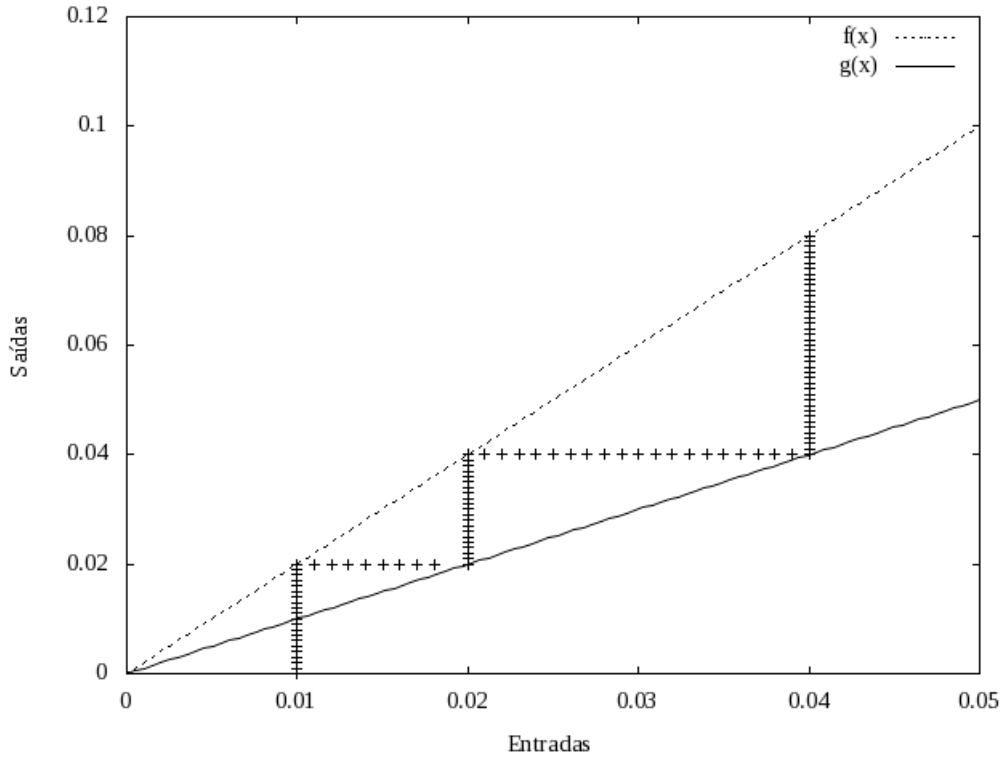


Figura 3.1: Órbita associada à função $f(x)$

a população não é simplesmente resultante de x, mas sim do produto de x pelo termo $(1-x)$, o que gera um efeito não linear no crescimento. Nessa circunstância, $h(x)$ define um crescimento logístico para a população, o que é mais próximo de situações reais (Alligood *et al.*, 1997).

Tabela 3.2: Tamanho da população aplicando a função $h^k(x)$

k	$f^k(x)$
0	$0,010$
1	$0,019$
2	$0,038$
3	$0,074$
4	$0,138$
5	$0,238$
6	$0,362$
7	$0,462$
8	$0,497$
9	$0,499$
10	$0,499$
11	$0,500$
12	$0,500$

A tabela 3.2 apresenta resultados dessa função para uma população inicial de $0,01$ milhão. Para estudar a órbita desse sistema aplica-se o mesmo método, Cobweb plot,

anteriormente abordado. A figura 3.2 permite observar que, em contraste com o mapeamento linear anteriormente considerado, para quaisquer valores entre $[0,0;0,5]$, a função $h(x)$ resulta em valores próximos de $0,5$.

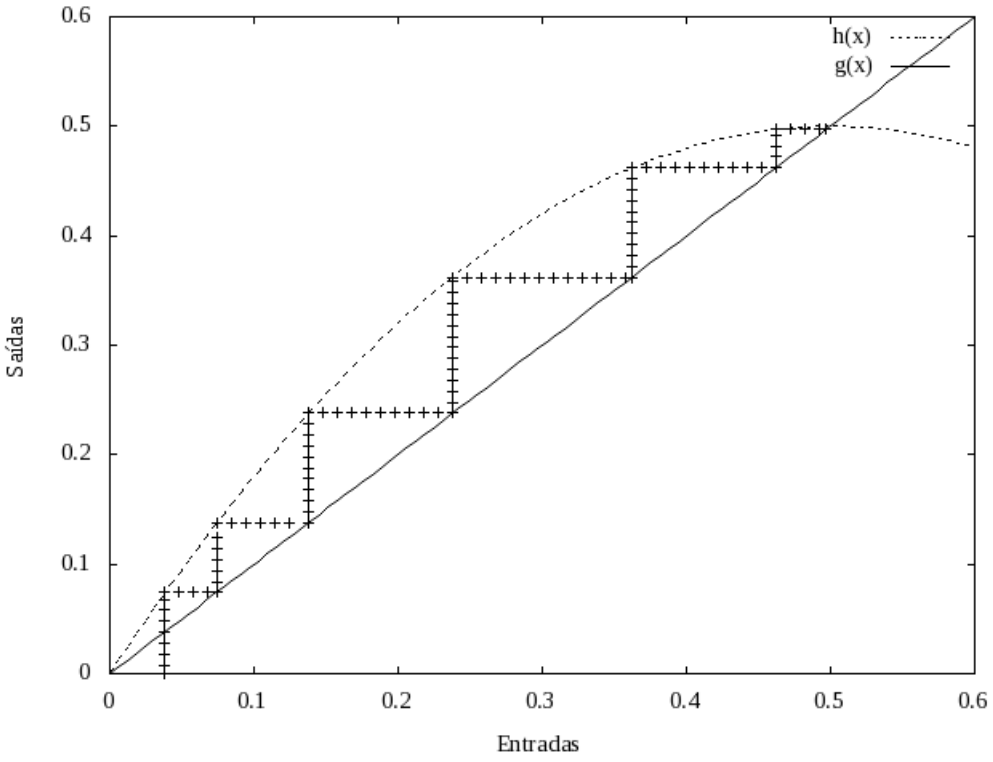


Figura 3.2: Órbita associada à função $h(x)$

Considere agora outro exemplo que permite melhor exemplificar a órbita de uma função em relação a condições iniciais. Seja um sistema definido pela função $q(x) = \frac{3x-x^3}{2}$ com valores iniciais $x = 1,6$ e $x = 1,8$. Traça-se o Cobweb plot para as duas condições iniciais, conforme a figura 3.3. Cabe ressaltar que, quando a curva estiver abaixo de $g(x)$, deve-se traçar linhas horizontais da órbita à esquerda e quando estiver acima, à direita, conforme realizado nos exemplos anteriores. Observa-se, nesse caso, que para a primeira condição inicial, tende-se ao ponto $x = 1$, enquanto para a segunda, a $x = -1$. Valores próximos ao ponto $x = 0$, contudo diferentes, fazem a órbita mover-se em direção à -1 ou 1. Esses pontos de atração, tais como $x = 1$ e $x = -1$, ou de repulsão, tais como regiões vizinhas de $x = 0$, são tratados por pesquisas adicionais em estabilidade de sistemas dinâmicos.

3.4 Estabilidade de Pontos Fixos

A seção anterior considerou três exemplos para estudar órbitas de sistemas dinâmicos. O primeiro, $f(x) = 2x$, apresenta comportamento linear. O segundo, $h(x) = 2x(1-x)$, e terceiro, $q(x) = \frac{3x-x^3}{2}$, contêm componentes não lineares. Esses exemplos utilizam uma

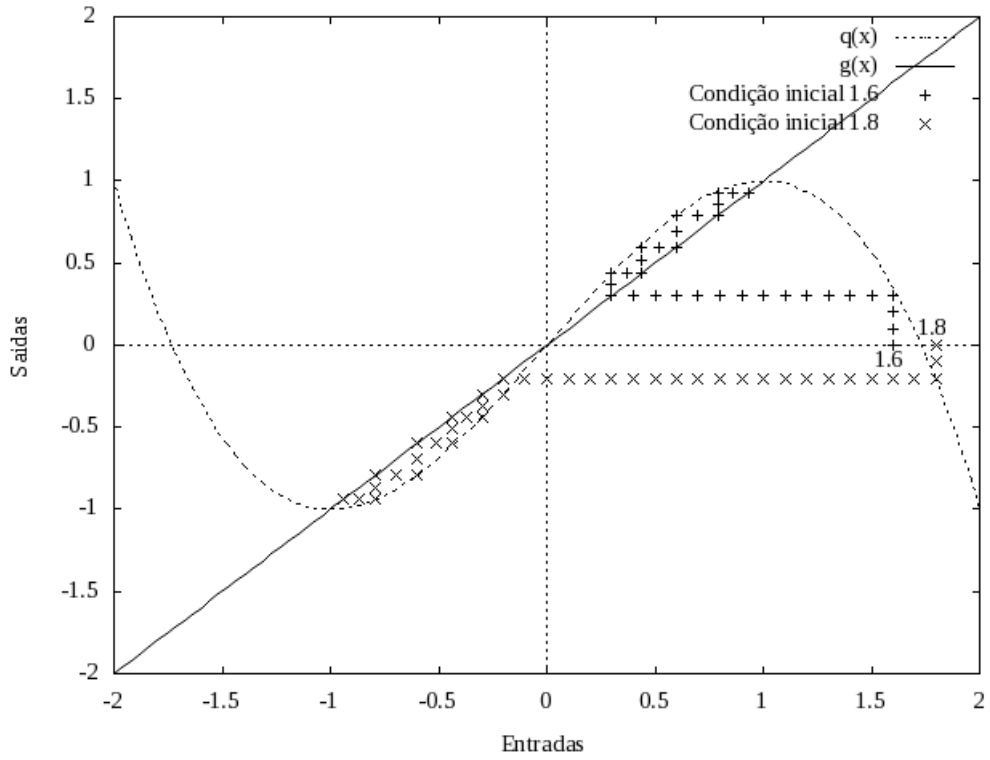


Figura 3.3: Órbita associada à função $q(x)$

reta $g(x)$ para auxiliar na observação gráfica das órbitas (simplifica a geração de próximas entradas e cálculo de saídas). Além de auxiliar na visualização de órbitas, $g(x)$ permite definir alguns pontos relevantes para os sistemas em estudo.

Percebe-se que todos os sistemas exemplificados, $f(x)$, $h(x)$ e $q(x)$ cruzam a função diagonal $g(x)$, ou seja, há pares de pontos, denominados pontos fixos, em que a abscissa e a ordenada são iguais. No caso de $f(x)$ o único ponto fixo é $x = 0$; para $h(x)$ há dois pontos fixos $x = 0$ e $x = 0,5$; finalmente, em $q(x)$ observa-se três pontos fixos $x = -1$, $x = 0$ e $x = 1$. Em $f(x)$, para qualquer população de entrada positiva diferente de zero, diverge-se do ponto fixo $x = 0$ e tende-se ao infinito. Para $h(x)$, a tendência é divergir de 0 e convergir para uma população máxima de $0,5$ milhão. Para $q(x)$, o sistema diverge de 0 e tende a -1 ou 1, dependendo das condições iniciais (tal como observado na seção anterior, onde duas condições são apresentadas: $x = 1,6$ e $x = 1,8$).

Os pontos fixos de convergência são denominados estáveis, enquanto os de divergência, instáveis. Por exemplo, uma esfera no topo de uma montanha está sobre um ponto instável, qualquer pequena perturbação faz com que ela se mova até encontrar um vale, ou ponto estável, onde o movimento cessa. Assim, pontos estáveis atraem a órbita de um sistema, enquanto os instáveis a repelem.

Para melhor definir estabilidade e instabilidade de pontos fixos, considere que o comprimento Euclidiano de um vetor $v = \{x_1, \ldots, x_m\} \in \mathbb{R}^m$ é dado por $|v| = \sqrt{{x_1}^2 + \ldots + {x_m}^2}$. Seja $p = \{p_1, \ldots, p_m\} \in \mathbb{R}^m$ um ponto em um plano e ϵ um número positivo. A vizinhança

ϵ, $N_\epsilon(p)$, é dada pelo conjunto $\{v \in \mathbb{R}^m : |v - p| < \epsilon\}$ de pontos dentro da distância Euclidiana ϵ de p. Seja f uma função em $\mathbb{R}^m$ e p, também em $\mathbb{R}^m$, um ponto fixo, onde $f(p) = p$. Se existe um $\epsilon > 0$ tal que, para todo v em $N_\epsilon(p)$, $\lim_{k\to\infty} f^k(v) = p$, então p é um ponto fixo estável do tipo sorvedouro ou atrator. Se existe uma vizinhança ϵ, $N_\epsilon(p)$, tal que para cada v em $N_\epsilon(p)$, exceto o próprio p, mapeiam-se pontos mais distantes e, conseqüentemente, fora de $N_\epsilon(p)$, então p é um ponto fixo instável do tipo fonte ou repulsor. Além de pontos fixos estáveis do tipo sorvedouro e fixos instáveis do tipo fonte, há um terceiro tipo, que ocorre somente em espaços multidimensionais, denominado sela. Esses pontos têm ao menos uma direção de atração e uma de repulsão (Alligood *et al.*, 1997).

Conhecer os pontos fixos de um sistema permite inferir suas tendências. Por exemplo, sejam as temperaturas globais da terra definidas como uma série temporal. Esses valores definem a órbita do sistema dinâmico em questão. Essa órbita pode ser utilizada para encontrar sua regra de origem. Ao conhecer essa regra pode-se definir seus pontos fixos e estudar, por exemplo, regiões estáveis de temperatura. Essas regiões definiriam máximos ou mínimos de temperatura para o planeta. Além disso, pode-se encontrar regiões instáveis, onde temperaturas tendem, sempre, a divergir. Ao compreender a função geradora dessa órbita e seus pontos fixos, pode-se, por exemplo, avaliar a influência do aquecimento global em outros sistemas, tais como fauna e flora (Clark, 1990; Perry *et al.*, 2001; King & Birk, 2004).

Os pontos fixos estáveis e instáveis são úteis, também, para estudar o comportamento de preços de ações em bolsa de valores. Considere os preços de uma ação, os quais definem a órbita do sistema. Deve-se obter a regra origem dessa órbita e encontrar seus pontos fixos. Esses pontos permitem avaliar, por exemplo, quando ações mudam suas tendências de preços, as quais permitem estimar os melhores momentos de compra e venda. Percebe-se, portanto, que, ao compreender a órbita e os pontos fixos de um sistema, pode-se predizer seu comportamento.

3.5 Expoente de Lyapunov

Formas de compreender o comportamento de sistemas dinâmicos motivaram diversos trabalhos (Casdagli, 1989; Shenshi *et al.*, 1999), dentre eles, estudos sobre variações em relação a condições iniciais, utilizando o expoente de Lyapunov (Edmonds, 1996). Esse expoente mede a taxa de variação de trajetórias vizinhas considerando uma separação inicial $\delta\mathbf{Z}_0$. Essa divergência dá-se por $|\delta\mathbf{Z}(t)| \approx e^{\lambda t}|\delta\mathbf{Z}_0|$, onde t é o instante de tempo e λ é o expoente de Lyapunov.

$$\lambda = \lim_{t\to\infty} \frac{1}{t} \ln \frac{|\delta\mathbf{Z}(t)|}{|\delta\mathbf{Z}_0|} \tag{3.1}$$

Considera-se o expoente de Lyapunov como maior valor de divergência entre trajetórias definido na equação 3.1. O valor desse expoente permite caracterizar um sistema dinâmico em (Elert, 2005; Rosenstein *et al.*, 1993):

1. $\lambda < 0$ – a série temporal é atraída para um ponto fixo estável ou para uma órbita periódica (onde pontos da órbita se repetem no tempo). Esses expoentes são característicos de sistemas dissipativos, ou não conservativos, tais como osciladores harmônicos. Quanto menor o valor de λ, mais rapidamente o sistema tende ao ponto de equilíbrio;

2. $\lambda = 0$ – a órbita do sistema tende a um ponto fixo neutro. Esses sistemas são conhecidos como conservativos. Por exemplo, considere dois osciladores harmônicos idênticos com diferentes amplitudes. Como a freqüência é independente da amplitude, a fase de ambos seria similar a dois círculos concêntricos, ou seja, as órbitas manteriam uma separação constante. Esse valor para o expoente indica que o sistema está no modo conhecido como estado fixo;

3. $\lambda > 0$ – o sistema apresenta órbita caótica e instável. Isso significa que a distância entre pontos da trajetória irá sempre divergir, em média, à uma taxa exponencial definida pelo expoente de Lyapunov (Rosenstein *et al.*, 1993; Eckmann & Ruelle, 1985).

Os valores do expoente máximo de Lyapunov permitem compreender tendências de sistemas dinâmicos. Eles podem convergir para pontos fixos ($\lambda < 0$), apresentar comportamento conservativo ($\lambda = 0$) ou divergir ($\lambda > 0$). Dado que esse expoente expressa a tendência de um sistema, pode-se utilizá-lo para compreender o horizonte máximo (número máximo de pontos) de predição de sistemas dinâmicos. Por exemplo, considere um modelo qualquer (e.g. equações que representam o comportamento do sistema) para um sistema com divergência dada pelo expoente de Lyapunov. Quanto maior a divergência, menos eventos podem ser previstos sem alterar ou incrementar o modelo inicial. Caso haja nenhuma divergência, pode-se predizer inúmeros eventos futuros com base no modelo inicial.

Até mesmo estimativas qualitativas são impossíveis para intervalos além desse horizonte (Elmer, 1998). O horizonte é definido por $-\frac{\ln \epsilon}{\lambda}$ onde ϵ é o erro medido no estado inicial, isto é, o ponto inicial de predições. Por exemplo, considere que qualquer técnica foi utilizada para modelar uma série temporal. Tal técnica foi treinada com um conjunto de dados e o erro de predição de um próximo valor é $\epsilon = 0,001$, o Lyapunov da mesma série é $0,692$, conseqüentemente seu horizonte de predição é igual a $-\frac{\ln\ 0,001}{0,692} = 9,98$. Isso significa que é possível predizer, no máximo, o comportamento dos próximos $9,98$ valores futuros.

A figura 3.4 apresenta o comportamento do horizonte de predição de acordo com o erro ϵ. Erros abaixo de $0,001$ foram considerados nessa circunstância. Observa-se que o horizonte é maior para pequenos valores de erro inicial.

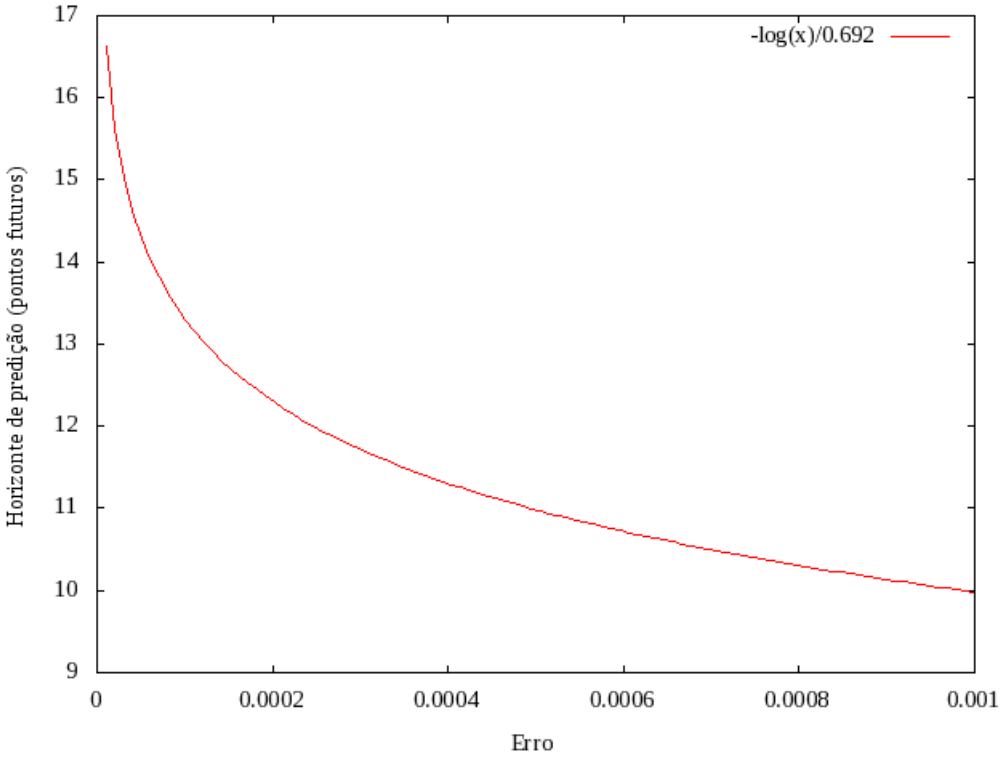


Figura 3.4: Horizonte de predição utilizando um expoente de Lyapunov igual a $0,692$

Esta tese considera o método de Kantz (1994) para calcular o expoente de Lyapunov de séries. Esse é um dos métodos implementados pelos autores do pacote *Nonlinear Time Series Analysis* (`TISEAN`) (Hegger *et al.*, 2009), que o recomendam em relação aos demais. O comando do pacote `TISEAN` que executa esse método é o `lyap_k`, o qual gera como saída logaritmos do fator de extensão (do inglês *stretching factor*), conforme proposto em (Kantz, 1994). Para computar o expoente de Lyapunov, deve-se traçar a regressão linear desses logaritmos. O ângulo da curva de regressão resultante define o valor do expoente.

3.6 Expoente de Hurst

Além do expoente máximo de Lyapunov, pode-se estimar o expoente de Hurst de um sistema dinâmico, o qual mede a aleatoriedade de um conjunto de dados. Esse expoente pode ser estimado por diferentes técnicas, a mais adotada é a *Rescaled Range* (`R/S`) (Kaplan, 2003). Para exemplificá-la, considere a série temporal $X(n) = \{x_0, x_1, ..., x_{n-1}\}$. O primeiro passo consiste em computar a média $AVG_{X(k)}$ para todos os valores de $X(k)$, onde $k = n$.

Em seguida, encontra-se o valor mínimo (min) e o máximo (max), em relação à média,

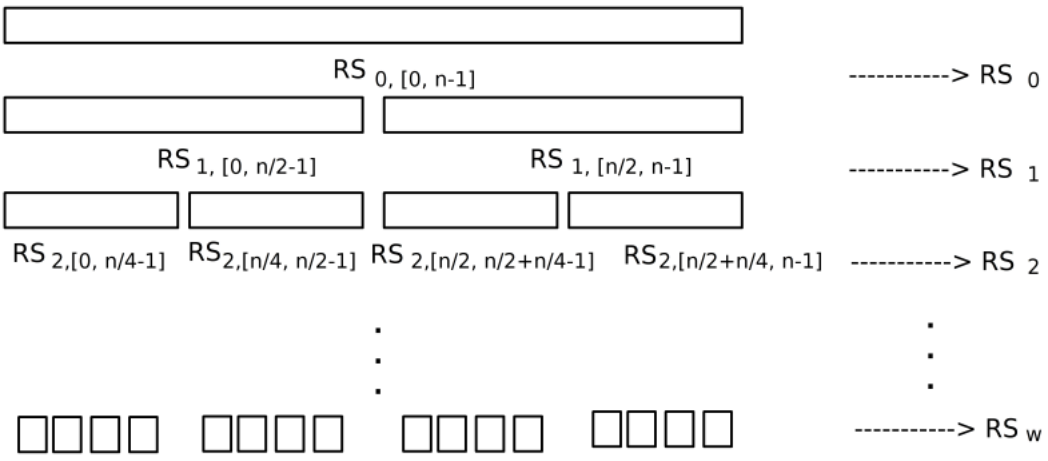


Figura 3.5: Cálculo de *Rescaled Range*

Algoritmo 1 *Rescaled Range*: Localizando os valores mínimo e máximo

```
float min = MAX_FLOAT;
float max = MIN_FLOAT;
float v = 0;
for i = first; i < k; i++ do
  v = v + (x_i - Avg_X(k));
  if v < min then
    min = v;
  end if
  if v > max then
    max = v;
  end if
end for
```

do conjunto de dados, conforme o algoritmo 1. Então, o desvio padrão $STDEV_{X(k)}$ é calculado para os termos $X(k)$ e o primeiro valor de `R/S` (denominado RS_0) para o subconjunto $k \in [0; n-1]$ é encontrado, definido por $RS_0 = \frac{|\max - \min|}{stdev_{X(k)}}$. O índice 0 em RS_0 faz referência à primeira iteração do cálculo.

A próxima iteração é iniciada pela divisão da série temporal original em dois subconjuntos. O primeiro com $k \in [0; \frac{n}{2} - 1]$ e o segundo com $k \in [\frac{n}{2}; n-1]$. Os mesmos passos são conduzidos para calcular `R/S` em cada subconjunto da série temporal e, conseqüentemente, obter-se-á dois valores (um para o subconjunto $[0, \frac{n}{2} - 1]$, o qual é definido por $RS_{1,[0;\frac{n}{2}-1]}$, e outro para o segundo subconjunto, $[\frac{n}{2}; n-1]$, representado por $RS_{1,[\frac{n}{2};n-1]}$). Calcula-se, então, a média dos dois valores e obtém-se RS_1, o qual representa o valor de `R/S` para a segunda iteração. A série temporal é recursivamente particionada a fim de calcular $RS_2, RS_3, ..., RS_w$, onde w é a última iteração. De acordo com Kaplan (2003), o particionamento pode ser feito, recursivamente, até que o menor subconjunto formado tenha um número mínimo de elementos no passo w (esse processo de particionamento é apresentado na figura 3.5). O mesmo autor conclui que há uma boa aproximação con-

siderando cerca de 8 elementos na menor partição (valor adotado no exemplo da tabela 3.3).

Depois de calcular todos os valores $RS = \{RS_0, RS_1, ..., RS_w\}$, obtém-se o expoente de Hurst por meio de uma regressão linear dos pontos $(\log_2(|X(\frac{n}{2^y})|),\ \log_2(RS_y))$, onde $y = 0, 1, 2, ..., w$ é a iteração e $|X(\frac{n}{2^y})|$ é o número de elementos por partição da série na iteração y. A tabela 3.3 apresenta um exemplo. Fazendo a regressão linear, onde a coluna 4 representa o eixo x e a coluna 5 o y, obtém-se a seguinte equação $y = 0,727024x - 0,745555$. O ângulo da curva, definido nesse caso por $H = 0,727024$, é o expoente de Hurst[1].

Tabela 3.3: *Rescaled Range*: estimativa do expoente de Hurst

iteração	**tamanho da partição**	RS	$\log_2$**(tamanho da partição)**	$\log_2(RS)$
0	1024	$RS_0 = 96,4451$	$10,0$	$6,5916$
1	512	$RS_1 = 55,7367$	$9,0$	$5,8006$
2	256	$RS_2 = 30,2581$	$8,0$	$4,9193$
3	128	$RS_3 = 20,9820$	$7,0$	$4,3911$
4	64	$RS_4 = 12,6513$	$6,0$	$3,6612$
5	32	$RS_5 = 7,2883$	$5,0$	$2,8656$
6	16	$RS_6 = 4,4608$	$4,0$	$2,1573$
7	8	$RS_7 = 2,7399$	$3,0$	$1,4541$

O intervalo de valores do expoente de Hurst é $H \in [0; 1]$. Um expoente próximo de 1 indica comportamento persistente, isso significa que há correlação entre um evento e a ocorrência de outro no futuro. Caso o valor seja próximo de zero, a série temporal apresenta comportamento anti-persistente, ou seja, há uma auto-correlação negativa entre pares de eventos. Nesse caso, um acréscimo em valor passado da série gera o decréscimo de outro elemento futuro e vice-versa. Valores próximos de $0,5$ indicam que a série temporal é uma caminhada aleatória (*random walk*), dessa forma, valores futuros não dependem do histórico. Observa-se, portanto, que o expoente de Hurst é uma ferramenta importante para avaliar dados históricos.

