UNIVERSIDADE FEDERAL RURAL DO SEMIÁRIDO
UNIVERSIDADE DO ESTADO DO RIO GRANDE DO NORTE
MESTRADO EM CIÊNCIA DA COMPUTAÇÃO

MAXIMILIANO ARAÚJO DA SILVA LOPES

CONTROLE DE CONCORRÊNCIA EM SISTEMAS DE TEMPO REAL UTILIZANDO REDES NEURAIS ARTIFICIAIS.

MOSSORÓ – RN

2011

MAXIMILIANO ARAÚJO DA SILVA LOPES

CONTROLE DE CONCORRÊNCIA EM SISTEMAS DE TEMPO REAL UTILIZANDO REDES NEURAIS ARTIFICIAIS.

Dissertação apresentada ao Mestrado de Ciência da Computação – associação ampla entre a Universidade do Estado do Rio Grande do Norte e a Universidade Federal Rural do Semiárido, para a obtenção do título de Mestre em Ciência da Computação.

Orientador: Prof. Dr. Pedro Fernandes Ribeiro Neto – UERN.

MOSSORÓ – RN

2011

Catalogação da Publicação na Fonte.

Lopes, Maximiliano Araújo da Silva.
Controle de concorrência em sistemas de tempo real utilizando redes neurais artificiais. / Maximiliano Araújo da Silva Lopes - Mossoró, RN, 2011

88f.
Orientador(a): Prof . Dsc. Pedro Fernandes Ribeiro Neto
Dissertação (Mestre) -.Universidade do Estado do Rio Grande do Norte./ Universidade Federal Rural do Semiárido. Mestrado em Ciência da Computação computação.

1. Sistema – tempo real - Monografia. 2. Controle de concorrência - Monografia. 3. Redes neurais artificiais - Monografia. I. Ribeiro Neto, Pedro Fernandes.II. Universidade do Estado do Rio Grande do Norte. III. Universidade Federal Rural Do Semi-árido. IV. Título

UERN/BC CDD 004

Bibliotecário: Sebastião Lopes Galvão Neto CRB 15 / 486

MAXIMILIANO ARAÚJO DA SILVA LOPES

CONTROLE DE CONCORRÊNCIA EM SISTEMAS DE TEMPO REAL UTILIZANDO REDES NEURAIS ARTIFICIAIS.

Dissertação apresentada ao Mestrado em Ciência da Computação para a obtenção do título de Mestre em Ciência da Computação.

APROVADA EM: ___ / ___ / _____.

BANCA EXAMINADORA

_____

Prof. Dr. PEDRO FERNANDES RIBEIRO NETO – UERN
Presidente

_____

Prof. Dr. FRANCISCO MILTON MENDES NETO – UFERSA
Primeiro Membro

_____

Prof. Dr. DIOGO PINHEIRO FERNANDES PEDROSA – UFRN
Segundo Membro

Dedico este trabalho às mulheres de minha vida, minha mãe, Dapaz Lopes (in memorian), Estela, minha esposa, e Maria Eduarda, minha pequena.

Agradecimentos

Em primeiro lugar a Deus!

Aos meus pais, Toinho e Dapaz (*in memorian*), por terem me preparado para a vida de maneira tão inteligente.

À Estela e Maria Eduarda, ambas fontes de inspiração para a minha vida.

Aos meus irmãos, Alexandre, Frederico e Maria Cecília, que sempre foram companheiros em todas as etapas da vida, mesmo considerando a distância que nos separou ou ainda nos separam.

Aos meus amigos, em especial à Cicília Maia e ao Jardim de Infância da Unimed Mossoró (Mírian, Pablo, Tatá, Cibelly, Naja e Alécia) por sempre proporcionarem momentos agradáveis.

Aos professores e colaboradores do Departamento de Informática da UERN, em particular ao Dr. Pedro Fernandes, pela paciência e compreensão com todas as minhas atividades.

Aos companheiros do Laboratório de Engenharia de Software, mesmo com a minha grande ausência neste ambiente.

Aos meus alunos, por proporcionarem excelentes momentos de aprendizagem e razão maior pela minha escolha pela docência, fazendo com que os momentos em sala de aula sejam sempre prazerosos, tornando especial aos que hoje além de ex-alunos são ou foram em algum momento colegas de trabalho: Cicília (de novo), Daniel Grospim, Jéssica Neiva, Heitor Liberalino, Abrahão Christophe, Íria Caline, Shirlly Macedo, Susanny Mirelly, Camila Araújo, Rodrigo Ronner, Lenardo Chaves, Gracon Lima, Luana Priscilla, Alisson Mendes, Gideom Couto, Karla Haryanna, Alessandro Firmino, Lilliany Freitas, Marilyn Christinne, Edsongley Varela, Gutemberg Aquino, Alderi Junior, dentre outros.

À Faculdade de Ciências e Tecnologia Mater Christi, por acreditar na minha capacidade. Em especial aos professores desta instituição e principalmente os do curso de Sistemas de Informação.

Ao corpo docente, discente e funcional do Mestrado em Ciência da Computação.

-Papai, papai: Te amo!!!
Maria Eduarda.

Resumo

A utilização de Sistemas de Tempo Real (STR) é realidade no mundo em que vivemos. As pessoas necessitam cada vez mais de programas de computador que funcionem de maneira a atender todas as suas restrições e uma dessas restrições mais evidenciadas é o tempo. Para atender os requisitos temporais não significa que o sistema tem que ser rápido e que seja prioritário comparado com os demais sistemas, mas sim que no tempo certo, deve atender as demandas solicitadas pelo ambiente. A questão é que para atender o tempo certo os sistemas devem ser desenvolvidos com técnicas específicas, onde as adaptações de sistemas convencionais são necessárias. Um dos desafios a ser trabalhado é o das técnicas de controle de concorrência, onde os algoritmos devem tratar de forma específica às tarefas que necessitam atuar sobre um mesmo item de dado ou tabela em um mesmo intervalo de tempo. Apesar da existência de várias técnicas que realizam este controle de concorrência elas são aplicadas isoladamente, fazendo com que o sistema sempre se adeque a elas. A utilização de Redes Neurais Artificiais (RNA) no processo de seleção, através da identificação de padrões, e das técnicas de controle de concorrência a serem aplicadas no sistema em tempo de execução é analisada como uma possível solução, para desta maneira evitar adequações dos sistemas às técnicas e faça com que as técnicas se adequem aos sistemas.

Palavras-Chave: Sistemas de Tempo Real, Controle de Concorrência, Redes Neurais Artificiais.

Abstract

The use of Real-Time Systems is a reality in the world in which we live. People need computer programs that work so as to meet all their restrictions and such a restriction is more evident over time. To meet time requirements does not mean that the system has to be fast and be a priority compared to other systems, but rather that in the right time, must meet the demands required by the environment. The point is that the right time to meet the systems must be developed with specific technical adaptations of conventional systems where needed. One of the challenges to be worked on are the techniques of concurrency control, where these algorithms should deal with specific tasks that need to act on the same data item or table at the same time interval. Altough the existence of various techniques that carry out this concurrency control they are applied separately, making the system always appropriate to them. The use of Artificial Neural Networks in the selection process by identification of patterns, techniques concurrency control to be applied in the system at runtime is examined as a possible solution to this way to avoid adjustments of systems techniques do with the techniques that are suited to the systems.

Wordkeys: Real-Time Systems, Concurrency Control, Artificial Neural Network

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

2PL – Bloqueio de Duas Fases

2PLHP – Bloqueio de Duas Fases com Alta Prioridade

ACID – Atomicidade, Consistência, Isolamento e Durabilidade

BD – Banco de Dados

CC – Controle de Concorrência

IA – Inteligência Artificial

OCC-APFO – *Optmistic Concurrency Control Adaptive Priority Fan Out*

OCC-APFS – *Optmistic Concurrency Control Adaptive Priority Fan-Out Sum*

pe – Período

pr – Prazo

RNA – Rede Neural Artificial

SGBD – Sistema de Gerenciamento de Banco de Dados

SNH – Sistema Nervoso Humano

SNNS – *Stutgard Neural Network Simulator*

STR – Sistema de Tempo Real

tc – Tempo Computacional

tl – Tempo de Liberação

Capítulo 1

Introdução

Os Sistemas em Tempo Real (STR) (FARINES et al., 2000) são realidade no dia a dia das pessoas. Esses sistemas estão presentes nas agências bancárias, nos sistemas de internet, no *e-commerce*, bem como em outros sistemas.

Como o tempo passa a ser uma restrição para esses sistemas, a conclusão de suas tarefas de maneira correta, e no seu tempo correto, passa a ser fundamental.

Por sua vez, os protocolos de Controle de Concorrência (CC) (SILBERSCHATZ, 2006) são elementos importantes nesses sistemas, visto que estes são um dos componentes dos Sistemas Gerenciadores de Banco de Dados (SGBD) que fazem com que se garanta a integridade dos dados.

A utilização de um protocolo de Controle de Concorrência isoladamente pode ser um obstáculo para o melhor funcionamento dos Sistemas de Tempo Real, visto que se pode observar que a cada instante a demanda pelo sistema pode funcionar de maneira diferente.

Baseado nestas informações, o uso de uma Rede Neural Artificial (RNA) (BRAGA, 2000) na seleção de uma técnica adequada para cada momento de funcionamento do sistema, orientada pela demanda imediata, pode ser capaz de melhorar o desempenho do Sistema de Tempo Real, visto que este trabalharia com o seu melhor protocolo a cada instante.

Para as aplicações do mundo real, uma das principais restrições é o tempo, não sendo este tempo exigência ou sinônimo de rapidez, já que existem situações em que um determinado prazo deve ser cumprido, mesmo que ele seja bastante dilatado.

Pela razão do tempo ser requisito para estes sistemas, algumas pessoas vinculam Sistemas de Tempo Real a Sistemas Críticos, o que é uma visão completamente errada, pois esses sistemas não necessariamente são críticos.

Como exemplo, podemos citar sistemas que funcionam na internet. Estes podem possuir o tempo como um requisito, já que as pessoas ao acessarem as informações necessitam de segurança e agilidade no que estão buscando, sem necessariamente serem críticos, causando algum dano à pessoa pela informação ter se atrasado alguns segundos.

Seguindo essas idéias, alguns dos mecanismos de um Sistema de Tempo Real, tais como Controle de Concorrência, escalonamento e qualidade de serviço, devem ser reconsiderados, já que nesta percepção o tempo se torna fundamental.

Assim, o foco deste trabalho está concentrado na abordagem do Controle de Concorrência.

> Os protocolos de Controle de Concorrência devem permitir que informações conflitantes sejam executadas concorrentemente, de acordo com a necessidade de cada aplicação, onde essa necessidade é definida através de funções de qualidade de serviço e métricas de desempenho (Ribeiro Neto, 2006).

A possibilidade de existirem tarefas que acessam os mesmos dados ou tabelas em um mesmo instante de tempo pode fazer com que os sistemas gerem informações inválidas. As técnicas de Controle de Concorrência buscam solucionar este problema, fazendo com que as tarefas sejam executadas como se estivessem de maneira escalar, produzindo assim o mesmo resultado final, independente de qual das tarefas se encerre primeiro.

Para que seja possível a boa utilização de Controles de Concorrência, existem diferentes protocolos que facilitam o uso mediante cada situação, dependendo da classificação dos sistemas.

De maneira geral, existem dois principais tipos de controle para tratar a concorrência. Um baseado em *TimeStamps* ou marcas de tempo, que prevê que dificilmente uma concorrência irá ocorrer, deixando para verificar a consistência do Banco de Dados antes de encerrar as suas tarefas, e o outro baseado em Bloqueios, que supõe que concorrências ocorrem a todo o momento e assim utiliza técnicas para prevenir que tarefas utilizem em um mesmo tempo o mesmo dado ou tabela, bloqueando estes itens no início de sua execução.

1.1 Motivação

Uma questão importante a ser tratada junto aos protocolos de Controle de Concorrência e seus algoritmos é que, selecionado um algoritmo em um banco de dados este será o único que tratará as possíveis concorrências de um sistema, podendo desta forma gerar algum impasse que o mesmo não consiga resolver como, por exemplo, um *deadlock*. Os impasses gerados em um determinado algoritmo certamente não seriam detectados se outro algoritmo pudesse tratá-lo, ou seja, algum que fosse mais indicado para verificar o momento de execução e as características, além do Sistema de Tempo Real, das suas Tarefas.

Estes impasses, quando trata-se de Sistemas de Tempo Real, geram maiores problemas como, por exemplo, a perda de prazos ou não execução de tarefas.

Por outro lado, as Redes Neurais Artificiais, como parte da Inteligência Computacional, resolvem problemas que podem ser definidos como reconhecimento de padrões, ou seja, definida a etapa de análise sobre um determinado problema, onde podem ser verificados determinados tipos de comportamentos (ou padrões) diferentes e um bom esboço desses, as redes neurais artificiais são capazes de identificar com uma margem de acerto bastante aceitável qual o melhor padrão em que um determinado modelo se encaixa.

Sendo assim, ao existir uma definição de padrões de como as concorrências acontecem dentro do funcionamento do sistema não seria necessária a seleção de um único protocolo para tratá-las.

Uma Rede Neural Artificial funcionando integrada a um Sistema Gerenciador de Banco de Dados poderia selecionar em tempo de execução o melhor algoritmo para cada situação, melhorando o desempenho do sistema.

1.2 Objetivos

1.2.1 Objetivo Geral

O Objetivo deste trabalho é buscar uma forma inteligente e automática para a seleção e uso de protocolos de Controle de Concorrência em Sistemas de Tempo Real, associando uma Rede Neural Artificial a estes sistemas.

1.2.2 Objetivos Específicos

Como objetivos específicos tem-se:

- Definir padrões para a utilização de técnicas de Controle de Concorrência em situações específicas para um Sistema de Tempo Real;
- Simular em uma Rede Neural Artificial um modelo neural que quando treinado com as características anteriores possa indicar qual o algoritmo indicado para ser utilizado a cada momento;
- Desenvolver um algoritmo para validação da técnica aplicada, bem como verificar o seu desempenho;
- Definir métricas comparativas para a análise dos protocolos de Controle de Concorrência para Sistemas de Tempo Real;
- Comparar a utilização dos protocolos de Controle de Concorrência isoladamente com a técnica desenvolvida usando a Rede Neural Artificial.

1.3 Metodologia

A metodologia utilizada nesta dissertação é apresentada da seguinte maneira:

- Revisão bibliográfica: revisão bibliográfica sobre Sistemas de Tempo real, suas características e aplicações, bem como os protocolos para o Controle de Concorrência de sistemas em geral, focando após isto em Sistemas de Tempo Real, visto que as mesmas técnicas são aplicadas a todos os sistemas, sendo estas adaptadas ao Tempo Real. Também foi realizado um estudo sobre os conceitos das Redes Neurais Artificiais, focado nos tipos de problemas que estas resolvem, seus componentes e do *Neural Planner*, uma ferramenta que realiza a simulação de Redes Neurais Artificiais para se aplicar o estudo em questão.
- Definição das características que fundamentam a escolha de uma técnica de Controle de Concorrência: baseado no referencial teórico foram identificadas quais as características mais importantes no processo de escolha de que técnica de Controle de Concorrência utilizar, que serviram como base para os padrões de entrada da Rede Neural Artificial.
- Definição do estudo de caso: a definição do estudo de caso se deu no domínio de Sistemas de Tempo Real, que podem ter características diferentes de acordo com o momento em que suas tarefas acontecem.
- Modelagem do estudo de caso: de acordo com as análises sobre as características de um Sistema de Tempo Real, foi modelada uma Rede Neural Artificial com o objetivo de selecionar o algoritmo de Controle de Concorrência adequado para cada modelo apresentado como padrão de entrada.
- Validação da técnica utilizando uma Rede Neural Artificial: o desenvolvimento, em linguagem C, de um algoritmo simulando o funcionamento de três situações distintas. A primeira utilizando um algoritmo de bloqueio, a segunda um de *TimeStamp* e, por fim, a utilização da técnica com a Rede Neural Artificial.

1.4 Estrutura do Documento

Capítulo 2

O Capítulo 2 apresenta uma introdução aos Sistemas de Tempo Real e as suas possíveis classificações em termos do próprio sistema e suas tarefas.

Capítulo 3

O Capítulo 3 trata exclusivamente sobre o Controle de Concorrência e a razão dos mesmos existirem como mecanismos de estudo para os Sistemas Gerenciadores de Banco de Dados, focando no problema do Tempo Real. Aqui são apresentados alguns algoritmos de resolução de Concorrência.

Capítulo 4

O Capítulo 4 descreve a fundamentação teórica relacionada à inteligência computacional, dando ênfase às Redes Neurais Artificiais, seus neurônios, arquiteturas e algoritmos de aprendizagens, bem como as áreas de pesquisa onde se desenvolvem estudos utilizando estas Redes. Uma ferramenta para simulação de Redes Neurais Artificiais também é apresentada.

Capítulo 5

O Capítulo 5 apresenta o desenvolvimento do estudo de caso, onde será discutido um modelo de Rede Neural Artificial que trata o problema da seleção de uma determinada técnica de Controle de Concorrência baseada nas características dos sistemas, das tarefas e do momento em que o sistema se encontra em execução.

Apresenta ainda os resultados comparativos entre a utilização de técnicas avulsas e da RNA através da utilização do simulador, bem como o resultado do algoritmo desenvolvido para simulação.

Capítulo 6

Finalmente, o Capítulo 6 apresenta a conclusão do trabalho onde os aspectos positivos e negativos desta pesquisa serão discutidos, levantando-se também trabalhos futuros.

