Bibliografia

[1] ANDREWS, G.. **Foundations of Multithreaded Parallel and Distributed Programming**. Addison-Wesley, 2000.

[2] BERNSTEIN, P.; BRODIE, M.; CERI, S.; DEWITT, D.; FRANKLIN, M.; GARCIA-MOLINA, H.; GRAY, J.; HELD, J.; HELLERSTEIN, J.; JAGADISH, H.; LESK, M.; MAIER, D.; NAUGHTON, J.; PIRAHESH, H.; STONEBRAKER, M. ; ULLMAN, J.. **The Asilomar report on database research**. ACM SIGMOD Record, 27(4):74–80, 1998.

[3] BROWN, K.; CAREY, M. ; LIVNY, M.. **Managing memory to meet multiclass workload response time goals**. Em: PROCEEDINGS OF THE INTERNATIONAL CONFERENCE ON VERY LARGE DATABASES (VLDB), 1993.

[4] BURNS, A.; DAVIES, G.. **Concurrent Programming**. Addison-Wesley, 1993.

[5] BAILEY, J.; GEORGEFF, M.; KEMP, D.; KINNY, D. ; RAMAMOHANARAO, K.. **Active databases and agent systems - a comparison**. Em: PROCEEDINGS OF THE INTERNATIONAL WORKSHOP ON RULES IN DATABASE SYSTEMS, LECTURE NOTES IN COMPUTER SCIENCE 985, p. 342–356. Springer, 1995.

[6] BERRY, R. F.; HELLERSTEIN, J. L.. **A flexible and scalable approach to navigating measurement data in performance management applications**. Em: PROCEEDINGS OF THE IEEE INTERNATIONAL WORKSHOP ON SYSTEMS MANAGEMENT (SMW), p. 92–103, 1996.

[7] BIGUS, J. P.; HELLERSTEIN, J. L.; JAYRAM, T. S. ; SQUILLANTE, M. S.. **Auto tune: A generic agent for automated performance tuning**. Em: PROCEEDINGS OF THE INTERNATIONAL CONFERENCE AND EXHIBITION ON THE PRACTICAL APPLICATION OF IN-

TELLIGENT AGENTS AND MULTI-AGENT SYSTEMS (PAAM), p. 33–52, 2000.

[8] BARRERA III, J. S.. **Self-tuning systems software**. Em: PROCEEDINGS OF THE WORKSHOP ON WORKSTATION OPERATING SYSTEMS, IEEE COMPUTER SOCIETY, p. 194–197, 1993.

[9] BENOIT, D.. **Automatic Diagnosis of Performance Problems in Database Management Systems**. PhD thesis, School of Computing, Queen's University, 2003.

[10] CHAUDHURI, S.; GUPTA, A. ; NARASAYYA, V.. **Compressing sql workloads**. Em: PROCEEDINGS OF THE ACM SIGMOD INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA, p. 488 – 499, 2002.

[11] COSTA, R. L. C.; LIFSCHITZ, S.. **Index self-tuning with agent-based databases**. Em: PROCEEDINGS OF THE LATIN-AMERICAN CONFERENCE ON INFORMATICS (CLEI), p. 76, Abstracts Proceedings; 12 pp. CD–ROM Proceedings, 2002.

[12] CHAUDHURI, S.; WEIKUM, G.. **Rethinking database system architecture: Towards a self-tuning risc-style database system**. Em: PROCEEDINGS OF THE INTERNATIONAL CONFERENCE ON VERY LARGE DATABASES (VLDB), p. 1–10, 2000.

[13] DIAO, Y.; HELLERSTEIN, J. L.; PAREKH, S. ; BIGUS, J. P.. **Managing web server performance with autotune agents**. IBM SYSTEMS JOURNAL, 42(1):136 – 149, 2003.

[14] EHRGOTT, M.; GANDIBLEUX, X.. **An Annotated Bibliography of Multi-objective Combinatorial Optimization**. Technical Report 62/2000, Universitat Kaiserslautern, 2000.

[15] ELMASRI, R.; NAVATHE, S.. **Fundamentals of Database Systems**. Benjamin-Cummings, 1994.

[16] FEITELSON, D. G.; NAAMAN, M.. **Self-tuning systems**. IEEE Software, 16(2):52–60, 1999.