3.7 Dimensão embutida e de separação

Além do auxílio dos expoentes de Lyapunov e Hurst na interpretação de órbitas, é necessário encontrar formas de estimar as regras, ou funções, que definem os comportamentos de sistemas dinâmicos. Para exemplificar a obtenção dessas funções, considere, inicialmente, a equação logística 3.2 com condições iniciais $t \in [0; 4000]$, $b = 3,8$ e $x_0 = 0,5$. A figura 3.6 apresenta as saídas, ou órbita percorrida no tempo, para essa regra, a qual tem

[1] O *dataset* utilizado foi obtido do livro *Chaos and Order in the Capital Markets*, segunda edição, Edgar Peters.

expoente de Hurst $H = 0,40535$ e Lyapunov $\lambda = 0,447039$[2]. O expoente de Hurst evidencia pequeno grau de anti-correlação, contudo a série apresenta, em geral, comportamento aleatório. O expoente de Lyapunov aponta seu comportamento caótico e instável (a distância entre pontos da trajetória tende sempre a divergir, o que dificulta a modelagem). Esses expoentes permitem concluir que ao aplicar, diretamente, uma técnica de predição sobre tal série, tende-se a obter resultados ruins. Pode-se, contudo, aplicar uma forma alternativa de reconstrução dessa órbita, a qual permite observar regularidades internas e simplificar a compreensão do sistema em estudo.

$$x_{t+1} = b \cdot x_t \cdot (1,0 - x_t) \tag{3.2}$$

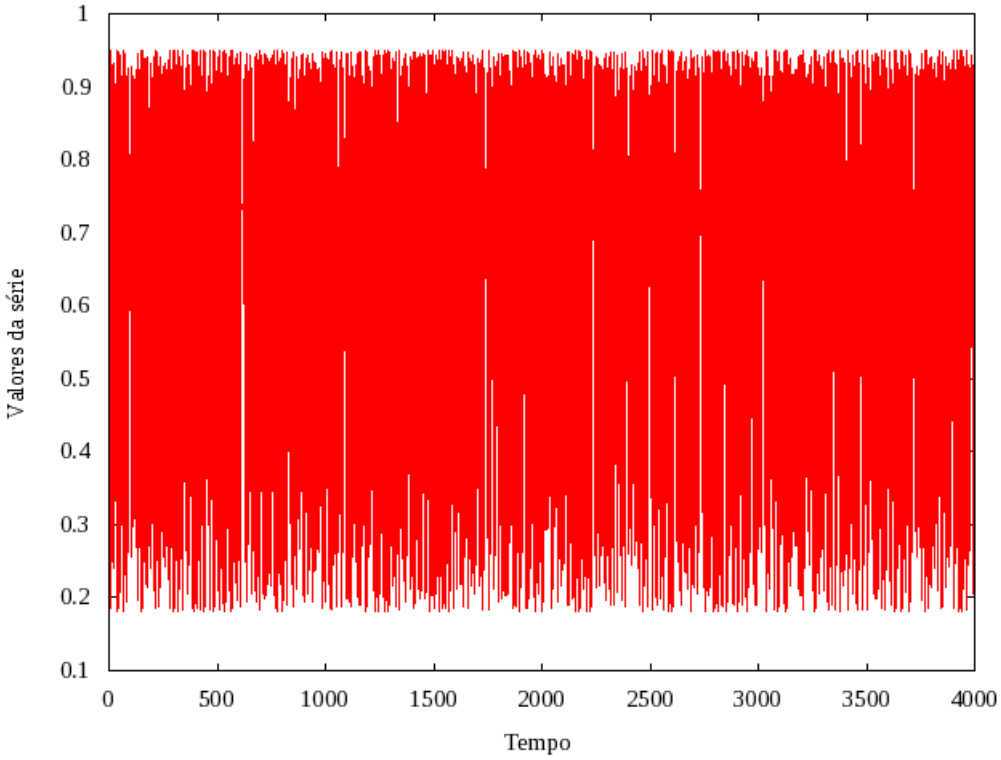


Figura 3.6: Saídas da função logística

Whitney (1936) aplicou variedades diferenciáveis como forma de reconstruir funções utilizando transformações para espaços Euclidianos multidimensionais. Matematicamente, diz-se que $M \subset \mathbb{R}^k$ é uma variedade diferenciável de dimensão m se, para cada ponto $p \in M$, existem uma vizinhança $U \subset M$ de p e um homeomorfismo $x : U \to U_0$, U_0 um aberto de $\mathbb{R}^m$, tais que o homeomorfismo inverso $x^{-1} : U_0 \to U \subset \mathbb{R}^k$ é uma imersão de classe $C^{\inf}$. Isto é, para cada $u \in U_0$, a derivada $dx^{-1}(u) : \mathbb{R}^m \to \mathbb{R}^k$ é biunívoca. Diz-se, nesse caso, que (x, U) é uma carta local em torno de p e U é uma vizinhança coordenada de p (Palis Jr. & Melo, 1978).

[2]Esse expoente de Lyapunov foi calculado utilizando 10 iterações para o programa `lyap_k` do pacote TISEAN (Hegger *et al.*, 2009). O número de iterações é informado por meio do argumento `-s`.

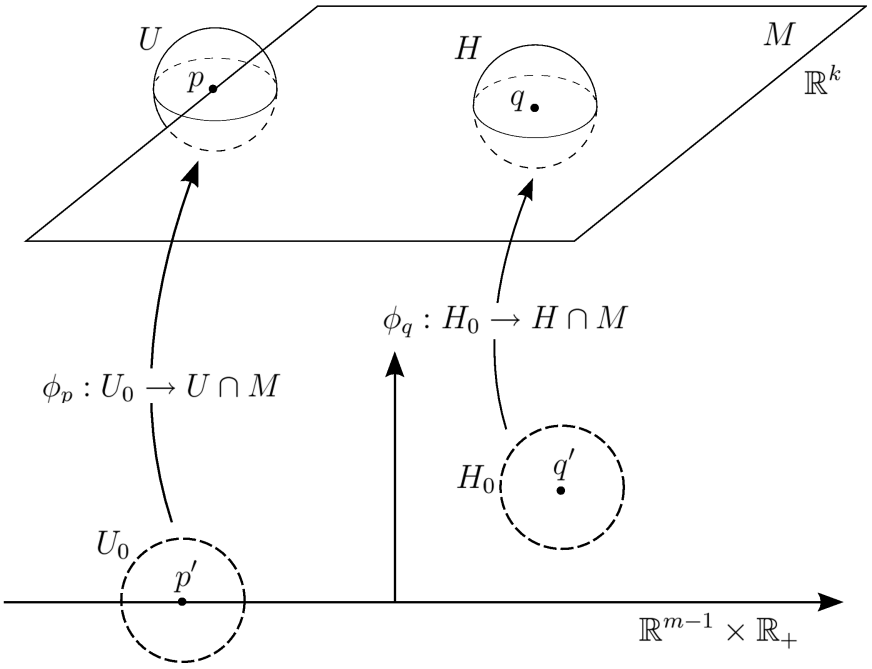


Figura 3.7: Exemplo de variedade

A figura 3.7 apresenta um exemplo de parametrização de um plano $\mathbb{R}^{m-1} \times \mathbb{R}_+$ para outro $\mathbb{R}^k$. Dado um ponto q' pode-se, por meio de $\phi_q : H_0 \to H \cap M$, encontrar um ponto q em M correspondente. O mesmo ocorre com p', contudo, nesse caso, obtém-se uma região na borda de M. Esse exemplo ilustra o mapeamento de um ponto e sua vizinhança em um plano com maior número de dimensões.

Whitney (1936) observou que esse mapeamento permitiria a compreensão de comportamentos não observáveis ou pouco representativos quando descritos sob número reduzido de dimensões. A partir disso, ele propôs seu teorema de imersão, segundo o qual qualquer variedade em n dimensões pode ser mapeada em espaço Euclidiano de $2n + 1$ dimensões.

Baseado nos estudos de Whitney (1936), Takens (1980) prova que, ao invés de mapear os estados de um sistema dinâmico em espaço Euclidiano de $2n + 1$ dimensões, pode-se reconstruí-lo considerando deslocamentos no tempo. Segundo o teorema de imersão de Takens (1980), uma série temporal $x_0, x_1, ..., x_{n-1}$ pode ser reconstruída em espaço multidimensional $x_n(m, \tau) = (x_n, x_{n+\tau}, ..., x_{n+(m-1)\tau})$, ou de coordenadas de atraso, onde m é a dimensão embutida e τ representa um *time delay* ou dimensão de separação. Essa técnica de mapeamento ou reconstrução permite transformar as saídas produzidas por sistemas dinâmicos, representadas por séries temporais unidimensionais, em um conjunto de pontos em espaço Euclidiano de m dimensões. Essa reconstrução foi, posteriormente, empregada para auxiliar na obtenção de regras de sistemas dinâmicos, simplificando, conseqüentemente, o estudo de comportamentos e sua aplicação para diferentes fins, tais como estudo de órbitas, tendências e predição (Alligood *et al.*, 1997).

Para melhor compreender as dimensões embutida e de separação, considere as saídas da função logística, anteriormente abordada, reconstruídas em um espaço multidimen-

sional onde $m = 2$ e $\tau = 1$, a qual resulta em pares de pontos (x_t, x_{t+1}) (figura 3.8). Após a reconstrução, o comportamento da função logística, que era aparentemente uma caminhada aleatória (figura 3.6), pode ser estudado, compreendido e modelado de forma mais simples. Ao realizar uma regressão dos pontos resultantes, pode-se obter a regra do sistema dinâmico e determinar seus comportamentos futuros. Tendo essa regra e um x_t, pode-se, por exemplo, definir o próximo valor da série, x_{t+1}, o qual pode ser retro-alimentado para gerar x_{t+2}, e assim sucessivamente.

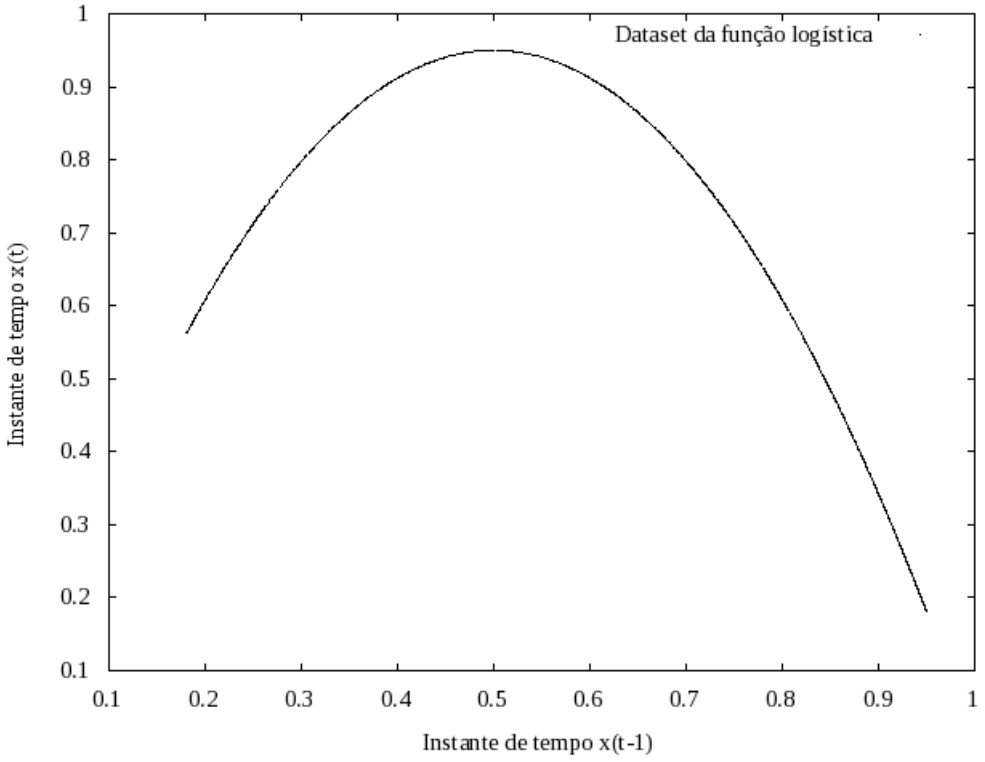


Figura 3.8: Função logística reconstruída em dimensão embutida 2 e de separação 1

A dimensão embutida define, basicamente, o número de eixos, do espaço de coordenadas de atraso, necessários para plotar o comportamento reconstruído da série. Nesse caso a série necessitou de duas dimensões, outras podem requerer espaços com mais eixos. Esse comportamento é, por exemplo, observado no atrator de Lorenz cujas saídas são apresentadas na figura 3.9.

Ao considerar a reconstrução da série utilizando dimensão embutida 2, obtém-se espaço de coordenadas de atraso similar ao da figura 3.10. Contudo, ao adicionar uma nova dimensão e reconstruí-la, portanto, com $m = 3$, desdobra-se todo o comportamento da série, simplificando sua compreensão (figura 3.11). Pára-se de adicionar dimensões caso não haja desdobramentos de novos comportamentos; nesse caso, cessa-se com $m = 3$, que representa a melhor dimensão para o atrator de Lorenz.

Além da dimensão embutida há ainda a de separação, que auxilia na extração de comportamentos periódicos de séries. A dimensão de separação informa o deslocamento

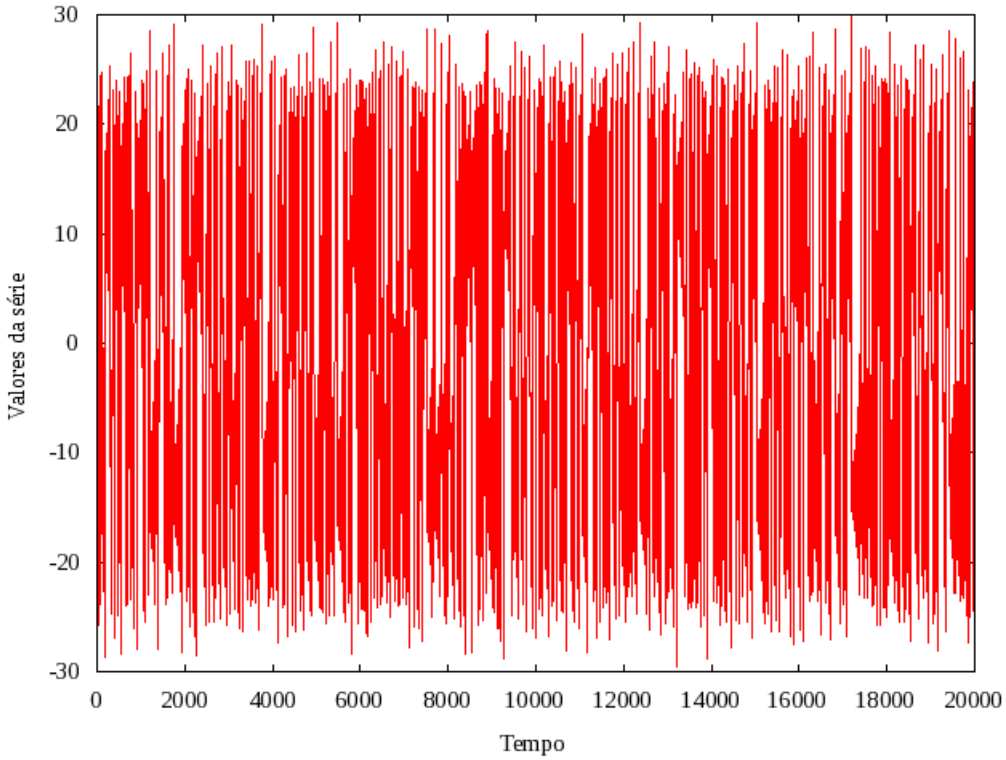


Figura 3.9: Saídas do atrator de Lorenz

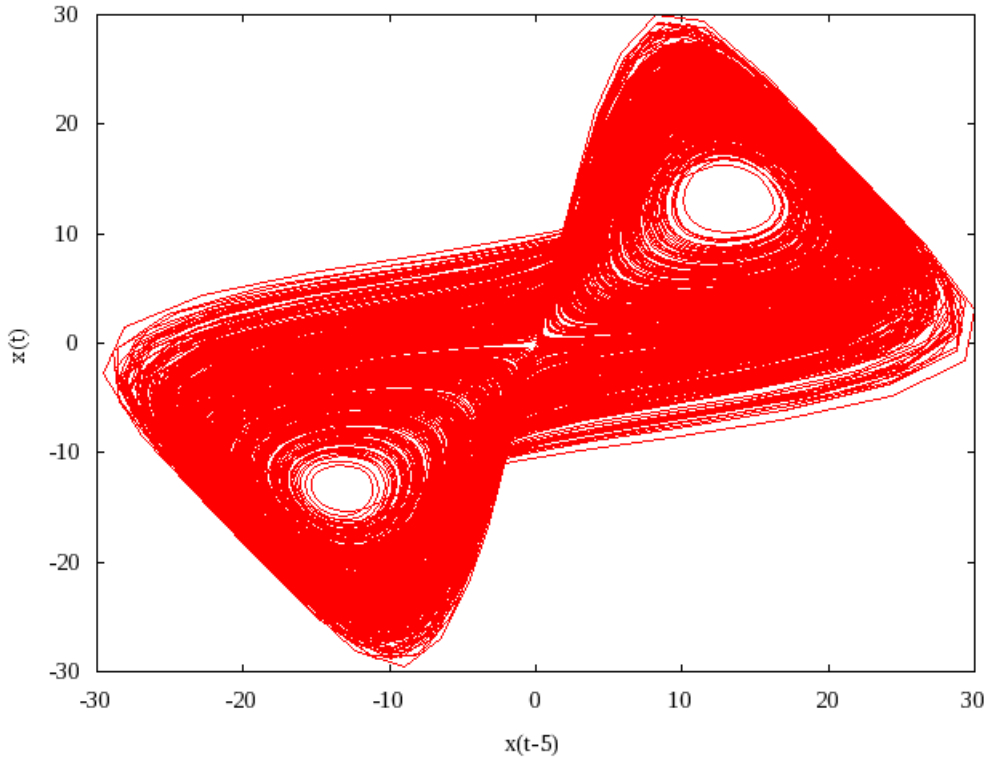


Figura 3.10: Atrator de Lorenz reconstruído em espaço de coordenadas de atraso com dimensão embutida 2 e de separação 5

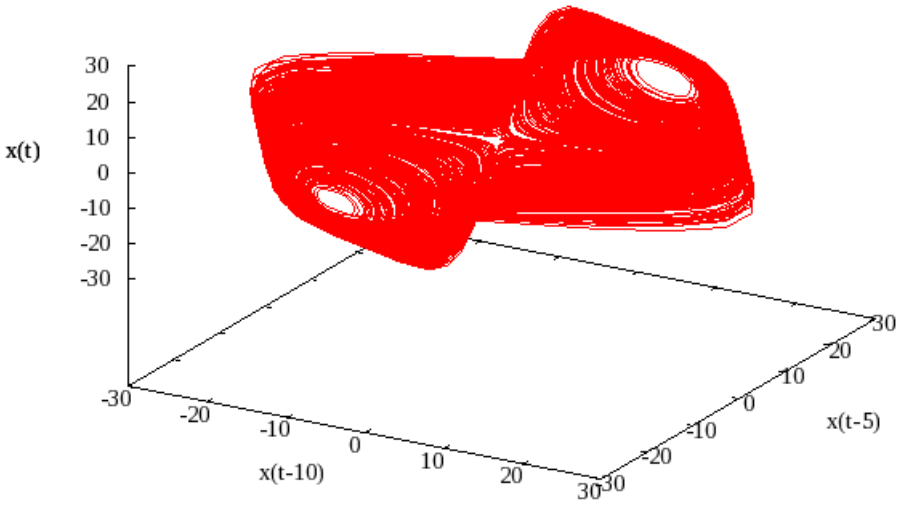


Figura 3.11: Atrator de Lorenz reconstruído em espaço de coordenadas de atraso com dimensão embutida 3 e de separação 5

de valores históricos que devem ser avaliados a fim de predizer comportamento futuro (ela permite encontrar a sazonabilidade da série). Por exemplo, a fim de predizer temperaturas de uma região do mundo na data de 12 de dezembro de 2007, pode-se observar medidas do ano anterior (em 12 de dezembro de 2006). Espera-se, portanto, que o próximo ano tenha temperaturas próximas aos dos anteriores. Nesse exemplo, a dimensão de separação seria igual a 365 dias.

Por outro lado, caso seja considerado o período de um mês para o problema abordado, observar-se-ia que o comportamento das temperaturas não se repetiria da maneira esperada[3]. Nesse caso seriam consideradas, por exemplo, as temperaturas de 12 de novembro de 2007, 12 de outubro de 2007, 12 de setembro de 2007 e 12 de agosto de 2007 que, muito provavelmente, não auxiliariam na predição das temperaturas de 12 de dezembro de 2007.

Exemplos anteriores evidenciam como as dimensões embutida e de separação auxiliam no estudo de algumas séries específicas. Contudo, é necessário determinar essas dimensões para quaisquer séries oriundas de dados experimentais. De acordo com Abarbanel *et al.* (1993), uma função de auto-correlação (equação 3.3, onde $E[.]$ é o valor esperado, μ é a média, k é o deslocamento no tempo e σ^2, a variância) auxilia na determinação da dimensão de separação de séries. A auto-correlação mensura a repetição de comportamento de um trecho da série em relação a seu histórico. Contudo, essa técnica é formulada para

[3]No último exemplo, envolvendo o atrator de Lorenz, considerou-se $t = 5$, o qual é indicado como dimensão de separação segundo Lorenz (1963); Kennel *et al.* (1992b).

séries lineares, e, conseqüentemente, não é adequada para outros tipos de séries.

$$ACF(k) = \frac{E[(X_i - \mu)(X_{i+k} - \mu)]}{\sigma^2} \tag{3.3}$$

Fraser & Swinney (1986) estudaram e confirmaram que a técnica de auto-informação mútua (*Auto Mutual Information* – `AMI`) apresenta melhores resultados na estimativa de dimensões de separação. Essa técnica não depende de séries lineares. Para obter a separação de uma série, aplica-se essa técnica considerando diferentes deslocamentos no tempo. Em seguida, traça-se uma curva em função dos deslocamentos (iniciando em 1 e incrementando) e adota-se seu primeiro mínimo como dimensão de separação.

A informação mútua média é definida pela equação 3.4, onde X e Y seguem, respectivamente, as funções de distribuição de probabilidades P_X e P_Y, e X e Y ocorrem em pares com distribuição conjunta P_{XY} (Kennel, 2002). Aplicando essa técnica sobre o conjunto de dados de Lorenz, previamente estudado, obtém-se a figura 3.12, a partir da qual se encontra o primeiro mínimo, igual a 5, confirmando os resultados apresentados por Lorenz (1963); Kennel *et al.* (1992b).

$$I(X;Y) = \int P_{XY}(x,y) \log_2 \frac{P_{XY}(x,y)}{P_X(x)P_Y(y)} dxdy \tag{3.4}$$

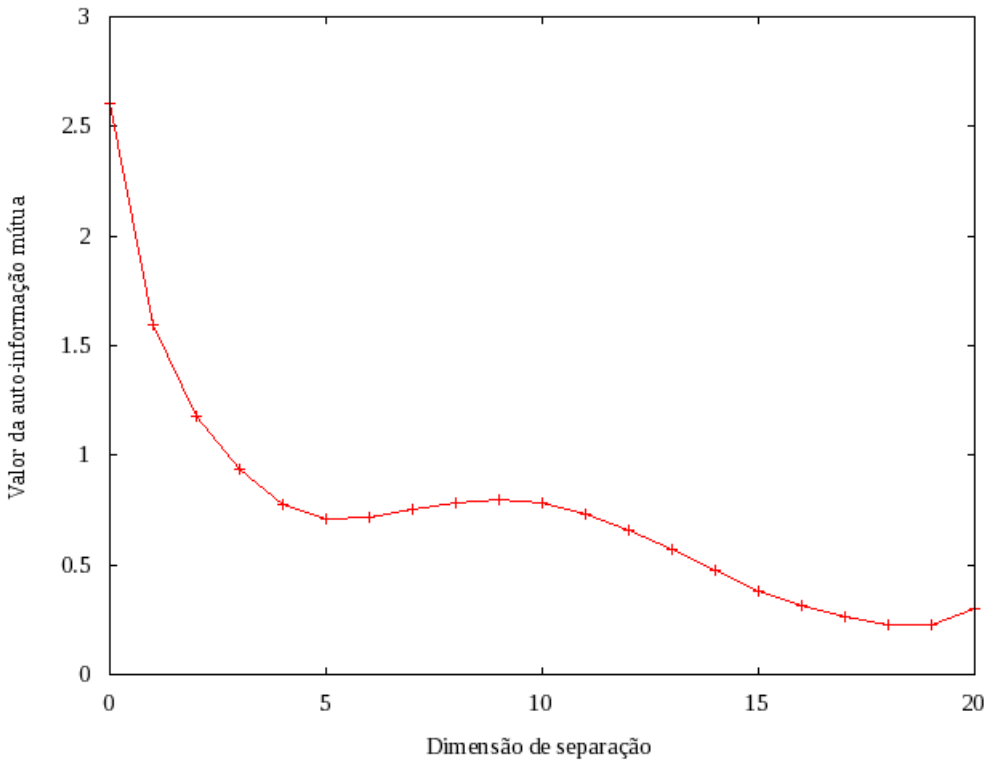


Figura 3.12: Lorenz – auto-informação mútua

Após definir a dimensão de separação de uma série, deve-se encontrar a dimensão embutida. Takens (1980) e Mañé (1980) estudaram e confirmaram que o limite superior da

dimensão embutida D_e (valor inteiro) pode ser estimado utilizando a dimensão fractal D_f, de acordo com a equação $D_e > 2,0 \cdot D_f$. Contudo, a dimensão resultante dessa equação é, em geral, maior que o necessário. Por exemplo, a dimensão fractal do atrator de Lorenz é 2,06 (Medio & Gallo, 1993), conseqüentemente, o limite superior da dimensão embutida seria $D_e > 2,0 \cdot 2,06$, que resulta em 5. Contudo, de acordo com (Kennel *et al.*, 1992b), o atrator pode ser representado em $D_e = 3$. Do ponto de vista matemático (Kennel *et al.*, 1992a,b), pode-se modelar esse sistema utilizando 3 ou 5 dimensões, pois, uma vez que todos os possíveis estados foram encontrados, pode-se conduzir a análise comportamental. Contudo, ao trabalhar, desnecessariamente, com mais dimensões, adiciona-se complexidade e tempo de processamento à modelagem e à análise de resultados (Kennel *et al.*, 1992b).

Uma forma alternativa para obter a dimensão embutida mínima é por meio do cálculo de invariantes do sistema (tais como o expoente de Lyapunov (Alligood *et al.*, 1997)) para diferentes valores de dimensão, observando a saturação dos resultados. A complexidade dessa abordagem motivou Kennel *et al.* (1992b) a propor o método *False Nearest Neighbors* (`FNN`), que calcula os vizinhos mais próximos de cada ponto, no espaço de coordenadas de atraso (iniciando com dimensão embutida igual a 1). Em seguida, uma nova dimensão é adicionada e a distância entre vizinhos mais próximos é novamente calculada. Caso haja acréscimo nessa distância, os pontos são considerados falsos vizinhos, o que evidencia a necessidade de mais dimensões para reconstruir o comportamento da série.