Capítulo 2

Sistemas de Tempo Real

No mundo real, o tempo é um fenômeno importante, visto que os eventos acontecem em pontos temporais. O mundo responde a estes estímulos de uma forma imediata, fazendo com que esta característica seja importante também aos sistemas computacionais.

Aplicações e tarefas em tempo real são caracterizadas por restrições de tempo, ou prazos, que devem ser respeitados para obterem o comportamento temporal desejado ou necessário. (LIU, 2000 *apud* FERNANDES, 2005).

Os Sistemas de Tempo Real (STR) devem poder fazer eficientemente o controle de tarefas e dados que possuam restrições temporais. Dentre essas aplicações estão os sistemas de internet, sistemas bancários, de aviação, redes de sensores etc.

Neste capítulo, serão vistos os conceitos fundamentais para o entendimento dos STR e as características necessárias para o funcionamento destes tipos de sistemas.

2.1 Introdução

Wermeister (2005) define Sistemas de Tempo Real como uma classe de sistemas computacionais onde o correto processamento dos algoritmos implementados não é suficiente para garantir o correto funcionamento do sistema. Sendo que para que

os resultados do sistema estejam corretos eles devem estar prontos nos tempos definidos pelos requisitos temporais do sistema.

Freitas (2007) acrescenta que os Sistemas de Tempo Real são sistemas que possuem fortes requisitos no atendimento de compromissos temporais e cita como exemplo o tempo máximo de execução, *jitter* e *deadline*, além de robustez e segurança.

Observando estes conceitos, entende-se que os Sistemas de Tempo Real são cada vez mais importantes para a sociedade, pois eles podem estar presentes desde simples eletrodomésticos ou aplicações *web* utilizadas no dia a dia até sistemas de controle de tráfego aéreo ou de transações bancárias.

Estando presente em lugares e situações tão diferentes é possível verificar que algumas aplicações não permitem falhas, como o acompanhamento de pacientes em leitos hospitalares. Outras não necessitam ser tão exigentes, como as videoconferências, onde pode haver um pequeno *delay* ou perda entre as partes envolvidas, desde que não atrapalhem o diálogo. Daí a necessidade de não confundir Sistemas de Tempo Real com sistemas rápidos.

Carvalho (2009) diz que esse tipo de sistema não implica que o tempo seja imediato. Implica que o tempo de resposta seja adequado, independentemente da unidade temporal.

Assim, mesmo o tempo sendo uma restrição, o não funcionamento deste por algum instante não o direciona a problemas críticos que podem acarretar em situações de difícil solução, dependendo da aplicação onde o mesmo está empregado.

Farines et al. (2000) apresenta diferentes interpretações para tempo computacional. Essa classificação pode se dar por:

- Tempo de execução e tempo na programação: recurso gasto durante a execução de um programa e uma grandeza manipulada pelo programa como outros tipos de variáveis, respectivamente;
- Tempo lógico e tempo físico: o lógico definido através de relações de precedências entre eventos e o físico um tempo métrico que permite estabelecer ordem e calcular o tempo entre eventos;

- Tempo denso e tempo discreto: o denso segue a natureza uniforme e contínua do tempo físico e isomorfo aos reais. Já o discreto funciona como uma simplificação do denso, sendo isomorfo aos naturais;
- Tempo global e tempo local: funciona para sistemas distribuídos, sendo o global uma noção abstrata ao instante de acesso a qualquer parte do sistema e o local observado em cada nó do sistema;
- Tempo absoluto e tempo relativo: referentes a eventos globais e locais, respectivamente.

Independente da caracterização de tempo a ser utilizada é interessante saber que os Sistemas de Tempo Real têm o tempo como um elemento extremamente relevante. A não entrega de um resultado esperado dentro de um período de tempo específico pode acarretar em falhas temporais, críticas ou não, no funcionamento do sistema.

> A corretude de muitos sistemas computacionais depende tanto dos resultados lógicos produzidos por eles, como do tempo em que esses resultados são produzidos. Esses sistemas são comumente denominados de sistemas em tempo-real. Em um STR, as especificações de tempo são impostas às tarefas na forma de tempo de liberação, tempo computacional, período e prazo (Ribeiro Neto,2006).

O tempo de liberação (tl) é o momento de tempo em que a tarefa está pronta para ser executada, sendo classificado como tempo de início mais cedo e tempo de início mais tarde da tarefa, sendo estes dois indicadores de quando a tarefa não pode iniciar e quando ela não deve iniciar, respectivamente. O tl pode detectar as violações nos prazos dos Sistemas de Tempo Real (Ribeiro Neto, 2006).

O tempo computacional (tc) é indicado pelo tempo necessário para que a tarefa seja realizada sem sofrer interrupções. É um tempo complexo de ser calculado, apesar de ser fundamental, pois ele depende como um todo do processamento da máquina. Utilização de processador, memórias principal e secundária, dispositivos de entrada e saída etc. podem influenciar diretamente sobre este tempo (Ribeiro Neto, 2006).

O período (pe) de uma tarefa é o tempo entre duas execuções consecutivas da mesma tarefa, podendo algumas tarefas não possuírem períodos fixos entre suas repetições (Ribeiro Neto, 2006).

O prazo (pr) é o tempo na qual a tarefa deve ser completada (Ribeiro Neto, 2006).

2.2 Classificação dos Sistemas de Tempo Real

Os Sistemas de Tempo Real podem ser classificados, segundo Farines et al. (2000), sob o ponto de vista da segurança e da implementação.

Do ponto de vista da segurança são divididos em sistemas não críticos de tempo real, ou brandos, quando uma falha temporal é da mesma grandeza para os benefícios do sistema funcionando normalmente; e em sistemas críticos de tempo real, ou duros, quando a consequência de uma falha temporal excede consideravelmente os benefícios do sistema, sendo esta falha considerada crítica (FARINES et al., 2000).

Sob o ponto de vista da implementação, os sistemas são separados em sistemas de resposta garantida, quando os recursos são suficientes para suportar picos do sistema e possui uma boa definição dos cenários de falhas; e sistemas de melhor esforço, quando a alocação dinâmica dos recursos é realizada probabilisticamente sobre cargas esperadas e cenários de falhas aceitáveis (FARINES et al., 2000).

Farines et al. (2000) ainda classifica as tarefas de tempo real como sendo de dois tipos: as tarefas críticas, que podem ocasionar falhas de grande porte no sistema se terminadas após o seu período, e as tarefas brandas, que se terminarem após o período, no máximo, apenas diminuem o seu desempenho.

Ampliando esta classificação, Ribeiro Neto (2006) e Carvalho (2009) apresentam uma classificação diferente para as tarefas, tratadas por ele como transações de tempo-real, que serão definidas neste trabalho seguindo os seguintes critérios:

- Restrições de tempo real: Nesta classificação, os prazos (pr) possuem suas restrições classificadas como estritas, suaves e firmes, que seguem

o mesmo padrão de contextualização dos sistemas do ponto de vista da segurança. Sendo para as transações de prazo estrito essencial o cumprimento dos prazos para não afetar diretamente o resultado do sistema; as suaves são transações que mesmo existindo a restrição temporal podem encerrar após o seu prazo; e as firmes são transações que serão canceladas caso não exista o cumprimento dos seus prazos;

- Padrões de Chegada das Transações: Esta classificação caracteriza os períodos (pe) de uma tarefa e mostra que elas podem ser Periódicas, Aperiódicas e Esporádicas, que determinam a periodicidade de chegada das transações. As periódicas chegam a intervalos de tempos previstos, as aperiódicas em intervalos irregulares de tempo e as esporádicas ocorrem apenas com um intervalo de tempo mínimo entre duas execuções, mas não é possível prever o intervalo entre duas transações;
- Tipo de Acesso de Dados: Transações de Escrita e Transações de Leitura

Ramamrithman (1993) *apud* Leite (2005) caracteriza as tarefas de um Sistema de Tempo Real baseado na natureza das tarefas de três diferentes maneiras:

- De acordo com a origem da restrição temporal imposta à tarefa. Possuem forma de requisitos de periodicidade;
- Baseadas nos prazos impostos às tarefas aperiódicas;
- Baseadas no efeito de perder o seu prazo como estrita, suave e firme, como acontece na classificação dos STR.

Além destas classificações, devem ser analisados outros tipos de restrições de tempo a serem aplicadas em Sistemas de Tempo Real, visto que o tempo da transação não é o único a ser observado. Dentre essas classificações, o tempo computacional, o de chegada e o de liberação, visto que qualquer transação sofre diretamente efeito destes outros tempos.

Para o desenvolvimento deste trabalho outra característica importante para ser levada em consideração é a prioridade das tarefas.

Supõe-se que algumas tarefas, por sua natureza, podem possuir diferentes níveis de prioridade, onde algumas devem, obrigatoriamente, ser executadas antes das

outras. Assim, a definição de prioridade das tarefas segue três padrões, podendo ser alta, média ou baixa prioridade, de acordo com a identificação realizada pelo próprio sistema.

As tarefas de alta prioridade devem ser executadas primeiro, as de prioridade média devem ser executadas tão logo possam e as de baixa prioridade podem ter um atraso um pouco maior, desde que haja tarefas com prioridade maior à sua frente.

Uma forma de organizar as tarefas em um sistema é através da aplicação de escalonamento, cuja ordenação acontece de uma maneira que aquelas consigam cumprir as suas restrições.

2.3 Escalonamento

Carvalho (2009) mostra que os requisitos dos Sistemas de Tempo Real apresentam informações temporais na forma de prazos, sendo estes executados para o perfeito funcionamento do sistema.

O escalonamento (SILBERSCHATZ et al., 2006) pode ser conceituado como uma ordenação das tarefas em uma fila na ordem em que elas devam ser executadas. A ordem em que elas são colocadas nesta fila depende da política aplicada para este intuito. O ideal é que essa ordenação realize todas as tarefas no tempo ideal, cumprindo assim, as suas restrições.

Em sistemas convencionais, o escalonador é essencial para a garantia de um funcionamento adequado e é ele que decide qual transação deve ser executada quando existe mais de uma pronta para ser executada.

Como as tarefas de um Sistema de Tempo Real possuem características diferentes dos sistemas convencionais como por exemplo possuem *deadline*, ou seja, o limite de tempo para execução da tarefa, onde esgotado o limite talvez não exista mais o interesse na realização da mesma, o papel do escalonador passa a ser ainda mais importante (FARINES, et al., 2000).

Segundo Carvalho (2009) para os Sistemas de Tempo Real o escalonamento difere um pouco dos sistemas convencionais. Nos sistemas convencionais, o escalonamento apenas tenta minimizar o tempo em que as transações acontecem. Já nos STR, o escalonamento tenta cumprir os requisitos temporais de cada tarefa.

A Tabela 2.1 apresenta os atributos de tempo real.

Para Farines et al. (2000), para programar um escalonamento é importante que se tenha regras ou políticas nesta ordenação. Assim, é mais provável que se consiga construir um escalonamento que cumpra de uma maneira mais adequada às restrições de tempo.

Tabela 2.1 - Atributos de um STR

Atributo	Definição
Tempo de Lançamento	Momento no qual a tarefa se encontra disponível para ser executada.
Tempo Máximo de Execução	Tempo máximo permitido para a execução de uma tarefa.
Deadline	Tempo no qual uma tarefa deve estar completa
***Deadline* relativo**	O tempo após o lançamento no qual a tarefa deve estar completa
Período	Período com a que novas instâncias das tarefas são lançadas.

Ainda segundo Farines et al.(2000), os algoritmos de escalonamento podem ser classificados de várias maneiras, tendo como base a forma de ordenação das tarefas:

- Preemptivo ou não preemptivo: Quando tarefas que estão em execução podem ou não ser interrompidas na chegada de uma outra com uma maior prioridade;
- Estático ou dinâmico: quando os parâmetros da escala desenvolvida são fixos e elaborados antes da execução da tarefa ou quando podem ser ajustados em tempo de execução, respectivamente.

É provável que determinadas situações inadequadas ocorram quando se trata exclusivamente de escalonamento, principalmente em STR, como, por exemplo, a perda

de prazos ou ainda no caso de sistemas preemptivos as tarefas que possuem uma prioridade baixa não serem realizadas nunca pela existência de tarefas com maior prioridade.

Pela quantidade de observações que devem ser realizadas para escalonamento Audsley e Burns (1990) trata este como um problema NP-completo.

A utilização de mecanismos para promover um melhor resultado na execução das tarefas é um estudo em constante evolução para o tratamento de sistemas de tempo real.

Um dos métodos que podem ser utilizados na melhora dos resultados é a utilização prática de técnicas de controle para evitar que várias tarefas tentem acessar e/ou modificar valores de mesmos itens de dados em um mesmo instante de tempo, alterando assim características importantes para o funcionamento do sistema, como a sua integridade.

As técnicas de Controle de Concorrência (CC) são apresentadas no próximo capítulo.

Capítulo 3

Controle de Concorrência

As propriedades fundamentais que um Banco de Dados (BD) deve possuir são conhecidas por ACID, que remetem a Atomicidade, Consistência, Isolamento e Durabilidade. Quando várias tarefas acessam ao mesmo tempo um BD uma destas regras pode ser quebrada: a de Isolamento (ELMASRI & NAVATHE, 2005).

Uns dos principais objetivos de um Sistema de Gerenciamento de Banco de Dados (SGBD) é permitir que várias transações acessem dados compartilhados concorrentemente (LINDSTROM, 2003 *apud* FERNANDES, 2005). Entretanto, quando várias transações estão acessando concorrentemente o SGBD, os resultados produzidos ou recuperados pelas mesmas podem ser incorretos. Como conseqüência, a integridade do BD pode ser violada (HUNG; HUONG, 2002 *apud* FERNANDES, 2005).

O Isolamento permite que cada uma das tarefas ou transações trabalhe com uma visão de um banco de dados consistente a cada momento, sem haver interferência de outra tarefa ou transação.

Para garantir o isolamento, o sistema precisa ter o controle sobre todas as tarefas que acessam simultaneamente o Banco de Dados. Esse controle pode ser obtido através de protocolos e seus algoritmos são conhecidos por Controle de Concorrência (CC).

Basicamente os algoritmos de Controle de Concorrência tratam aspectos que garantem a seriação das tarefas, e essas técnicas, normalmente, são baseadas em bloqueios ou *timestamps* (ELMASRI & NAVATHE, 2005).

Visualizados os protocolos de Controle de Concorrência junto aos STR, o desafio é maior. Além de todas as atribuições já citadas, o tempo passa a ser fundamental para uma boa execução dos sistemas. Em certos momentos, a consistência do banco de dados passará a depender do descarte de tarefas e não tão somente do registro destas, já que as tarefas com operações fora de prazo podem não ter mais nenhum valor a agregar ao STR.

Neste capítulo serão apresentados de uma maneira breve os protocolos básicos de Controle de Concorrência baseados em bloqueio e *timestamp*, bem como alguns dos algoritmos desses controles aplicados a Sistemas de Tempo Real.

3.1 Concorrência

Silberschatz et al.(2006), Date (1999) e Elmasri e Navathe (2005) apresentam três problemas iniciais para a concorrência, considerando que mesmo que as tarefas ou transações estejam corretas elas podem vir a permitir resultados inválidos, se uma outra tarefa ou transação interferir nela. Estes problemas são:

- Atualização perdida;
- Dependência não comprometida;
- Análise inconsistente.

No problema da atualização perdida, duas tarefas “A” e “B” lêem o mesmo dado. Após as leituras uma delas, por exemplo, a “A”, realiza uma operação de atualização do dado e depois a outra tarefa, a “B”, também atualiza. Como esta segunda tarefa estava com o valor inicial do dado, implica diretamente na perda do valor da alteração realizada pela tarefa “A”. A Tabela 3.1 apresenta esta situação.

Tabela 3.1 - Atualização Perdida

Valor do Dado	Operação da Tarefa "A"	Valor do dado para a Tarefa "A"	Operação da Tarefa "B"	Valor do dado para a Tarefa "B"	Situação do BD
5	Ler dado	5	-	-	OK
5	-	5	Ler dado	5	OK
5	Atualizar dado (Soma +3)	8	-	5	OK
8	*Commit*	8	-	5	OK
8	-	-	Atualizar (Soma -2)	3	OK
3	-	-	*Commit*	3	Inconsistente

É possível observar que no momento do *commit* (encerramento de uma tarefa confirmando as alterações realizadas por ela) da tarefa "B" o valor do dado no banco era para ser atualizado para 6 (seis), porém como a leitura foi realizada anteriormente a escrita de "A" gerou um resultado errado para o dado.

Considerando que o comprometimento de um dado só é realizado no caso da sua operação de término for um *commit*, que garante a alteração do dado, a dependência não comprometida faz com que uma tarefa "B" leia o valor de um dado já atualizado, mas ainda não comprometido pela tarefa "A", chegando este dado nunca estar comprometido, quando se encerra a tarefa retomando as alterações, com um *rollback*. Desta maneira, a tarefa "B" trabalharia com um dado que nunca existiu. Acompanhando a Tabela 3.2, é possível perceber este problema.

No momento em que a tarefa "A" é retomada, é possível observar que a tarefa "B" já possuía o valor da atualização realizada por "A" e realizou operações sobre um valor de dado inexistente (8), fazendo com que o resultado final da operação ficasse com o valor 6 (seis), sendo que o seu valor correto seria 3 (três).