[17] FRANK, M.; OMIECINSKI, E. ; NAVATHE, S.. **Adaptive and Automated Index Selection in RDBMS**. Em: PROCEEDINGS OF THE INTERNATIONAL CONFERENCE ON EXTENDING DATABASE TECHNOLOGY (EDBT), p. 277–292, 1992.

[18] FERRARI, D.. **Computer Systems Performance Evaluation**. Prentice Hall, 1978.

[19] GARCIA, A.; CHAVEZ, C.; SILVA, V. ; LUCENA, C.. **Developing Multi-Agent Software: an Aspect-Based Approach and a Pattern-Based Approach**. Technical report, PUC-RIO, November 2001. MCC40/01.

[20] **GDB: The GNU Project Debugger**. http://www.gnu.org/software/gdb/gdb.html, acessado em 5/03/2004.

[21] GAMMA, E.; HELM, R.; JOHNSON, R. ; VLISSIDES, J.. **Padrões de projeto: soluções reutilizáveis de software orientado a objetos**. Porto Alegre: Bookman, 2000.

[22] GREY, J.. **The Benchmark Handbook**. Morgan Kaufmann Publishers, Inc., second edição, 1998.

[23] HARIZOPOULOS, S.; AILAMAKI, A.. **A case for staged database systems**. Em: PROCEEDINGS OF THE BIENNIAL CONFERENCE ON INNOVATIVE DATA SYSTEMS RESEARCH (CIDR), 2003.

[24] HUHNS, M.; STEPHENS, L.. **Multiagent systems and societies of agents**. Em Weiss [70], capítulo 2, p. 79–120.

[25] HEISS, H.; WAGNER, R.. **Adaptive load control in transaction processing systems**. Em: PROCEEDINGS OF THE INTERNATIONAL CONFERENCE ON VERY LARGE DATABASES (VLDB), p. 47–54, 1991.

[26] HARVEY, A.. **Time series models**. MIT Press, second edição, 1993.

[27] HELLERSTEIN, J. L.. **A comparison of techniques for diagnosing performance problems in information systems**. Em: PROCEEDINGS OF THE SIGMETRICS CONFERENCE ON MEASUREMENT AND MODELING OF COMPUTER SYSTEMS, p. 278–279. ACM, 1994.

[28] HELLERSTEIN, J. L.. **Automated tuning systems: Beyond decision support**. Em: PROCEEDINGS OF THE COMPUTER MEASUREMENT GROUP, p. 277–292, 1997.

[29] HORN, P.. **Autonomic computing: IBM's perspective on the state of information technology**, 2001. http://www.research.ibm.com/autonomic/manifesto, acessado em 11/01/2004.

[30] **IBM and Autonomic Computing**, 2003. http://www-306.ibm.com/autonomic/pdfs/ACwpFinal.pdf, acessado em 11/01/2004.

[31] JAIN, R. K.. **The Art of Computer Systems Performance Analysis**. John Wiley & Sons, 1991.

[32] KWAN, E.; LIGHTSTONE, S.; SCHIEFER, B.; STORM, A. ; WU, L.. **Automatic database configuration for DB2 universal database: Compressing years of performance expertise into seconds of execution**. Em: PROCEEDINGS OF THE CONFERENCE ON DATABASE SYSTEMS FOR BUSINESS, TECHNOLOGY AND WEB (BTW 2003), 2003.

[33] KENDALL, E.; MURALI KRISHNA, P.; C.V., P. ; SURESH, C.. **A framework for agent systems**, capítulo 6, p. 113–154. John Wiley & Sons, 1999.

[34] LIFSCHITZ, S.; MILANÉS, A. ; SALLES, M. V.. **Estado da arte em auto-sintonia de sgbd relacionais**. Preprint, 2004.

[35] LAVENDER, R. G.; SCHMIDT, D. C.. **Active object: an object behavioral pattern for concurrent programming**. Em: Vlissides, J.; Coplien, J. ; Kerth, N., editors, PATTERN LANGUAGES OF PROGRAM DESIGN 2, capítulo 30, p. 483Ů–499. AddisonŮWesley, 1996.

[36] LERNER, A.. **Troubleshooting**, capítulo 7. Em SB03 [49], 2003.

[37] MENASCE, D.; ALMEIDA, V.. **Capacity Planning for Web Performance: Metrics, Models, and Methods**. Prentice Hall, 1998.