Kennel *et al.* (1992b) consideram uma dimensão embutida d onde o r-ésimo vizinho mais próximo de $y(n)$ é dado por $y^{(r)}(n)$. A distância Euclidiana entre o ponto $y(n)$ e seu r-ésimo vizinho mais próximo é dada pela equação 3.5. Ao adicionar uma nova dimensão, reconstrói-se a série em $d+1$ e adiciona-se a coordenada $(d+1)$ em cada vetor $y(n)$, a qual é incluída na equação de distância Euclidiana (termo $x(n + dT)$ da equação 3.6). Dessa forma, o critério mede a variação de distância ao adicionar uma nova dimensão, conforme descrito pela equação 3.7.

$$R_d^2(n,r) = \sum_{k=0}^{d-1} (x(n+kT) - x^{(r)}(n+kT))^2 \tag{3.5}$$

$$R_{d+1}^2(n,r) = R_d^2(n,r) + (x(n+dT) - x^{(r)}(n+dT))^2 \tag{3.6}$$

$$V_{n,r} = \sqrt{\frac{R_{d+1}^2(n,r) - R_d^2(n,r)}{R_d^2(n,r)}} = \frac{|x(n+dT) - x^{(n)}(n+dT)|}{R_d^2(n,r)} \tag{3.7}$$

Segundo os autores, se $V_{n,r} > R_{tol}$ então os pontos são considerados falsos vizinhos, onde R_{tol} é um limiar. Eles ainda concluem, empiricamente, que $R_{tol} \geq 10,0$ é um bom limite para a geração de resultados.

Aplicando o método `FNN` sobre o conjunto de dados do atrator de Lorenz (utilizando

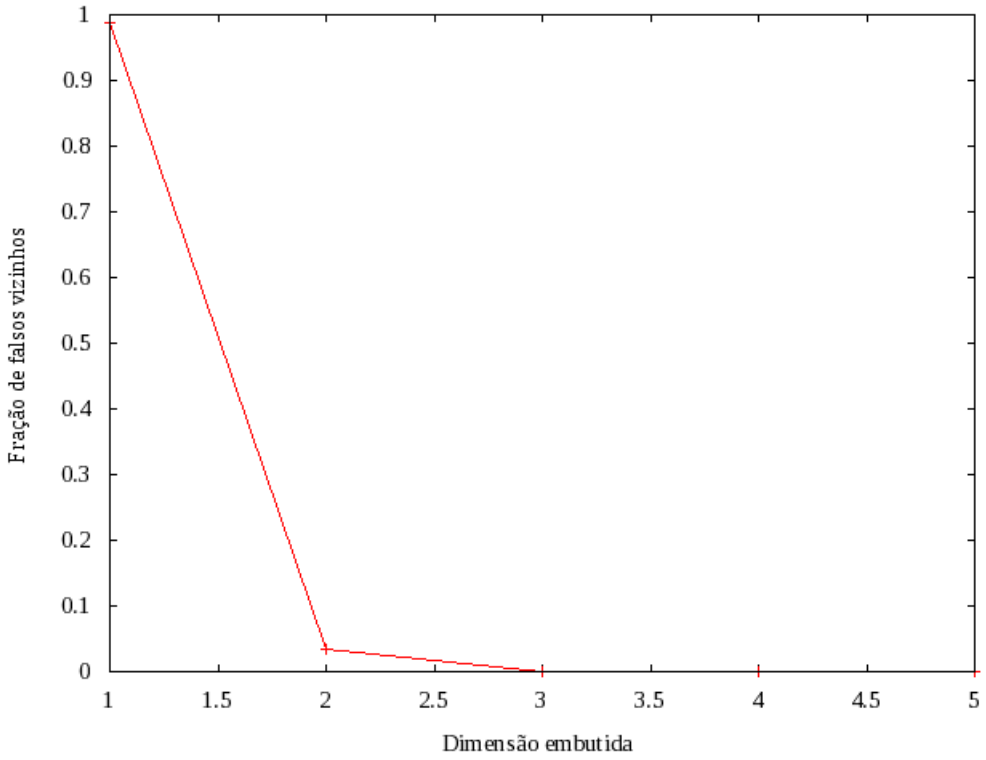


Figura 3.13: Atrator de Lorenz – estudos para encontrar a dimensão embutida

a dimensão de separação 5 previamente obtida), obtém-se os resultados apresentados na figura 3.13. Essa figura traça a fração dos falsos vizinhos versus a dimensão embutida considerada. Quando a fração é igual a zero, encontra-se a melhor dimensão embutida. Nesse caso, a dimensão embutida encontrada é 3, o que confirma os resultados obtidos por Kennel *et al.* (1992b).

Após definir as duas dimensões, aplica-se a teoria de imersão de Takens (1980), conforme apresentado anteriormente, onde a série temporal $x_0, x_1, ..., x_{n-1}$ é reconstruída no espaço multidimensional, ou de coordenadas de atraso, $x_n(m, \tau) = (x_n, x_{n+\tau}, ..., x_{n+(m-1)\tau})$ (O termo m representa a dimensão embutida e τ, a separação). A reconstrução desdobra, completamente, o comportamento da regra desse sistema dinâmico. Para exemplificar esse desdobramento, considere o conjunto de dados de Lorenz apresentado na tabela 3.4. Depois de reconstruir esse conjunto com dimensão embutida 3 e separação 5, obtém-se a tabela 3.5 (a curva resultante dessa reconstrução é apresentada na figura 3.11)[4].

Essa reconstrução permite desdobrar o comportamento da série e obter sua regra, ou seja, a função que define sua órbita no tempo. Obtendo tal função, pode-se estudar seus pontos fixos estáveis e instáveis, compreender suas tendências e, também, predizer seu comportamento futuro.

[4]Essa tabela apresenta mais valores que a tabela 3.4. Esse é apenas um exemplo de desdobramento dos dados originais a fim de obter a função geradora do sistema dinâmico em questão.

Tabela 3.4: Conjunto de dados original do atrator de Lorenz

Dimensão 1
$-9,6559617$
$-6,9902085$
$-4,9834927$
$-3,5773619$
$-2,6589215$
$-2,1120568$
$-1,8411753$
$-1,7784935$
$-1,8834828$
$-2,1397586$
$-2,5521791$
$-3,1453527$
$-3,9638112$
$-5,0733551$
$-6,5619076$
$-8,5356685$
$-11,100864$
$-14,311700$
$-18,056232$
$-21,873802$
$-24,819411$

Tabela 3.5: Conjunto de dados do atrator de Lorenz reconstruído segundo a dimensão embutida e de separação encontradas ($m = 3$ e $\tau = 5$)

Dimensão 1	**Dimensão 2**	**Dimensão 3**
$-9,655962$	$-2,112057$	$-2,552179$
$-6,990209$	$-1,841175$	$-3,145353$
$-4,983493$	$-1,778494$	$-3,963811$
$-3,577362$	$-1,883483$	$-5,073355$
$-2,658921$	$-2,139759$	$-6,561908$
$-2,112057$	$-2,552179$	$-8,535668$
$-1,841175$	$-3,145353$	$-11,100864$
$-1,778494$	$-3,963811$	$-14,311700$
$-1,883483$	$-5,073355$	$-18,056232$
$-2,139759$	$-6,561908$	$-21,873802$
$-2,552179$	$-8,535668$	$-24,819410$

3.8 Aproximação de funções

Após reconstruir uma órbita em espaço de coordenadas de atraso, obtém-se um conjunto de pontos que define a regra do sistema dinâmico em estudo. Por exemplo, considere a órbita da função logística da figura 3.14, após calcular a dimensão de separação pelo método de auto-informação mútua (obtendo, neste caso, $\tau = 1$) e a dimensão embutida

pelo método dos falsos vizinhos (obtendo $m = 2$), reconstrói-se sua órbita no espaço de coordenadas de atraso tal como apresentado na figura 3.15.

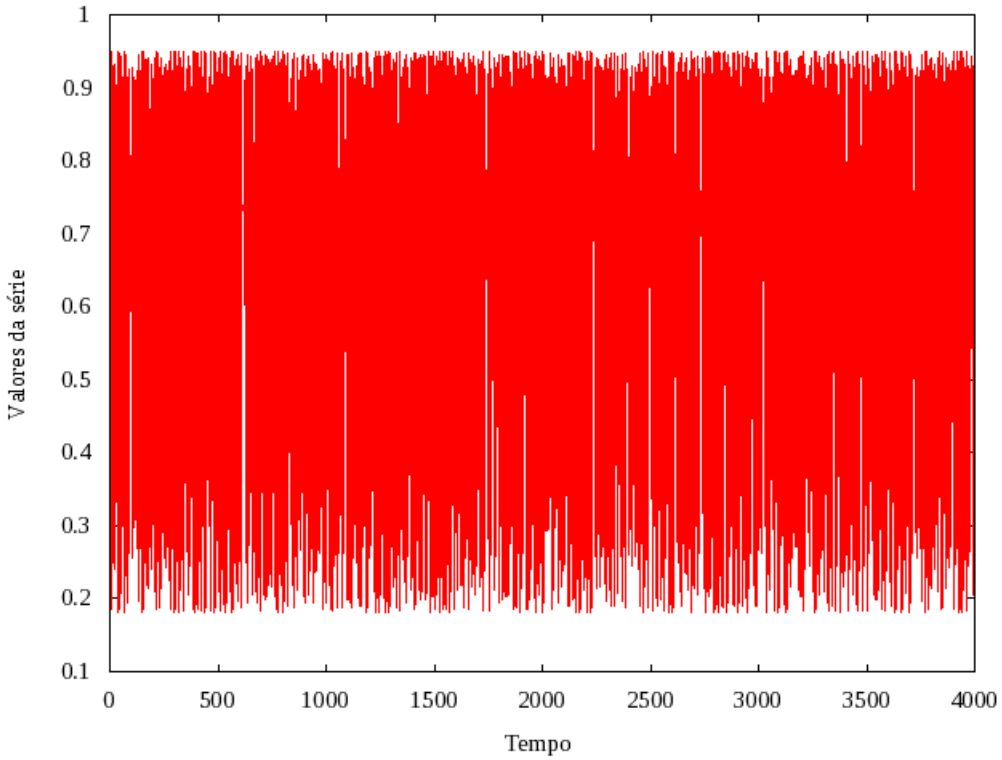


Figura 3.14: Órbita da função logística

A órbita reconstruída permite encontrar a regra origem do sistema dinâmico, o que auxilia no estudo de suas tendências. Conhecendo as tendências pode-se, por exemplo, classificar, predizer e determinar esse comportamento e influências em demais sistemas interdependentes. Para obter a regra origem, deve-se realizar a regressão dos pontos do espaço de coordenadas de atraso.

A figura 3.16 exemplifica os pontos obtidos em espaço de coordenadas de atraso e sua regra origem, a qual é definida pela equação $f(x) = -3,8x^2 + 3,8x - 1,49236 \cdot 10^{-07}$ (regressão realizada pelo método dos mínimos quadrados (Bretscher, 2004)). Pode-se aplicar uma condição inicial, ou seja, x_t, à essa regra e estimar saídas para k instantes de tempo futuros. Pode-se, também, empregar mapas lineares e não lineares (Alligood *et al.*, 1997), como matrizes Jacobianas, a fim de estudar pontos fixos, períodos de órbitas e tendências globais do sistema.

No exemplo anterior foi adotado o método dos mínimos quadrados para a regressão dos pontos em espaço multidimensional. No entanto, outras abordagens poderiam adotadas, tais como métodos auto-regressivos (Box *et al.*, 1994), funções radiais (Buhmann, 2003), filtros de Kalman (Wan & Van Der Merwe, 2000; Bianchi & Tinnirello, 2003; Kohler, 1997; Doblinger, 1998; Piovoso & Laplante, 2003), cadeias de Markov (Meng, 2003), redes neurais artificiais `TDNN`, `ATNN`, *Long Short-Term Memory* (`LSTM`), *Radial Basis Function*

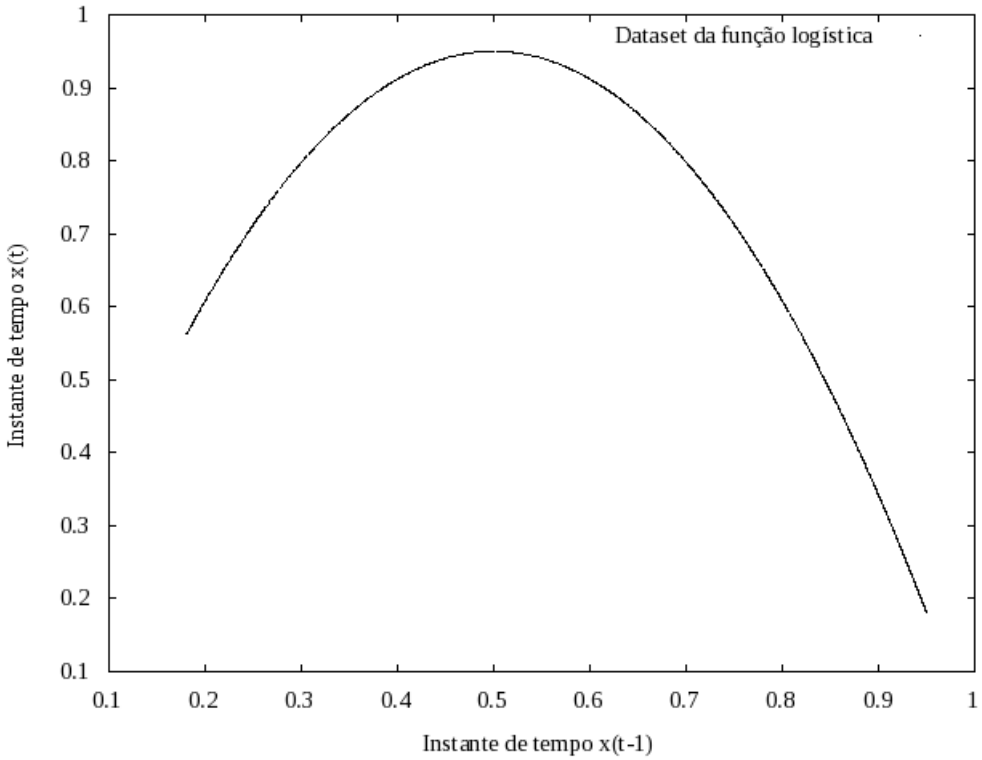


Figura 3.15: Função logística reconstruída em dimensão embutida 2 e de separação 1

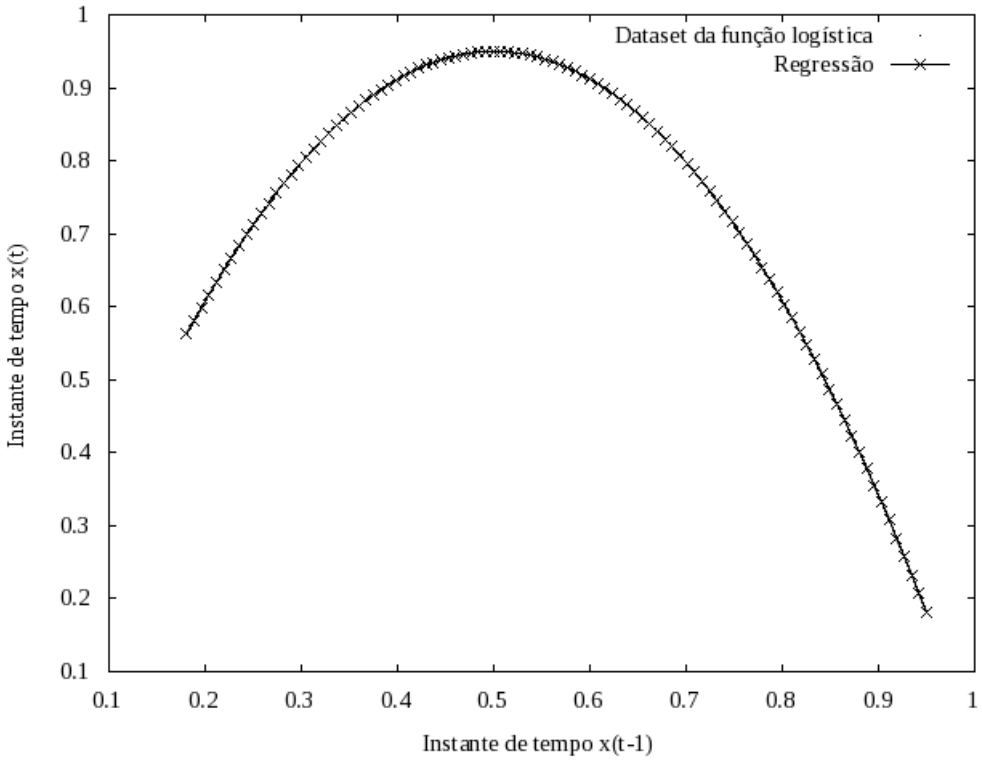


Figura 3.16: Regressão dos pontos no espaço de coordenadas de atraso

(`RBF`) e `RRBF` (Dodonov & Mello, 2008). Essas abordagens aproximam, de forma distinta, funções sobre o conjunto de pontos em plano multidimensional.

Parte dessas técnicas, como mínimos quadrados e métodos auto-regressivos, apresentam baixo desempenho para séries com comportamento não linear e variação abrupta. Outras, ainda, são indicadas somente para a predição de um próximo estado, ou valor, tais como filtros de Kalman e cadeias de Markov. Demais abordagens tais como funções radiais, `TDNN`, `ATNN`, `LSTM`, `RBF` e `RRBF` apresentam bons resultados para aproximação de funções, contudo, a maioria delas tem alto custo computacional (`TDNN` e `ATNN`) ou parametrização complexa (`LSTM`). Técnicas como funções radiais e redes neurais `RBF` apresentam boa aproximação de pontos, contudo desconsideram componentes dinâmicos, os quais geram perturbações responsáveis por modificar o comportamento de séries. A rede neural `RRBF` aborda tal dinamicidade por meio de uma primeira camada de neurônios em sua arquitetura (Zemouri *et al.*, 2003). A comparação dessas abordagens auxiliou na escolha da rede neural `RRBF`, a qual se mostra adequada no contexto deste trabalho.

A rede neural `RRBF` estende a `RBF` tradicional, introduzindo uma primeira camada recorrente (figura 3.17) para proporcionar funcionalidades de memória dinâmica. Essa camada auxilia na representação de reações causadas por perturbações em séries temporais e, portanto, na modificação de tendências para predição de comportamento futuro. A segunda camada dessa arquitetura é composta por neurônios com funções radiais de ativação.

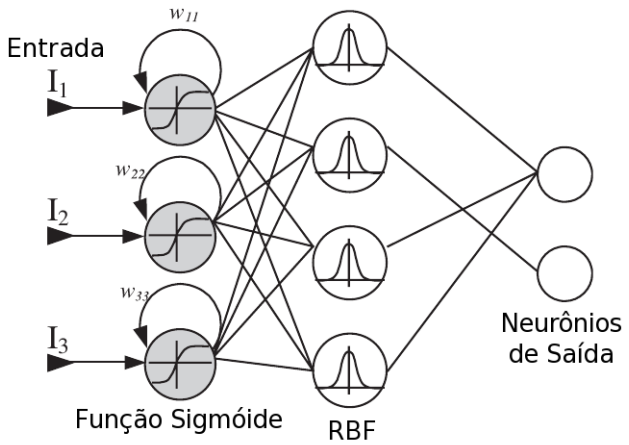


Figura 3.17: Arquitetura da rede neural artificial RRBF

$$x(t) = \frac{1 - exp(-kw_{ii}x(t-1))}{1 + exp(-kw_{ii}x(t-1))} \tag{3.8}$$

A primeira camada calcula a relevância de padrões de entrada históricos sobre um novo padrão, considerando a função sigmóide apresentada na equação 3.8, onde $x(t-1)$ representa o valor de uma entrada anterior, k parametriza a função de ativação, e w_{ii} é o peso dado para conexões recorrentes entre neurônios (Zemouri *et al.*, 2003). Valores resultantes da camada de recorrência são processados pela próxima camada, constituída

por neurônios de uma rede neural RBF tradicional, a qual emprega funções radiais para ativação (geralmente baseadas em Gaussianas e Multiquadráticas). Essas funções são tipicamente projetadas para aproximar funções em relação a dados de entrada. Resultados da RBF são gerados pela equação 3.9, que realiza o produto das saídas das N funções radiais $\phi(.)$, centradas em c_i, pelos coeficientes de peso w_i (Powell, 1987).

$$y(x) = \sum_{i=1}^{N} w_i \cdot \phi(||x - c_i||) \tag{3.9}$$

O algoritmo de uma rede RBF consiste de duas fases: treinamento e execução. A etapa de treinamento determina o número de neurônios necessários para representar uma série de entrada, os valores para os centros c_i das funções radiais $\phi(.)$ e os coeficientes de peso w_i. Os centros podem ser definidos por valores fixos ou por uma heurística. Coeficientes de peso são calculados de acordo com a equação 3.10, onde w representa os coeficientes, $\mathbf{b} = [c_0, c_1, .., c_n]$ os centros de funções, e $\mathbf{G}$ a matriz de pesos na forma $g_{ij} = \phi(||c_j - c_i||)$. Durante a fase de execução, essa rede neural avalia padrões de entrada e computa suas similaridades em relação aos centros (determinadas pelos coeficientes de peso w), utilizando a equação 3.9.

No contexto deste trabalho, adotou-se a rede neural artificial auto-organizável e não supervisionada SONDE como heurística para definir centros (Albertini & Mello, 2007) para as funções radiais $\phi(.)$. SONDE é apresentada, em detalhes, na próxima seção.

$$w = \mathbf{G}^{-1} \cdot \mathbf{b} \tag{3.10}$$

Resultados apresentados por Zemouri *et al.* (2003) motivam a adoção da rede neural RRBF, pois comprovam sua capacidade em adaptar saídas geradas em função da dinâmica de séries. Esse fato motivou a adoção da RRBF para obter regras de sistemas dinâmicos formados por comportamentos de processos, tal como abordado nesta tese.

3.8.1 Rede neural artificial SONDE

A rede neural artificial SONDE foi adotada neste trabalho como heurística para encontrar agrupamentos, os quais são utilizados para definir centros para neurônios da segunda camada da RRBF. A SONDE agrupa padrões de entrada de forma não-supervisionada e *on-line* e foi inicialmente proposta para detectar novidades em séries temporais (Albertini & Mello, 2007).

A rede SONDE foi projetada com o objetivo de integrar características presentes nas técnicas SOM, Grow When Required (GWR) e *Adaptive Resonance Theory* (ART). A arquitetura dessa rede é dividida em três camadas (figura 3.18): camada de entrada e pré-processamento – na qual padrões podem ser opcionalmente normalizados; camada de neurônios competitivos – na qual ocorre a ativação de neurônios segundo padrões de

entrada; a última para escolher a unidade mais representativa `BMU` (*Best Matching Unit*) – na qual o melhor neurônio (com ativação mais alta) é estimulado para melhor representar o padrão de entrada recebido.

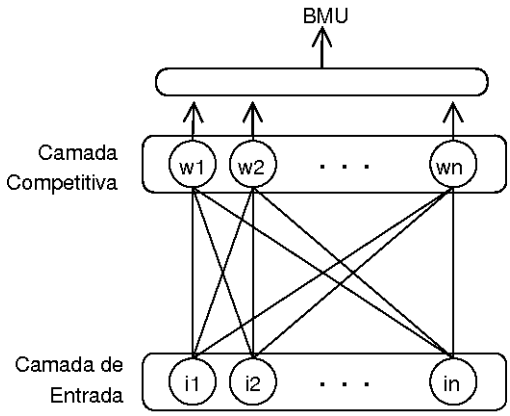


Figura 3.18: Arquitetura da rede neural artificial SONDE

A `SONDE` representa dados de entrada em neurônios adaptativos. Novos neurônios são criados, conforme novidades são detectadas. Cada neurônio c é definido pelo centróide médio $\vec{w}_c$ de seus padrões, raio médio rad_c em relação à dispersão desses padrões e um grau mínimo de similaridade α_c para reconhecer novos padrões.

Essa rede agrupa, de maneira não supervisionada, padrões de entrada similares. Após um padrão ser agrupado em um neurônio, essa unidade é estimulada para representar um histórico de dados de entrada. Todos os neurônios têm um raio máximo de ação para agrupamento de padrões.

O raio de ativação e o centróide de um neurônio adaptam-se conforme um novo padrão é agrupado. Quando nenhum neurônio é capaz de agrupar um padrão, um novo é criado, indicando novidade. Entretanto, de acordo com alterações nas entradas, neurônios adaptam-se, esquecendo informações passadas. As taxas de esquecimento e de adaptação dos neurônios são parâmetros da rede.

Esses mecanismos de adaptação e esquecimento são descritos por equações relacionando a entrada atual e o armazenamento dos resultados passados. Para cada padrão $\vec{I}_t$ recebido (opcionalmente normalizado na camada 1 pela equação 3.11) no instante t, o valor de ativação a_c de cada neurônio da camada competitiva é calculado segundo a equação 3.12.

Quando nenhum neurônio é capaz de representar $\vec{I}_t$, ou seja, quando a equação 3.13 é satisfeita, um novo neurônio é criado. O centróide do novo neurônio $\vec{w}_{new}$ é definido segundo os valores dos padrões responsáveis por sua criação (resultando em ativação máxima com $a_{new} = 1$). O nível de similaridade mínimo, α_{new}, é definido segundo a constante α_0. O raio médio inicial rad_{new} recebe o valor $-\ln(\alpha_0)$.

$$\vec{I}_t = \frac{\vec{I}_t}{\|\vec{I}_t\|} \tag{3.11}$$

$$a_c = \exp(-\|\vec{I}_t - \vec{w}_c\|) \tag{3.12}$$

$$a_c < \alpha_c, \forall c \tag{3.13}$$

Quando a equação 3.14 é satisfeita, o neurônio vencedor (ou `BMU`) é obtido por $BMU = argmax_c(\exp -\|\vec{I} - \vec{w}_c\|)$. Esse neurônio é aquele que melhor representa o padrão de entrada.

$$\|\vec{I} - \vec{w}_c\| \leq -\ln \alpha_c \tag{3.14}$$

O neurônio vencedor é adaptado a fim de especializar-se (equação 3.15, com t sendo o instante de tempo). Essa especialização é realizada de tal maneira que o raio e centróide do neurônio representem os valores de entrada, segundo as médias móveis exponenciais ponderadas (*Exponential Weighted Moving Average* – `EWMA`) (Achelis, 2000) apresentadas, respectivamente, nas equações 3.15 e 3.16. Os parâmetros γ e Ω dessas equações definem a influência de padrões passados na situação atual. Quanto maior é o valor desses parâmetros, maior é a taxa de esquecimento da `SONDE`.

Após atualizar o centróide do neurônio vencedor, seu grau de similaridade mínimo (α_c) é modificado para melhor representar padrões de entrada em próximas ativações. Após serem especializados, qualquer padrão divergente pode ser detectado como novidade e ser utilizado na criação de um novo neurônio. Essa atualização é proporcional à taxa de modificação p (equação 3.17) do novo raio médio em relação ao anterior seguindo a equação 3.18.

Essa equação assegura dois comportamentos de adaptação diferentes para α_c, os quais seguem a distância entre o raio (que define um limite interno para a especialização do neurônio) e o limite de similaridade $-\ln(\alpha_c)$. Quanto maior é essa distância, mais acelerada é a especialização a qual converge para representar padrões de entrada.

Quando padrões são uniformemente agrupados pela área de cobertura, o limite de similaridade torna-se próximo ao interno, o que implica que o neurônio tem boa adaptação aos padrões de entrada. Esse comportamento é similar ao treinamento da técnica *Support Vector Machine* (`SVM`) (Manevitz & Yousef, 2001), embora sua fase de aprendizado não seja *on-line*, nem sua execução adaptativa.

$$\vec{w}_{BMU_t} = \vec{w}_{BMU_{t-1}} \cdot (1 - \gamma) + \vec{I}_t \cdot \gamma \tag{3.15}$$

$$rad_{BMU_t} = rad_{BMU_{t-1}} \cdot (1 - \Omega) + \|\vec{I_t} - \vec{w}_{BMU_{t-1}}\| \cdot \Omega \tag{3.16}$$

$$p = \|\frac{rad_{BMU_t} - rad_{BMU_{t-1}}}{\max(rad_{BMU_t}, rad_{BMU_{t-1}})}\| \tag{3.17}$$

$$\alpha_{BMU_t} = \min\left((1 + p) \cdot \alpha_{BMU_{t-1}}, \exp^{(-rad_{BMU_t} \cdot (1+p))}\right) \tag{3.18}$$

Neurônios são criados para representar conjuntos de padrões de entrada. Os centróides $\vec{w}_c$, de cada neurônio c criado pela rede neural `SONDE`, são utilizados para definir os centros das funções radiais $\phi(.)$ presentes na segunda camada da rede `RRBF` (apresentada na seção anterior).