Tabela 3.2 - Dependência não Comprometida

Valor do Dado	Operação da Tarefa "A"	Valor do dado para a Tarefa "A"	Operação da Tarefa "B"	Valor do dado para a Tarefa "B"	Situação do BD
5	Ler dado	5	-	-	OK
5	Atualizar dado (Soma +3)	8	-	-	OK
8	-	8	Ler dado	8	OK
8	*Rollback*	-	-	8	OK
5	-	-	Atualizar (Soma -2)	6	Inconsistente
6	-	-	*Commit*	-	Inconsistente

O último problema, o da análise inconsistente, ocorre quando duas tarefas estão a realizar operações no mesmo conjunto de dados e esta sequência de operações levam a resultados inconsistentes na análise dos dados. A Tabela 3.3 apresenta uma adaptação deste problema.

Tabela 3.3 - Análise Inconsistente. Adaptado de Date (1999)

Tarefa "A"	Valores			Tarefa "B"
	Conta 1	Conta 2	Conta 3	
Ler Conta 1 Soma = 40	40	50	30	-
Ler Conta 2 Soma = 90	40	50	30	-
-	40	50	30	**Ler Conta 3**
-	40	50	20	**Atualizar Conta 3 (30 → 20)**
-	40	50	20	**Ler Conta 1**
-	50	50	20	**Atualizar Conta 1 (40 → 50)**
-	50	50	20	***commit***
Ler Conta 3 Soma = 110?	50	50	20	-
Valor Total das Contas	**120**			

A Tarefa "A" realiza um somatório do valor de todas as contas. Porém, antes que essa operação fosse encerrada, a Tarefa "B" atualiza os valores dos dados e os compromete, fazendo com que a análise realizada por "A" se tornasse inconsistente

totalizando o seu valor em 110 (Cento e Dez), quando este valor deveria ser 120 (Cento e Vinte).

Casanova e Moura (1999) trata estes mesmos problemas.

A partir destes exemplos, se verifica a importância de se realizar o controle de concorrência das transações, visto que estas operações realizadas podem implicar em resultados inconsistentes para um Banco de Dados.

3.1.1 Serialização

Segundo Silberschatz et al.(2006), o Sistema de Gerenciamento de Banco de Dados (SGBD) precisa controlar a execução de duas ou mais tarefas ou transações concorrentes mantendo as características dos dados em um estado consistente, ou seja, semelhante ao qual seria se estas fossem executadas sequencialmente em uma ordem específica. Isso garante que o conjunto de tarefas é seriável, produzindo um mesmo resultado que a execução serial das mesmas transações.

Dependendo do tipo de operação a ser realizada por uma tarefa em um banco de dados, estas podem gerar conflitos e impedir que a sequência seja seriável. O conflito é gerado quando temos em pelo menos uma das tarefas uma operação de escrita sobre um dado, que foi ou vai ser lido por outra tarefa antes desta ser consolidada, pois isto pode afetar diretamente nas propriedades ACID do Banco de Dados (SILBERSCHATZ et al., 2006).

Para evitar tais conflitos as técnicas de Controle de Concorrência evitam que problemas mais graves ocorram em Banco de Dados e permitem que a teoria da serialização seja sempre válida.

Desta forma, as principais abordagens que são utilizadas para a realização do Controle de Concorrência em um Banco de Dados são baseadas em duas técnicas: bloqueio e *TimeStamp*.

Os algoritmos de bloqueio fazem com que antes que uma tarefa realize uma operação sobre um item de dado ou uma tabela este realize um bloqueio sobre o mesmo, impedindo assim que outra tarefa possa alterar os dados sem autorização para tal.

Já os algoritmos de *TimeStamp*, ou marca de tempo, são baseados em marcas sobre as tarefas, normalmente inseridas no momento em que elas se iniciam. Essas marcas garantem que uma tarefa iniciou antes que a outra.

Ao contrário do bloqueio, antes de encerrar a tarefa, o algoritmo verifica se o item de dado que está sendo alterado sofreu alguma outra modificação. Dependendo da resposta confirma ou aborta a tarefa, fazendo com que a mesma reinicie.

As seções seguintes apresentam a contextualização geral de cada uma das técnicas citadas, visto que estas serão utilizadas no estudo de caso do corrente trabalho.

3.1.2 Técnicas de Bloqueio

As técnicas de bloqueio são caracterizadas pelo bloqueio/liberação de dados, fazendo com que esses dados, compartilhados, sejam acessados apenas por uma tarefa, no caso da existência de uma operação de escrita.

Segundo Elmasri e Navathe (2005), um bloqueio ou *lock* é uma variável associada a um item de dados que descreve a condição do item em relação às possíveis operações que podem ser aplicadas a ele.

O bloqueio é realizado sobre um dado quando uma tarefa necessita utilizá-lo para realizar uma operação de leitura ou escrita.

Silberschatz ET AL. (2006) mostra que os dois principais modos de bloqueio são o compartilhado, quando este permite que várias transações realizem operações de leitura concorrentemente no mesmo objeto, e o exclusivo, usado quando uma transação necessita escrever em um item de dado. Desse modo, o bloqueio evita que outras transações acessem esse dado enquanto a transação que o bloqueia não o liberar. Essa

mesma caracterização é adotada por Elmasri e Navathe (2005), Date (1999), Koshafian (1994), Casanova e Moura (1999) e Özsu e Valduriez (2001).

A compatibilidade entre os modos de bloqueio se dá quando duas transações podem obter bloqueios concorrentemente sobre o mesmo item. A princípio, bloqueios de leitura são compatíveis a de leitura e gravação; e gravação e gravação não o são. A Tabela 3.4 apresenta as compatibilidades.

Tabela 3.4 - Compatibilidade de Bloqueios

	Compartilhado (Leitura)	Exclusivo (Gravação)
Compartilhado (Leitura)	Compatível	Não Compatível
Exclusivo (Gravação)	Não Compatível	Não Compatível

Casanova e Moura (1999) apresenta um protocolo de bloqueio e liberação de objetos, que considera uma tabela de fila de bloqueios, baseado nos seguintes passos:

1. Inicialmente a tabela de objetos está vazia;
2. Ao receber a solicitação para bloquear o objeto x por uma tarefa/transação T através de uma ação B(x), pesquise a tabela de bloqueios procurando alguma tripla, cujo primeiro elemento seja x:
    a. Se nenhuma tripla for encontrada, x está livre, então bloqueie x para T acrescentando a sua tripla à tabela.
    b. Senão, acrescente T ao final da fila de acessos a x, após a tripla encontrada.
3. Ao receber solicitação da tarefa/transação T para liberar x, pesquise na tabela de bloqueios procurando uma tripla cujos dois primeiros elementos sejam x e T:
    a. Se nenhuma tripla for encontrada, ignore a liberação de x;
    b. Caso contrário:
        i. Se a fila estiver vazia, retire a tripla liberando x;
        ii. Se a fila não estiver vazia, substitua a tripla pela próxima tripla da tabela de bloqueios.

Algumas melhorias podem ser aplicadas ao modo de bloqueio ao se adotar métodos ou critérios diferentes para tipos de bloqueios diferentes.

A utilização dos algoritmos de bloqueio como definidos não resolve completamente os problemas da atualização perdida, dependência não comprometida e análise inconsistente. É possível identificar a continuidade dos problemas. As tabelas serão reeditadas com as modificações impostas pelo algoritmo de bloqueio e a explicação será apresentada em seguida.

A Tabela 3.5 apresenta uma revisão no problema da atualização perdida, neste caso utilizando os bloqueios. É possível verificar que até o momento em que a Tarefa "A" solicita o bloqueio exclusivo, que não pode ser cedido, pois a Tarefa "B" possui um bloqueio compartilhado junto ao item de dado, neste momento a Tarefa "A" fica aguardando a liberação do dado, o que não ocorre. Em um momento seguinte "B" também solicita o bloqueio exclusivo, impossibilitado por "A". As duas operações ficam na espera, uma pela outra. Como nenhuma das duas encerra temos um problema chamado de impasse ou *deadlock*.

Tabela 3.5 – Atualização Perdida com técnicas de bloqueio

Valor do dado	Operação da Tarefa "A"	Valor do dado para a Tarefa "A"	Operação da Tarefa "B"	Valor do dado para a Tarefa "B"	Situação do BD
5	Ler dado (Bloqueio Compartilhado)	5	-	-	OK
5	-	5	Ler dado (Bloqueio Compartilhado)	5	OK
5	Atualizar Dado (Soma +3) (Solicita Bloqueio Exclusivo)	5	-	5	OK
5	Espera	5	-	5	OK
5	Espera	-	Atualizar (Soma -2) (Solicita Bloqueio Exclusivo)	5	OK
	Espera			Espera	
	Espera			Espera	

O impasse ocorre no momento em que um conjunto de duas ou mais tarefas se colocam em um estado de espera, aguardando que uma das outras encerre o seu bloqueio para poder prosseguir. Desta maneira, o ideal é que uma ou mais destas tarefas sejam retomadas para o restante dar prosseguimento.

Para o problema da dependência não comprometida os danos são menores. O fato de uma tarefa, após realizar ou solicitar um bloqueio exclusivo, retardar o seu encerramento, seja comprometendo ou reiniciando, pode acarretar em um tempo de espera grande para a tarefa que aguarda, o que para Banco de Dados de Tempo Real pode ocasionar problemas de grande perda. A Tabela 3.6 apresenta um exemplo desta espera.

Tabela 3.6 - Dependência não Comprometida com Técnicas de Bloqueio

Valor do dado	**Operação da Tarefa "A"**	**Valor do dado para a Tarefa "A"**	**Operação da Tarefa "B"**	**Valor do dado para a Tarefa "B"**	**Situação do BD**
5	Ler dado (Adquire Bloqueio Compartilhado)	5	-	-	OK
5	Atualizar dado (Soma +3) (Adquire Bloqueio Exclusivo)	8	-	-	OK
8	-	8	Ler dado (Solicita Bloqueio Compartilhado)	-	OK
8	-	8	Espera	-	OK
8	*Commit* (Libera Bloqueio)	-	Espera	-	OK
8	-	-	Ler dado Adquire Bloqueio Compartilhado)	8	OK
6			Atualizar (Soma -2)	6	OK
6			*Commit*		OK

No caso do problema de análise inconsistente, o mesmo problema de impasse, citado anteriormente pode ocorrer. Na Tabela 3.7 é apresentada esta situação. No momento em que a Tarefa "B" solicita Compartilhamento Exclusivo para a Conta 1, a mesma está com um bloqueio compartilhado pela Tarefa "A", então "B" aguarda "A" encerrar para dar continuidade. Porém a Tarefa "A", requisita um Bloqueio compartilhado para a conta 3, que já mantém um Bloqueio Exclusivo de "B", o que faz com que "A" necessite aguardar o encerramento de "B" para dar continuidade. Assim, nenhuma das duas consegue encerrar, gerando o impasse.

Tabela 3.7 - Análise Inconsistente com Técnicas de Bloqueio. Adaptado de Date (1999)

Tarefa "A"	Valores			**Tarefa "B"**
	Conta 1	Conta 2	Conta 3	
Ler Conta 1 (Adquire Bloqueio Compartilhado sobre Conta 1) Soma = 40	40	50	30	-
Ler Conta 2 (Adquire Bloqueio Compartilhado Sobre Conta 2) Soma = 90	40	50	30	-
-	40	50	30	**Ler Conta 3 (Adquire Bloqueio Compartilhado Sobre Conta 3)**
-	40	50	20	**Atualizar Conta 3 (30 → 20) (Adquire Bloqueio Exclusivo Sobre Conta 3)**
-	40	50	20	**Ler Conta 1 (Adquire Bloqueio Compartilhado Sobre Conta 1)**
-	50	50	20	**Atualizar Conta 1 (40 → 50) (Solicita Bloqueio Exclusivo Sobre Conta 1)**
-	50	50	20	**Espera**
Ler Conta 3 (Solicita Bloqueio Compartilhado sobre Conta 3)	50	50	20	**Espera**
Espera				**Espera**
Espera				**Espera**

Outro algoritmo que realiza o controle de concorrência através de bloqueios é conhecido como Bloqueio de duas fases, ou 2PL (T*wo-Phase Locking*). Este algoritmo foi elaborado para garantir a serialização das transações.

Segundo Koshafian (1994), o algoritmo 2PL consegue separar uma tarefa em fase de crescimento[1] e outra de redução[2] de forma clara. De maneira geral, a regra de bloqueio de duas fases faz com que nenhuma transação que já tenha liberado um bloqueio solicite bloquear algum item de dado.

Silberschatz ET AL. (2006), Elsmari e Navathe (2005) e Özsu e Valduriez (2001) mostram que cada tarefa possui duas fases, uma de crescimento, onde se adquire os bloqueios e se acessam os dados, e outra de redução, onde os bloqueios são liberados. A Figura 3.1 apresenta um gráfico que apresenta essas duas fases.

É interessante perceber que o ponto de bloqueio é atingido no momento em que a transação adquire todos os seus bloqueios e antes que seja iniciada a liberação destes, sendo este o fator determinante para a mudança de fases.

Elsmari e Navathe (2005) afirmam que toda transação em um plano de execução que segue o protocolo de bloqueio em duas fases é garantida a sua serialização.

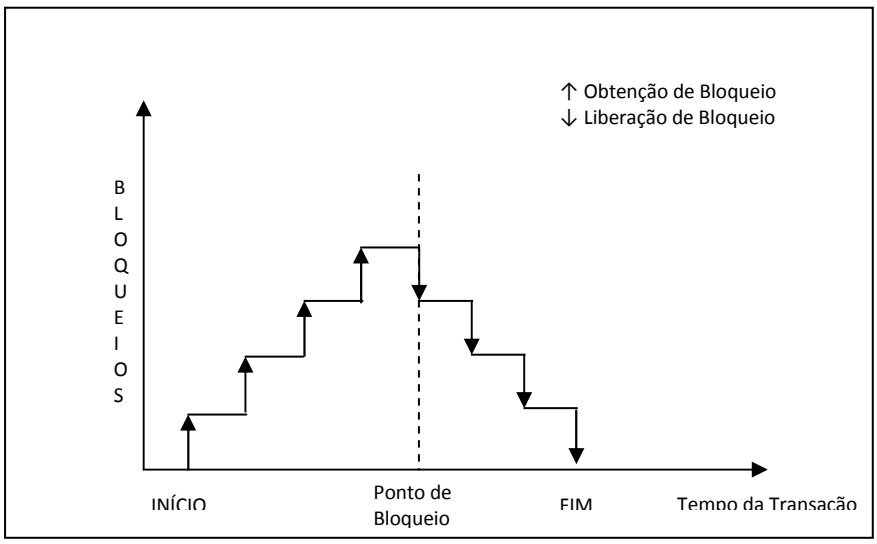

Figura 3.1 - Bloqueio de Duas Fases. Özsu e Valduriez (2001)

[1] Fase em que o algoritmo 2PL realiza todos os bloqueios necessários para a execução de uma tarefa.
[2] Fase em que o algoritmo 2PL realiza a liberação dos bloqueios realizados para a execução de uma tarefa. Após a primeira liberação nenhum outro item pode ser bloqueado.

Tanto Elsmari e Navathe (2005) quanto Silberschatz et al. (2006) apresentam aprimoramentos do algoritmo de Bloqueio de duas fases que foram desenvolvidos na tentativa de otimizar o seu funcionamento. Abaixo segue uma relação destes:

- 2PL Conservador: Requer que uma tarefa bloqueie todos os itens que ela vai acessar antes de iniciar;
- 2PL Estrito: Uma tarefa não libera nenhum de seus bloqueios exclusivos até que ela se encerre, seja efetivando a tarefa ou abortando a mesma;
- 2PL Rigoroso: Semelhante ao 2PL Estrito, este modelo não libera nenhum dos bloqueios até o fim de sua execução.

Segundo Silberschatz et al. (2006), o algoritmo 2PL não garante a liberdade ao impasse.

Elsmari e Navathe (2005) apresentam uma abordagem chamada detecção de *deadlock* para tratar este problema do impasse baseada em grafos. Eles afirmam ainda que esta é uma solução atrativa se previamente for sabido que existirão poucas interferências entre as transações. A possibilidade de poucas interferências se dá se as transações forem curtas e se cada transação bloquear poucos itens de dados, ou ainda se a carga de transações for leve.

3.1.2 Técnicas de *TimeStamp*

Segundo Özsu e Valduriez (2001), as técnicas que utilizam *TimeStamp* não tentam manter a serialização por exclusão mútua, diferente do que ocorre com os algoritmos de bloqueio.

Elsmari e Navathe (2005) apresentam um *TimeStamp* como um identificador único criado pelo SGBD para identificar uma tarefa, o que torna cada uma delas única. Outra característica importante é que, como estes identificadores são gerados pelo

mesmo gerenciador e baseados na ordem em que cada tarefa é submetida ao sistema essa geração tem caráter crescente, permitindo uma ordenação de maneira mais simples.

Casanova (1999) intitula essas técnicas de pré-ordenação e faz uma observação interessante com relação ao momento em relação à ordem de serialização das tarefas. Já na comparação com o modelo de bloqueio de duas fases é mostrado que a ordem de serialização é realizada posteriormente, enquanto que nas técnicas *TimeStamp* a serialização é realizada anteriormente.

A utilização de técnicas de *TimeStamp* apresenta duas garantias às tarefas. A primeira é que não ocorrerá o *deadlock*, garantia não dada pelas técnicas de bloqueio, já que estas não realizam bloqueios (Locks), e a segunda é que a tarefa, ao receber o seu identificador primeiro, também inicializará primeiro, passando uma ideia de que está sempre à frente das demais tarefas que receberam o seu identificador posteriormente.