3.9 Considerações finais

Este capítulo apresentou conceitos sobre sistemas dinâmicos, focando, principalmente, na reconstrução de órbitas em espaço de coordenadas de atraso. A partir de tal reconstrução, pode-se, por meio de aproximação de funções, obter a regra que dá origem ao sistema dinâmico em estudo. Essa regra permite estudar tendências, classificar e predizer comportamentos. Além da reconstrução de séries, pode-se empregar os expoentes de Lyapunov para estudar separações de pontos de trajetórias e horizontes de predição, e de Hurst para compreender quão aleatório é um conjunto de dados. Esta tese de livre docência aplica tais conceitos sobre dados experimentais a fim de predizer comportamentos de processos e empregá-los em otimização de escalonamento em grades.

Capítulo

4

Otimização de Escalonamento de Processos em Grades por meio de Sistemas Dinâmicos e Técnicas Inteligentes

4.1 Considerações Iniciais

Diversas iniciativas têm proposto arquiteturas e integrações de ferramentas e serviços buscando prover autonomia para grades computacionais (Parashar & Hariri, 2006; Othman *et al.*, 2003; Cheng *et al.*, 2002; Shirose *et al.*, 2004; Champrasert *et al.*, 2005). No entanto, poucas dessas têm sido voltadas para o estudo, projeto e avaliação de desempenho de técnicas e modelos com o objetivo de melhorar aspectos autônomos em grades (Lee & Suzuki, 2006; Suzuki & Suda, 2005; Champrasert & Suzuki, 2006a,b). A falta desse tipo de investigação motivou, em 2003, o projeto `MidHPC` (Andrade Filho *et al.*, 2008), que visa prover autonomia a grades computacionais por meio da otimização de escalonamento, distribuição de dados, rebalanceamento de cargas, configuração de serviços para suportar múltiplos níveis de carga, segurança por meio da avaliação de comportamento de processos e usuários, e tolerância a falhas.

Pesquisas resultantes do projeto `MidHPC` apresentaram contribuições significativas (Senger *et al.*, 2005a, 2007; Mello & Senger, 2006; Mello *et al.*, 2006b,a, 2007c,b; Andrade Filho *et al.*, 2008), principalmente no que se refere à otimização de desempenho de aplicações em grades. Esses resultados devem-se, essencialmente, ao emprego de estimativas de comportamento de processos. As contribuições oriundas de tais estimativas motivaram esta tese de livre docência que emprega conceitos sobre sistemas dinâmicos e

técnicas inteligentes para modelar e predizer o comportamento de processos. Essas predições visam otimizar operações de escalonamento de processos em ambientes de grade computacional.

Neste contexto, este capítulo apresenta trabalhos relacionados que, também, empregam conhecimento sobre o comportamento de aplicações. Essas trabalhos apoiaram o desenvolvimento de várias pesquisas do projeto `MidHPC`, inclusive desta tese. Em seguida, são apresentados o objetivo, a metodologia adotada, e a proposta de uma nova política de escalonamento de processos que emprega comportamentos previstos. Resultados obtidos com tal política, bem como sua comparação com demais abordagens, são descritos a seguir. Uma avaliação do custo e complexidade da abordagem é, também, realizada.

4.2 Conhecimento sobre Aplicações

Muitos pesquisadores têm demonstrado que a utilização de conhecimento sobre aplicações e recursos computacionais auxilia na alocação otimizada de recursos em ambientes distribuídos (Harchol-Balter & Downey, 1997; Brecht & Guha, 1996; Naik *et al.*, 1997; Sevcik, 1989). Três principais fontes têm sido consideradas para a obtenção de conhecimento de aplicações: a descrição de requisitos computacionais provida por usuários (ou programadores); traços de execução (do inglês *traces*, o qual contém detalhes sobre operações executadas por processos) de todas as aplicações previamente executadas no ambiente; e traços de execução específicos de cada aplicação, obtidos por meio de monitoração. Dentre essas fontes, traços de execução têm demonstrado grande potencial para a descoberta de informações relevantes e contribuído, significativamente, na tomada de decisões (Senger *et al.*, 2007).

Técnicas que utilizam tais fontes para caracterização e estimação de comportamento de aplicações têm sido propostas na literatura (Devarakonda & Iyer, 1989; Brecht & Guha, 1996; Feitelson & Nitzberg, 1995; Feitelson *et al.*, 1997; Smith *et al.*, 1998; Harchol-Balter & Downey, 1997; Mello *et al.*, 2006b; Yarkhan & Dongarra, 2002; Abraham *et al.*, 2000; Senger, 2005). Muitas dessas confrontam características de aplicações à capacidade oferecida por um conjunto de recursos computacionais, a fim de otimizar a alocação e minimizar o tempo de resposta de processos. Em um dos primeiros trabalhos nessa linha, Sevcik (1989) demonstra, por meio de modelos analíticos e de simulação, que é possível melhorar, substancialmente, decisões de escalonamento empregando conhecimento sobre aplicações.

Para identificar estados de utilização de recursos em processos seqüenciais, Devarakonda & Iyer (1989) empregam uma técnica de reconhecimento de padrões para prever o consumo de processamento, operações de entrada/saída e memória. Essa técnica utiliza o algoritmo *k-means* (Hartigan & Wong, 1979), uma forma de agrupamento estatístico, para identificar estados de ocupação de recursos em um sistema `UNIX` uniprocessado.

Bons resultados foram obtidos, contudo os autores não realizam estudos complementares considerando comunicação e sincronização entre processos.

Brecht & Guha (1996) utilizam informações sobre o grau de paralelismo e tempo de execução de aplicações paralelas com o objetivo de melhorar decisões de escalonamento. Eles concluem que, adotando estimativas de ocupação de recursos, obtém-se desempenho próximo ao observado em situações em que o comportamento da aplicação é previamente conhecido.

Feitelson & Nitzberg (1995) demonstram que o tempo de execução de aplicações paralelas pode ser estimado a partir de execuções repetidas. Essa conclusão indica que o histórico de uma aplicação pode ser empregado para melhorar futuras decisões de escalonamento. No mesmo estudo, eles verificam que aproximadamente dois terços das aplicações paralelas, em ambiente de produção, são executadas múltiplas vezes. Posteriormente, Feitelson *et al.* (1997) observam que execuções repetidas de uma mesma aplicação tendem a apresentar padrões similares de consumo de recursos. Conseqüentemente, informações relativas ao comportamento das aplicações podem ser descobertas sem a cooperação explícita do usuário, por meio de histórico de execuções.

Smith *et al.* (1998) utilizam traços a fim de predizer o tempo de execução de aplicações paralelas. Esses traços são tratados como uma base de experiências, a partir da qual são definidas categorias de aplicações mais adequadas para gerar predições. A mesma metodologia é seguida por Krishnaswamy *et al.* (2004), que propõem algoritmos para estimar o tempo de execução de processos, e por Downey (1997), que categoriza aplicações utilizando o identificador de fila do escalonador e propõe funções de distribuição de probabilidades para caracterizar cargas de trabalho.

Harchol-Balter & Downey (1997) mostram como conhecimento sobre o tempo de vida de processos pode ser empregado para melhorar decisões de escalonamento. Eles analisam traços de execução de um sistema computacional baseado em estações de trabalho e modelam uma distribuição de probabilidades para o tempo de vida de aplicações seqüenciais. Por meio desses modelos, propõem um algoritmo de balanceamento de carga, que emprega conhecimento sobre processos seqüenciais para tomar decisões de migração de cargas entre processadores. Esse trabalho é, posteriormente, estendido por Mello & Senger (2004), que acrescentam uma análise de utilização de recursos na tomada de decisões.

Senger (2005) propõe dois modelos de extração de características de aplicações paralelas. O primeiro considera uma arquitetura de redes neurais da família `ART` (Carpenter & Grossberg, 1989) para a categorização automática de estados de utilização de recursos de processos. Conceitos de aprendizado baseado em instâncias são empregados para construir o segundo modelo, que permite predizer características de execução de aplicações paralelas. O diferencial desse trabalho é a aquisição incremental de conhecimento, de forma que o modelo e o conhecimento obtido podem ser atualizados na medida que novas informações são apresentadas. Esse trabalho ainda corrobora que algoritmos de escalona-

mento que adotam conhecimento sobre aplicações apresentam desempenho superior aos que não o fazem.

Mello *et al.* (2006b) propõem o algoritmo `Route` para o balanceamento de carga (Shivaratri *et al.*, 1992) em ambientes de grade computacional. Esse algoritmo utiliza conceitos de roteamento em redes de computadores para definir uma vizinhança e localizar os recursos mais adequados para receber cargas de trabalho. Experimentos comprovam que o algoritmo proposto superou outros propostos na literatura, quando avaliados em ambientes heterogêneos de larga escala. Esse algoritmo é estendido por Mello *et al.* (2007c), que propõem uma nova abordagem, denominada `RouteGA`, a qual utiliza um algoritmo genético para otimizar a alocação inicial de processos. Essa alocação reduz o tempo necessário para estabilizar a distribuição eqüitativa de cargas no sistema.

Yarkhan & Dongarra (2002) comparam uma abordagem baseada em *Simulated Annealing* (`SA`) com outra de busca gananciosa (*greedy search*). Esse trabalho considera o problema de escalonamento de uma aplicação numérica específica (*ScaLAPACK LU solver*) executando em grades computacionais. Eles concluem que a metaheurística `SA` melhora os resultados de escalonamento, pois permite uma busca mais flexível em relação aos mínimos locais da função de custo (Mello & Senger, 2008). Esse trabalho considera apenas uma aplicação paralela e não adota predições sobre características de aplicações. Para avaliar as soluções obtidas, os autores consideram apenas o impacto na carga de processamento dos computadores da grade.

Abraham *et al.* (2000) utilizam predições sobre o tempo de execução de aplicações paralelas e heurísticas inspiradas em processos comuns da natureza, tais como `SA` e algoritmos genéticos, a fim de escalonar aplicações em grades computacionais. Esse trabalho considera apenas aplicações independentes (*Bag-of-Tasks*), que não se comunicam, e assim, não modelam o impacto imposto em rede. Heurísticas similares para otimizar o escalonamento de processos independentes são, também, exploradas por Lee & Zomaya (2007) e Mika *et al.* (2004).

Esses diversos trabalhos evidenciam a importância de empregar conhecimento sobre o comportamento de aplicações. Um estudo mais profundo sobre a dinâmica desses comportamentos motivou esta tese de livre docência.

4.3 Objetivo

Como discutido, diversas frentes de pesquisa têm comprovado os benefícios do emprego de conhecimento sobre aplicações a fim de otimizar operações em sistemas computacionais. Trabalhos anteriores consideram, contudo, médias comportamentais que não representam todas as variações de ocupação geradas por processos (Devarakonda & Iyer, 1989; Brecht & Guha, 1996; Feitelson & Nitzberg, 1995; Feitelson *et al.*, 1997; Smith *et al.*, 1998; Harchol-Balter & Downey, 1997; Mello *et al.*, 2006b; Yarkhan & Dongarra, 2002; Abraham

et al., 2000; Senger, 2005). Otimizações conduzidas com base nessas médias melhoram o desempenho global de sistemas, entretanto dificultam a reconfiguração do ambiente para aumentar o desempenho de determinadas operações.

Por exemplo, considere uma aplicação $A = \{p_0, \ldots, p_{n-1}\}$ composta por n processos, em que cada um deles tem comportamentos de ocupação de processamento e canais de comunicação (a ocupação de demais recursos é suprimida para simplificar o entendimento), os quais podem ser representados por séries temporais. Assume-se que eles processam um alto volume v de operações e, em seguida, comunicam-se, trocando m unidades de informação. Considere, então, que o volume v supera m em algumas ordens de grandeza, contudo, mesmo assim, m apresenta alto valor e influencia, significativamente, no tempo total de execução. Empregar o comportamento médio, nesse caso, faria com que as operações de processamento sobrepusessem as de comunicação e, muito provavelmente, o otimizador alocaria processos em regiões da grade a fim de maximizar a ocupação de processadores. Canais de comunicação entre tais regiões possivelmente apresentam altas latências e baixa largura de banda, o que geraria altos atrasos de sincronização em momentos de comunicação entre processos.

Melhores soluções seriam encontradas ao compreender a variação comportamental de processos e adaptar decisões de escalonamento, antecipando a necessidade de rebalanceamento de cargas em função de operações futuras. Essa variação comportamental depende da dinâmica envolvida nas séries temporais que caracterizam a ocupação de recursos. Esses fatores motivaram esta tese de livre docência que emprega conceitos sobre sistemas dinâmicos, em conjunto com técnicas inteligentes, para modelar e predizer o comportamento de processos com o objetivo de otimizar operações de escalonamento em ambientes de grade computacional. Detalhes sobre a metodologia aplicada e resultados são apresentados a seguir.

4.4 Metodologia Aplicada

As pesquisas envolvidas nesta tese foram iniciadas com a exploração de técnicas para obtenção de conhecimentos por meio de redes neurais artificiais. Sabe-se que esses modelos conexionistas permitem classificação, agrupamento e regressão de dados, como também a otimização, o acúmulo, e a representação de conhecimento. Essas redes extraem regras comportamentais de dados e as armazenam, internamente, em neurônios ou em seus pesos de conexões (Freeman & Skapura, 1991; Haykin, 2008). Essas regras são extraídas em fase de treinamento, na qual a rede é configurada para gerar saídas esperadas. Dentre as principais redes neurais que aprendem sobre o comportamento de sistemas dinâmicos estão: `TDNN` (Waibel *et al.*, 1989), `ATNN` (Lin *et al.*, 1995) e `RRBF` (Ryad *et al.*, 2001). Essas redes neurais são, essencialmente, voltadas para a predição de órbitas de sistemas dinâmicos.

Esses modelos conexionistas, contudo, requerem que usuários parametrizem, de acordo com as características da série em estudo, diversos aspectos topológicos, tais como o número de neurônios, sua organização em camadas e conexões. Esses parâmetros regem a forma com que a dinâmica de padrões é extraída. Essa dependência de parametrização motivou outras abordagens para encontrar o número ideal de neurônios, camadas e conexões por meio de técnicas de otimização, tais como algoritmos genéticos (Han & May, 1996). Contudo, essas abordagens avaliam diversas organizações para arquiteturas de redes neurais, o que apresenta alto custo computacional.

Alguns exemplos da dificuldade em parametrizar e extrair a dinâmica de séries temporais são apresentados por Wan (1994) e Zemouri *et al.* (2003). No primeiro trabalho, Wan (1994) destaca a necessidade de definir diferentes *time delays*, número de neurônios e conexões para a `TDNN`. No caso da `RRBF`, Zemouri *et al.* (2003) sugerem variar o número de neurônios da camada de entrada da rede neural até obter resultados satisfatórios. Tais neurônios influenciam na memória dinâmica do modelo. A escolha desses parâmetros é, em geral, feita com base em estudos sobre conjuntos de treinamento e validação, os quais permitem analisar se uma topologia de rede neural atende a determinado problema. Esses aspectos dificultam a determinação da dinâmica de repetição de padrões, principalmente em casos em que não há longos conjuntos para treinamento e validação.

O estudo de comportamentos de processos em ambientes de grade computacional pode passar por alguns desses problemas, pois nem sempre há informações históricas suficientes para validar uma arquitetura de rede neural. Além disso, processos tendem a apresentar comportamentos distintos de utilização de recursos no tempo, os quais influenciam a dinâmica e, portanto, na topologia das redes neurais adotadas. Dessa forma, os comportamentos de determinado processo podem ser melhor previstos utilizando x neurônios interconectados em y camadas por meio de z ligações, enquanto outro requer uma topologia distinta. Outro fator decisivo é que a dinâmica envolvida nessas séries pode desatualizar, rapidamente, os modelos de conhecimento utilizados, o que requer novos treinamentos e validações para minimizar erros de representação de comportamento.

Esses problemas dificultam a adoção automática e transparente de modelos conexionistas, que necessitam de intervenção para parametrização. Essa intervenção evidencia a necessidade de definir uma topologia adequada para extrair a dinâmica de séries. A partir desses estudos investigou-se formas alternativas de compreensão da dinâmica envolvida no comportamento de processos. Observou-se, então, que conceitos sobre sistemas dinâmicos permitiriam a compreensão de órbitas, pontos estáveis e instáveis, os quais caracterizam tendências de séries em função de condições iniciais e auxiliam na predição de eventos. Pode-se, dessa forma, compreender, por exemplo, como evoluem as operações de comunicação ou utilização de `CPU` de determinado processo.

Para avaliar as órbitas de sistemas dinâmicos, esta tese considera os expoentes de Lyapunov, Hurst e técnicas de reconstrução de comportamento por meio de planos multi-

dimensionais (Jackson, 1989), denominados espaços de coordenadas de atraso. O expoente de Lyapunov identifica as tendências gerais de um sistema, as quais podem levar a um estado fixo (ou conservativo de comportamento), à dissipação de energia, ou à divergência instável e caótica (Alligood *et al.*, 1997). Sistemas conservativos são mais fáceis de serem estudados e modelados, pois desenvolvem órbitas que se repetem periodicamente e, assim, tornam-se previsíveis[1]. Sistemas que dissipam energia apresentam tendência contínua para um ponto, tal como um pêndulo que, após determinado tempo, tende a parar. Sistemas divergentes são inerentemente mais complexos de serem modelados e estudados, pois modificam seus comportamentos e apresentam componentes não lineares.

Além do expoente de Lyapunov, há o de Hurst, que mede a aleatoriedade de um sistema dinâmico (Kaplan, 2003). Esse expoente avalia o quanto padrões históricos auxiliam na estimativa de comportamentos futuros, por meio de correlações da série em janelas de tempo distintas. O resultado do expoente de Hurst permite concluir se um sistema apresenta auto-correlação, ou seja, dados históricos tendem a se repetir no futuro; anti-correlação, quando eventos futuros tendem a ocorrer de forma inversa; ou comportamento aleatório, o qual dificulta estimativas futuras.

Esses expoentes direcionam conclusões iniciais sobre o comportamento de sistemas dinâmicos. Contudo, para complementar tais conclusões, faz-se necessário reconstruir as séries em espaço de coordenadas de atraso. Essa reconstrução permite o estudo de órbitas, influências de pontos fixos e tendências nos valores das séries. Segundo o teorema de imersão de Whitney (1936), estendido, posteriormente, por Takens (1980), reconstruir um sistema em espaço com maior número de dimensões permite encontrar todos seus estados, bem como as relações entre eles. Essa mudança de espaço simplifica o estudo e modelagem de sistemas.

Para reconstruir um sistema dinâmico deve-se encontrar suas dimensões de separação e embutida, as quais definem como reorganizá-lo em espaço multidimensional, como proposto por Takens (1980), de tal forma que seja possível compreender as influências que valores passados causam em eventos futuros.

Após reconstruir o comportamento de uma série, aplicam-se técnicas de regressão sobre pontos do espaço de coordenadas de atraso. As funções resultantes permitem compreender a dinâmica e, portanto, predizer o comportamento de séries com base em estados passados.

Diversas técnicas de regressão podem ser empregadas, parte delas oferece baixo desempenho para séries com comportamento não linear e com alta variabilidade, tais como mínimos quadrados (Bretscher, 2004) e métodos auto-regressivos (Box *et al.*, 1994). Outras ainda são indicadas somente para a predição de um próximo estado, ou valor, tais como filtros de Kalman (Wan & Van Der Merwe, 2000; Bianchi & Tinnirello, 2003; Kohler, 1997; Doblinger, 1998; Piovoso & Laplante, 2003) e cadeias de Markov (Meng, 2003).

Outras abordagens como funções radiais, `TDNN`, `ATNN`, `LSTM`, `RBF` e `RRBF` (Dodonov &

[1]Essa afirmação considera sistemas unidimensionais, tais como os abordados nesta tese.

Mello, 2008) apresentam bons resultados para a aproximação de funções, contudo, a maioria tem alto custo computacional (TDNN e ATNN) ou dificuldade de parametrização (LSTM). Técnicas como funções radiais e redes neurais RBF apresentam boa aproximação de pontos, contudo desconsideram componentes dinâmicos, que geram perturbações responsáveis por modificar o comportamento de séries. A rede neural RRBF aborda essa questão por meio de sua arquitetura que apresenta uma camada de entrada para tratar aspectos de memória dinâmica (Zemouri *et al.*, 2003).

Devido a essas questões, nesta tese considera-se a rede neural RRBF para aproximar funções a partir de pontos em espaço de coordenadas de atraso.

4.5 Estudos sobre Comportamento de Processos

A metodologia apresentada foi aplicada sobre dois conjuntos de dados de comunicação entre processos de aplicações paralelas científicas. Esses conjuntos, obtidos no *site* do *National Energy Research Scientific Computing Center* (NERSC)[2], são resultantes da execução das seguintes aplicações (NERSC, 2008):

1. GTC[3] – esse software é utilizado para estudar microturbulências em plasmas de fusão toroidal magneticamente confinados por meio do método de partícula em célula (*Particle-In-Cell* – PIC), considerando espaço tridimensional. Essa aplicação resolve a equação gyro-média de Vlasov em espaço real, utilizando abordagens girotécnicas globais e uma aproximação eletrostática. O código fonte do GTC, otimizado para alta eficiência em máquinas superescalares e vetoriais, foi escrito em Fortran90/95, tendo cerca de 4.300 linhas (17 arquivos fonte);

2. PMEMD[4] – esse software executa várias simulações incluindo dinâmica molecular, refinamentos e minimizações. O código fonte tem cerca de 30.000 linhas em Fortran90 e cerca de 50 em linguagem C.

Os conjuntos de dados gerados por tais aplicações têm o formato apresentado na tabela 4.1[5]. As duas primeiras colunas informam, respectivamente, o computador origem e destino da mensagem. A terceira define qual chamada MPI foi solicitada, a próxima apresenta o tamanho do *buffer* de mensagem. Em seguida é apresentado o número de vezes que a chamada foi continuamente realizada, tempo total consumido com o conjunto de chamadas, tempos mínimo e máximo despendidos em tais chamadas.

[2] NERSC – *Petascale Data Storage Institute* – http://pdsi.nersc.gov/benchmarks.htm.

[3] Autores do GTC: Zhihong Lin, Stephane Ethier e Jerome Lewandowski do *Princeton Plasma Physics Laboratory*.

[4] Autores do PMEMD: Robert E. Duke e Lee G. Pedersen da Universidade da Carolina do Norte – Departamento de Química de Chapel Hill, baseado na versão do Sander/Amber 6.

[5] Essa tabela apenas ilustra o formato. Ela não contém todos os dados de comunicação entre processos.

Tabela 4.1: Trecho do conjunto de dados GTC

#org	#dst	chamada MPI	tamanho *buffer*(bytes)	#chamadas	tempo total(segs)	tempo mínimo(segs)	tempo máximo(segs)
197	196	MPI_Sendrecv	2595216	1	$1,6917e-02$	$1,6917e-02$	$1,6917e-02$
228	229	MPI_Sendrecv	2595216	1	$5,8828e-02$	$5,8828e-02$	$5,8828e-02$
248	247	MPI_Sendrecv	2595216	1	$7,4673e-03$	$7,4673e-03$	$7,4673e-03$
160	161	MPI_Sendrecv	2595216	1	$1,9284e-02$	$1,9284e-02$	$1,9284e-02$
211	212	MPI_Sendrecv	2595216	1	$1,5988e-02$	$1,5988e-02$	$1,5988e-02$
14	15	MPI_Sendrecv	2595216	1	$1,0592e-02$	$1,0592e-02$	$1,0592e-02$
109	110	MPI_Sendrecv	2595216	1	$1,2633e-02$	$1,2633e-02$	$1,2633e-02$
215	216	MPI_Sendrecv	2595216	1	$1,5624e-02$	$1,5624e-02$	$1,5624e-02$
60	59	MPI_Sendrecv	2595216	1	$6,4287e-03$	$6,4287e-03$	$6,4287e-03$

O *site* do NERSC contém diversos conjuntos de dados para as aplicações GTC e PMEMD, no entanto, esta tese considerou os denominados *Large*, ou seja, de grande porte, para ambas aplicações. Esses conjuntos apresentam as seguintes características:

1. GTC – contém o perfil dessa aplicação executando 256 tarefas sobre 16 computadores. O tempo total de execução é de 9.702,25 segundos. O tempo consumido com operações de comunicação é de 14,68% do total. A aplicação calcula, aproximadamente, 32 Gflops por segundo. O *switch* que interconectou os computadores recebeu um total de 411,25 e enviou 411,44 Gbytes durante toda a execução;

2. PMEMD – contém o perfil dessa aplicação executando 256 tarefas sobre 16 computadores. O tempo total de execução foi de 2.474,90 segundos. O percentual de comunicação foi de 53,14% do tempo total. Essa aplicação calcula, aproximadamente, 15,62 Gflops por segundo. O *switch* utilizado enviou e recebeu, respectivamente, um total de 1.130,76 e 1.130,35 Gbytes.

A fim de estudar o comportamento de comunicação de processos utilizando tais conjuntos de dados, resolveu-se selecionar chamadas oriundas de um mesmo processo e tipo (qual chamada MPI foi executada), permitindo caracterizar o comportamento de um sistema dinâmico. No caso do conjunto de dados GTC, considerou-se as chamadas MPI_Sendrecv oriundas do processo de identificador 197. No caso do PMEMD, considerou-se as chamadas MPI_Isend feitas pelo processo de número 181. As chamadas extraídas foram transformadas em séries temporais unidimensionais contendo o número de bytes enviados, ou seja, o tamanho do *buffer* da mensagem (coluna 4 da tabela 4.1) repetido pelo número de vezes que essa foi enviada (coluna 5). Tais séries são apresentadas, em escala logarítmica (a fim de simplificar visualização), nas figuras 4.1 e 4.2.

Primeiramente, calculou-se o expoente máximo de Lyapunov a fim de avaliar as tendências das órbitas de ambas séries unidimensionais. Os seguintes resultados[6] foram obtidos: GTC com $\lambda = 1,7886$ e PMEMD com $\lambda = 1,4659$. Esses valores confirmam a presença de

[6]Esses expoentes foram calculados utilizando 2 iterações para o método lyap_k do pacote TISEAN (Hegger *et al.*, 2009).

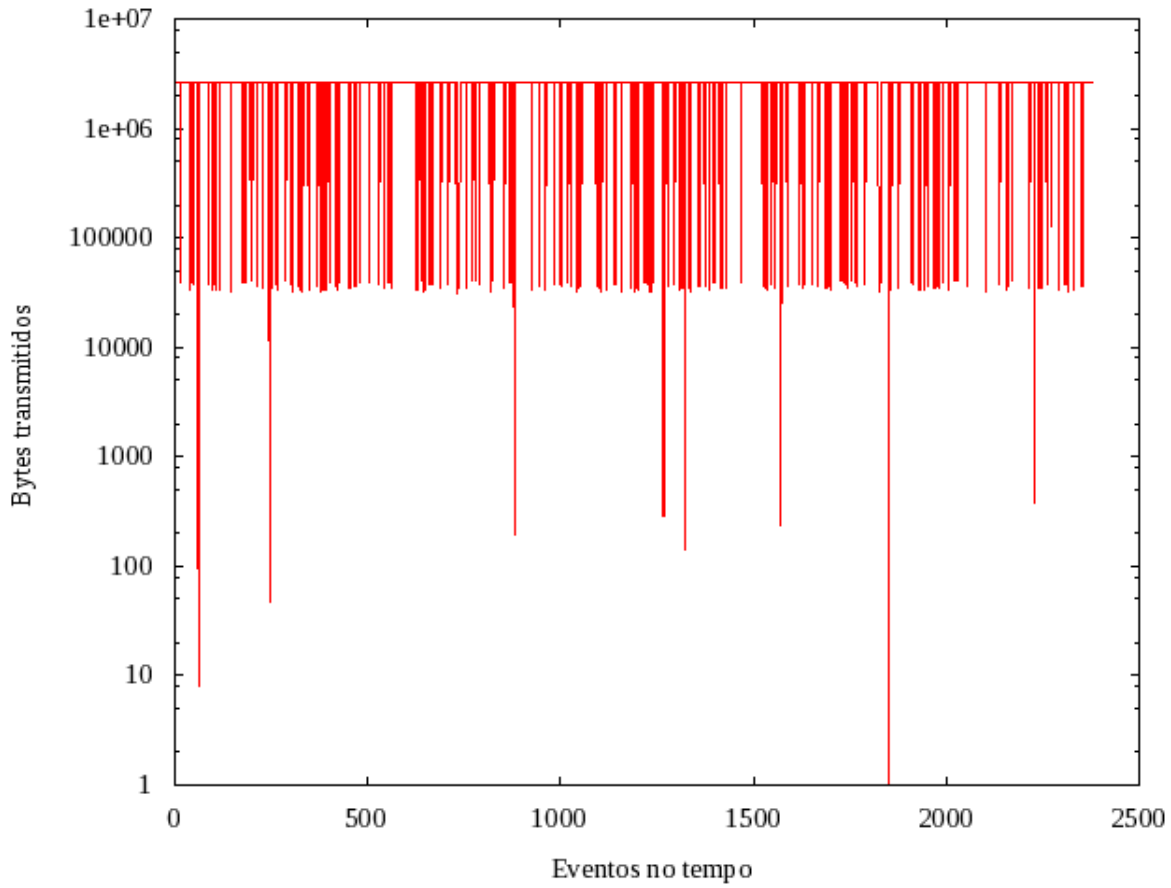


Figura 4.1: Conjunto de dados GTC

comportamento caótico e instável em ambos conjuntos de dados, o que dificulta estimação por meio de técnicas lineares tais como métodos auto-regressivos (Penny & Harrison, 2006).

Em seguida, calculou-se o expoente de Hurst a fim de avaliar o quão útil é o histórico de comportamento de uma série para realizar estimativas. O expoente de Hurst para `GTC` foi de $0,500801$ e para `PMEMD`, $0,65207$. O primeiro caracteriza o conjunto de dados `GTC` como caminhada aleatória (definido pelo valor do expoente igual a $0,5$), o que evidencia falta de correlação com eventos históricos, dificultando estimativas (Alligood *et al.*, 1997). O segundo expoente apresenta alguma, mesmo que pouca, correlação de dados históricos, o que tende a gerar melhores resultados em predições.

Após calcular esses expoentes, prosseguiu-se com o estudo das séries a fim de compreender suas tendências comportamentais por meio de reconstrução no espaço de coordenadas de atraso. Para isso, reconstruiu-se cada série $x_0, x_1, ..., x_{n-1}$ em um espaço multidimensional $x_n(m, \tau) = (x_n, x_{n+\tau}, ..., x_{n+(m-1)\tau})$, onde m é a dimensão embutida e τ, um *time delay* ou dimensão de separação (Jackson, 1989).

Para reconstruir as séries, utilizou-se técnicas de auto-informação mútua (Fraser & Swinney, 1986) e falsos vizinhos mais próximos (Kennel *et al.*, 1992b). A primeira permite encontrar o *time delay*, ou τ, para determinada série. Esse valor define a sazonabilidade

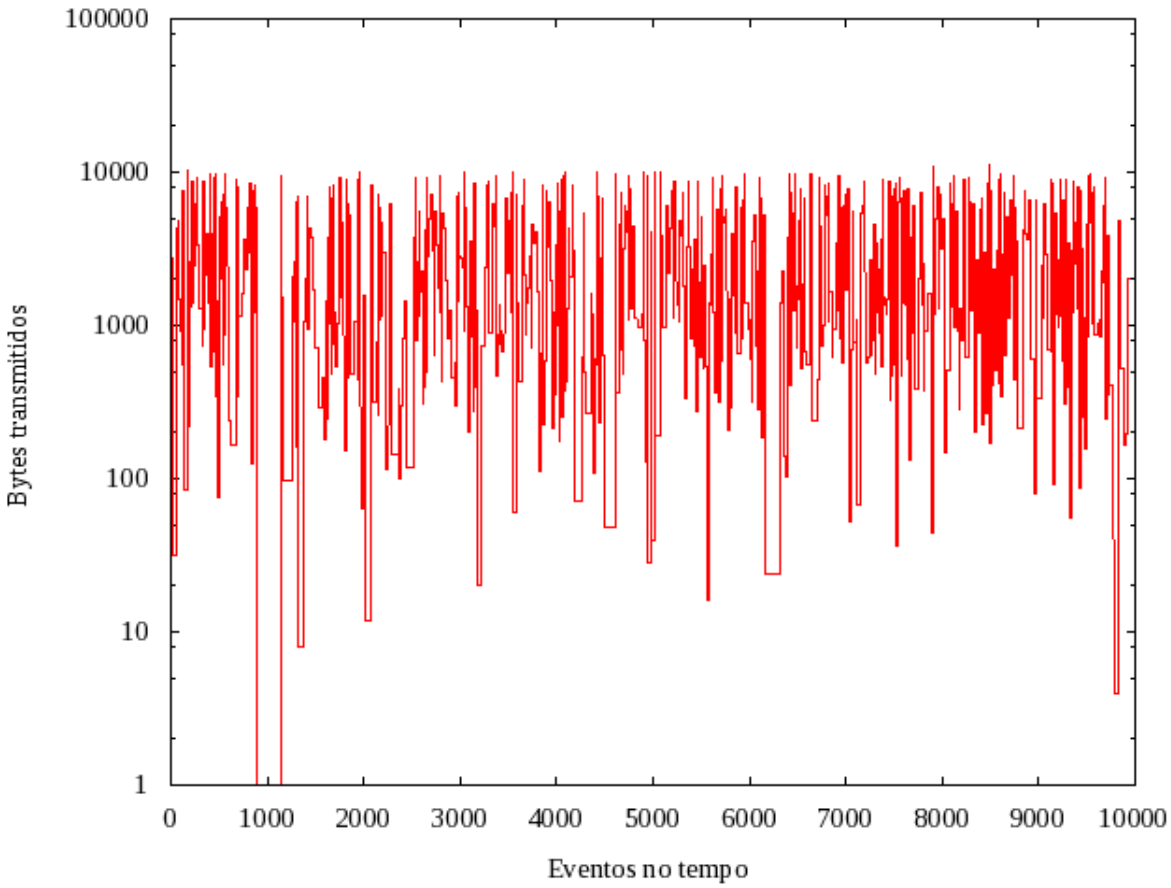


Figura 4.2: Conjunto de dados PMEMD

do sistema dinâmico, ou seja, como a série se repete no tempo, tal como o exemplo de temperaturas em dada região do mundo apresentado na seção 3.4. Reconstruir uma série com base nesse deslocamento permite compreender suas relações, tendências e influências temporais. A segunda técnica foi utilizada para encontrar o número de dimensões necessárias para reconstruir o comportamento da série em espaço de coordenadas de atraso, ou seja, definir o valor para m, ou dimensão embutida.

Os resultados para a técnica de auto-informação mútua são apresentados nas figuras 4.3 e 4.4. Observa-se, pelo primeiro mínimo obtido, que o conjunto de dados `GTC` apresenta dimensão de separação igual a dois, ou seja, necessita-se deslocar duas unidades de tempo da série para compreender seu comportamento repetitivo em espaço de coordenadas de atraso. A dimensão de separação obtida para o `PMEMD` foi igual a 37.

Após calcular as dimensões de separação para ambas séries, calculou-se as dimensões embutidas. As figuras 4.5 e 4.6 apresentam essas dimensões em função das separações anteriormente obtidas. Observa-se que ambas séries não tocam o eixo das ordenadas, portanto, não se encontra um número exato mínimo de dimensões para reconstruir seus comportamentos. Estudos comprovam que séries ruidosas apresentam esse comportamento, dificultando a definição de um valor exato para o número de dimensões. Liebert *et al.* (1991) investigaram tais séries e propuseram a escolha do número de dimensões quando

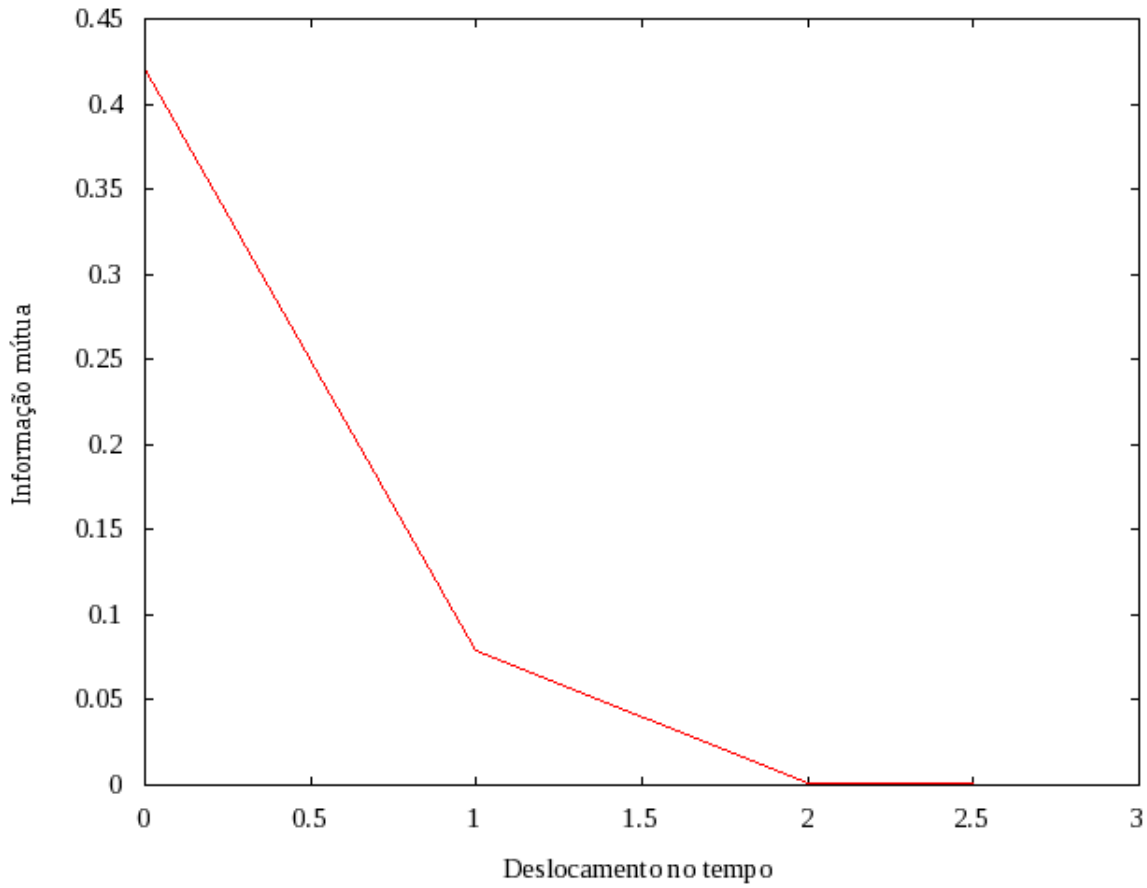


Figura 4.3: GTC – dimensão de separação

a fração de falsos vizinhos estiver abaixo de 30%. Nessas circunstâncias, os valores das dimensões embutidas para `GTC` e `PMEMD` foram, respectivamente, iguais a 4 e 3.

Após calcular essas dimensões, pôde-se reconstruir as séries em espaço de coordenadas de atraso. Para isso, aplicou-se a teoria de imersão de Takens (1980), em que uma série temporal $x_0, x_1, ..., x_{n-1}$ é reconstruída em espaço multidimensional $x_n(m, \tau) = (x_n, x_{n+\tau}, ..., x_{n+(m-1)\tau})$. A tabela 4.2 exemplifica a série `GTC` antes da reconstrução, enquanto a tabela 4.3, o resultado após reconstrução com os seguintes parâmetros: `GTC` – dimensão embutida de 4 e separação 2. O mesmo foi realizado para o `PMEMD` com dimensão embutida de 5 e separação 37 (somente o exemplo do `GTC` é apresentado, para fins ilustrativos).

O espaço de coordenadas de atraso das séries `GTC`, com 4 dimensões, e `PMEMD`, com 3 dimensões, são traçados nas figuras 4.7 e 4.8. O comportamento da `GTC` é parcialmente reconstruído, pois necessita de 4 dimensões. Observa-se que o comportamento do `GTC` segue formas retangulares, em que pontos de borda tendem a ser visitados. O `PMEMD` é composto por uma nuvem de pontos que se concentra ao redor de valores próximos de zero, para quaisquer dimensões.

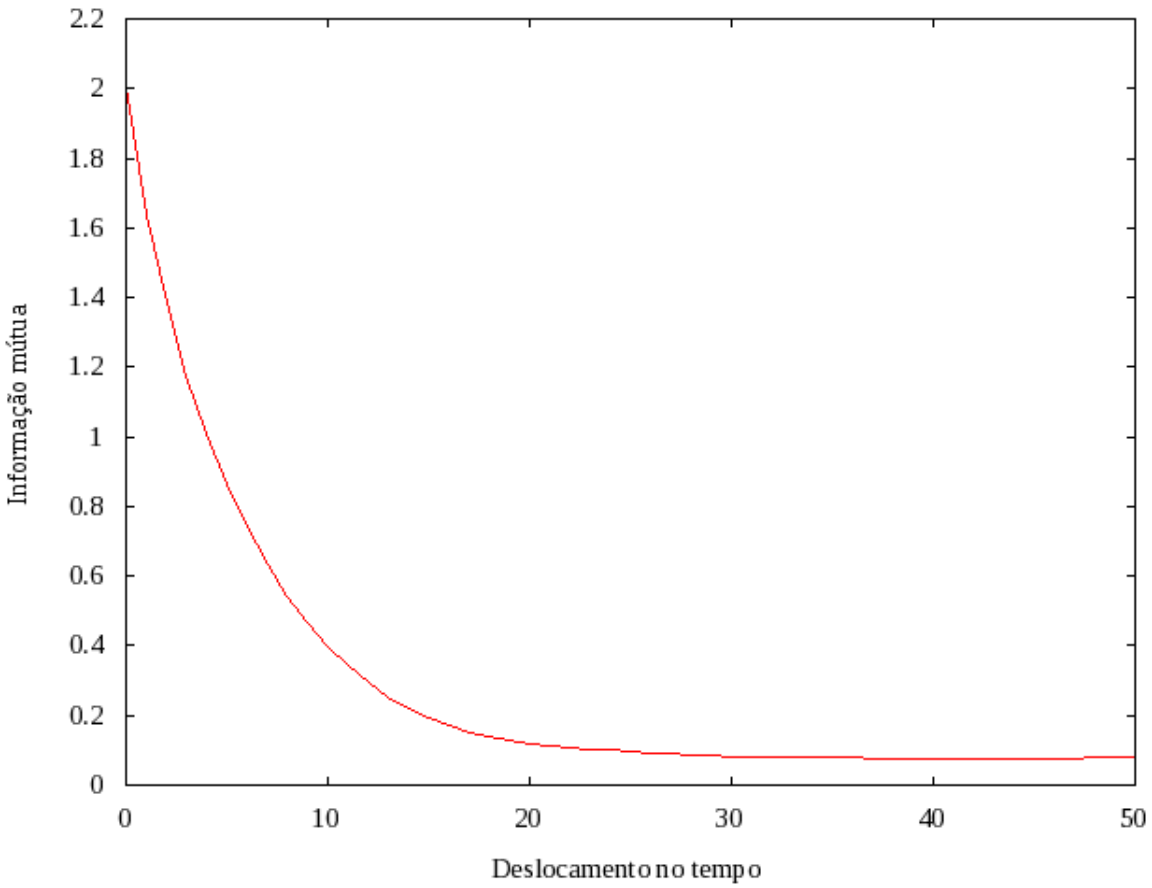


Figura 4.4: PMEMD – dimensão de separação

Tabela 4.2: Trecho do conjunto de dados GTC

Dimensão 1
2595216
2620896
2620896
2659152
2659152
2648160
2648160
2660496
2648064
2648064

4.6 Aplicando Comportamento de Comunicação em Escalonamento de Processos

Experimentos foram realizados a fim de comprovar os benefícios que a metodologia proposta proveria para aspectos autônomos em grades. O aspecto selecionado para estudo foi de auto-otimização, particularmente, aquele voltado para a redução do tempo de resposta (ou execução) de aplicações que executam sobre grades computacionais. Para

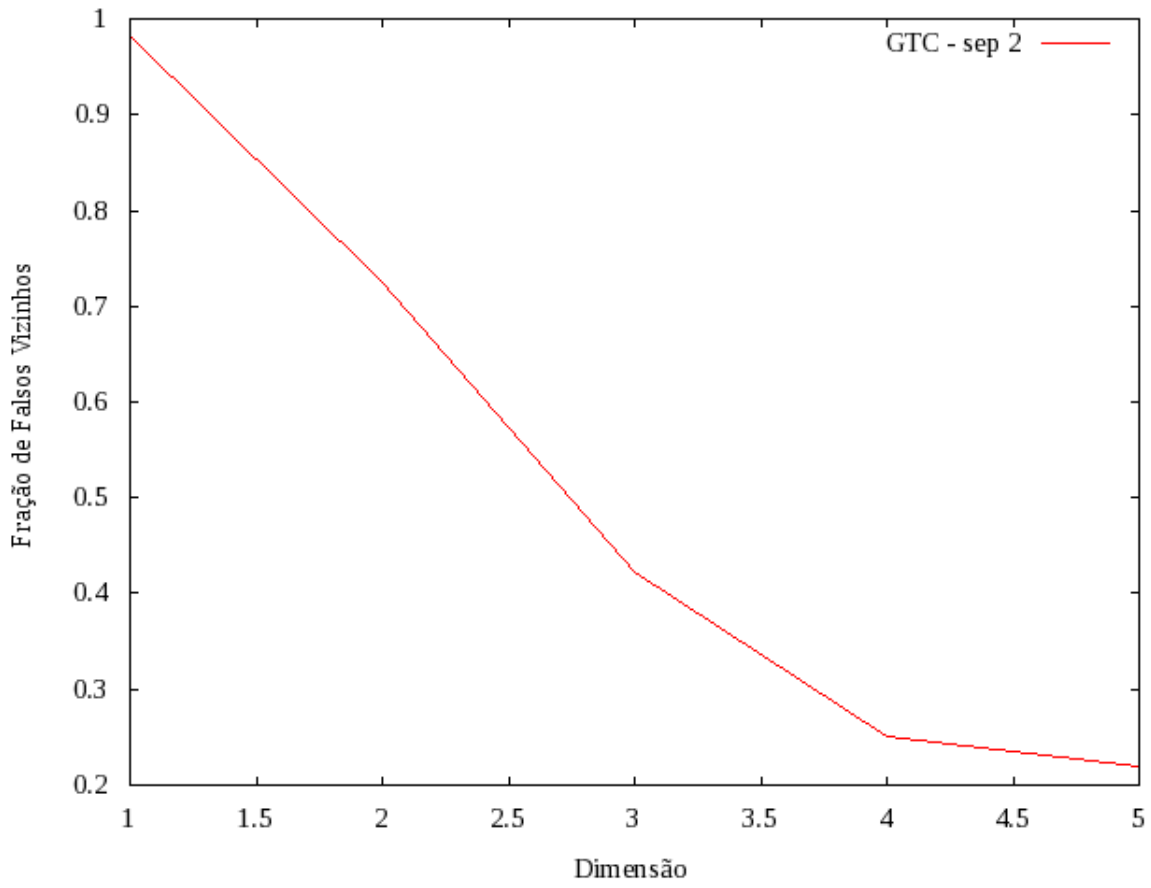


Figura 4.5: GTC – dimensão embutida

Tabela 4.3: Trecho do conjunto de dados `GTC` reconstruído com dimensão embutida 4 e separação 2

Dimensão 1	**Dimensão 2**	**Dimensão 3**	**Dimensão 4**
2595216	2620896	2659152	2648160
2620896	2659152	2648160	2660496
2620896	2659152	2648160	2648064
2659152	2648160	2660496	2648064
2659152	2648160	2648064	2635680
2648160	2660496	2648064	2635680
2648160	2648064	2635680	2645232
2660496	2648064	2635680	2645232

avaliar essas otimizações, uma nova política de escalonamento foi proposta, denominada `PredRoute`. Essa política considera o comportamento histórico médio de utilização de `CPU`, tal como o protótipo do `MidHPC` faz atualmente (baseado em `IBL` (Andrade Filho *et al.*, 2008)), contudo, utiliza, também, informações sobre eventos futuros de comunicação (obtidos segundo a metodologia). O problema de escalonamento de processos em grades e a abordagem da política `PredRoute` são descritos nas próximas seções.

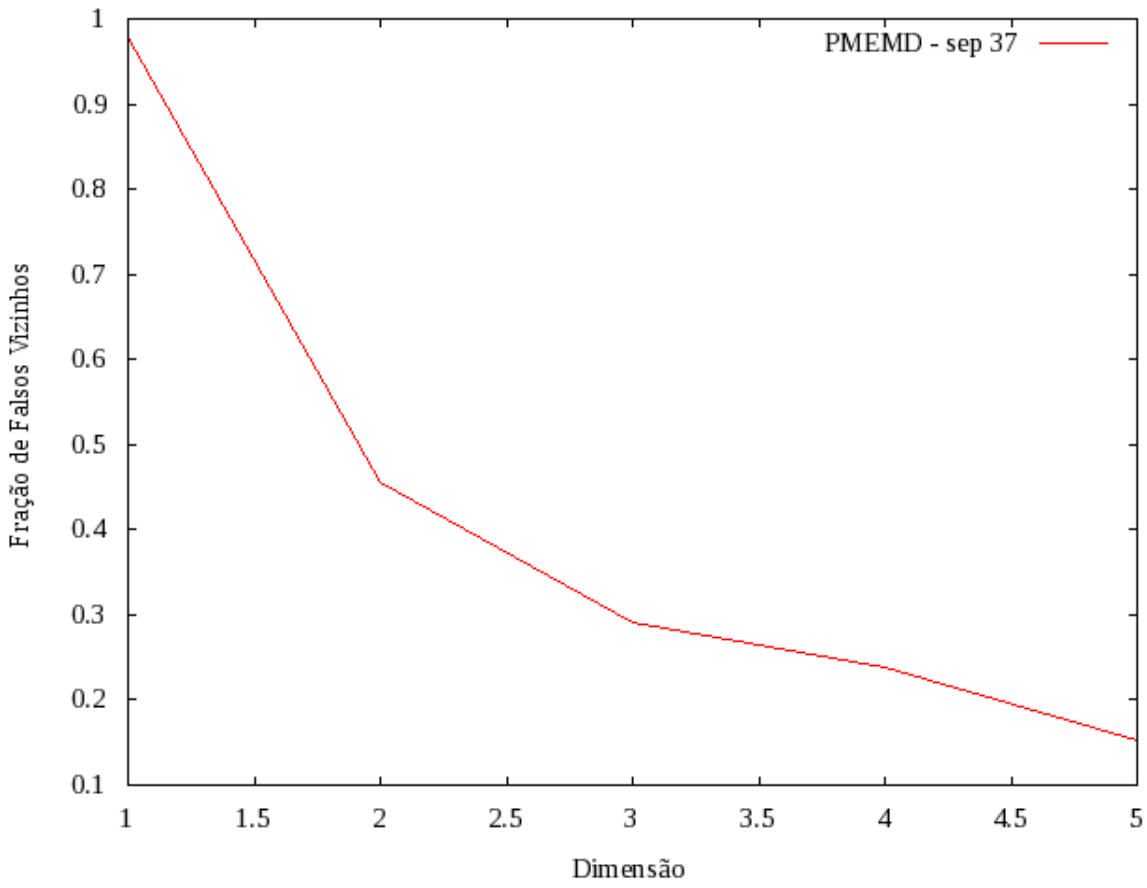


Figura 4.6: PMEMD – dimensão embutida

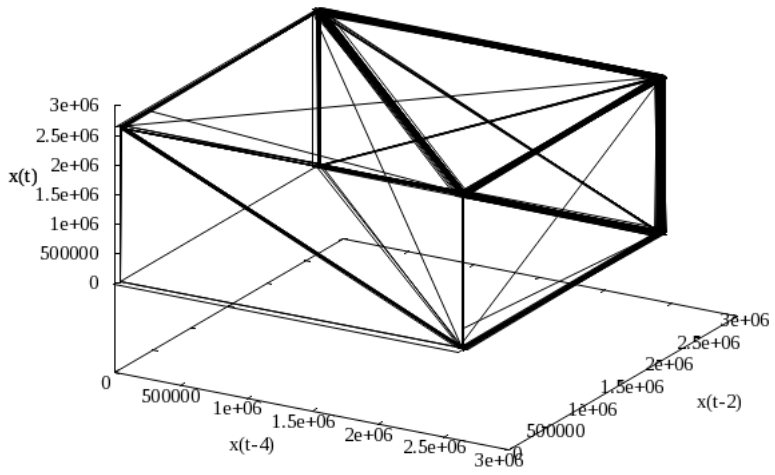


Figura 4.7: GTC – espaço de coordenadas de atraso

4.6.1 Problema de Escalonamento de Processos

Seguindo a formalização de Garey & Johnson (1979), caracteriza-se o problema de otimização de escalonamento de processos como descrito a seguir. Seja o conjunto A de

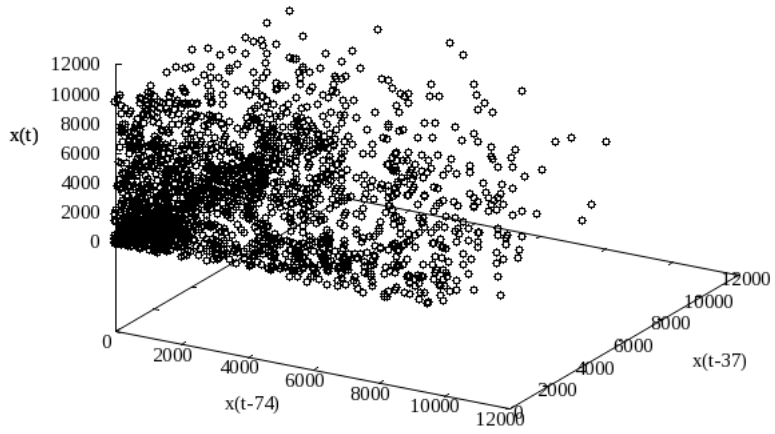


Figura 4.8: PMEMD – espaço de coordenadas de atraso

aplicações paralelas $A = \{a_0, a_1, \ldots, a_{k-1}\}$ e a função $\mathbf{tam}(.)$, que define o número de processos que compõe cada aplicação. Assim, cada uma das k aplicações é constituída por uma diferente quantidade de processos, por exemplo, $\mathbf{tam}(a_0) = 5$, $\mathbf{tam}(a_1) = 8$ etc. Considere, então, que o conjunto P contém todos os processos de todas as k aplicações paralelas. Sendo assim, a quantidade de elementos em P é igual a $|P| = \sum_{i=0}^{k-1} \mathbf{tam}(a_i)$.