O funcionamento de um algoritmo de controle de *TimeStamp* é apresentado por meio de uma verificação do identificador cada vez que uma tarefa T tentar realizar uma operação, seja de leitura ou escrita, de maneira que nenhuma ordem da execução dos identificadores será quebrada.

No caso de alguma outra tarefa já estar operando um item de dado necessário à transação T, essa operação é abortada e enviada para obter um novo identificador para tentar realizar a sua operação.

O identificador do *TimeStamp* pode ser gerado a partir do *clock* do computador ou de um contador lógico iniciado pelo próprio SGBD.

Silberschatz et al. (2006) apresenta dois valores de *TimeStamp* utilizados para implementar esse algoritmo: o W-*TimeStamp*, que indica o maior *TimeStamp* de qualquer transação que executou uma escrita com sucesso e o R-*TimeStamp*, que indica o maior *TimeStamp* de qualquer transação que efetuou uma leitura com sucesso.

3.2 Trabalhos Relacionados

As técnicas apresentadas até este momento são gerais e podem ser discutidas no âmbito de qualquer sistema informatizado, porém existem estudos específicos que tratam o Controle de Concorrência para Banco de Dados de Tempo Real. Alguns destes trabalhos serão apresentados nesta seção.

Em Datta e Son (2002) é realizado um estudo comparativo de duas técnicas otimistas para o controle de concorrência em Sistemas de Tempo Real. O artigo é iniciado com uma abordagem geral sobre Sistemas de Tempo Real e o porquê da necessidade de novos algoritmos de Controle de Concorrência para estes sistemas.

As técnicas otimistas prevêem que dificilmente haverá conflitos. Sendo assim, as tarefas são executadas e no seu final verificado se existe ou não a concorrência, assim como o *TimeStamp*.

Essas duas técnicas apresentadas são: a *Optmistic Concurrency Control Adaptive Priority Fan Out* (OCC-APFO), que realiza ajustes dinâmicos na ordem da serialização das transações, e a *Optmistic Concurrency Control Adaptive Priority Fan-Out Sum* (OCC-APFS), que é bastante semelhante a anterior, porém realiza alguns cálculos para decidir qual a tarefa com maior prioridade a ser operada. Nas duas técnicas, a ideia é diminuir o trabalho perdido.

Em Kot (2008), é utilizada uma ferramenta chamada Uppaal para avaliar um algoritmo de bloqueio de duas fases pessimista, proposto por Nyström e Nolin (2004), de maneira formal, através de autômatos finitos.

As técnicas pessimistas prevêem que os conflitos ocorrerão a todo o momento e por isso verifica-se antes de iniciar uma tarefa se a mesma pode ser executada, assim como os algoritmos de bloqueio.

É justificada a utilização desta ferramenta por ser uma ferramenta projetada para sistemas em tempo real pela Universidade de Uppsala e pela Universidade de Aalborg, um modelo de Controle de Concorrência chamado *Pessimistc Protocol Two-Phase-Locking High-Priority* (2PLHP) (NYSTRÖM & NOLIM, 2004), que acrescenta ao bloqueio de duas fases a característica da prioridade da transação, ou seja, se uma transação de alta prioridade requisitar o bloqueio de um dado que está bloqueado por uma transação de menor prioridade, a de menos prioridade é abortada e reiniciada.

Xiao et al. (2009) apresenta um algoritmo otimista, seguro e híbrido para o controle de concorrência em sistemas móveis de tempo real, chamado SHORTCCP. Este algoritmo combina técnicas otimistas com o 2PLHP buscando o aperfeiçoamento da perda de transações por *deadline* com a garantia da segurança dos dados. Nele, a ideia de similaridade é colocada, pois para os autores a imprecisão dos sistemas de tempo real é visível, visto que existe um *delay* entre o tempo do acontecimento do fato e o registro deste no Sistema de Banco de Dados, daí existe um relaxamento no conceito de serialização para a introdução dos conceitos de similaridade de dados e similaridade de operações.

A principal diferença entre os algoritmos otimistas e o SHORTCC é que além de adotar mecanismos de bloqueio, este último ainda incorpora um mecanismo de checagem segura da informação.

Moiz e Rajamani (2010) mostram um algoritmo otimista para sistemas de tempo real móvel que garante consistência dos dados e introduz uma aplicação específica para detecção de conflitos e estratégias de resolução. Este algoritmo é baseado em *on-demand multicasting*.

Nos métodos *on-demand,* os dados são transmitidos apenas quando os clientes solicitam e nos métodos de *broadcast*, que segundo o autor são os dois métodos abordados pela literatura. O *host* transmite dados periodicamente, ficando a critério do cliente selecionar quais são os dados de seu interesse.

Em Pavlova e Hung (1999) é realizada uma especificação formal utilizando uma técnica chamada *Duration calculus* aplicada no algoritmo de bloqueio de duas fases, realizando, assim, correções e otimizações no mesmo.

Algumas outras técnicas Otimistas para o Controle de Concorrência em Sistemas de Tempo Real podem ser observadas em: Qilong e Zhongxiao (2005) e Adya et al. (1995). Cada um dos autores citados trata uma técnica ou protocolo específico para o Controle de Concorrência em Sistemas de Tempo Real. Desta forma, apesar de cada método resolver problemas deixados em aberto por outros, eles são muito específicos a tipos de problemas (sistemas), não considerando que o mesmo pode passar por períodos com características e/ou demandas diferentes.

A utilização de técnicas que possam prever a utilização de mais de um protocolo de Controle de Concorrência poderia ser uma saída, fazendo com que em momentos distintos os sistemas possam trabalhar também de maneira distinta e assim evitar a possibilidade de inconsistências ou bloqueios intermináveis, prejudicando o desempenho do mesmo.

Por sua vez, Redes Neurais Artificiais é um mecanismo da Inteligência Artificial (IA) que quando bem treinada consegue de maneira satisfatória atender ao reconhecimento de padrões de comportamento para situações específicas.

A utilização das Redes Neurais Artificiais na seleção de protocolos de CC para STR caracteriza-se em uma maneira eficiente de realizar esta escolha.

Capítulo 4

Redes Neurais Artificiais

Quando se visualiza a palavra Inteligência, o modelo que logo vem à mente é a forma de funcionamento do Sistema Nervoso Humano (SNH), que é formado por células, denominadas neurônios, capazes de coletar, processar e disseminar sinais elétricos (RUSSEL e NORVIG, 2004) e pelo cérebro que tem a função de processar, armazenar e organizar os dados e informações.

Logo, seria óbvio que a Inteligência Artificial (IA) tentaria de alguma forma simular o funcionamento do SNH do homem em uma máquina, daí surge a ideia da Rede Neural Artificial (RNA).

As Redes Neurais Artificiais possuem unidades de processamento bastante simples, que simulam o funcionamento do neurônio humano que, segundo Haykin (2001) tem a propensão natural para armazenar conhecimento experimental e torná-lo disponível para uso e adquirir conhecimento a partir do ambiente através de um processo de aprendizagem.

Este capítulo visa entender os conceitos fundamentais e o funcionamento das RNAs, bem como conhecer alguns modelos e entender em que casos estes se aplicam, visualizando assim as diversas aplicações e a aplicabilidade destas soluções para o problema proposto.

4.1 Sistema Nervoso Humano

A célula fundamental do SNH é o neurônio. Apenas no cérebro existem cerca de 10^{11} neurônios que se comunicam uns com os outros em paralelo e de forma contínua. Esses neurônios possuem três componentes, como mostra a Figura 4.1: o corpo celular, os dentritos e os axônios (BRAGA et al., 2000).

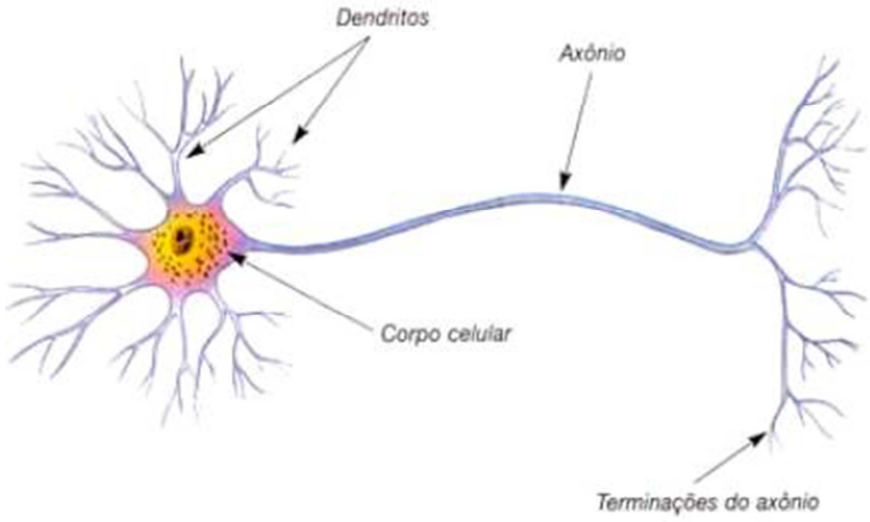

Figura 4.1 – Neurônio Humano.

O corpo celular mede alguns milésimos de centímetro e é responsável pelo processamento das informações. Os dentritos, por sua vez, medem alguns milímetros e são filamentos irregulares com ligações bastante complexas e é o meio pelo qual o neurônio recebe sinais advindos de outros neurônios. O axônio é o mais longo dos componentes, normalmente de largura uniforme, e que serve como canal de saída dos sinais do corpo celular (BRAGA et al., 2000).

Os neurônios se comunicam entre si através de ligações entre o axônio e os dentritos de outro neurônio. Estas ligações são chamadas de sinapses e recebem o sinal vindo do axônio e o repassa para a célula seguinte, por seus dentritos. Existem dois tipos de sinapses: as excitatórias, que ativam os neurônios, e as inibitórias, que dificultam o processo de ativação (BRAGA et al., 2000).

Baseadas na estrutura e no comportamento dos neurônios do SNH foram extraídas as características fundamentais dos modelos de neurônios artificiais.

4.2 Redes Neurais Artificiais

As Redes Neurais Artificiais podem ser conceituadas como uma forma de computação não algorítmica, sendo caracterizada pela representação de estruturas que lembram o funcionamento do SNH.

Braga et al. (2000) define Redes Neurais Artificiais como sendo sistemas paralelos distribuídos compostos por unidades de processamento simples, que calculam determinadas funções matemáticas.

As Redes Neurais Artificiais possuem características que a auxiliam na resolução de problemas complexos, dentre as quais se podem citar:

- Generalização: capacidade de abstrair e direcionar modelos semelhantes para uma única saída através de interpolação e extrapolação;

- Aprendizagem por exemplos: o treinamento de uma Rede Neural Artificial é realizado se apresentando padrões, e, dependendo do caso, a saída desejada;

- Tolerância a falhas: caso ocorra alguma falha com um neurônio ou com uma sinapse, a Rede Neural Artificial continua funcionando. O dano deve ser extenso o bastante para que a resposta global da rede seja afetada (HAYKIN, 2001);

- Auto organização: os pesos das conexões de uma Rede Neural Artificial não são postos pelo desenvolvedor. Eles são auto ajustados de acordo com os modelos apresentados a rede;

- Independência ao contexto: uma Rede Neural Artificial pode ser utilizada para vários contextos diferentes, dependendo, normalmente, da quantidade de entradas e saídas relevantes para ela;

- Processamento paralelo e distribuído: essa característica ocorre pelo fato da Rede Neural Artificial ser formada por um conjunto de neurônios artificiais interligados, que trabalham paralelamente e de forma distribuída.

Essas redes são formadas por três componentes: o neurônio, a arquitetura e o algoritmo de aprendizagem. Nas seções seguintes esses componentes são apresentados

separadamente e alguns exemplos de cada um deles são apresentados para facilitar o entendimento geral das Redes Neurais Artificiais.

4.2.1 O Neurônio

O neurônio artificial é a unidade matemática de uma rede neural. Simulando um Neurônio Biológico quanto à sua forma, funções e comportamento, recebendo entradas que estimulam ou inibem a ativação de suas sinapses.

O Neurônio de McCulloch & Pitts (McP) (McCULLOCH & PITTS, 1943), o primeiro proposto, funciona de uma maneira de fácil entendimento e o seu funcionamento será detalhado a seguir.

Segundo Braga et al. (2000), a definição matemática resultou em um modelo matemático com n terminais de entrada, representando os dentritos e um terminal de saída, representando o axônio.

Cada uma das sinapses possui um peso associado a ela. A representação gráfica de um neurônio de McCulloch & Pitts é apresentado na Figura 4.2.

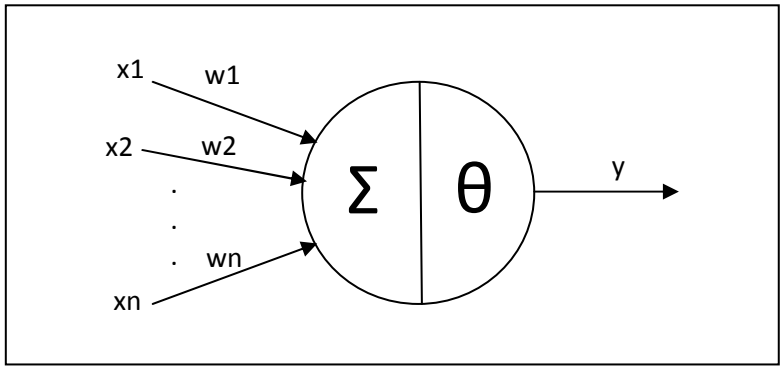

Figura 4.2 - Neurônio de McCullhoch & Pitts. Adaptado de Braga et al. (2000)

Na figura, x1, x2, ..., xn representam as entradas do neurônio; w1, w2, ..., wn, os pesos referentes a cada uma das entradas; y representa a saída do neurônio; Σ representa o somatório dos produtos das entradas pelos respectivos pesos; e θ, o limiar do neurônio.

O limiar funciona como uma função de ativação da saída do neurônio, que se torna ativa no somatório das entradas caso ultrapasse o seu valor de ativação ou inativa no caso contrário.

O funcionamento do neurônio McP é realizado com a excitação ou inibição de cada uma das entradas através de seus pesos, que podem ser positivos (excitatórios) ou negativos (inibitórios).

Semelhante ao ocorrido no nosso cérebro, o neurônio realiza um somatório ponderado dos produtos entre entradas e pesos recebidos pelo neurônio. Soma esta que é comparada a um limiar estabelecido. Se porventura o somatório ultrapassar o limiar, o neurônio fica ativo propagando o sinal 1, caso contrário, caracteriza a inatividade da célula processadora e o sinal transmitido é 0 (OLIVEIRA, 2005).

4.2.2 A Arquitetura

Como o neurônio artificial é a unidade de uma Rede Neural Artificial, é necessário dispô-los de alguma maneira que eles possam trabalhar em conjunto e assim formar a rede. A disposição dos neurônios é chamada de arquitetura.

Braga et al. (2000) mostram a importância da definição da arquitetura de uma RNA, visto que ela pode restringir o tipo de problema que pode ser tratado pela rede.

Uma arquitetura pode ser classificada de acordo com o número de camadas da rede, o número de neurônios por camada, o grau de conectividade e o tipo de conexões da rede. Em seguida, são apresentadas estas classificações:

- De acordo com o número de camadas:
    - Redes com uma camada: só existe um neurônio entre as entradas e a saída. Resolvem problemas linearmente separáveis;
    - Redes de múltiplas camadas: existe mais de um neurônio entre pelo menos uma entrada e a saída. As camadas intermediárias entre a entrada e a saída são chamadas de camadas ocultas.

Resolvem problemas complexos por conseguirem extrair estatísticas de ordem elevada (HAYKIN, 2001).

- De acordo com o número de neurônios por camada:
  - Piramidal: o número de neurônios diminui a cada camada;
  - Bloco: o número de neurônios permanece o mesmo a cada camada.
- De acordo com o grau de conectividade:
  - Completamente Conectada: todos os neurônios da camada anterior são conectados com todos os neurônios da camada posterior. Este tipo de rede aumenta a capacidade de generalização da rede, visto que aumenta a quantidade de conexões.
  - Parcialmente Conectada: não existe a conexão completa de todos os neurônios de uma camada anterior para com os da camada posterior.
- De acordo com o tipo de conectividade da rede:
  - *Feedforward*: as saídas dos neurônios apenas se conectam aos neurônios da camada posterior.
  - *Feedbackward*: as saídas dos neurônios de uma camada n se conectam aos da camada n+1, mas também podem existir conexões entre os neurônios de uma mesma camada.