Cada processo $p_j \in P$, com $j = 0, 1, \ldots, |P| - 1$, contém características particulares, aqui denominadas comportamento, de utilização dos recursos de processamento, memória, e entrada e saída. Cada processo, portanto, requer quantidades distintas de recursos oferecidos por um conjunto V de computadores do ambiente distribuído (onde $|V|$ define o total de computadores). Além disso, cada computador $v_w \in V$, onde $w = 0, 1, \ldots, |V| - 1$, tem capacidades distintas no que se refere ao poder de processamento, latência de acesso e capacidade de armazenamento da memória primária, vazão para leitura e escrita da memória secundária, e interface de rede com tecnologia própria que define largura de banda, latência e *overhead* para empacotamento de mensagens.

Os computadores em V estão conectados por diferentes redes de comunicação. Todo esse ambiente pode ser modelado por um grafo não dirigido $G = (V, E)$, onde cada vértice representa um computador $v_w \in V$ e os canais de comunicação entre pares de vértices são arcos desse grafo $\{v_w, v_m\} \in E$. Tais canais de comunicação têm propriedades de largura de banda e latência associadas.

O problema de otimização consiste, portanto, em escalonar o conjunto P de processos sobre os vértices do grafo G a fim de minimizar o tempo total de execução de cada aplicação em A. Esse tempo total é caracterizado pelo somatório de custos de operações de processamento, acesso a memória, disco rígido e rede.

Para simplificar o entendimento, considere uma instância do problema com duas aplicações paralelas compostas pelos processos apresentados na tabela 4.4 (MI representa o número de instruções executadas pelos processos em milhões; ML e ME são, respectivamente, o número de Kbytes por segundo lidos e escritos na memória principal; HDL e HDE são, respectivamente, a quantidade em Kbytes por segundo lida e escrita em disco rígido – memória secundária; NETR e NETE são, respectivamente, a quantidade em Kbytes por segundo recebida e enviada na rede de comunicação – nesse caso, também é apresentado o processo origem dos dados para NETR e o destino para NETE).

Tabela 4.4: Comportamento dos processos das aplicações paralelas

Aplicação 0							
Processo	CPU (MI)	ML Kb/s	ME Kb/s	HDL Kb/s	HDE Kb/s	NETR Kb/s	NETE Kb/s
p_0	1.234	123,78	0,00	78,21	0,00	12,50 – p_1	532,12 – p_1
p_1	1.537	23,54	89,45	0,00	12,30	532,12 – p_0	12,50 – p_0
Aplicação 1							
Processo	CPU (MI)	ML Kb/s	ME Kb/s	HDL Kb/s	HDE Kb/s	NETR Kb/s	NETE Kb/s
p_2	1.221	823,78	70,00	78,21	543,00	10,92 – p_3	321,12 – p_4
p_3	1.137	223,54	179,45	324,00	212,31	423,12 – p_4	10,92 – p_2
p_4	2.237	23,54	17,45	12,00	0,00	321,12 – p_2	423,12 – p_3

Tabela 4.5: Características dos computadores

Computador	CPU (MIPS)	ML Kb/s	ME Kb/s	HDL Kb/s	HDE Kb/s
v_0	1.200	100.000	40.000	32.000	17.00
v_1	2.100	120.000	50.000	42.000	19.00
v_2	1.800	100.000	30.000	22.000	9.00
v_3	1.700	95.000	20.000	25.000	11.00
v_4	2.500	110.000	60.000	62.000	30.00

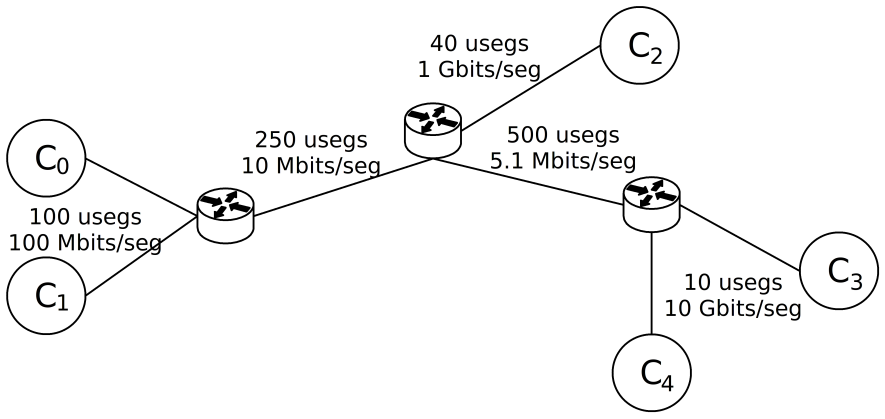


Figura 4.9: Exemplo de rede de interconexão e alocação de processos

Considere que o ambiente é composto pelos cinco computadores descritos na tabela 4.5 (onde MIPS representa a capacidade de processamento em milhões de instruções por segundo; ML e ME são, respectivamente, a vazão de leitura e escrita na memória principal em Kbytes por segundo; HDL e HDE são, respectivamente, a vazão de leitura e escrita

em disco rígido em Kbytes por segundo – memória secundária). Esses computadores, do conjunto V, estão interconectados conforme a figura 4.9, que também apresenta a largura de banda e latência média dos canais de comunicação. Além disso, considere o operador de alocação, ou escalonamento, definido por $\propto$.

O problema de escalonamento consiste em definir sobre qual computador v_w cada processo p_j será alocado considerando capacidades e ocupações de recursos. Um exemplo de solução para essa instância é dada por $p_0 \propto v_0$, $p_1 \propto v_1$, $p_2 \propto v_2$, $p_3 \propto v_3$ e $p_4 \propto v_4$. Dessa maneira, para cada instância do problema deve-se escalonar $|P|$ processos sobre um ambiente composto por $|V|$ computadores, conseqüentemente, o universo de possíveis soluções de escalonamento é igual a $|V|^{|P|}$. Para a instância, anteriormente apresentada, o problema teria um universo de soluções igual a $5^5 = 3.125$. Instâncias reais para o problema podem considerar, por exemplo, 1.024 computadores e 64 aplicações, contendo 512 processos cada. Nesse caso o espaço de soluções seria igual a $1.024^{64 \cdot 512} = 1.024^{32.768}$.

Nesse contexto, observa-se que o espaço de soluções para o problema de escalonamento é exponencial e, deve-se, portanto, propor abordagens capazes de solucionar o problema em tempo computacional aceitável. Não se conhece um algoritmo de tempo polinomial que seja capaz de solucionar o problema de forma ótima (ou seja, encontrar a melhor solução), o que permite caracterizá-lo como intratável (Garey & Johnson, 1979). Dado esse fato é necessário adotar algoritmos que explorem parte do espaço de soluções e encontrem, por meio de suposições e suas avaliações, uma boa candidata em tempo polinomial não determinístico (Garey & Johnson, 1979).

Após caracterizar o problema como intratável é importante compreender o quão difícil é solucioná-lo. Para isso considere o problema NP-completo equivalente estudado por Lageweg & Lenstra (1977), também abordado por Garey & Johnson (1979). Esse problema define o escalonamento de processos sobre m processadores conforme:

Instância: *Seja o conjunto de tarefas T, um número $m \in \mathbb{Z}^+$ de processadores, para cada tarefa $t \in T$ um comprimento $l(t) \in \mathbb{Z}^+$ e um peso $w(t) \in \mathbb{Z}^+$, e um inteiro positivo K.*

Questão: *Há um escalonamento σ sobre m processadores para T, tal que o somatório, para todo $t \in T$, de $(\sigma(t) + l(t)) \cdot w(t)$ não é maior que K.*

De fato: *No contexto deste trabalho, ao invés de considerar m processadores, considera-se m computadores com suas capacidades específicas, e T, ao invés de representar tarefas, representa processos. O comprimento $l(t)$ define a duração do processo, nesse caso, o consumo de recursos. Além disso, deve-se ressaltar que segundo Garey & Johnson (1979) o problema permanece NP-completo para qualquer instância com $m \geq 2$. O problema pode, somente, ser resolvido em tempo polinomial caso as durações de processos sejam*

idênticas, o que não é esperado em ambiente de grade computacional, onde há grande variedade de aplicações para execução.

A melhor solução para o problema de escalonamento seria encontrada por um método exato, que avaliaria todas as possibilidades. Contudo, ele consumiria muitos recursos. Como alternativa a esses métodos, algoritmos de aproximação têm sido propostos Blum & Roli (2003), os quais buscam por soluções satisfatórias em tempo polinomial. Nos últimos anos, alguns algoritmos de aproximação têm sido mais explorados e comumente adotados para solucionar tais problemas, dentre eles estão os algoritmos genéticos (Goldberg, 1989; Semenov & Terkel, 2003), algoritmos baseados em colônias (Bonabeau *et al.*, 2000; Dorigo & Di Caro, 1999) e *Simulated Annealing* (Kirkpatrick *et al.*, 1983).

Para validar a metodologia proposta, esta tese propõe uma política de escalonamento, denominada `PredRoute`, que é baseada na técnica de otimização de algoritmos genéticos (uma descrição sobre essa técnica é apresentada no Apêndice A).

4.6.2 A política de escalonamento PredRoute

Esta seção apresenta a política de escalonamento `PredRoute` que visa encontrar boas soluções utilizando uma metaheurística baseada em algoritmos genéticos e informações preditas sobre o comportamento de eventos de comunicação entre processos. Esta política é composta pelas seguintes etapas:

1. Ao iniciar o sistema, calcula-se a latência de comunicação entre todos computadores do ambiente, determinando uma matriz de atrasos global. Essa matriz é, periodicamente, recalculada e utilizada para avaliar o custo envolvido em transferências (migração) e comunicação entre processos;

2. Processos são inicialmente distribuídos sobre os computadores do ambiente utilizando o algoritmo `RouteGA` (apresentado na seção 2.4.3). Após o início de execução, extrai-se, contínua e transparentemente, operações de comunicação empregando a abordagem de interceptação de eventos utilizada no projeto `MidHPC` (Dodonov *et al.*, 2006; Dodonov & Mello, 2007; Andrade Filho *et al.*, 2008);

3. Assim que um número mínimo de eventos é capturado (definido manualmente ou por meio de uma heurística), a série temporal de comportamentos de comunicação de determinado processo é avaliada, determinando as dimensões de separação e embutida;

4. Inicia-se a montagem da arquitetura da rede neural artificial `RRBF` (seção 3.8) empregada na regressão de séries temporais de comportamento de processos. O primeiro passo consiste em definir o número de neurônios na primeira camada em função da dimensão de separação. Os proponentes dessa rede sugerem avaliar diferentes números de neurônios para essa camada, até obter bons resultados. Contudo,

observou-se que adotando a dimensão de separação, encontra-se, automaticamente, o número ideal de neurônios. Esse fato, resultante de conclusões sobre os experimentos conduzidos nesta tese, deve-se à necessidade da primeira camada ter um número de neurônios capaz de representar a influência de padrões históricos sobre eventos futuros, tal como expresso pela dimensão de separação. Em seguida, agrupa-se esses dados utilizando a rede neural artificial `SONDE` (Albertini & Mello, 2007), proposta no contexto do projeto `MidHPC`. Essa rede utiliza aprendizado contínuo, adaptativo, auto-organizável e não supervisionado, a fim de gerar protótipos, ou centróides, para os agrupamentos de dados, além da dispersão dos padrões agrupados em cada um desses agrupamentos. Esses parâmetros são utilizados para definir os parâmetros (centros e abrangência) dos neurônios da segunda camada da arquitetura `RRBF`, os quais são compostos por funções radiais (tal como uma rede neural artificial `RBF` tradicional);

5. A arquitetura `RRBF` é treinada com os conjuntos de dados de comportamento de processos `GTC` e `PMEMD`. Próximos eventos de comunicação são utilizados para quantificar o erro médio quadrático (*Mean Squared Error* – `MSE`) (Lehmann & Casella, 2003). Assim que o erro de treinamento estiver abaixo de um nível pré-estabelecido (por exemplo, 5%), o treinamento cessa, e a rede prediz os próximos N eventos de comunicação, informando a quantidade de tráfego e os computadores destino das mensagens. Esses resultados são submetidos ao mecanismo de escalonamento;

6. Ao receber dados previstos, o mecanismo de escalonamento avalia possíveis decisões a serem tomadas, empregando um algoritmo genético. Esse mecanismo calcula o custo de processamento e a latência de comunicação entre computadores para cada solução proposta pelo algoritmo genético. O desempenho esperado de processos segundo tal solução é avaliado, permitindo a otimização específica de cada aplicação de usuário. O algoritmo genético empregado, baseado no `RouteGA` (Mello *et al.*, 2007c), define soluções segundo cromossomos na forma $cr = \{g_0, g_1, \ldots, g_{n-1}\}$, onde índices representam processos a serem escalonados e cada termo, ou gene, g_k propõe um computador destino para o processo k. Esse algoritmo utiliza a função de aptidão definida na equação 4.1, onde ic é o índice do cromossomo a ser avaliado, LV_{ic} é a variação de carga dos computadores do ambiente (definida na equação 4.2), MRT_{ic} representa o tempo de resposta do processo com maior custo de execução (equação 4.3), e η é uma constante utilizada para evitar divisões por zero, caso alguma das variáveis seja nula.

$$F_{ic} = \frac{1}{MRT_{ic} + \eta} + \frac{1}{LV_{ic} + \eta} \tag{4.1}$$

$$LV_{ic} = \sum_{j=1}^{i} (C_j - \frac{\sum_{j=1}^{|ic|} C_j}{j})^2 j \tag{4.2}$$

$$MRT_{ic} = max(C_j), j = 0, .., |j| - 1 \tag{4.3}$$

$$C_j = \frac{PM_j}{CAP_d} + CC_j \tag{4.4}$$

$$CC_j = \sum_{w=1}^{|i|} \sum_{k>w}^{|i|} CPS(i_w, i_k) \tag{4.5}$$

$$CPS(i_w, i_k) = \frac{D_{i_w,i_k}}{LB_{i_w,i_k}} \tag{4.6}$$

Nas equações 4.2, 4.3 e 4.4, C_j representa o tempo de execução total da tarefa j, $|ic|$ é o número de termos do cromossomo a ser avaliado, PM_j é o custo total de processamento dessa tarefa, CAP_d representa a capacidade de processamento do computador d, onde o processo j está alocado, CC_j é o custo de comunicação da tarefa j para outras localizadas nos demais computadores, e $CPS(i_w, i_k)$ é o custo de comunicação entre as tarefas w e k executadas, respectivamente, pelos computadores i_w e i_k. $CPS(i_w, i_k)$ é definida na equação 4.6, em que D_{i_w,i_k} representa a quantidade de dados a ser transmitida e LB_{i_w,i_k} a largura de banda entre os computadores que executam as tarefas de índices w e k do cromossomo i.

A principal diferença entre o mecanismo proposto e o algoritmo utilizado no `RouteGA` (Mello *et al.*, 2007c) consiste na utilização de operações preditas de comunicação de acordo com a equação 4.5, as quais consideram a quantidade de dados a serem transferidos e os computadores destino.

Dessa forma, a função de aptidão F_{ic} (equação 4.1) visa a minimização do tempo de resposta de processos com alto custo de execução (MRT_{ic}), e, ao mesmo tempo, busca a menor variação da carga dos computadores do ambiente. Além disso, o algoritmo utiliza informações comportamentais para alocar tarefas em computadores que apresentem latências de comunicação capazes de atender trocas de mensagens. Com isso, atrasos de sincronização, provenientes de comunicação, são reduzidos, o que diminui o tempo total de execução de processos. Após executar o algoritmo genético por diversas gerações, sendo que em cada uma n indivíduos são criados, cruzados e expostos a mutações, ocorre a convergência para uma solução de escalonamento de processos, onde altos valores para a função F_{ic} representam melhores resultados;

7. Processos são transferidos (migrados), conforme a solução proposta pelo algoritmo genético. Em seguida, retoma-se, transparentemente, a execução desses contextos;

8. Novas transferências de processos são periodicamente avaliadas, segundo o modelo de migração proposto por Mello & Senger (2004). Esse mesmo modelo é utilizado pelas políticas `Route` e `RouteGA`.

4.7 Simulações

Predições sobre eventos dos conjuntos de dados `GTC` e `PMEMD`, anteriormente estudados, foram empregadas a fim de avaliar a eficiência da política de escalonamento proposta. Resultados obtidos por meio de simulações utilizando a ferramenta `SchedSim` (Mello & Senger, 2006) foram comparados a demais abordagens da literatura. Em seguida, avaliou-se o custo envolvido na predição de comportamentos de processos.

4.7.1 Avaliação da Eficiência da Política de Escalonamento

Para avaliar a política `PredRoute`, considerou-se dois tipos de ambientes distribuídos:

1. *Cluster* – caracterizado por computadores de configuração e latência de comunicação homogêneas. Dois cenários foram simulados, o primeiro com 32 e o segundo com 128 computadores, apresentando latência de comunicação de $40ms$ (valor extraído em experimentos sobre redes locais (Ferreira & Mello, 2004; Dodonov *et al.*, 2005));

2. Grade – caracterizado por computadores de configuração heterogênea e latência de comunicação variável, a qual foi parametrizada segundo resultados obtidos em (Dodonov *et al.*, 2005) (Latência de redes `LAN`, `MAN` e `WAN` de acordo com funções de distribuição de probabilidades *General Pareto* com valores $k = 0,7104, \theta = 9,9009 \cdot 10^{-6}, \mu = 2,4299 \cdot 10^{-4}$; $k = 0,603, \theta = 0,00323, \mu = 0,03648$ e $k = 0,41876, \theta = 0,00532, \mu = 0,19831$). Dois ambientes de grade compostos, respectivamente, por 256 e 1.024 computadores foram considerados.

As capacidades dos computadores desses ambientes foram parametrizadas segundo resultados de experimentos conduzidos em laboratório (Mello & Senger, 2006). A partir desses resultados definiu-se a função de distribuição de probabilidades Normal para caracterizar capacidades de recursos, utilizando as seguintes médias:

1. `CPU` – 1.500 MIPS (milhões de instruções por segundo);

2. Memória principal – 1.024 MBytes;

3. Memória virtual – 1.024 MBytes;

4. Disco rígido: leitura – 40 MBytes (taxa de transferência para leitura de arquivos do disco rígido);

5. Disco rígido: escrita – 30 MBytes (taxa de transferência para escrita em arquivos do disco rígido).

O algoritmo genético empregado pela política `PredRoute` considerou populações compostas por 50 cromossomos, tendo probabilidade de mutação de 1% e de cruzamento de 70%. Soluções foram obtidas após a execução de 100 gerações desse algoritmo. Adotou-se o método de seleção por *ranking* (Apêndice A).

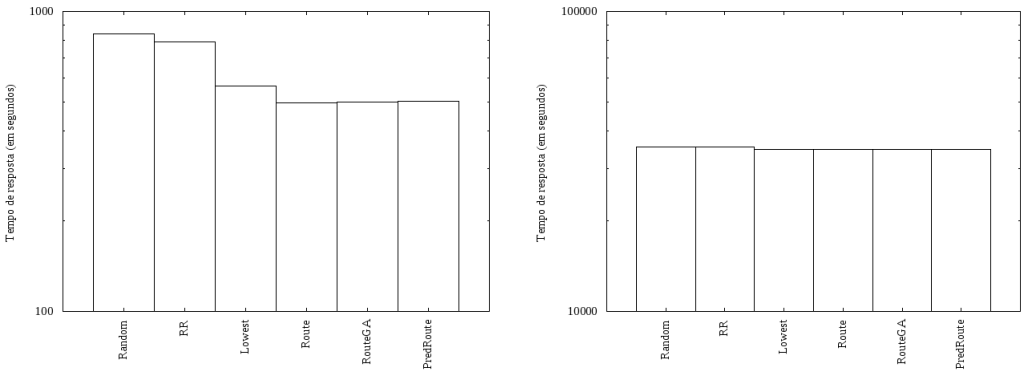
Comparou-se a `PredRoute` a cinco outras políticas: `Random`, `Round-Robin`, `Lowest` (Shivaratri *et al.*, 1992; Zhou & Ferrari, 1987), `Route` (Mello *et al.*, 2006b) e `RouteGA` (Mello *et al.*, 2007c). As três primeiras políticas são amplamente difundidas e freqüentemente adotadas na literatura. `Random` determina, aleatoriamente, a alocação de processos sobre computadores. `Round-Robin` distribui, sucessivamente, cargas sobre computadores organizados segundo uma lista circular. `Lowest` comunica com um subgrupo de computadores e escolhe aquele com maior nível de ociosidade para receber processos.

`Route` e `RouteGA` utilizam estimativas históricas, obtidas por meio de um algoritmo de aprendizado baseado em instâncias, de comportamento de processos a fim de otimizar sua distribuição. O `Route` inicia essa distribuição em uma região do ambiente até estabilizar, eqüitativamente, a carga de computadores. O `RouteGA` estende o anterior, adicionando um algoritmo genético para melhorar o desempenho da etapa de estabilização de cargas.

Simulações foram executadas 30 vezes, o que permite, de acordo com o teorema do limite central, gerar medidas resumo estatísticas representativas (Shefler, 1988). Elas adotaram uma janela de 100 pontos históricos, ou passados, para predizer os próximos 100 eventos de comunicação. Os 100 eventos previstos foram empregados pela política `PredRoute` para escalonar aplicações sobre os ambientes computacionais distribuídos. Tais simulações avaliaram o tempo médio de execução de processos, cujos resultados são apresentados nas figuras 4.10, 4.11, 4.12 e 4.13.

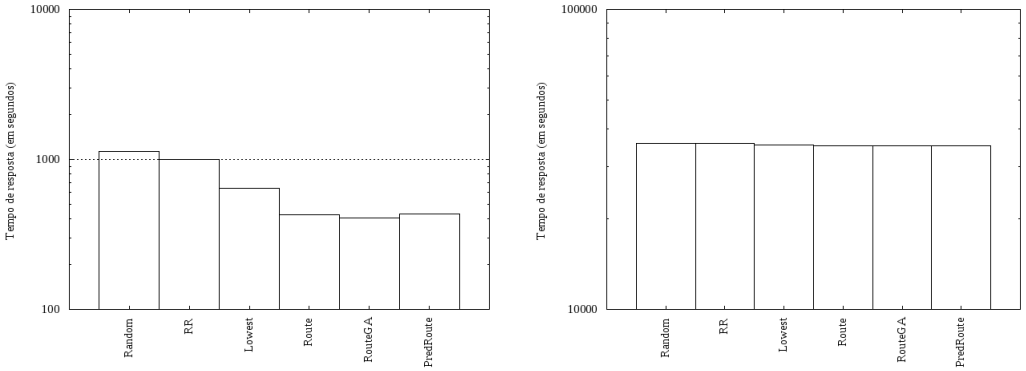
Em *clusters* (figuras 4.10 e 4.11), a política proposta apresentou resultados equivalentes às políticas `Route` e `RouteGA`. Esses resultados devem-se ao fato de latências de comunicação entre computadores serem homogêneas. Isso não permite que a `PredRoute` encontre canais de comunicação que minimizem, ainda mais, custos envolvidos em trocas de mensagens.

No entanto, em ambientes de grade, a política proposta apresentou melhores resultados, superando as demais em até 4 ordens de grandeza, oferecendo, portanto, tempos de execução até 10.000 vezes mais rápidos. Analisando esses resultados, percebe-se que a `PredRoute` foi capaz de predizer a quantidade de dados e os processos destino de mensagens. Essas informações foram empregadas pela política a fim de alocar processos em computadores cuja latência pudesse reduzir custos de comunicação. Dessa forma, os



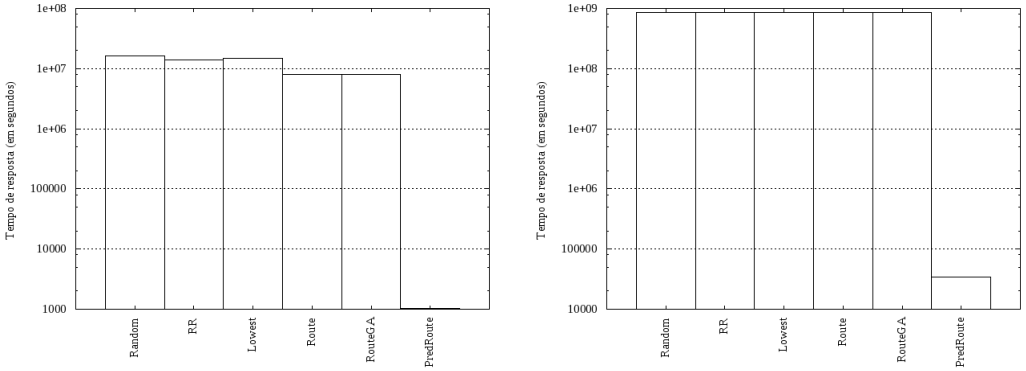
(a) GTC **(b) PMEMD**

Figura 4.10: Escalonamento em *cluster* com 32 computadores



(a) GTC **(b) PMEMD**

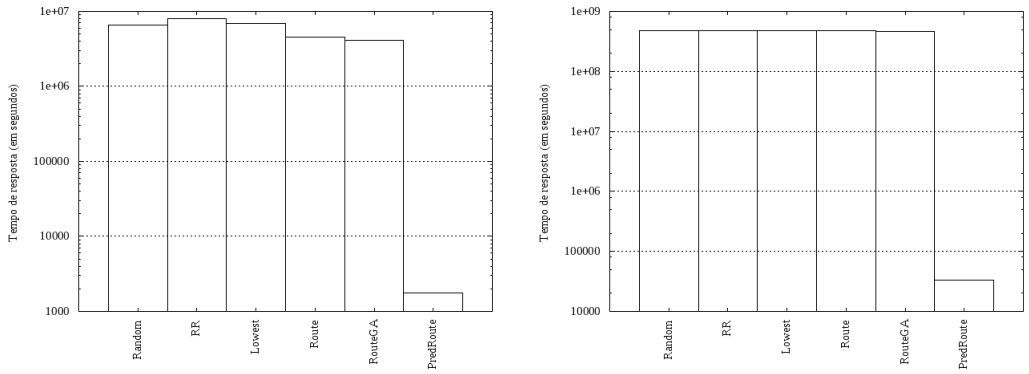
Figura 4.11: Escalonamento em *cluster* com 128 computadores



(a) GTC **(b) PMEMD**

Figura 4.12: Escalonamento em grade com 256 computadores

tempos de execução de ambientes de grade tornaram-se comparáveis aos observados em *clusters*, tal como pode ser observado na tabela resumo 4.6.



(a) GTC **(b) PMEMD**

Figura 4.13: Escalonamento em grade com 1.024 computadores

Tabela 4.6: Resultados de escalonamento

Conjunto	*Random*	*RoundRobin*	*Lowest*	*Route*	*RouteGA*	*PredRoute*
Cluster 32						
GTC	$8,42e+02$	$7,93e+02$	$5,64e+02$	$4,96e+02$	$4,98e+02$	$5,01e+02$
PMEMD	$3,53e+04$	$3,53e+04$	$3,47e+04$	$3,47e+04$	$3,47e+04$	$3,47e+04$
Cluster 128						
GTC	$1,12e+03$	$1,00e+03$	$6,39e+02$	$4,25e+02$	$4,04e+02$	$4,34e+02$
PMEMD	$3,58e+04$	$3,56e+04$	$3,52e+04$	$3,51e+04$	$3,51e+04$	$3,51e+04$
Grid 256						
GTC	$1,62e+07$	$1,39e+07$	$1,50e+07$	$8,03e+06$	$7,89e+06$	$1,01e+03$
PMEMD	$8,65e+08$	$8,61e+08$	$8,67e+08$	$8,58e+08$	$8,53e+08$	$3,37e+04$
Grid 1024						
GTC	$6,61e+06$	$7,95e+06$	$6,90e+06$	$4,59e+06$	$4,07e+06$	$1,76e+03$
PMEMD	$4,75e+08$	$4,73e+08$	$4,73e+08$	$4,77e+08$	$4,68e+08$	$3,36e+04$

4.7.2 Custos da Abordagem de Otimização

As próximas seções apresentam um estudo sobre a ordem de complexidade e os custos, em termos de tempo de execução, da abordagem de otimização de escalonamento de processos.

Ordem de Complexidade

A complexidade da função de aptidão considerada pela política de escalonamento `PredRoute`, definida pela equação 4.1, depende das equações 4.3 e 4.2. A equação 4.2 depende, por sua vez, da equação 4.4, que necessita da equação 4.5. A análise de complexidade deve, portanto, iniciar pelas equações mais simples e resolver as dependências até encontrar a complexidade da equação 4.1 (as equações são definidas na seção 4.6.2).

A equação 4.5 tem ordem de complexidade dada por $O(n^2)$. A equação 4.4 é composta por uma fração e pelo resultado da equação 4.5. Sua complexidade é, portanto, dominada pela equação 4.5, sendo, conseqüentemente, igual a $O(n^2)$. A equação 4.3 deve percorrer todos os termos de C_j para encontrar o valor máximo, portanto, requer um laço de

repetição externo ao cálculo de C_j. Conseqüentemente, a equação 4.3 apresenta ordem de complexidade $O(n^3)$. A equação 4.2 realiza um somatório sobre C_j o que resulta em ordem de complexidade $O(n^3)$, contudo, há outro somatório mais externo que acrescenta complexidade e faz com que a equação 4.2 seja da ordem de $O(n^4)$. A equação 4.1 é dominada pela equação 4.2, dessa forma, sua ordem de complexidade é de $O(n^4)$.

Essas equações são calculadas para cada indivíduo proposto pelo algoritmo genético adotado pela política `PredRoute`. Esse algoritmo propõe n gerações, onde para cada uma z indivíduos são gerados. Dessa maneira, a complexidade do algoritmo genético é de $n \cdot z \cdot O(n^4)$ o que tende, assintoticamente, a um algoritmo de complexidade polinomial da ordem de $O(n^6)$.

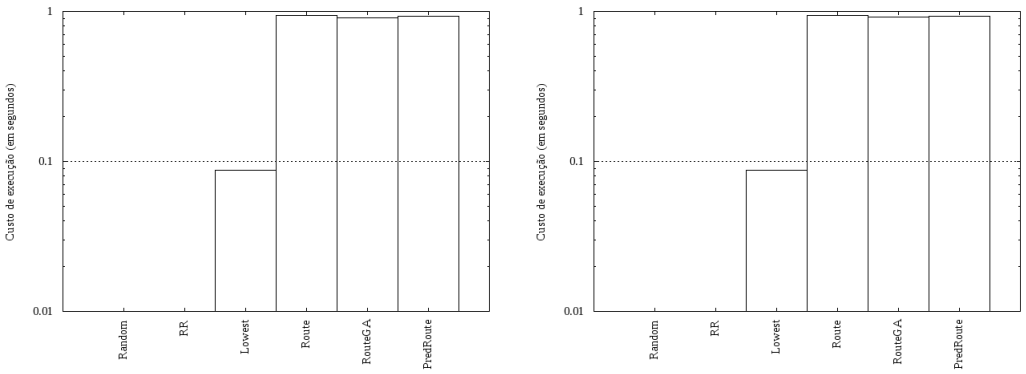
Custos de Execução

As figuras 4.14, 4.15, 4.16 e 4.17 apresentam os tempos, em segundos, consumidos para a execução de cada uma das políticas de escalonamento. Esses custos são reduzidos para ambientes com 32 computadores. *Clusters* com 128 computadores começam a apresentar maior custo, em torno de 21 segundos para a política `RouteGA` e 27 segundos para `Route` e `PredRoute`.

Em ambiente de grade com 256 computadores, os tempos consumidos pelas políticas `Route`, `RouteGA` e `PredRoute` ficam em torno de 2 segundos. Essa queda em relação a *clusters* com 128 máquinas deve-se ao fato que tais políticas consideram uma vizinhança de computadores, em função da latência de rede, na busca por soluções. Dessa forma, reduz-se, significativamente, o espaço de busca por alternativas de escalonamento. A mesma situação é observada em ambientes de grade com 1.024 computadores, onde os tempos consumidos pelas políticas `Route`, `RouteGA` e `PredRoute` foram próximos a 7 segundos.

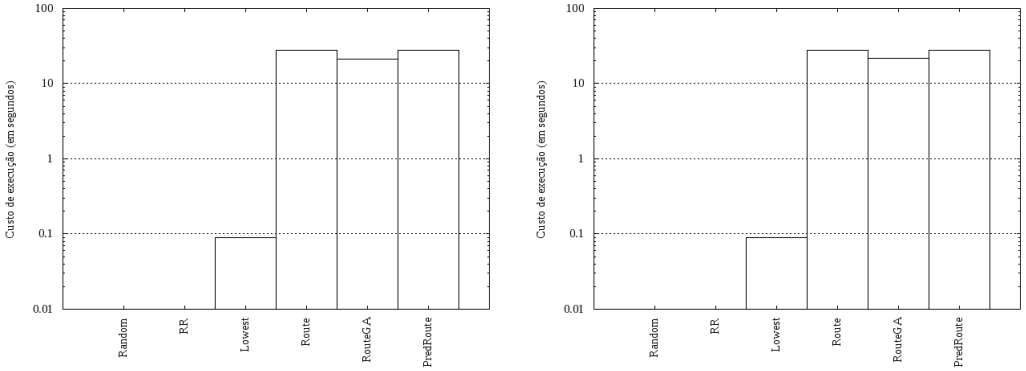
A política `Lowest` apresentou maior consumo de recursos para realizar escalonamentos em ambientes de grade, no entanto seu custo não é significativo em ambientes de *cluster*. O custo das políticas `Random` e `Round-Robin` é zero, pois nenhum parâmetro do ambiente é avaliado na tomada de decisões.

Observa-se que, apesar de apresentar custo superior a algumas abordagens, tais como `Round-Robin` e `Random`, a política `PredRoute` reduz, em algumas ordens de grandeza, o tempo de execução de aplicações em ambientes heterogêneos de larga escala, tais como grades computacionais (seção 4.7.1). Em ambientes de grade com 256 e 1.024 computadores, a política `PredRoute` reduz, significativamente, o tempo de execução de processos. Isso evidencia que a predição de eventos de comunicação auxilia na tomada de decisões de escalonamento.



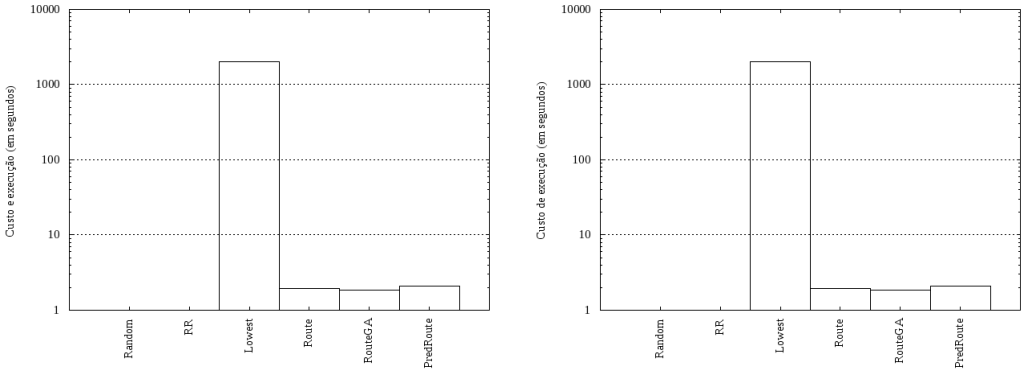
(a) GTC **(b) PMEMD**

Figura 4.14: Custo de execução das políticas de escalonamento em *cluster* com 32 computadores



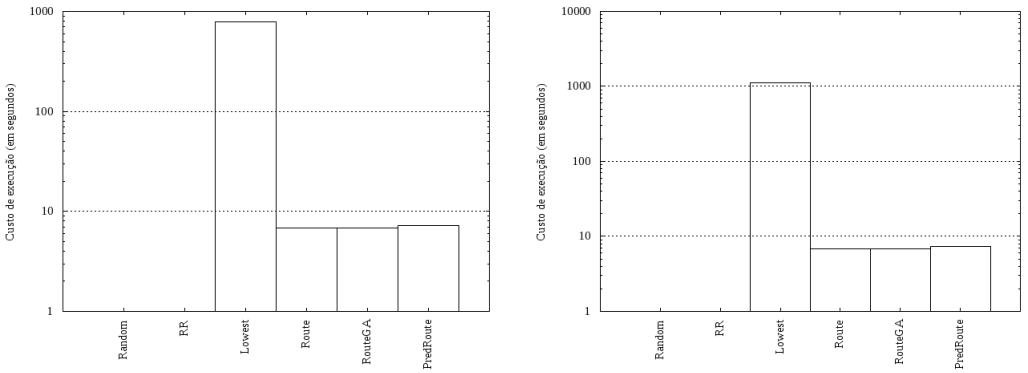
(a) GTC **(b) PMEMD**

Figura 4.15: Custo de execução das políticas de escalonamento em *cluster* com 128 computadores



(a) GTC **(b) PMEMD**

Figura 4.16: Custo de execução das políticas de escalonamento em grade com 256 computadores



(a) GTC **(b) PMEMD**

Figura 4.17: Custo de execução das políticas de escalonamento em grade com 1.024 computadores

4.7.3 Custos da Abordagem de Predição

As próximas seções apresentam um estudo sobre a ordem de complexidade e sobre custos, em termos de tempo de execução, da abordagem de predição de eventos de comunicação.

Ordem de Complexidade

A abordagem de predição é composta pelas etapas de: cálculo da dimensão de separação pela técnica de auto-informação mútua; cálculo da dimensão embutida pela técnica dos falsos vizinhos; reconstrução da série temporal segundo o teorema de imersão de Takens (1980); modelagem da série reconstruída utilizando a rede neural `RRBF`.

Na primeira etapa, aplica-se a equação 3.4 (seção 3.7) para computar a auto-informação mútua. O cálculo de probabilidades dessa equação considera transições entre estados de uma cadeia de Markov (Kennel, 2002). Nesse caso, deve-se calcular as probabilidades de transição de cada estado X para Y, o que apresenta ordem de complexidade igual a $O(n^2)$. Além disso, a equação calcula a integral desses valores o que aumenta a complexidade total da equação para $O(n^3)$.

A segunda etapa consiste no cálculo da dimensão embutida por meio da técnica de falsos vizinhos. Essa técnica encontra, primeiramente, os vizinhos mais próximos para cada ponto. Dessa forma, deve-se avaliar a distância de cada ponto p em relação aos demais, o que apresenta ordem de complexidade $O(n^2)$. Essas distâncias são calculadas para cada dimensão d, o que aumenta a complexidade total da técnica para $O(n^3)$.

Na terceira etapa, a série é reconstruída segundo o teorema de imersão de Takens (1980), que apresenta ordem de complexidade de $O(n^2)$, dado que para cada valor de entrada deve-se encontrar m outros segundo um deslocamento τ no tempo.

Em seguida, a série reconstruída é modelada pela rede neural `RRBF`, responsável por

predições de eventos. Essa rede é composta por uma etapa de treinamento onde são definidos parâmetros para as funções $\phi(.)$ de sua segunda camada (seção 3.8). Esses parâmetros são encontrados após a execução da rede neural `SONDE`, que apresenta ordem de complexidade $O(n^2)$. Em seguida, define-se as funções $\phi(.)$ e inicia-se o treinamento da segunda camada da rede neural `RRBF`, a qual calcula a equação 3.10 (seção 3.8) como forma de configurar pesos de conexões entre neurônios. Essa etapa realiza uma multiplicação de matrizes para cada termo modelado da série de entrada, o que resulta em complexidade $O(n^3)$.

Por meio dessa análise, observa-se que a complexidade total da etapa de predição é da ordem de $O(n^3)$.

Custos de Execução

A abordagem proposta foi avaliada com o objetivo de determinar os custos envolvidos na predição de eventos. Para isso, contabilizou-se os tempos necessários para determinar as dimensões de separação e embutida, realizar o agrupamento de dados utilizando a rede neural `SONDE`, treinar o modelo e predizer eventos. Eventos históricos foram enviados para a abordagem na forma de vetores no formato *{Bytes trafegados, ID do computador de destino}*, em seguida, submeteu-se um número inteiro representando a quantidade de eventos a serem previstos. Após predição, dados resultantes foram retornados para o mecanismo de escalonamento.

O modelo de predição foi implementado utilizando a linguagem `Python 2.5` e as aplicações *false_ nearest* e *mutual* do pacote de análise de séries não lineares `TISEAN 3.0.1` (Hegger *et al.*, 2009). Essas duas aplicações foram utilizadas para calcular, respectivamente, as dimensões embutida e de separação das séries em estudo. Os experimentos foram executados 30 vezes em um computador Intel Core2Duo T7250 com 3GB de memória RAM (menos de 10MB de RAM foram utilizados durante os experimentos) sobre o *kernel* do sistema operacional Linux versão 2.6.25. Essas repetidas execuções permitem estatísticas mais representativas (teorema do limite central) (Shefler, 1988).

A figura 4.18 apresenta os custos médios envolvidos nas etapas de cálculo das dimensões de separação e embutida, agrupamento pela rede neural `SONDE` (essas etapas são aqui denominadas pré-processamento) e treinamento do modelo de predição para um conjunto de dados com 5.000 pontos. Esse conjunto representa o tamanho total médio do `GTC` e do `PMEMD`. Quanto maior o tamanho desse conjunto, maior o custo para pré-processar e treinar o modelo.

Outros experimentos foram realizados para avaliar o custo envolvido na predição de operações de comunicação. Nessa situação, treinou-se o modelo com 100 pontos históricos, assim como nos experimentos que comparam a nova abordagem a outras existentes[7],

[7]Esse fato comprova que 100 pontos históricos foram suficientes para que a abordagem proposta obtivesse bons resultados em relação a outras existentes.

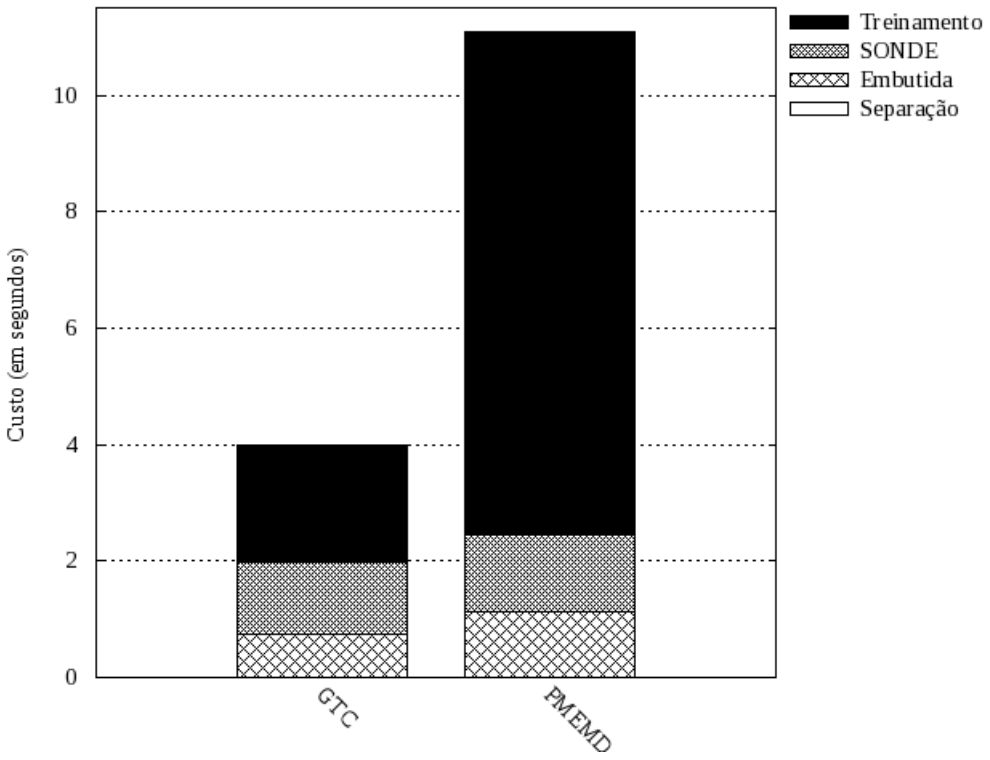


Figura 4.18: Custo computacional envolvido em pré-processamento e treinamento

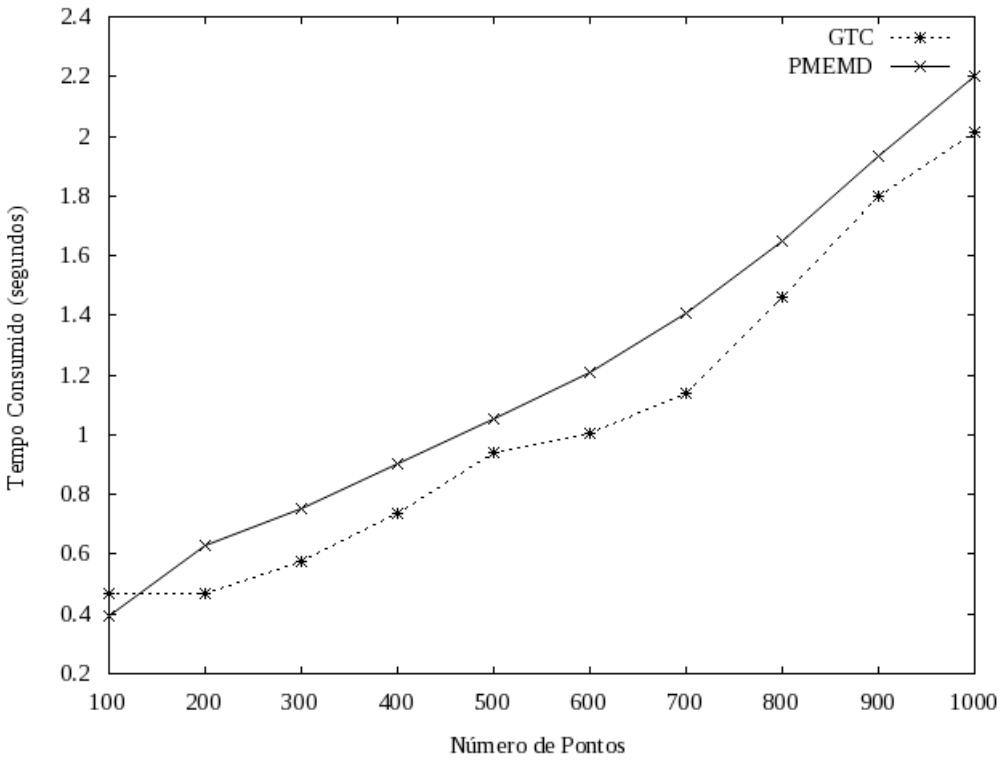


Figura 4.19: Custo computacional de operações de predição

para predizer os próximos 100 a 1.000 pontos (adotando intervalos de 100 pontos). Os resultados para os conjuntos de dados `GTC` e `PMEMD` são apresentados nas figuras 4.19(a)

e 4.19(b). Observou-se que a abordagem proposta apresenta custo máximo igual a 2 segundos para a predição de eventos, além de tendência linear.

Tanto os custos envolvimento no pré-processamento e treinamento do modelo, quanto aqueles referentes à predição de eventos, são baixos, principalmente, para ambientes de grade computacional, onde a melhor alocação de processos pode reduzir, em várias ordens de grandeza (tabela 4.6), o tempo de execução de processos. Conseqüentemente, esses custos não tornam proibitiva a adoção dessa abordagem em ambientes reais.

4.8 Considerações finais

Este capítulo apresentou o objetivo, metodologia e a aplicação dos estudos conduzidos a fim de modelar comportamentos de comunicação entre processos. Em seguida, uma política de escalonamento, baseada na `RouteGA` (Mello *et al.*, 2007c), foi proposta com o objetivo de validar os benefícios relativos à predição de eventos de comunicação. Essa política, denominada `PredRoute`, foi avaliada por meio de simulações em ambientes de *cluster* e grade computacional. Os resultados comprovam o impacto positivo de tal abordagem no desempenho de ambientes heterogêneos de larga escala.

Capítulo

5

Conclusões e Trabalhos Futuros

A escassez de pesquisas relacionadas ao estudo, projeto e avaliação de técnicas e modelos a fim de prover funcionalidades autônomas para grades motivou o projeto `MidHPC`, que investiga aspectos de otimização de escalonamento, distribuição de dados, rebalanceamento de cargas, configuração de serviços, segurança, e tolerância a falhas, a fim de prover autonomia para grades computacionais.

Diversas pesquisas foram conduzidas no contexto desse projeto, as principais focaram em aspectos de otimização de escalonamento adotando conhecimento sobre operações executadas por processos. Esses conhecimentos são provenientes de médias históricas de ocupação de `CPU`, memória, disco e rede. Essas informações são submetidas a diferentes otimizadores capazes de selecionar o melhor conjunto de computadores para receber novas execuções. Resultados obtidos comprovam que a utilização de conhecimento minimiza tempos de execução de aplicações, ou seja, maximiza o desempenho do sistema.

Por outro lado, médias históricas não evidenciam as variações comportamentais de utilização de recursos, o que limita ferramentas de otimização a escolherem soluções que, na medida do possível, melhoram o desempenho global das aplicações. Essas soluções podem privilegiar, por exemplo, a ocupação de processamento em detrimento de comunicação via rede, o que tende a aumentar os tempos de execução de aplicações. Melhores soluções seriam encontradas ao compreender a variação de comportamento de processos no tempo e adaptar o escalonamento, antecipando a necessidade de rebalanceamento de cargas em função das operações executadas. Para isso, deve-se detectar e compreender a dinâmica de padrões repetitivos envolvidos nas séries temporais de comportamento de processos.

Essa compreensão motivou esta tese de livre docência que emprega conceitos sobre sistemas dinâmicos, em conjunto com técnicas inteligentes, para modelar e predizer o

comportamento de processos com o objetivo de otimizar operações de escalonamento em ambientes de grade computacional.

Estudos realizados consideraram conjuntos de dados sobre comportamento de comunicação entre processos. Operações preditas foram empregadas por uma nova política de escalonamento, denominada `PredRoute`, que otimiza a distribuição de cargas com o objetivo de minimizar custos de processamento e comunicação. Simulações realizadas avaliaram o desempenho de ambientes de *cluster* e grade computacional adotando tal política. Observou-se que, em *clusters*, a nova abordagem gera resultados similares às demais propostas na literatura. Avaliações conduzidas em ambientes de grade computacional comprovaram ótimos resultados em relação a demais políticas.

Esses resultados se devem ao fato da `PredRoute` otimizar a alocação de processos em função dos canais de comunicação que oferecem menor latência. No caso de *clusters*, canais de latência homogêneos foram considerados, portanto, não há reduções significativas no tempo de execução de processos. Em ambientes de grade, onde latências variam, a nova política consegue encontrar melhores regiões para receber processos, de acordo com seus comportamentos de comunicação. Além de reduzir o tempo de execução de processos, a abordagem proposta apresenta baixo custo computacional, sendo, portanto, recomendada para ambientes de grade.

Esses resultados motivam trabalhos futuros a explorarem aspectos de estabilidade de pontos fixos, órbitas e tendências de sistemas dinâmicos com o objetivo de abordar outros aspectos da computação autônoma em grades. Além disso, esses resultados já têm direcionado algumas pesquisas de mestrado e doutorado. Nesse contexto, tem-se explorado técnicas de parametrização adaptativa, por meio do expoente de Lyapunov, para a rede neural artificial `SONDE`, e emprego de predições para otimizar a distribuição de dados em ambientes de grade computacional.

Apêndice

A

Algoritmos Genéticos

Algoritmos genéticos são aplicados como técnicas de busca e otimização em diversas áreas. Eles são baseados no mecanismo de seleção natural, focando na sobrevivência do indivíduo mais apto. Esses algoritmos nem sempre encontram a melhor solução possível, porém, fornecem boas soluções locais para problemas NP-completos (Papadimitriou, 1994).

A solução de problemas utilizando algoritmos genéticos envolve dois aspectos: codificação da solução na forma de cromossomos, onde cada cromossomo representa uma possível solução, e uma função de aptidão (*fitness*) que é aplicada para encontrar a melhor solução.

Várias técnicas de codificação podem ser utilizadas para diferentes tipos de problemas, como *strings*, binárias, *bitmaps*, números reais dentre outras. A função de aptidão é responsável por avaliar as possíveis soluções. Essa função recebe um cromossomo como parâmetro e retorna um número real que representa a qualidade da solução obtida, por exemplo, o quão adequada é a solução para o problema em estudo.

Cromossomos mais aptos são identificados e armazenados durante o processo de evolução. Os menos aptos, por outro lado, são eliminados. Diferentes técnicas podem ser aplicadas para a identificação dos melhores cromossomos, como a seleção proporcional, seleção por *ranking* e a seleção baseada em torneio (Back *et al.*, 1999).

Na seleção proporcional, indivíduos são transferidos para a próxima geração de acordo com o valor proporcional ao resultado de sua função de aptidão. Uma das possíveis implementações dessa técnica consiste no uso de uma roleta, dividida em N partes, com N sendo o número de indivíduos (cromossomos) da população atual. O tamanho de cada parte é proporcional ao valor da função de aptidão de cada indivíduo. A roleta é girada N vezes e, a cada turno, o indivíduo apontado é selecionado e inserido na próxima geração.

A seleção baseada em *ranking* pode ser dividida em duas etapas. Durante a primeira, as soluções são ordenadas de acordo com seu valor da função de aptidão. Com a lista ordenada, cada indivíduo recebe um novo valor de aptidão de acordo com sua posição no *ranking*. Em seguida, aplica-se um procedimento que seleciona os indivíduos conforme sua posição no *ranking*. Assim, os indivíduos de melhores posições têm maiores chances de serem selecionados.

Na seleção baseada em torneio não se atribui automaticamente probabilidades a indivíduos. Um torneio de tamanho k é definido, com $k \geq 2$ indivíduos. Então, k indivíduos são aleatoriamente escolhidos na população atual, e seus valores de aptidão são comparados e o indivíduo com maior aptidão é selecionado para reprodução. O valor de k é definido pelo usuário, representando a pressão de seleção, isto é, a velocidade com a qual os indivíduos mais aptos vão dominar a população, eliminando os menos adaptados.

Uma vez selecionados os indivíduos para reprodução, é necessário modificar suas características genéticas usando técnicas de reprodução conhecidas como operadores genéticos. Os operadores mais comuns são o cruzamento e a mutação. O operador de cruzamento permite a troca de material genético entre dois indivíduos, conhecidos como pais, combinando suas informações de modo a aumentar a possibilidade de gerar um novo indivíduo com melhores características que os originais (Hinterding, 2000). O operador de mutação é utilizado para alterar um único gene por um valor aleatório. Quando um indivíduo é representado por um *bitmap*, esse operador escolhe de forma aleatória um gene do cromossomo e troca seu valor de 1 para 0 e vice-versa. O objetivo do operador de mutação é manter a diversidade da população, sempre permitindo que um cromossomo cubra um amplo espaço de busca (Hinterding, 2000). Esse operador é geralmente aplicado com baixa probabilidade, pois, com alta, o resultado tende a ser aleatório.

Referências

Abarbanel, H. D. I.; Brown, R.; Sidorowich, J. J.; Tsimring, L. S. (1993). The analysis of observed chaotic data in physical systems. *Reviews of Modern Physics*, v.65, p.1331–1392.

Abawajy, J. H. (2005). Autonomic job scheduling policy for grid computing. *International Conference on Computational Science (3)*, p. 213–220.

Abraham, A.; Buyya, R.; Nath, B. (2000). Nature's heuristics for scheduling jobs on computational grids. *8th IEEE International Conference on Advanced Computing and Communications (ADCOM 2000)*, Índia.

Achelis, S. B. (2000). *Technical Analysis from A to Z.* McGraw-Hill, 2 edição.