A Figura 4.3 mostra alguns modelos de arquiteturas de RNAs e a descrição de sua classificação logo a seguir:

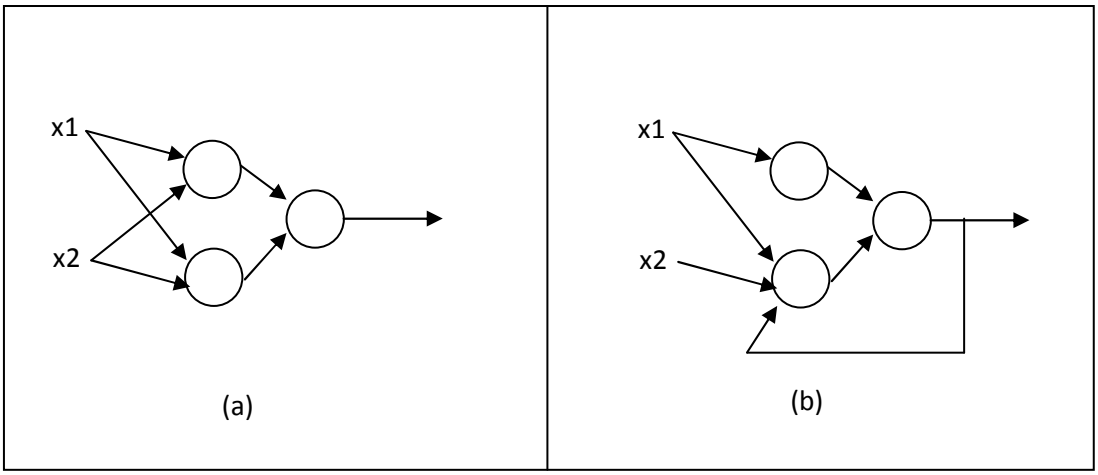

Figura 4.3 - Arquiteturas de RNAs

A Figura 4.3 (a) apresenta uma Rede Neural Artificial de múltiplas camadas, piramidal, completamente conectada e *feedforward*. Já a Figura 4.3 (b) mostra uma Rede Neural Artificial de múltiplas camadas, piramidal, parcialmente conectada e *feedbackward*.

4.2.3 Algoritmo de Aprendizagem

Segundo Braga et al. (2000), a aprendizagem é o processo pelo qual os parâmetros de uma Rede Neural Artificial são ajustados através de uma forma continuada de estímulo pelo ambiente no qual a rede está operando, sendo o tipo específico de aprendizagem definido pela maneira particular de como ocorrem os ajustes realizados nos parâmetros.

No processo de aprendizagem, os pesos sinápticos da rede são ajustados para encontrar as relações entre entradas e saídas.

Existe uma significativa quantidade de diferentes algoritmos de aprendizagem para Redes Neurais Artificiais, mas todos divididos em dois grandes grupos: os algoritmos supervisionados e os não supervisionados.

Os algoritmos supervisionados, que são os mais comuns, são fornecidos à RNA. Tanto as entradas quanto as saídas desejadas para cada uma dessas entradas fazem o ajuste dos pesos sinápticos para encontrar a relação entre as entradas e saídas desejadas, na proporção do erro máximo indicado. O supervisor da rede indica quando cada saída calculada pela rede teve um desempenho bom ou ruim, facilitando o ajuste dos pesos. A Figura 4.4 apresenta o modelo do aprendizado supervisionado.

Apesar de bastante eficiente, os algoritmos supervisionados não são capazes de, por própria iniciativa, aprender novos conhecimentos, visto que todos eles devem ser apresentados pelo tutor.

Já no aprendizado não supervisionado, o papel do tutor para acompanhar o treinamento da RNA inexiste, fazendo com que apenas os padrões de entradas sejam

apresentados à rede, tornando a própria rede capaz de classificar a saída da maneira mais adequada.

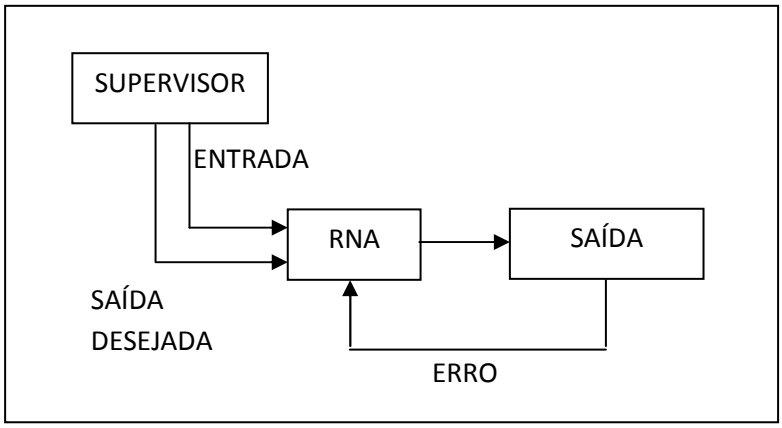

Figura 4.4 - Algoritmo Supervisionado - Adaptado de Braga et al. (2000)

Segundo Braga et al. (2000), quando uma rede estabelece uma harmonia com a regularidade estatística das entradas de dados, inicia o processo de formação de representações internas para representar as características criando novas classes ou grupos.

De acordo com Haykin (1991) um dos algoritmos de aprendizagem mais utilizados é o *BackPropagation*, que possui algumas variações.

Uma dessas variações, e a que será utilizada no estudo de caso deste trabalho, é a *Batch BackPropagation*, onde MELCIADES et al. (1999) o apresenta como um algoritmo supervisonado onde a atualização dos seus pesos apenas é realizada quando todo o conjunto de treinamento é apresentado a Rede Neural Artificial

4.3 Construção de uma RNA

O processo de construção de uma Rede Neural Artificial, segundo Tafner et al. (1995), possui quatro etapas: a definição, o treinamento, a utilização e a manutenção. Neste processo, diferentes dos sistemas convencionais, todas as etapas são desenvolvidas com foco nos dados, já que estes são os elementos essenciais para o

funcionamento da Rede Neural Artificial. A Figura 4.5 apresenta o processo de construção de uma Rede Neural Artificial.

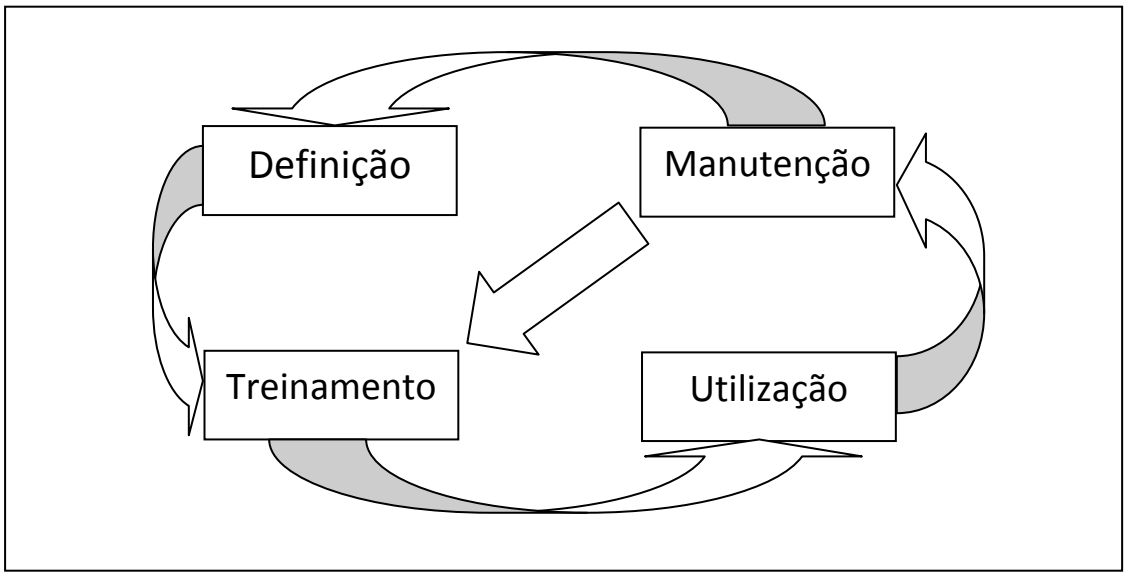

Figura 4.5 – Processo de Construção de uma RNA. Adaptado de Tafner et al. (1995)

A etapa da definição é responsável pelo estudo e pela descrição do problema a ser trabalhado. Normalmente este problema é bem direcionado, evitando generalizações de soluções, visto que os modelos neurais são pontuais em seu contexto. Além disso, ainda nesta etapa é formulado o modelo da Rede Neural Artificial a ser usado, entradas, saídas, tipo de neurônio e arquitetura da rede, bem como o algoritmo de aprendizagem a ser utilizado.

Com toda a definição desenvolvida é realizado o treinamento da Rede Neural Artificial, onde o treinamento se dá através de exemplos modelos de padrões de entrada e, se necessário, no caso dos algoritmos supervisionados, as saídas para cada entrada. O bom funcionamento da Rede Neural Artificial se dará mediante a convergência deste aprendizado.

Na etapa de utilização, a Rede Neural Artificial é colocada à prova. Ela é posta em ambiente de trabalho de onde se avaliará os resultados para a aplicação desenvolvida. No caso de um funcionamento insatisfatório existe a necessidade da

manutenção, que pode ser realizada tanto na definição quanto no treinamento, dependendo do problema identificado.

4.4 Aplicações de Redes Neurais Artificiais

Existe uma grande diversidade para as aplicações em Redes Neurais Artificiais. As mais comuns são prognósticos acerca de reconhecimento ótico de caracteres, análise de mercado utilizada por bancos e seguradoras e o auxílio na tomada de decisões na diagnose dos médicos. É importante ressaltar que estas aplicações não devem substituir os profissionais, mas apenas auxiliá-los no processo decisório.

A seguir são apresentadas algumas aplicações para Redes Neurais Artificiais desenvolvidas:

Mendes (2008) utilizou redes *perceptron* de múltiplas camadas para auxiliar o diagnóstico de pacientes com cefaléia, ou dor de cabeça, apresentando resultados satisfatórios neste diagnóstico.

O Centro Médico Social Comunitário de Vila Lobato, de Ribeirão Preto – SP, desenvolveu um sistema baseado em Redes Neurais Artificiais para a identificação do grau de vigilância de pacientes pediátricos. Nos testes preliminares a taxa de acerto chegou a 61,76% dos casos (POLLETTINI ET al., 2008).

Freiman e Pamplona (2005) construiu uma Rede Neural Artificial de múltiplas camadas treinada com algoritmo *back-propagation* para realizar a previsão do valor de *commodity* no agronegócio, salientando que para a realização de previsões é utilizada atividades cognitivas de baixo nível.

A identificação de áreas cafeeiras no interior de Minas Gerais, através de aplicação de Rede Neural Artificial para a classificação de dados de sensoriamento remoto, foi trabalho de Andrade et al. (2010), chegando a atingir o percentual de 60,29% de acerto nessa identificação.

Raviran e Wahidabanu (2008) utiliza uma Rede Neural Artificial para realizar o Controle de Concorrência de banco de dados integrados de manufaturas. Este tipo de banco de dados sofre um número constante e alto de atualizações, onde os *designers* que alteram os produtos o tentam fazê-lo mediante a opinião de diversos clientes.

A questão é que se dois ou mais *designers* estiverem trabalhando no mesmo projeto algumas alterações podem ser perdidas. Este trabalho consiste no desenvolvimento de uma Rede Neural Artificial que controle o bloqueio dos objetos através de suas transações. Os autores ainda justificam o fato de que as Redes Neurais Artificiais ainda consumirem menos espaço de memória no armazenamento de informações de bloqueio.

Foi utilizado um algoritmo de aprendizagem de *backpropagation*. Este trabalho, dentro das pesquisas realizadas, foi o único a tratar a utilização de Rede Neural Artificial para técnicas de Controle de Concorrência.

Outros estudos realizados que comprovam a diversidade da utilização de Redes Neurais Artificiais são: Zanella Junior (2009) utilizou Rede Neural Artificial para o estudo de equilíbrio de troca iônica de sistemas binários e ternários; Corrêa et al. (2005) realizaram uma análise do tempo de formação de graduandos, usando as Redes Neurais Artificiais; Araújo et al. (2006) utilizaram Rede Neural Artificial para indexar e recuperar automaticamente imagens médicas com diferentes padrões de texturas; O uso de Rede Neural Artificial para a determinação da necessidade de adubação da goiabeira com agricultura de precisão, para o elemento químico fósforo, foi desenvolvido por Silva et al. (2004).

4.5 Implementação de uma Rede Neural Artificial

O processo de construção de uma Rede Neural Artificial pode ocorrer de três maneiras distintas: através de hardware, desenvolvimento de software específico ou utilização de um simulador. Cada uma das maneiras possui vantagens e desvantagens onde as principais são abordadas na Tabela 4.1.

Tabela 4.1 - Vantagens e Desvantagens dos Métodos de Implementação de uma RNA.

	Vantagens	Desvantagens
Hardware	• Velocidade de processamento • Aplicações específicas	• Alto custo • Alta tecnologia • Demora na visualização de resultados
Software	• Aplicações específicas • Possibilidades diversas na utilização de funções de ativações de neurônios e algoritmos de aprendizagem	• Maior complexidade de desenvolvimento • Requer conhecimento específico de uma linguagem
Simulador	• Simplicidade de configuração e manutenção • Fácil visualização preliminar dos resultados obtidos pela RNA • Baixo custo	• Pouco aplicável, visto a sua interface com outros sistemas não ser amigável • Limitação no uso de neurônios ou algoritmos de aprendizagem

Considerando o descrito na tabela é importante observar que tanto o desenvolvimento em hardware quanto em software de uma Rede Neural Artificial possuem vantagem no desenvolvimento de aplicações específicas, porém o resultado da execução da rede é demorado, pois necessita de todo um processo de construção do mesmo.

Em outra via, o desenvolvimento de uma Rede Neural Artificial através de simulação tem como um ponto negativo a dificuldade de integração desta com outros sistemas, já que a adequação deve ser realizada de acordo com o simulador que está se utilizando e dificilmente este possui interfaces amigáveis com outros softwares.

A opção por utilizar um simulador neste trabalho advém do fato de que a análise do resultado final do processo de seleção entre um algoritmo de controle de concorrência é o ponto fundamental e os simuladores tornam mais fáceis as possíveis reconfigurações da Rede Neural Artificial até se encontrar um conjunto de parâmetros ideal para estes resultados.

Existem vários simuladores de Redes Neurais Artificiais disponíveis na Internet, alguns livres, como é o caso do *Studgard Neural Network Simulator* (SNNS), desenvolvido pela Universidade de Studgard, e alguns pagos, que disponibilizam a sua versão *trial* para verificação. Nestes últimos se encontram diversos como o *Neural Planner,* o E*asyNN-Plus* e o *NeuroDiet*.

Após a realização de uma breve avaliação dos softwares SNNS, *Neural Planner* e *EasyNN-Plus*, foi constatado que para fins deste trabalho a utilização de qualquer um deles seria adequada, pois conseguiriam trazer os resultados esperados para a simulação desejada, daí a escolha pelo *Neural Planner* por ter uma interface simples de se utilizar.

Na seção seguinte é realizada uma breve descrição sobre este simulador.

4.5.1 O Neural Planner

O *Neural Planner* 4.1 (versão Trial) sem limitações de uso é um simulador para desenvolvimento de Rede Neural Artificial desenvolvido por Stephen Wolstenholme. Ele possui uma interface amigável e utiliza neurônios que têm como função de ativação a sigmóide, formada pelo somatório da multiplicação das entradas dos neurônios pelos respectivos pesos dos seus dentritos. Este software foi a base para construção de outro simulador citado anteriormente, o *EasyNN-Plus*, hoje da empresa *Neural Planner Software Ltda.*

A utilização do software se dá basicamente no processo de criação de dois arquivos: um onde é criada a arquitetura da rede, chamado pelo autor de *Network File*, com extensão .nnp, e o outro usado para inserir os modelos de treinamento, teste e interrogação da Rede Neural, chamado de *Training, Testing, Interrogating File* e com extensão .tti.

O software fornece alguns modelos de exemplos, de onde será utilizado neste trabalho o exemplo *logic* para apresentação do software.

O exemplo citado é bastante simples e de fácil entendimento. É uma rede neural que possui dois neurônios de entrada, que representam dois valores lógicos, chamados de “A” e “B” e três neurônios de saída, que formam as portas lógicas *AND*, *OR* e *XOR*. Na Figura 4.6, pode-se observar a apresentação dos dois arquivos através da interface do *Neural Planner*.

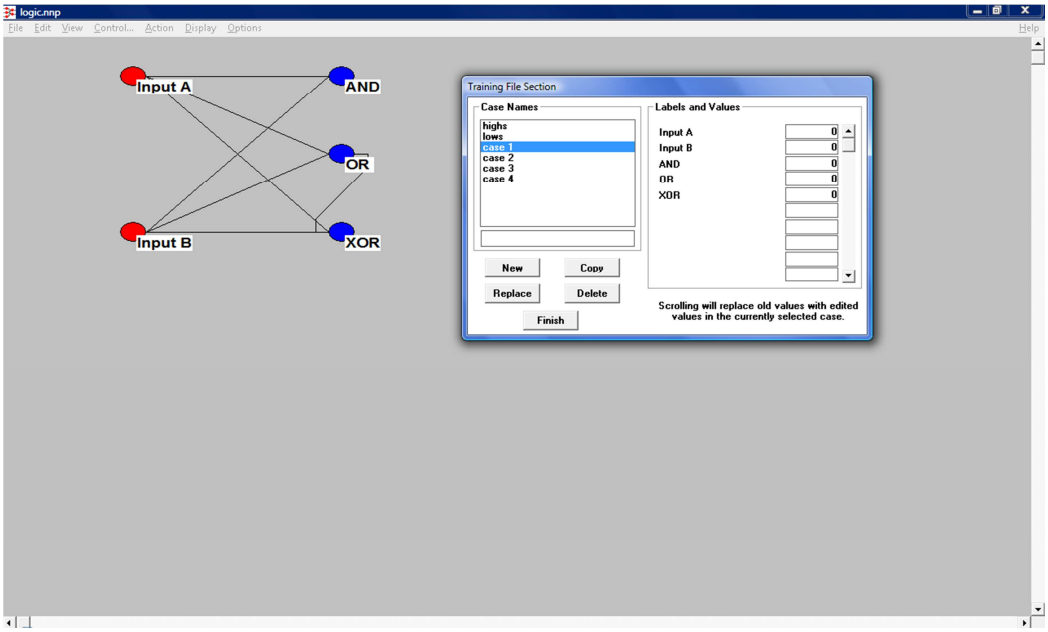

Figura 4.6 - Tela do Neural Planner com o exemplo logic.nnp

A parte esquerda da Figura 4.6 apresenta o arquivo da rede neural (.nnp), os neurônios de entrada e saída, bem como as sinapses entre eles e a parte direita da mesma figura mostra o arquivo de treinamento, teste e interrogação (.tti) com os seus valores setados.