Adve, S. V.; Gharachorloo, K. (1996). Shared memory consistency models: A tutorial. *IEEE Computer*, v.29, n.12, p.66–76.

Aha, D. W.; Kibler, D.; Albert, M. K. (1991). Instance-based learning algorithms. *Machine Learning*, v.6, n.1, p.37–66.

Albertini, M. K.; Mello, R. F. (2007). A self-organizing neural network for detecting novelties. *SAC '07: Proceedings of the 2007 ACM symposium on Applied computing*, p. 462–466, Nova Iorque, EUA. ACM.

Alligood, K. T.; Sauer, T. D.; Yorke, J. A. (1997). *Chaos: An Introduction to Dynamical Systems.* Springer.

Amir, Y. (2000). An opportunity cost approach for job assignment in a scalable computing cluster. *IEEE Transactions on Parallel and Distributed Systems*, v.11, n.7, p.760–768.

Andrade Filho, J. A.; Mello, R. F.; Dodonov, E.; Senger, L. J.; Yang, L. T.; Li, K.-C. (2008). Toward an efficient middleware for multithreaded applications in computational

grid. *CSE '08: Proceedings of the 2008 11th IEEE International Conference on Computational Science and Engineering*, p. 147–154, Washington, DC, EUA. IEEE Computer Society.

Back, T.; Fogel, D. B.; Michalewicz, Z., editores (1999). *Basic Algorithms and Operators*. IOP Publishing Ltd., Bristol, Reino Unido.

Bianchi, G.; Tinnirello, I. (2003). Kalman filter estimation of the number of competing terminals in an ieee 802.11 network. *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies. IEEE*, v. 2, p. 844–852 vol.2.

Blackham, B. (2008). Cryopid, disponível em: http://cryopid.berlios.de. acesso em: 20 outubro 2008.

Blum, C.; Roli, A. (2003). Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Comput. Surv.*, v.35, n.3, p.268–308.

Bonabeau, E.; Dorigo, M.; Theraulaz, G. (2000). Inspiration for optimization from social insect behaviour. *Nature*, v.406, n.6791, p.39–42.

Box, G.; Jenkins, G. M.; Reinsel, G. (1994). *Time Series Analysis: Forecasting & Control (3rd Edition)*. Prentice Hall.

Brecht, T.; Guha, K. (1996). Using Parallel Program Characteristics in Dynamic Processor Allocation Policies. *Performance Evaluation*, v.27/28, n.4, p.519–539.

Bretscher, O. (2004). *Linear Algebra with Applications*. Prentice Hall.

Buhmann, M. D. (2003). *Radial Basis Functions: Theory and Implementations*. Cambridge University Press.

Carpenter, G. A.; Grossberg, S. (1989). ART 2: Self-organization of Stable Category Recognition Codes for Analog Input Patterns. *Applied Optics*, v.26, n.23, p.4919–4930.

Casdagli, M. (1989). Nonlinear prediction of chaotic time series. *Physica D Nonlinear Phenomena*, v.35, p.335–356.

Champrasert, P.; Itao, T.; Suzuki, J. (2005). Symbioticsphere: a biologically-inspired network architecture for autonomic grid computing. v. 2, p. 1395–1404.

Champrasert, P.; Suzuki, J. (2006a). A biologically-inspired autonomic architecture for self-healing data centers. *COMPSAC '06: Proceedings of the 30th Annual International Computer Software and Applications Conference (COMPSAC'06)*, p. 103–112, Washington, DC, EUA. IEEE Computer Society.

Champrasert, P.; Suzuki, J. (2006b). Towards green grids: A biologically-inspired adaptive architecture for power efficient server farms. *ICAS '06: Proceedings of the International Conference on Autonomic and Autonomous Systems*, p. 39, Washington, DC, EUA. IEEE Computer Society.

Champrasert, P.; Suzuki, J. (2007). Building self-configuring data centers with cross layer coevolution. *Journal of Software*, v.2, n.5, p.29–43.

Cheng, S.-W.; Garlan, D.; Schmerl, B.; Steenkiste, P.; Hu, N. (2002). Software architecture-based adaptation for grid computing. *High Performance Distributed Computing, 2002. HPDC-11 2002. Proceedings. 11th IEEE International Symposium on*, v., p.389–398.

Chodnekar, S.; Srinivasan, V.; Vaidya, A. S.; Sivasubramaniam, A.; Das, C. R. (1997). Towards a communication characterization methodology for parallel applications. *Proceedings of the 3rd IEEE Symposium on High-Performance Computer Architecture (HPCA '97)*, p. 310–321. IEEE Computer Society.

Clark, J. D. (1990). Modeling and simulating complex spatial dynamic systems: a framework for application in environmental analysis. *SIGSIM Simul. Dig.*, v.21, n.2, p.9–19.

Culler, D.; Karp, R.; Patterson, D.; Sahay, A.; Schauser, K. E.; Santos, E.; Subramonian, R.; von Eicken, T. (1993). Logp: towards a realistic model of parallel computation. *Proceedings of the fourth ACM SIGPLAN symposium on Principles and practice of parallel programming*, p. 1–12, San Diego, California, United States. ACM Press.

Danelutto, M.; Dazzi, P. (2006). Joint Structured/Unstructured Parallelism Exploitation in Muskel. Alexandrov, V.; van Albada, D.; Sloot, P.; Dongarra, J., editores, *International Conference on Computational Science (ICCS 2006)*, LNCS. Springer.

Devarakonda, M. V.; Iyer, R. K. (1989). Predictability of Process Resource Usage: A Measurement-based Study on UNIX. *IEEE Transactions on Software Engineering*, v.15, n.12, p.1579–1586.

Doblinger, G. (1998). An adaptive kalman filter for the enhancement of noisy ar signals. *Circuits and Systems, 1998. ISCAS '98. Proceedings of the 1998 IEEE International Symposium on*, v.5, p.305–308.

Dodonov, E.; Mello, R. F. (2007). A model for automatic on-line process behavior extraction, classification and prediction in heterogeneous distributed systems. *Seventh IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2007)*, p. 899–904. IEEE Computer Society.

Dodonov, E.; Mello, R. F. (2008). Estudo sobre abordagens de extração, classificação e predição de comportamento de processos. Relatório Técnico 322, Instituto de Ciências Matemáticas e de Computação, USP, São Carlos, SP, Brazil.

Dodonov, E.; Mello, R. F.; Yang, L. T. (2005). A network evaluation for LAN, MAN and WAN grid environments. *EUC*, v. 3824 de *Lecture Notes in Computer Science*, p. 1133–1146. Springer.

Dodonov, E.; Mello, R. F.; Yang, L. T. (2006). Adaptive technique for automatic communication access pattern discovery applied to data prefetching in distributed applications using neural networks and stochastic models. *The International Symposium on Parallel and Distributed Processing and Application – ISPA*, p. 292–303.

Dodonov, E.; Sousa, J. Q.; Guardia, H. C. (2004). Gridbox: securing hosts from malicious and greedy applications. *Proceedings of the 2nd workshop on Middleware for grid computing*, p. 17–22, Nova Iorque. ACM Press.

Dorigo, M.; Di Caro, G. (1999). The ant colony optimization meta-heuristic. Corne, D.; Dorigo, M.; Glover, F., editores, *New Ideas in Optimization*, p. 11–32. McGraw-Hill, Londres.

Downey, A. B. (1997). Predicting queue times on space-sharing parallel computers. *11th International Parallel Processing Symposium*. Disponível como University of California technical report number CSD-96-906.

Dubie, D. (2003). *IBM grabs Think Dynamics, NetworkWorld.com – Disponível em: http://www.networkworld.com/news/2003/0514ibmthink.html. Acesso em: 10 de novembro de 2008.*

Eckmann, J.-P.; Ruelle, D. (1985). Ergodic theory of chaos and strange attractors. *Reviews of Modern Physics*, v.57, p.617–656.

Edmonds, A. N. (1996). *Time Series Prediction Using Supervised Learning and Tools from Chaos Theory*. Tese (Doutorado), University of Luton.

Elert, G. (2005). *The Chaos Hypertextbook: Measuring Chaos – Disponível em: http://hypertextbook.com/chaos/. Acesso em 10 de setembro de 2007.*

Elmer, F.-J. (1998). The lyapunov exponent – disponível em: http://monet.unibas.ch/ elmer/pendulum/lyapexp.htm. acesso em 10 de setembro de 2007.

Feitelson, D. G.; Nitzberg, B. (1995). Job characteristics of a production parallel scientific workload on the NASA ames iPSC/860. Feitelson, D. G.; Rudolph, L., editores, *Job Scheduling Strategies for Parallel Processing – IPPS'95 Workshop*, v. 949, p. 337–360. Springer.

Feitelson, D. G.; Rudolph, L.; Schwiegelshohn, U.; Sevcik, K. C.; Wong, P. (1997). Theory and Practice in Parallel Job Scheduling. *Job Scheduling Strategies for Parallel Processing*, v. 1291, p. 1–34. LNCS 1291.

Ferreira, E. A.; Mello, R. F. (2004). Avaliação de desempenho de protocolos e bibliotecas de comunicação entre processos. *International Information and Telecommunication Technologies Symposium*, p. 8.

Foster, J.; Subramanian, K.; Herring, R.; Ahn, G. (2004). Interactive exploration of the afs file system. *INFOVIS '04: Proceedings of the IEEE Symposium on Information Visualization*, p. 215.7, Washington, DC, EUA. IEEE Computer Society.

Fraser, A. M.; Swinney, H. L. (1986). Independent coordinates for strange attractors from mutual information. *Phys. Rev. A*, v.33, n.2, p.1134–1140.

Freeman, J. A.; Skapura, D. M. (1991). *Neural networks: algorithms, applications, and programming techniques*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, EUA.

Garey, M. R.; Johnson, D. S. (1979). *Computers and Intractability : A Guide to the Theory of NP-Completeness*. Series of Books in the Mathematical Sciences. W. H. Freeman.

Gibbs, W. W. (2002). *Autonomic Computing, Scientific American – Disponível em: http://www.sciam.com/article.cfm?id=autonomic-computing. Acesso em: 10 de outubro de 2008.*

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, EUA.

Han, S.-S.; May, G. S. (1996). Optimization of neural network structure and learning parameters using genetic algorithms. *ICTAI '96: Proceedings of the 8th International Conference on Tools with Artificial Intelligence (ICTAI '96)*, p. 200, Washington, DC, EUA. IEEE Computer Society.

Harchol-Balter, M.; Downey, A. B. (1997). Exploiting Process Lifetimes Distributions for Dynamic Load Balancing. *ACM Transactions on Computer Systems*, v.15, n.3, p.253–285.

Hartigan, J. A.; Wong, M. A. (1979). A K-means clustering algorithm. *Applied Statistics*, v.28, p.100–108.

Haykin, S. (2008). *Neural Networks and Learning Machines*. Prentice-Hall, 3 edição.

Hegger, R.; Kantz, H.; Schreiber, T. (2009). Tisean 3.0.1: Nonlinear time series analysis. Disponível em: http://www.mpipks-dresden.mpg.de/~tisean/Tisean_3.0.1/index.html. Acesso em: 26 de janeiro de 2009.

Hinterding, R. (2000). Representation, mutation and crossover issues in evolutionary computation. *Proc. of the 2000 Congress on Evolutionary Computation*, p. 916–923, Piscataway, NJ. IEEE Service Center.

Hockney, R. W. (1996). *The Science of Computer Benchmarking*. Soc for Industrial & Applied Math.

Hoskins, M. E. (2006). Sshfs: super easy file access over ssh. *Linux Journal*, v.2006, n.146, p.4.

Hussain, A.; Heidemann, J.; Papadopoulos, C. (2003). A framework for classifying denial of service attacks. *SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, p. 99–110, Nova Iorque, NY, EUA. ACM.

IEEE (1993). *9945-2: 1993 (ISO/IEC) [IEEE/ANSI Std 1003.2-1992 and IEEE/ANSI 1003.2a-1992] Information Technology-Portable Operating System Interface (POSIX) — Part 2: Shell and Utilities*. IEEE CS.

Intel (2008). Proactive computing – disponível em: http://www.intel.com/cd/corporate/techtrends/emea/eng/209579.htm. acesso em: 19 de novembro de 2008.

Ishii, R. P.; Mello, R. F.; Senger, L. J.; Santana, M. J.; Santana, R. H. C.; Yang, L. T. (2005). Improving scheduling of communication intensive parallel applications on heterogeneous computing environments. *Parallel Processing Letters*, v.15, n.4, p.423–438.

Jackson, E. A. (1989). *Perspectives of Nonlinear Dynamics*. Cambridge University Press, Cambridge.

Kantz, H. (1994). A robust method to estimate the maximal Lyapunov exponent of a time series. *Physics Letters A*, v.185, p.77–87.

Kaplan, I. (2003). Estimating the hurst exponent – disponível em http://www.bearcave.com/misl/misl_tech/wavelets/hurst/index.html. acesso em 10 de setembro de 2007.

Kennel, M. (2002). The multiple-dimensions mutual information program – disponível em: http://www-ncsl.postech.ac.kr/en/softwares/archives/mmi.tar.z. acesso em: 10 de setembro de 2007.

Kennel, M.; Brown, R.; Abarbanel, H. (1992a). Determining embedding dimension for phase space reconstruction using the method of false nearest neighbors. Relatório técnico, Institute for Nonlinear Science and Department of Physics, University of California, San Diego, Mail Code R-002, La Jolla, CA, EUA.

Kennel, M. B.; Brown, R.; Abarbanel, H. D. I. (1992b). Determining embedding dimension for phase-space reconstruction using a geometrical construction. *Phys. Rev. A*, v.45, n.6, p.3403–3411.

King, R. L.; Birk, R. J. (2004). Developing earth system science knowledge to manage earth's natural resources. *Computing in Science and Engineering*, v.6, n.1, p.45–51.

Kirkpatrick, S.; Gelatt, C. D.; Vecchi, M. P. (1983). Optimization by simulated annealing. *Science, Number 4598, 13 Maio 1983*, v.220, 4598, p.671–680.

Kohler, M. (1997). Using the kalman filter to track human interactive motion - modelling and initialization of the kalman filter for translational motion. Relatório Técnico 629/1997, Fachbereich Informatik, Universität Dortmund, Fachbereich Informatik, Universität Dortmund, 44221 Dortmund, Alemanha.

Krishnaswamy, S.; Zaslavsky, A.; Loke, S. W. (2004). Estimating computation times to support scheduling of data intensive applications. *IEEE Distributed Systems Online (Special issue on Data Management, I/O Techniques and Storage Systems for Large-scale Data Intensive Applications)*, v.5, n.4.

Lageweg, B. J.; Lenstra, J. K. (1977). Private communication.

Lee, C.; Suzuki, J. (2006). Biologically-inspired design of autonomous and adaptive grid services. *2nd IEEE International Conference on Autonomic and Autonomous Systems*, Santa Clara, CA, EUA.

Lee, Y. C.; Zomaya, A. Y. (2007). Practical scheduling of bag-of-tasks applications on grids with dynamic resilience. *IEEE Transactions on Computers*, v.56, n.6, p.815–825.

Lehmann, E.; Casella, G. (2003). *Theory of Point Estimation*. Springer.

Liebert, W.; Pawelzik, K.; Schuster, H. G. (1991). Optimal embeddings of chaotic attractors from topological considerations. *Europhysics Letters*, v.14, n.6, p.521–526.

Lin, D.-T.; Dayhoff, J. E.; Ligomenides, P. A. (1995). Trajectory production with the adaptive time-delay neural network. *Neural Netw.*, v.8, n.3, p.447–461.

Lorenz, E. (1963). Deterministic nonperiodic flow. *Journal of Atmospheric Science*, v.20, p.130–141.

Manevitz, L. M.; Yousef, M. (2001). One-class svms for document classification. *Journal of Machine Learning Research*, v.2, p.139–154.

Mañé, R. (1980). *On the dimension of the compact invariant sets of certain nonlinear maps*. Springer.

Medio, A.; Gallo, G. (1993). *Chaotic Dynamics: Theory and Applications to Economics*. Cambridge University Press.

Mello, R. F.; Andrade Filho, J. A.; Dodonov, E.; Ishii, R. P.; Yang, L. T. (2007a). Optimizing distributed data access in grid environments by using artificial intelligence techniques. *The International Symposium on Parallel and Distributed Processing and Application – ISPA*, p. 1–12.

Mello, R. F.; Andrade Filho, J. A.; Dodonov, E.; Li, K.-C.; Yang, L. T. (2007b). Supporting the transparent execution of high performance applications on grids. p. 1–4.

Mello, R. F.; Andrade Filho, J. A.; Senger, L. J.; Yang, L. T. (2007c). RouteGA: A grid load balancing algorithm with genetic support. *The IEEE 21th International Conference on Advanced Information Networking and Applications (AINA 2007)*, p. 885–892. IEEE Computer Society.

Mello, R. F.; Andrade Filho, J. A.; Senger, L. J.; Yang, L. T. (2008). Grid job scheduling using route with genetic algorithm support. *Telecommunication Systems*, v.38, n.3-4, p.147–160.

Mello, R. F.; Paiva, M. S. V.; Trevelin, L. C.; Gonzaga, A. (2002). Analysis on the significant information to update the tables on occupation of resources by using a peer-to-peer protocol. *16th Annual International Symposium on High Performance Computing Systems and Applications*, Moncton, NB, Canadá.

Mello, R. F.; Senger, L. J. (2004). A new migration model based on the evaluation of processes load and lifetime on heterogeneous computing environments. *International Symposium on Computer Architecture and High Performance Computing - SBAC-PAD*, p. 222–227. IEEE Computer Society.

Mello, R. F.; Senger, L. J. (2006). Model for simulation of heterogeneous high-performance computing environments. *7th International Conference on High Performance Computing in Computational Sciences – VECPAR 2006*, p. 1–11.

Mello, R. F.; Senger, L. J. (2008). On simulated annealing applied on the scheduling of parallel applications. *International Symposium on Computer Architecture and High Performance Computing - SBAC-PAD*, p. 1–8. IEEE Computer Society.

Mello, R. F.; Senger, L. J.; Yang, L. T. (2006a). Performance evaluation of route: A load balancing algorithm for grid computing. *RITA – Revista de Informática Téorica e Aplicada*, v.13, n.1, p.87–108.

Mello, R. F.; Senger, L. J.; Yang, L. T. (2006b). A routing load balancing policy for grid computing environments. *The IEEE 20th International Conference on Advanced Information Networking and Applications (AINA 2006)*, p. 153–158. IEEE Computer Society.

Meng, A. (2003). An introduction to markov and hidden markov models. http://www2.imm.dtu.dk/pubdb/p.php?3313.

Meyler, K. (2008). *The Dynamic Datacenter – Disponível em: http://www.networkworld.com/community/node/27354. Acesso em: 10 de novembro de 2008.*

Mika, M.; Waligóra, G.; Weglarz, J. (2004). A metaheuristic approach to scheduling workflow jobs on a grid. *Grid resource management: state of the art and future trends*, p. 295–318, Norwell. Kluwer Academic Publishers.

Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill.

Murch, R. (2004). *Autonomic Computing*. IBM Press.

Naik, V. K.; Setia, S. K.; Squillante, M. S. (1997). Processor Allocation in Multiprogrammed Distributed-memory Parallel Computer Systems. *Journal of Parallel and Distributed Computing*, v.47, n.1, p.28–47.

NERSC (2008). National energy research scientific computing center. Disponível em `http://pdsi.nersc.gov/benchmarks.htm`. Acesso em: 12 de setembro de 2008.

Nery, B. R.; Mello, R. F.; de Leon Ferreira de Carvalho, A. C. P. (2006). Escalonamento de processos utilizando técnicas de ACO. *Revista Eletrônica de Iniciação Científica*, v.1, p.1–15.

Nieuwpoort, R.; Maassen, J.; Kielmann, T.; Bal, H. E. (2005). Satin: Simple and efficient java-based grid programming. *In AGridM 2003 Workshop on Adaptive Grid Middleware.*

Omar, W. M.; Taleb-Bendiab, A.; Karam, Y. (2006). Autonomic middleware services for just-in-time grid services provisioning. *Journal of Computer Science*, v.6, n.2, p.521–527.

Oprescu, A.-M.; Kielmann, T.; Danelutto, M.; Aldinucci, M. (2008). Autonomic behavior of grid applications using component platforms. Relatório Técnico TR-0156, Institute on Grid Systems, Tools and Environments, CoreGRID - Network of Excellence.

Othman, A.; Dew, P.; Djemame, K.; Gourlay, I. (2003). Adaptive grid resource brokering. *cluster*, v.00, p.172.

Palis Jr., J.; Melo, W. (1978). *Introdução aos Sistemas Dinâmicos*. Edgard Blücher, 1 edição.

Papadimitriou, C. (1994). *Computational Complexity*. Addison-Wesley, Reading, Massachusetts.

Parashar, M.; Hariri, S. (2006). *Autonomic Computing: Concepts, Infrastructure, and Applications*. CRC Press.

Penny, W.; Harrison, L. (2006). Multivariate autoregressive models. Friston, K.; Ashburner, J.; Kiebel, S.; Nichols, T.; Penny, W., editores, *Statistical Parametric Mapping: The analysis of functional brain images*. Elsevier, Londres.

Perry, G. L. W.; Enright, N. J.; Jaffre, T. (2001). Spatial modelling of landscape-scale vegetation dynamics, mont do, new caledonia. *S. Afr. J. Sci.*, v.97, n.11-12, p.501–509+.

Piovoso, M.; Laplante, P. A. (2003). Kalman filter recipes for real-time image processing. *Real-Time Imaging*, v.9, n.6, p.433–439.

Powell, M. J. D. (1987). Radial basis functions for multivariable interpolation: a review. v.1, p.143–167.

Rosenstein, M. T.; Collins, J. J.; Luca, C. J. D. (1993). A practical method for calculating largest lyapunov exponents from small data sets. *Physica D*, v.65, p.117–134.

Ryad, Z.; Daniel, R.; Noureddine, Z. (2001). The rrbf. dynamic representation of time in radial basis function network. *Emerging Technologies and Factory Automation, 2001. Proceedings. 2001 8th IEEE International Conference on*, v.2, p.737–740 vol.2.

Santos, M. L.; Mello, R. F.; Yang, L. T. (2007). Extraction and classification of user behavior. *Embedded and Ubiquitous Computing*, v. 4808, p. 493–506.

Satyanarayanan, M.; Kistler, J.; Kumar, P.; Okasaki, M.; Siegel, E.; Streere, D. (1990). Coda: A highly available file system for distributed workstation environments. *IEEE Transactions on Computers*, v.39, n.4.

Semenov, M. A.; Terkel, D. A. (2003). Analysis of convergence of an evolutionary algorithm with self-adaptation using a stochastic lyapunov function. *Evol. Comput.*, v.11, n.4, p.363–379.

Senger, L. J. (2005). *Escalonamento de processos: uma abordagem dinâmica e incremental para a exploração de características de aplicações paralelas.* Tese (Doutorado), ICMC/USP, São Carlos, SP, Brasil.

Senger, L. J.; Mello, R. F.; Santana, M. J.; Santana, R. H. C. (2005a). An on-line approach for classifying and extracting application behavior on linux. *High Performance Computing: Paradigm and Infrastructure*, p. 381–402. John Wiley and Sons.

Senger, L. J.; Mello, R. F.; Santana, M. J.; Santana, R. H. C. (2007). Aprendizado baseado em instâncias aplicado à predição de características de execução de aplicações paralelas. *Revista de Informática Teórica e Aplicada*, v.14, p.44–68.

Senger, L. J.; Mello, R. F.; Santana, M. J.; Santana, R. H. C.; Yang, L. T. (2005b). Improving scheduling decisions by using knowledge about parallel applications resource usage. *High Performance Computing and Communcations: First International Conference (HPCC), LNCS 3726*, v. 3726, Sorrento, Itália.

Sevcik, K. C. (1989). Characterizations of Parallelism in Applications and their use in Scheduling. *Performance Evaluation Review*, v.17, n.1, p.171–180.

Shefler, W. C. (1988). *Statistics: Concepts and Applications.* The Benjamin/Cummings.

Shenshi, G.; Zhiqian, W.; Jitai, C. (1999). The fractal research and predicating on the times series of sunspot relative number. *Applied Mathematics and Mechanics*, v.20, n.1, p.84–89.

Shirose, K.; Matsuoka, S.; Nakada, H.; Ogawa, H. (2004). Autonomous configuration of grid monitoring systems. *Applications and the Internet Workshops, 2004. SAINT 2004 Workshops. 2004 International Symposium on*, v., p.651–657.

Shivaratri, N. G.; Krueger, P.; Singhal, M. (1992). Load distributing for locally distributed systems. *IEEE Computer*, v.25, n.12, p.33–44.

Sivasubramaniam, A.; Singla, A.; Ramachandran, U.; Venkateswaran, H. (1994). An approach to scalability study of shared memory parallel systems. *Measurement and Modeling of Computer Systems*, p. 171–180.

Smith, W.; Foster, I.; Taylor, V. (1998). Predicting application run times using historical information. Feitelson, D. G.; Rudolph, L., editores, *Job Scheduling Strategies for Parallel Processing*, p. 122–142. Springer. Lect. Notes Comput. Sci. vol. 1459.

Sun Microsystems, Inc. (1989). RFC 1094: NFS: Network File System Protocol specification.

Sun Microsystems, Inc. (2008). Sun grid engine – disponível em: http://www.sun.com/software/gridware. acesso em: 19 de novembro de 2008.

Suzuki, J.; Suda, T. (2005). A middleware platform for a biologically inspired network architecture supporting autonomous and adaptive applications.

Takens, F. (1980). Detecting strange attractors in turbulence. *Dynamical Systems and Turbulence*, p. 366–381. Springer.

van Nieuwpoort, R. V.; Kielmann, T.; Bal, H. E. (2007). User-friendly and reliable grid computing based on imperfect middleware. *SC '07: Proceedings of the 2007 ACM/IEEE conference on Supercomputing*, p. 1–11, New York, NY, USA. ACM.

Vetter, J. S.; Mueller, F. (2003). Communication characteristics of large-scale scientific applications for contemporary cluster architectures. *J. Parallel Distrib. Comput.*, v.63, n.9, p.853–865.

Waibel, A.; Hanazawa, T.; Hinton, G.; Shikano, K.; Lang, K. (1989). Phoneme recognition using time delay neural networks. *IEEE Transactions on Accoustics, Speech and Signal Processing*, v.37, p.328–339.

Wan, E. A. (1994). Time series prediction by using a connectionist network with internal delay lines. *Time Series Prediction*, p. 195–217. Addison-Wesley.

Wan, E. A.; Van Der Merwe, R. (2000). The unscented kalman filter for nonlinear estimation. *Adaptive Systems for Signal Processing, Communications, and Control Symposium 2000. AS-SPCC. The IEEE 2000*, p. 153–158.

Whitney, H. (1936). Differentiable manifolds. *The Annals of Mathematics*, v.37, n.3, p.645–680.

Wilson, D. R.; Martinez, T. R. (1997). Improved heterogeneous distance functions. *Journal of Artificial Intelligence Research*, v.6, p.1–34.

Yarkhan, A.; Dongarra, J. (2002). Experiments with scheduling using simulated annealing in a grid environment. *Third International Workshop on grid computing*, p. 232–242, Baltimore.

Zeevi, A. J.; Meir, R.; Adler, R. J. (1998). Non-linear models for time series using mixtures of autoregressive models. Relatório técnico.

Zemouri, R.; Racoceanu, D.; Zerhouni, N. (2003). Recurrent radial basis function network for time-series prediction. *Engineering Applications of Artificial Intelligence*, v.16, n.5–6, p.453–463.

Zhou, S.; Ferrari, D. (1987). An experimental study of load balancing performance. Relatório Técnico UCB/CSD 87/336, Technical Report N.o 86.8, Computer Science Division (EECS), Universidade da California, Berkeley, California 94720.