O aprendizado é dado por meio da opção *Action/Learn From File* onde a rede neural é treinada através do arquivo de treinamento, ou seja, é realizado um treinamento supervisionado a partir de modelos de entrada e saídas fornecidos anteriormente.

Sendo este o aplicativo a ser utilizado, o capítulo seguinte trata do desenvolvimento do estudo de caso deste trabalho.

Capítulo 5

Redes Neurais Artificiais para o Controle de Concorrência em Sistemas de Tempo Real

Pela razão de um Sistema de Tempo Real poder apresentar durante o seu funcionamento uma série de momentos, onde, é possível existir uma diferença de situações nesses momentos influenciados pelo fluxo de tarefas processadas, pela prioridade destas tarefas ou ainda outros fatores relacionados a questões técnicas, não necessariamente é necessário utilizar o mesmo protocolo de Controle de Concorrência em todos os instantes.

Exemplificando esta diferença de situações, pode-se considerar uma agência bancária, que possui no seu dia a dia uma gama de situações distintas, por exemplo:

- Ao iniciar o dia, ainda com a agência fechada, o fluxo de transações bancárias é executado apenas nos terminais de autoatendimento e através da Internet;
- Ao se iniciar o expediente ao público, existe uma mudança no fluxo de ocorrência, visto que além dos métodos já citados, os caixas da agência também iniciam seu fluxo normal de trabalho, persistindo assim até o horário de fechamento das agencias;

- Quando acontece o fechamento da agência, o fluxo passa novamente a ficar restrito aos terminais e à Internet, não significando dizer que existe uma diminuição no fluxo de tarefas processadas;
- Após um horário específico, por questões de segurança pública, os terminais de autoatendimento são desligados, sendo as transações bancárias executadas apenas pela Internet;
- Durante a noite, as câmaras de compensação, nome dado ao setor que realiza o processamento entre bancos e agências, realiza os seus processamentos de troca de informações e conferência de documentos.

Além do exemplo acima, pode-se observar em outros sistemas o mesmo tipo de diferentes fluxos, como no comércio eletrônico, onde dependendo da hora do dia, pode-se ter um maior ou menor fluxo de acessos.

Para este estudo de caso foram considerados dois protocolos de Controle de Concorrência. Um baseado em *TimeStamp*, para ser utilizado em momentos no qual as características de um sistema remetam trabalhar em baixos fluxos de tarefas ou com tarefas de baixa prioridade, e um algoritmo baseado em bloqueios para funcionar nos horários de maior fluxo de tarefas ou com várias tarefas de alta prioridade.

5.1 Caracterização do Problema

Considerando que os sistemas possuem padrões de funcionamento distintos, dependendo do instante de tempo em que eles estão sendo utilizados, e o uso das técnicas de Controle de Concorrência poderiam se adequar a estes momentos, torna-se importante definir a situação de atuação da Rede Neural Artificial.

Para este estudo de caso foi considerado que as tarefas do sistema possuem uma tupla, que é formada por: $<T_{LE}, T_P, T_I, T_E, T_T, T_{PC}>$, onde:

- T_{LE} – Indica se a tarefa é de leitura ou Escrita;
- T_P – Define se a prioridade da tarefa é alta, média ou baixa;
- T_I – Item de dado que a tarefa acessa;

- T_E – Tempo da tarefa;
- T_T – Indica se a tarefa é crítica ou Branda;
- T_{PC} – Define o padrão de chegada entre periódico, aperiódico e esporádico.

Essas características são importantes para a modelagem da arquitetura da Rede Neural Artificial.

Além desta tupla, foi definido que a cada cinquenta instantes de tempo seriam realizadas a verificação de qual algoritmo de Controle de Concorrência utilizar no Sistema de Tempo Real. A Figura 5.1 mostra um diagrama de sequencia de mensagem que representa essa verificação.

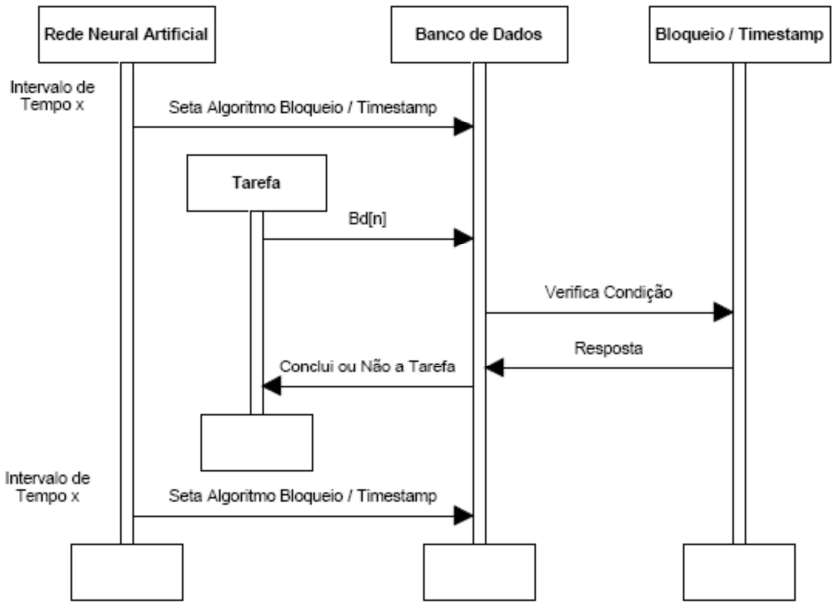

Figura 5.1 - Diagrama de Sequencia de Mensagens RNA

A cada intervalo de tempo x definido, no caso deste estudo cinquenta, a Rede Neural Artificial verifica e seta no banco de dados qual o algoritmo de Controle de Concorrência deve ser utilizado no próximo intervalo temporal.

A partir daí o Sistema de Tempo Real funciona normalmente até que outro tempo x chegue e interfira novamente analisando qual o algoritmo utilizar.

A definição de um intervalo de tempo para esta verificação se deu pelo fato de que poderia ser muito oneroso ao sistema, em relação ao tempo, se a cada concorrência verificada fosse analisado que algoritmo utilizar. Como aqui se fala de Sistemas de Tempo Real, o tempo perdido a cada verificação poderia ser de extrema relevância para o não funcionamento do sistema.

5.2 Definição da Rede Neural Artificial

Como visto na Figura 4.5, que apresenta o processo de construção de uma RNA. O trabalho deve ser iniciado pela etapa de definição, onde é construída a arquitetura da rede, com seus neurônios de entrada e saída bem definidos.

A escolha pela representação destes neurônios se deu com base nas classificações dos Sistemas de Tempo Real inseridas na Seção 2.2 deste trabalho, sendo escolhidos os cinco padrões para entrada e dois para saída, de acordo com o que é apresentado na Tabela 5.1.

Tabela 5.1- Padrões de Entrada/Saída para a RNA

Padrão	Entrada ou Saída	Possíveis Valores
Tipo da Operação da Tarefa	Entrada	1- Leitura **2- Escrita**
Prioridade das Tarefas	Entrada	**1- Alta** 2- Média 3- Baixa
Tempo Computacional da Tarefa	Entrada	**1- Alto** 2- Baixo
Tipo das Restrições de Tempo Real	Entrada	1- Branda **2- Crítica**
Periodicidade da Tarefa	Entrada	1- Periódica **2- Aperiódica** 3- **Esporádico**
Algoritmo de *TimeStamp*	Saída	0- Não Utiliza 1- Utiliza
Algoritmo de Bloqueio de Duas Fases (2PLHP)	Saída	0- Utiliza 1- Não Utiliza

É importante visualizar que dentro destes padrões de entrada foram colocadas questões relevantes sobre as tarefas e o instante de funcionamento do sistema, visto que ele usa os últimos cinquenta tempos computacionais para a tomada de decisão.

O valor correspondente para cada uma das entradas da rede neural artificial é o quantitativo de tarefas que aconteceu no intervalo de tempo anterior com as seguintes características: Para o tipo de operação da tarefa, as tarefas de escrita; para a prioridade das tarefas, as de alta prioridade; para o tempo computacional, as que possuíam um alto tempo; Para as restrições, as tarefas críticas; e para a periodicidade da tarefa, as que possuem periodicidade esporádica adicionadas as aperiódicas.

Esses padrões apresentam a criticidade do momento de funcionamento da aplicação, quanto mais tarefas possuírem essas características mais existe a possibilidade de se encontrar concorrência entre as mesmas.

Os dois padrões de saída representam os modelos de Controle de Concorrência utilizados para este estudo de caso: *TimeStamp* e 2PLHP.

Após a definição dos padrões de Entrada e Saída, foi realizada a montagem da arquitetura da Rede Neural Artificial utilizando a ferramenta *Neural Planner*. Esta construção se deu utilizando uma arquitetura contendo além dos neurônios de entrada e saída, definidos anteriormente, uma camada oculta contendo três neurônios para o melhor balanceamento da Rede Neural Artificial. A Figura 5.2 apresenta as telas do N*eural* Planner com esta arquitetura.

Com a definição da arquitetura da Rede Neural Artificial, foi elaborado o arquivo de treinamento da mesma com o intuito de realizá-lo de maneira supervisionada, visto que foram apresentados os modelos de Entrada e Saída da rede.

O arquivo de treinamento consta de 30 (trinta) modelos de treinamento e 11 (onze) para teste.

A tabela com todos os modelos de treinamento e testes aplicados a este trabalho está no anexo A deste trabalho.

A tabela de treinamento foi construída visando a rigidez e criticidade do momento da aplicação.

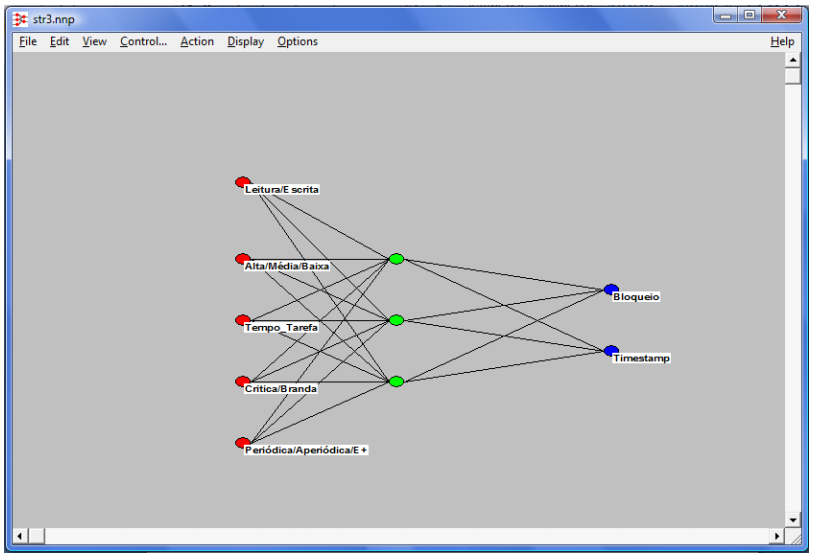

Figura 5.2 - Arquitetura da RNA

Quanto mais rígidos estivessem os parâmetros em um determinado instante, ou seja quanto mais tarefas acontecessem com características que aumentassem a probabilidade da geração de concorrência, o algoritmo que teria maior importância a ser utilizado seria o 2PLHP e quanto mais suave o mesmo estivesse seria utilizado o algoritmo de *Timestamp*.

Após a elaboração do arquivo de treinamento e teste, foi realizado o processo de aprendizagem através do algoritmo *Batch Back Propagation*. Nele, a mudança dos pesos nas sinapses é acumulada a cada apresentação dos modelos, porém os pesos apenas são ajustados uma vez, ao final do ciclo de apresentação. O método de treinamento *On-Line Back Propagation*, que atualiza os pesos das sinapses após cada ciclo de aprendizagem e também existente no simulador, não foi utilizado por apresentar uma taxa de aprendizado mais lenta que o anterior.

A princípio uma Rede Neural Artificial de uma camada (sem a camada oculta) foi testada nas mesmas condições e com o mesmo arquivo de treinamento, porém esta não obteve convergência para o problema.

5.3 Verificação do Funcionamento da Rede Neural Artificial

A aplicação dos modelos para teste, além de utilizar os 11(onze) modelos previamente definidos, ainda utilizou de quatro dos modelos do treinamento, selecionados aleatoriamente.

Essa metodologia foi realizada através da comparação dos resultados produzidos pela Rede Neural Artificial com os resultados esperados para que esta produzisse a real situação de seu funcionamento.

Na avaliação dos resultados apresentados pelo *Neural Planner*, foram percebidos os seguintes pontos relacionados ao treinamento:

- A convergência da Rede Neural Artificial se deu rapidamente, com 3642 ciclos, atingindo em 96% dos casos a meta do menor erro indicado pelo simulador, setado em 0,05, como mostra a Figura 5.3.

Para a utilização da Rede Neural Artificial não existiu um comprometimento do processamento da máquina, visto que neste apenas a saída foi calculada a partir da entrada apresentada. Sendo assim, não existe um impacto direto sobre o processamento do Sistema de Tempo Real a partir do processamento da máquina.

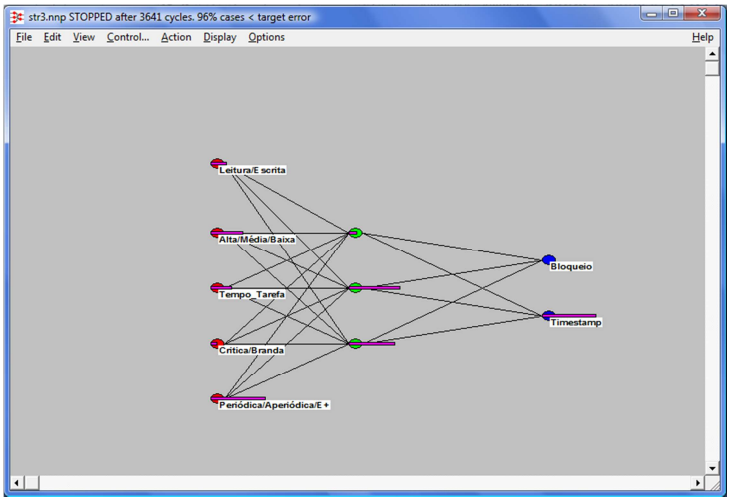

Figura 5.3 - Aprendizado da RNA

Na análise dos resultados onde foram aplicados os modelos de testes, pode-se observar do aprendizado a seguinte caracterização. De acordo com a Tabela 5.2, B significa Bloqueio e T *TimeStamp*.

Tabela 5.2 - Verificação do Aprendizado da RNA

Exemplo	Resultado Esperado	Resultado Obtido com a RNA
01	T	T
02	B	B
03	T	T
04	T	T
05	B	B
06	B	B
07	B	B
08	T	B
09	B	B
10	B	B
11	T	T
12	B	T
13	T	T
14	B	B
15	T	T

De acordo com a Tabela 5.2, em treze dos quinze modelos apresentados, a Rede Neural Artificial apresentou o resultado esperado, o que equivale a um percentual de 86,6% de acerto.

Nos dois modelos apresentados que ocorreram os erros na seleção do algoritmo, os valores para a decisão estavam muito próximos do limiar da Rede Neural Artificial, pois existia uma boa divisão entre as características apresentadas como ideais na utilização do bloqueio com as características apresentadas como ideais para a utilização do *TimeStamp*.

Verificando o bom aprendizado realizado pela Rede Neural Artificial, que significa a seleção do algoritmo adequado para utilizar a cada momento, falta realizar a verificação se a técnica de realizar as trocas de algoritmos no processo de solução dos Sistema de Tempo Real pode ser efetuada com sucesso. Essa verificação é realizada na seção que segue.

5.4 Validação

Para a validação do modelo apresentado foi desenvolvido um programa em linguagem C que simula os três algoritmos citados: O de bloqueio, o de *Timestamp* e utilização dos dois algoritmos juntamente com a Rede Neural Artificial.

Assim, um vetor bidimensional com mil e oitocentas instâncias é gerado aleatoriamente com Tuplas $<T_{LE}, T_P, T_I, T_E, T_T, T_{PC}>$, onde percentualmente se tem:

- T_{LE} – Leitura (50%) ou Escrita (50%);
- T_P – Alta (20%), Média (40%) ou Baixa (40%);
- T_I – Item de dado que a tarefa acessa, entre dez itens;
- T_E – Tempo da tarefa, entre um e dez;
- T_T – Crítica (20%) ou Branda (80%);
- T_{PC} – Periódico (40%), Aperiódico (30%) ou Esporádico (30%).

Cada uma das instâncias representa uma unidade de tempo. Após a geração deste vetor, é realizada uma cópia do mesmo a fim de manter a estrutura para as três simulações supracitadas. Cada um dos algoritmos é utilizado no mesmo vetor e ao final é gerada uma indicação de quantas das tarefas foram concluídas e quantas não conseguiram terminar.

Tanto o algoritmo de Bloqueio quanto o de *Timestamp* levam em consideração para o encerramento das tarefas o seu tempo computacional e o seu prazo, acrescendo este tempo em mais três unidades, sendo este o parâmetro utilizado para a definição da aceitação de atraso de cada tarefa.

O algoritmo que simula a Rede Neural Artificial a cada cinquenta instantes de tempo realiza uma verificação das características, dentre as que formam a tupla, que oneram o Banco de Dados.

Foram atribuídos pesos positivos e negativos para cada uma das características e ao final é feito um somatório para a seleção de qual dos dois algoritmos utilizar. Daí o algoritmo selecionado será executado durante as cinquenta interações seguintes, quando uma nova seleção é realizada.

O algoritmo foi executado vinte vezes para buscar uma média dos resultados e assim poder gerar uma comparação entre os três métodos utilizados, tendo a Tabela 5.3 como indicativa do resultado.

Observando os resultados, percebe-se que o algoritmo utilizando a Rede Neural Artificial tem um resultado semelhante ao algoritmo de bloqueio, onde em um universo de trinta e seis mil tarefas executadas apenas cento e setenta e uma (0,475%) deixaram de ser comprometidas com o Banco de Dados. No algoritmo de bloqueio sessenta (0,1667%) não se comprometeram.

Comparando o resultado com o algoritmo de *TimeStamp*, o resultado foi bem superior, deixando de comprometer duas mil, oitocentos e oitenta e uma tarefas, chegando a um percentual de 8,002% das tarefas.

Deve ser considerado que os valores para o vetor que representa as tuplas das tarefas foi gerado aleatoriamente, não representando com fidedignidade os ambientes onde os Sistemas de Tempo Real trabalham. Dessa maneira, o resultado esperado pelo funcionamento do algoritmo fica um pouco aquém do imaginado, pois não existe a padronização esperada para o ambiente.

Outro ponto importante a ser verificado é que no algoritmo com a Rede Neural Artificial existem os momentos mais críticos, que é onde ocorre a análise de qual algoritmo utilizar, em intervalo de tempos regulares.

Tabela 5.3 - Resultado da Simulação

	Bloqueio			*Timestamp*			RNA		
Interação	Não completos	Completos	% acerto	Não completos	Completos	% acerto	Não completos	Completos	% acerto
1	2	1798	99,8889	1	1799	99,9444	6	1794	99,6667
2	3	1797	99,8333	167	1633	90,7222	10	1790	99,4444
3	4	1796	99,7778	1	1799	99,9444	10	1790	99,4444
4	3	1797	99,8333	2	1798	99,8889	5	1795	99,7222
5	3	1797	99,8333	1	1799	99,9444	5	1795	99,7222
6	3	1797	99,8333	350	1450	80,5556	11	1789	99,3889
7	2	1798	99,8889	355	1445	80,2778	10	1790	99,4444
8	4	1796	99,7778	1	1799	99,9444	5	1795	99,7222
9	2	1798	99,8889	383	1417	78,7222	9	1791	99,5
10	3	1797	99,8333	178	1622	90,1111	10	1790	99,4444
11	2	1798	99,8889	176	1624	90,2222	9	1791	99,5
12	2	1798	99,8889	1	1799	99,9444	5	1795	99,7222
13	3	1797	99,8333	173	1627	90,3889	8	1792	99,5556
14	5	1795	99,7222	533	1267	70,3889	17	1783	99,0556
15	5	1795	99,7222	1	1799	99,9444	8	1792	99,5556
16	1	1799	99,9444	1	1799	99,9444	4	1796	99,7778
17	4	1796	99,7778	183	1617	89,8333	12	1788	99,3333
18	3	1797	99,8333	185	1615	89,7222	10	1790	99,4444
19	3	1797	99,8333	2	1798	99,8889	7	1793	99,6111
20	3	1797	99,8333	187	1613	89,6111	10	1790	99,4444
Totais	60	35940	99,8333	2881	33119	91,9972	171	35829	99,525

Esses instantes justificam a perda de algumas das tarefas, visto que antes de mudar o algoritmo todas as tarefas iniciadas anteriormente devem ser encerradas, comprometidas ou não, o que gera um pequeno atraso nas que se iniciam.

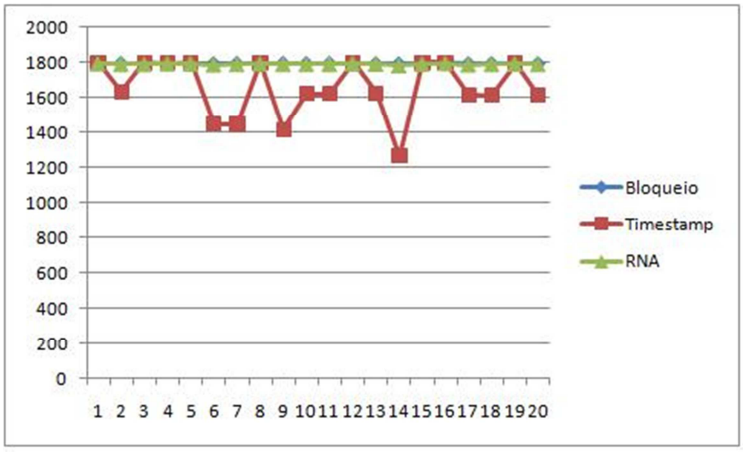

Figura 5.4 - Resultado da simulação

Observando graficamente os resultados atingidos na Figura 5.4 pode-se observar que praticamente não existe diferença entre a solução de Bloqueio com a utilização da Rede Neural Artificial, enquanto que o algoritmo que utiliza a técnica de *TimeStamp* em vários momentos se torna menos eficaz que a solução proposta.

Desta maneira a utilização do algoritmo que utiliza a Rede Neural Artificial para a seleção de qual a técnica de Controle de Concorrência utilizar pode ser considerada bastante satisfatória, visto que esta técnica ainda não é explorada na literatura e com resultados bastante semelhantes ao obtido com a técnica de bloqueio, que dentre as duas utilizadas foi a que apresentou melhores resultados.

Capítulo 6

Contribuições e Trabalhos Futuros

A utilização de Sistemas de Tempo Real está cada vez mais presente no dia a dia das pessoas, sejam em aplicações rígidas como o acompanhamento de pacientes em leitos hospitalares, onde estes são monitorados por sistemas, sejam em aplicações simples, como os presentes nos eletrodomésticos.

Como o tempo é uma das maiores restrições para estes sistemas, é importante que eles funcionem da maneira mais adequada possível, tornando as suas aplicações inertes aos ruídos que os sistemas podem ocasionar.

Baseado neste preâmbulo foi verificado que um dos problemas que podem afetar diretamente este resultado é a escolha da técnica de Controle de Concorrência a ser utilizada, que apesar de existirem várias técnicas eficientes, elas não podem ser executadas em um mesmo sistema de maneira concomitante.

A utilização de uma única técnica específica pode, de alguma maneira, resultar em impasses do sistema, no caso de técnicas de bloqueio, ou operações que são reiniciadas a todo o momento ou sequer conseguem ser realizadas nos algoritmos de *timestamp*.

Algumas dificuldades foram verificadas no decorrer desta pesquisa, inerentes aos próprios Sistemas de Tempo Real e a utilização de Redes Neurais Artificiais para resolução de seus problemas, visto que nenhum material bibliográfico foi encontrado

tratando estes dois assuntos de forma evidenciada e o único encontrado que trabalhava com Controle de Concorrência e Redes Neurais Artificiais não eram direcionados ao estudo de Sistemas de Tempo Real.

6.1 Contribuições

A principal contribuição desta dissertação se dá pelo fato desta abrir uma discussão sobre a utilização de mais de uma técnica de controle de concorrência para a utilização em Sistemas de Tempo Real, através de uma seleção prévia e sistemática realizada por uma Rede Neural Artificial, baseada no reconhecimento de padrões de concorrências. Este fato pode fazer com que atrasos ou perdas consideráveis na utilização deste tipo de sistemas sejam minimizados.

Pode-se destacar ainda:

- A especificação de um modelo neural simplificado para tratar questões relativas ao controle de concorrência e;
- A definição de padrões para a utilização de técnicas específicas para o controle de concorrência.

6.2 Trabalhos Futuros

Analisando os aspectos teóricos e os resultados obtidos através da utilização de Redes Neurais Artificiais no tratamento do Controle de Concorrência dos Sistemas de Tempo Real, prevemos que este processo pode ser bastante útil no desenvolvimento de Controles e aplicado aos sistemas nos seus respectivos domínios.

Para que isto possa ser possível, é essencial que esta pesquisa seja amadurecida, já que é pioneira. Daí terem sidos identificados alguns pontos nesta

dissertação que podem ser explorados para uma melhor observação dos resultados, sendo:

- Análise de outras arquiteturas de Redes Neurais Artificiais, bem como outras composições de Neurônios e Algoritmos de Aprendizagem que possam melhorar os resultados obtidos;
- Especificação formal do problema tratado, utilizando uma linguagem específica para tal;
- Desenvolver uma Rede Neural Artificial através de uma linguagem de programação e de como realizar a interface da mesma com um Banco de Dados de Sistemas de Tempo Real, fazendo com que realmente haja a interferência da mesma no processo de escolha da técnica de Controle de Concorrência.

REFERÊNCIAS

ADYA, A., GRUBER, R., LISKOV, B., MAHESHWARE, U. **Efficient Optimistic Concurrency Control Using Losely Synchronized Clocks**. ACM SIGMODInternational Conference on Manegement of Data. San Jose, CA, Maio, 1995.

ANDRADE, L. N., VIEIRA, T. G. C., ALVES, H. M. R., VOLPATO, M. M. L., SOUZA, V. C. O. **Aplicação de Redes Neurais Artificiais (RNA) na Análise e Classificação de Áreas Cafeeiras da Região de Machado – MG.** IX Congreso Latinoamericano y Del Caribe de Ingeniería Agrícola. Vitória – ES, 2010.

ARAÚJO, S. A. de; LIBRANTZ, A. F. H.; ALVES, W. A. L. **Ferramenta para Indexação e Recuperação Automática de Imagens Médicas.** Exacta, São Paulo, v.4, n. especial, p. 75-77, 25 nov. 2006.

AUDSLEY, N.; BURNS, A. ***Real-Time System Scheduling.***Predictably Dependable Computer Systems, Cap. 2, Vol. 2, mai. 1990.

BRAGA, A. P.; CARVALHO, A. C. P. L. F., LUDEMIR, T. B. **Redes Neurais Artificiais: Teoria e Aplicações**. Rio de Janeiro: LTC, 2000.

CASANOVA, M. A., MOURA, A. V., **Princípios de Sistemas de Banco de Dados Distribuídos.** Rio de Janeiro: PUC-Rio: 1999.

CARVALHO, J. S. **Verificação Automatizada de Sistemas de Tempo Real Críticos**. Dissertação de Mestrado. Universidade da Beira Interior, 2009.

CORREA, A. C. C., CAMARGO, D. R. de, VIANA, A. B. N. **Aplicação de Redes Neurais Artificiais para Previsão do Tempo de Titulação dos Graduandos.** XXV Encontro Nacional de Engenharia da Produção. Porto Alegre – RS, 2005.

DATE, C. J. **Uma Introdução a Sistemas e Banco de Dados.** São Paulo: Addison Wesley, 1999.

DATTA, A., SON, S. H. **A Study of Concurrency Control in Real-Time, Active Database Systems**. IEEE Transactionson Knowledge and DataEngineering, vol. 14 n° 3, Maio/Junho, 2002.

ELMASRI, R. NAVATHE, S. B. **Sistemas de Banco de Dados.** São paulo: Pearson Addison Wesley, 2005.

FARINES, J.M., FRAGA, J da S., OLIVEIRA, R. S. **Sistemas de Tempo Real.** Universidade Federal de Santa Catarina, Jul, 2000.

FERNANDES, Y. Y. M. P. **Técnica de Controle de Concorrência Semântico para Sistemas de Gerenciamento de Banco de Dados em Tempo-Real.** Dissertação de Mestrado, UFCG. Campina Grande, 2005.

FREIMAN, J. P., PAMPLONA, E. de O. **Redes Neurais Artificiais na Previsão do Valor de *Commodity* do Agronegócio.** V Encuentro Internacional de Finanzas. Santiago, Chile, 2005.

FREITAS, E. P. **Metodologia Orientada a Aspectos para a Especificação de Sistemas Tempo-Real Embarcados Distribuídos.** Dissertação de Mestrado, UFRGS. Porto Alegre, 2007.

HAYKIN, S. **Redes Neurais: Princípios e Práticas.** 2 ed. Porto Alegre: Bookman, 2001.

HAYKIN, S. **Neural Networks. A Comprehensive Foundation.** Macmilian College Publishing Company, 1991.

HUNG, V. HUONG, V. **Modeling Real-time database system in Duration Calculus**. [S.l.]. August, 2002.

JOSEPH, M. Problems, ***Promises and Performance: Some Questions for real-time system specification***.Lecture Notes. Ed. Springer-Verlag, Junho, 1991.

KOSHAFIAN, S. **Banco de Dados Orientado a Objeto**. Tryte Informática. – Rio de Janeiro: Infobook, 1994.

KOT, M. **Modeling Real-TimeDatabase Concurrency Control Protocol Two-Phase-Locking in Uppaal**. Proceedings of The International Multicinference on Computer Science and Information Technology pp. 673-678. IEEE, 2008.

LEITE, C. R. M. **Linguagem de Consulta para Aplicações em Tempo-Real.** Dissertação de Mestrado, UFCG. Campina Grande, 2005.

LINDSTROM, J. **Optimistic Concurrency Control Methods for Real-time Database**. Tese (doutorado) – Department of Computer Science, University of Helsinki – Finland, January, 2003.

LIU, J. W. S. **Real-Time Systems**. 1. ed. 2000.

McCULLOCH, W; PITTS, W. *A logical Calculus of the Ideas Immanent in Nervous Activity. Bulletin of mathematical Biophysics*, 5:115-133, 1943.

MELCIADES, W. FIALLOS, M., PIMENTEL, C. **Paralelização do algoritmo "*BackPropagation*" em Clusters de Estações de Trabalho.** IV Congresso Brasileiro de Redes Neurais PP. 231-236. ITA, São José dos Campos, São Paulo – Brasil, 1999.

MENDES, K. B. **O Uso de Redes Neurais Artificiais no Diagnóstico Preditivo dos Tipos Mais Frequentes de Cefaléia.** Dissertação de Mestrado, Universidade Federal do Paraná, PR, 2008.

MOIZ, S. A., RAJAMANI, L. **A Real Time Optimistc Strategy to Achieve Concurrency Control in Mobile Enviroments Using On-Demand Multicasting**. International Journal of Wireless & Mobile Network (IJWMN), Vol. 2, N° 2, Maio, 2010.

MOTUS, L. ***Time Concepts in real-time software***. WRTP'92. Bruge, junho, 1992.

NYSTRÖM, D., NOLIN, M., Tesanovic, A., NORSTRÖM, Ch., HANSSON, J.: **Pessimistic Concurrency-Control and Versioning to Support Database Pointers in Real-Time Databases**. Proc. of the 16th Euromicro Conference on Real-Time Systems, pages 261-270, IEEE Computer Society, 2004.

OLIVEIRA, J. D. F. **Um Estudo Sobre o Reconhecimento de Caracteres Utilizando Redes Neurais.** Monografia de Graduação, UERN, 2005.

ÖZSU, M. T., VALDURIEZ, P. **Princípios de Bancos de Dados Distribuídos.** Rio de Janeiro: Campus, 2001.

PAVLOVA, E., HUNG, D.V. **A Formal Specification of the Concurrency Control in Real-Time Databases**. IEEE, 1999.

POLLETTINI, J.T., TINÓS, R. PANICO, S., DANELUZZI, J. C., MACEDO, A. A. **Classificação Automática de Pacientes para Atendimento Médico Pediátrico Multidisciplinar a Partir do Seu Grau de Vigilância.** Anais do XXVIII Congresso da SBC – WIN – Workshop de Informática Médica. P. 61-70. Belém do Pará – PA, 2008.

QILONG, H., ZHONGXIAO, H. **Real Time Optmistic Concurrency Control based on Transaction Finish Degree**. Journal of Computer Science 1, p. 471-476. Science Publications, 2005.

RAMAMRITHMAN, K. **Real-Time Databases**. Distributed and Parallel Databases. v. 1, 1993. 199-226 p.

RAVIRAM, P.; WAHIDABANU, R. S. D. **Implementation of Artificial Neural Network in Concurrency Control of Computer Integrated Manufacturing (CIM) Database.** International Journal of Computer Science & Security, Vol. 1, n. 5, 2008.

RIBEIRO NETO, P. F. **Mecanismos de Qualidade de Serviço para o Gerenciamento de Dadose Transações em Tempo-Real.** Tese de Doutorado. Campina Grande – PB, 2006.

RUSSEL, S.; NORVIG, P. **Inteligência Artificial.** Rio de Janeiro: Elsevier, 2004.

SILBERSCHATZ, A., KORTH, H. F., SUDARSHAN, S. **Sistema de Banco de Dados.** Rio de Janeiro: Elsevier, 2006.

SILVA, S. H. M.G., PONCE de LEON, A. C., ROMERO, R. A. F., CRUVINEL, P. E., NATALE, W. **Redes Neurais Artificiais e Agricultura de Precisão para Recomendação de Adubação da Cultura da Goiabeira.** Revista Brasileira de Agrocomputação, v.2, n.1, p. 37-42. Ponta Grossa – PR. Jun, 2004.

TAFNER, M. A., XEREZ, M., RODRIGUES FILHO, I. W. **Redes Neurais Artificiais: Introdução e Princípios de Neurocomputação.** EKO: Ed. Da FURB, Blumenau, SC, 1995.

WEHRMEISTER, M. A. **Framework Orientado a Objetos para projetos de Hardware e Software Embarcado para Sistemas Tempo-Real.** Dissertação de Mestrado, UFRGS. Porto Alegre, 2005.

XIAO, Y., LIU, Y., LIAO, G. **A Secure Real-Time Concurrency Control Protocol for Mobile Distributed Real-Time Databases**. IJCSNS – International Journal of Computer Science and Network Security, VOL.7 N° 3, Março, 2007.

ZANNELA JÚNIOR, E. A. **Estudo de Equilíbrio de Troca Iônicade Sistemas Binários e Ternários por Meio de Redes Neurais.** Dissertação de Mestrado, UNIOESTE. Toledo – PR, 2009.

ANEXOS

Apêndice A: Parâmetros de configuração utilizados para o treinamento da Rede Neural Artificial

Interação	Tipo de Operação	Prioridade da Tarefa	Tempo da Tarefa	Tipo da Tarefa	Padrão de Chegada		
	Leitura/Escrita	Alta/Média/Baixa	<=4/>4	Crítica/Branda	Periódica/Aperiódica/Esporádica	Bloqueio	*Timestamp*
1	10	15	12	20	18	0	1
2	20	8	25	13	29	0	1
3	20	20	20	20	20	1	0
4	25	15	18	30	15	1	0
5	5	2	8	6	12	0	1
6	30	12	10	12	15	0	1
7	3	18	15	18	18	0	1
8	18	5	8	24	22	0	1
9	25	3	25	21	5	0	1
10	31	24	13	3	7	0	1
11	27	12	28	30	28	1	0
12	13	15	31	28	12	1	0
13	17	8	2	10	7	0	1
14	1	7	7	8	10	0	1
15	28	30	15	10	19	1	0
16	13	28	13	22	14	0	1
17	15	22	18	17	30	1	0
18	27	3	9	13	27	0	1
19	9	27	13	21	30	1	0
20	4	19	13	19	22	0	1
21	40	35	40	50	29	1	0
22	32	28	30	12	29	1	0
23	7	13	8	4	9	0	1
24	15	28	34	19	14	1	0
25	10	10	10	10	10	0	1
26	20	10	10	18	14	0	1

Interação	Tipo de Operação	Prioridade da Tarefa	Tempo da Tarefa	Tipo da Tarefa	Padrão de Chegada		
	Leitura/Escrita	Alta/Média/Baixa	<=4/>4	Crítica/Branda	Periódica/Aperiódica/Esporádica	Bloqueio	*Timestamp*
27	31	28	16	14	20	1	0
28	13	43	28	19	22	1	0
29	12	26	16	7	30	0	1
30	20	20	20	20	20	1	0

Apêndice B: Parâmetros utilizados para validação da Rede Neural Artificial através de Testes

Interação	Tipo de Operação	Prioridade da Tarefa	Tempo da Tarefa	Tipo da Tarefa	Padrão de Chegada	Resultado Esperado
	Leitura/Escrita	Alta/Média/Baixa	<=4/>4	Crítica/Branda	Periódica/Aperiódica/Esporádica	
1	10	15	12	20	18	*TimeStamp*
2	15	27	28	12	30	Bloqueio
3	13	13	13	9	20	*TimeStamp*
4	8	7	7	23	18	TimeStamp
5	28	14	31	28	29	Bloqueio
6	13	43	28	19	22	Bloqueio
7	32	18	22	14	26	Bloqueio
8	19	20	29	12	15	*TimeStamp*
9	26	32	12	22	19	Bloqueio
10	28	30	15	10	19	Bloqueio
11	5	14	8	3	12	*Timestamp*
12	29	8	7	25	31	Bloqueio
13	8	2	10	7	8	*TimeStamp*
14	29	12	10	23	31	*Bloqueio*
15	15	18	10	10	12	*TimeStamp*

Apêndice C: Algoritmo utilizado para validação

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <time.h>
#define TAM 1800

void bloqueio();
void timestamp();

int simul[8][TAM], copia[8][TAM], i, j, BD[20], flag_escr[20], sim, nao;
int k, alg_t, alg_b, conc, tempos[TAM];
/*Simul e copia
  [0] - 0 a 4 - Leitura / 5 - Escrita
  [1] - Prioridade: Alta / Media / Baixa
  [2] - item de dado - 0 a 19
  [3] - Tempo da tarefa: 1 a 5
  [4] - Tipo tarefa: 0 - Critica ou 1 - Branda
  [5] - Padrao de Chegada: Periodica / Aperiodica ou Esporadica
  [6] - Tempo Fim
  [7] - Concluida: -1 - Não Iniciada, 0 - Sim, 1 - Não, 2 - Em andamento ou 3 - Atrasada*/
/*BD
  Valores: >=1 - Bloqueio Leitura / 0 - Bloqueio Escrita / -1 - Livre
    valor = qtde tarefas acessando (timestamp)*/
/*Flag_escr
1 - Identifica se existe op de leitura
0 - se nao existe*/
/*Alg: 1 - Bloqueio / 2 - Timestamp*/
/*conc - Nivel de concorroncia*/



main()
  {
  clrscr();
  srand(time(NULL));
  /*iniciailizando flag_escr*/
  for(i=0;i<20;i++)
    flag_escr[i] = 0;
  /*Simulacao dos itens de dados*/
  for(i=0;i<20;i++)
    BD[i] = -1;
  /*Geracao dos dados para Simulacao*/
  for(i=0;i<TAM;i++) /*Leitura 50% - Gravacao 50%*/
    simul[0][i] = (rand()%10)+1;
  for(i=0;i<TAM;i++) /*Prioridade: Alta 20% - Media 40% - Baixa 40%*/
    simul[1][i] = (rand()%5)+1;
  for(i=0;i<TAM;i++) /*item de dado: 1 a 10*/
    simul[2][i] = (rand()%10);
  for(i=0;i<TAM;i++) /*Tempo da tarefa: 1 a 10*/
    simul[3][i] = (rand()%10)+1;
  for(i=0;i<TAM;i++) /*Tipo Tarefa: Critica 20% - Branda 80%*/
    simul[4][i] = (rand()%5)+1;
  for(i=0;i<TAM;i++) /*Periodica 40% - Aperiodica 30% - Esporádica 30% */
    simul[5][i] = (rand()%10)+1;
```

```
for(i=0;i<TAM;i++) /*Calcula tempo de término*/
  simul[6][i] = simul[3][i]-1+i;
for(i=0;i<TAM;i++)/*Verifica se foi concluída*/
  simul[7][i] = -1;
for(i=0;i<TAM;i++)/*Guarda os tempos para o timestamp*/
  tempos[i] = simul[3][i];
/*Fazendo cópia dos dados*/
for(i=0;i<8;i++)
  for(j=0;j<TAM;j++)
    copia[i][j] = simul[i][j];
/*Aplicando Algoritmo de Bloqueio*/
bloqueio();
/*Verificando resultado Bloqueio*/
sim=0;
nao=0;
for(i=0;i<TAM;i++)
  {
  if(simul[7][i] == 0)
            sim++;
    else
            nao++;
  }
printf("\nBloqueio: Sim - %d  Nao - %d", sim, nao);
/*Retornando Matriz Original*/
for(i=0;i<8;i++)
  for(j=0;j<TAM;j++)
    simul[i][j] = copia[i][j];
/*Aplicando Algoritmo de Timestamp*/
timestamp();
/*Verificando resultado Timestamp*/
sim=0;
nao=0;
for(i=0;i<TAM;i++)
  {
  if(simul[7][i] == 0)
    sim++;
  else
    nao++;
  }
printf("\nTimestamp: Sim - %d  Nao - %d", sim, nao);
/*Retornando Matriz Original*/
for(i=0;i<8;i++)
  for(j=0;j<TAM;j++)
    simul[i][j] = copia[i][j];
/*Aplicando Algoritmo da RNA*/
for(i=0;i<TAM;i++)
  {
  /*Verifica ultimas 50 tarefas*/
  if(i%50==0 && i!=0)
    {
    alg_t=0;
    alg_b=0;
    for(k=i-50;k<i;k++);
            {
            if(simul[0][i]>4)
              alg_b++;
```

```
        else
          alg_t++;
        if(simul[1][j] ==1)
          alg_b++;
        else if(simul[1][j]>3)
          alg_t++;
        if(simul[3][j]>=4)
          alg_b++;
        else if(simul[3][j]<=2)
          alg_t++;
        if(simul[4][j]==1)
          alg_b++;
        else
          alg_t++;
        if(simul[5][j]<=7)
          alg_b++;
        else
          alg_t++;
        }
 }
if(alg_t>alg_b)
 conc = 1;
else
 conc = 2;
if(conc == 2)
 {
 /*Usar Bloqueio*/
for(j=i+5;j>=0;j--)
        {
        if(j<=i)
         {
         /*Verificar se tarefa está encerrada*/
         if(simul[7][j] != 0 || simul[7][j] != 1)
          {
          /*Realiza Bloqueio caso não exista nenhum bloqueio*/
          if(BD[simul[2][j]] == -1)
           {
           if(simul[0][j] <= 4 && (simul[7][j] != 0 && simul[7][j] != 1))
                  {
                  BD[simul[2][j]] = 1;
                  simul[7][j] = 2;
                  }
           else if(simul[7][j] != 0 && simul[7][j] != 1)
                  {
                  BD[simul[2][j]] = 0;
                  simul[7][j] = 2;
                  }
           }
          /*Verifica se existe bloqueio de leitura e se a tarefa é de leitura*/
          else if(BD[simul[2][j]] >= 1 && simul[0][j] <= 4 && j == i)
           {
           simul[7][j] = 2;
           BD[simul[2][j]]++;
           }
          /*Se existe bloqueio de escrita ou ítem bloqueado e tarefa de escrita*/
          else if(i == j)
```

```
            simul[7][j] = 3;
           }
          /*Muda status da tarefa Concluida ou não*/
          if(i>=j+3 && (simul[7][j] == 3 || simul[7][j] == -1)) /*3 Tempo dado como aceitação de atraso*/
           simul[7][j] = 1;
          if(simul[3][j]>0 && simul[7][j] == 2)
           {
           simul[3][j]--;
           if(simul[3][j]==0)
            simul[7][j] = 0;
           }
          /*Libera os Bloqueios*/
          if((simul[7][j] == 0 || simul[7][j] == 1) && BD[simul[2][j]] > 1)
           BD[simul[2][j]]--;
          else if ((simul[7][j] == 0 || simul[7][j] == 1) && (BD[simul[2][j]] == 0 || BD[simul[2][j]] == 1))
           BD[simul[2][j]] = -1;
          }
         }
 }
else
/*Usar Timestamp*/
 for(j=i+5;j>=0;j--)
        {
        if(j<=i)
         {
         if(simul[7][j] == -1)
          /*Inicia tarefa com ítem de dado não acessado*/
          if(BD[simul[2][j]] == -1)
           {
           BD[simul[2][j]] = 1;
           simul[7][j] = 2;
           if(simul[0][j]>4)
                  flag_escr[simul[2][j]] = 1;
           }
          /*Tarefa já inicia com ítem de dado utilizado*/
          else
           {
           BD[simul[2][j]]++;
           simul[7][j] = 2;
           if(simul[0][j]>4)
                  flag_escr[simul[2][j]]++;
           }
          /*Muda Status Tarefa*/
          if(i>=j+3 && simul[7][j] == -1) /*3 Tempo dado como aceitacao de atraso*/
           simul[7][j] = 1;
          /*Se tarefa concluiu*/
          if(simul[3][j]>0 && simul[7][j] == 2)
           {
           simul[3][j]--;
           if(simul[3][j]==0)
            simul[7][j] = 0;
           }
          /*Encerrando tarefas*/
          if(simul[7][j] == 0 || simul[7][j] == 1)
           /*Se apenas a tarefa esta acessando o item de dado*/
           if(BD[simul[2][j]] == 1)
```

```
                {
                BD[simul[2][j]] = -1;
                if(simul[0][j]>4)
                        flag_escr[simul[2][j]]--;
                }
               /*Se existe mais de uma tarefa todas de leitura*/
               else if(BD[simul[2][j]] > 1 && flag_escr[simul[2][j]] == 0)
                BD[simul[2][j]]--;
               /*Se existe mais de uma tarefa e esta eh a unica de escrita*/
               else if(BD[simul[2][j]] > 1 && flag_escr[simul[2][j]] == 1 && simul[0][j] > 4)
                {
                BD[simul[2][j]]--;
                flag_escr[simul[2][j]]=0;
                }
               /*Se existe mais de uma tarefa de escrita acessando o  item*/
               /*Ou uma de leitura com uma de escrita pendente - Reinicia Tarefa*/
               else if(BD[simul[2][j]] > 1 && flag_escr[simul[2][j]] >= 1)
                {
                simul[7][j] = -1;
                simul[3][j] = tempos[simul[2][j]];
                BD[simul[2][j]]--;
                if(simul[0][j]>4)
                        flag_escr[simul[2][j]]--;
                }
               }
              }
             }
 /*Verificando resultado RNA*/
 sim=0;
 nao=0;
 for(i=0;i<TAM;i++)
  {
  if(simul[7][i] == 0)
    sim++;
  else
    nao++;
  }
 printf("\nRNA: Sim - %d  Nao - %d", sim, nao);
 getch();
 }


void bloqueio()
 {
 for(i=0;i<TAM+5;i++)
  {
   for(j=i;j>=0;j--)
    {
    /*Verificar se tarefa está encerrada*/
    if(simul[7][j] != 0 || simul[7][j] != 1)
            {
            /*Realiza Bloqueio caso não exista nenhum bloqueio*/
            if(BD[simul[2][j]] == -1)
             {
             if(simul[0][j] <= 4 && (simul[7][j] != 0 && simul[7][j] != 1))
              {
```

```
              BD[simul[2][j]] = 1;
              simul[7][j] = 2;
              }
             else if(simul[7][j] != 0 && simul[7][j] != 1)
              {
              BD[simul[2][j]] = 0;
              simul[7][j] = 2;
              }
             }
            /*Verifica se existe bloqueio de leitura e se a tarefa é de leitura*/
            else if(BD[simul[2][j]] >= 1 && simul[0][j] <= 4 && j == i)
             {
             simul[7][j] = 2;
             BD[simul[2][j]]++;
             }
            /*Se existe bloqueio de escrita ou ítem bloqueado e tarefa de escrita*/
            else if(i == j)
             simul[7][j] = 3;
            }
    /*Muda status da tarefa Concluida ou não*/
    if(i>=j+3 && (simul[7][j] == 3 || simul[7][j] == -1)) /*3 Tempo dado como aceitação de atraso*/
            simul[7][j] = 1;
    if(simul[3][j]>0 && simul[7][j] == 2)
            {
            simul[3][j]--;
            if(simul[3][j]==0)
             simul[7][j] = 0;
            }
    /*Libera os Bloqueios*/
    if((simul[7][j] == 0 || simul[7][j] == 1) && BD[simul[2][j]] > 1)
            BD[simul[2][j]]--;
   else if ((simul[7][j] == 0 || simul[7][j] == 1) && (BD[simul[2][j]] == 0 || BD[simul[2][j]] == 1))
            BD[simul[2][j]] = -1;
    }
   }
  }

void timestamp()
 {
 for(i=0;i<TAM+5;i++)
  for(j=i;j>=0;j--)
   {
   if(simul[7][j] == -1)
            /*Inicia tarefa com ítem de dado não acessado*/
            if(BD[simul[2][j]] == -1)
             {
             BD[simul[2][j]] = 1;
             simul[7][j] = 2;
             if(simul[0][j]>4)
              flag_escr[simul[2][j]] = 1;
             }
            /*Tarefa já inicia com ítem de dado utilizado*/
            else
             {
             BD[simul[2][j]]++;
             simul[7][j] = 2;
```

```
            if(simul[0][j]>4)
              flag_escr[simul[2][j]]++;
            }
  /*Muda Status Tarefa*/
  if(i>=j+3 && simul[7][j] == -1) /*3 Tempo dado como aceitacao de atraso*/
        simul[7][j] = 1;
  /*Se tarefa concluiu*/
  if(simul[3][j]>0 && simul[7][j] == 2)
        {
        simul[3][j]--;
        if(simul[3][j]==0)
          simul[7][j] = 0;
        }
  /*Encerrando tarefas*/
  if(simul[7][j] == 0 || simul[7][j] == 1)
        /*Se apenas a tarefa esta acessando o item de dado*/
        if(BD[simul[2][j]] == 1)
          {
          BD[simul[2][j]] = -1;
          if(simul[0][j]>4)
            flag_escr[simul[2][j]]--;
          }
        /*Se existe mais de uma tarefa todas de leitura*/
        else if(BD[simul[2][j]] > 1 && flag_escr[simul[2][j]] == 0)
          BD[simul[2][j]]--;
        /*Se existe mais de uma tarefa e esta eh a unica de escrita*/
        else if(BD[simul[2][j]] > 1 && flag_escr[simul[2][j]] == 1 && simul[0][j] > 4)
          {
          BD[simul[2][j]]--;
          flag_escr[simul[2][j]]=0;
          }
        /*Se existe mais de uma tarefa de escrita acessando o  item*/
        /*Ou uma de leitura com uma de escrita pendente - Reinicia Tarefa*/
        else if(BD[simul[2][j]] > 1 && flag_escr[simul[2][j]] >= 1)
          {
          simul[7][j] = -1;
          simul[3][j] = tempos[simul[2][j]];
          BD[simul[2][j]]--;
          if(simul[0][j]>4)
            flag_escr[simul[2][j]]--;
          }
 }
}
```