[38] MENASCÉ, D. A.; BARBARÁ, D. ; DODGE, R.. **Preserving QoS of E-commerce Sites Through Self-Tuning: A Performance Model Approach**. Em: PROCEEDINGS OF THE ACM CONFERENCE ON ELECTRONIC COMMERCE (ACM EC), p. 224 – 234. ACM Press, 2001.

[39] MARTIN, P.; POWLEY, W.; LI, H. ; ROMANUFA, K.. **Managing database server performance to meet QoS requirements in electronic commerce systems**. International Journal on Digital Libraries, 3(4):316–324, 2002.

[40] MACÊDO, J. A. F.. **Um estudo de SGBDs baseados em agentes**. Master's thesis, Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio), 2000.

[41] MCCANN, J. A.. **The database machine: Old story, new slant?** Em: PROCEEDINGS OF THE BIENNIAL CONFERENCE ON INNOVATIVE DATA SYSTEMS RESEARCH (CIDR), 2003.

[42] MOHAN, C.. **Interactions between query optimizations and concurrency control**. Em: PROCEEDINGS OF THE INTERNATIONAL WORKSHOP ON RESEARCH ISSUES ON DATA ENGINEERING: TRANSACTION AND QUERY PROCESSING (RIDE-TQP), p. 26–35, 1992.

[43] **MySQL**. http://www.mysql.com, acessado em 11/01/2004.

[44] NIEMIEC, R.; BROWN, B. ; TREZZO, J.. **Oracle 9i Performance Tuning: dicas e técnicas**. Campus, 2003.

[45] OGATA, K.. **Modern Control Engineering**. Prentice Hall, 3rd edição, 1997.

[46] **Patrol(r) knowledge module (tm) for unix v8.3 - features and faqs**. http://www.softwareinnovations.co.uk/downloads/patrol/9066.pdf, acessado em 11/01/2004.

[47] **PostgreSQL**. http://www.postgresql.com, acessado em 11/01/2004.

[48] RUSSELL, S. J.; NORVIG, P.. **Artificial Intelligence: A Modern Approach**. Prentice Hall, 2nd edição, 1995.

[49] SHASHA, D.; BONNET, P.. **Database Tuning: Principles, Experiments and Troubleshooting Techniques**. Morgan Kaufmann, 2003.

[50] SALLES, M. V.. **Uma arquitetura baseada em agentes para a resolução do problema de auto-sintonia de índices em sgbds relacionais**. Documento apresentado na defesa de proposta de Dissertação, Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio), 2003.

[51] SALLES, M. V.. **Agente de auto-sintonia de Índices baseado em diferenças**. Projeto Final de Programação, Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio), 2003.

[52] SALLES, M. V.. **Criação autônoma de Índices em bancos de dados**. Master's thesis, Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio), 2004.

[53] SANDHOLM, T.. **Distributed rational decision making**. Em Weiss [70], capítulo 5, p. 201–258.

[54] SEVCIK, K. C.. **Database system performance prediction using an analytical model (invited paper)**. Em: PROCEEDINGS OF THE INTERNATIONAL CONFERENCE ON VERY LARGE DATABASES (VLDB), p. 182–198. IEEE Computer Society, 1981.

[55] SHALLAHAMER, C.. **Total performance management**. Oracle Magazine, 69(2), 1995.

[56] SHASHA, D.. **Tuning databases for high performance**. ACM Computing Surveys, 28(1):113–115, March 1996.

[57] STEVENS, R.. **Unix Network Programming**, volume 2. Prentice Hall, 2 edição, 1999. Interprocess Communications.

[58] **Transaction processing council**. http://www.tpc.org, acessado em 11/02/2004.

[59] THOMASIAN, A.. **Two-phase locking performance and its thrashing behavior**. ACM Transactions in Database Systems, 18(4):579–625, 1993.

[60] **Together**. http://www.togethersoft.com, acessado em 11/02/2004.

[61] VILALTA, R.; APTE, C.; HELLERSTEIN, J.; MA, S. ; WEISS, S.. **Predictive algorithms in the management of computer systems**. IBM Systems Journal, special issue in Artificial Intelligence, 41(3):461–474, 2002.

[62] VAN DEN AKKER, J.; SIEBES, A.. **Enriching active databases with agent technology**. Em: PROCEEDINGS OF THE INTERNATIONAL WORKSHOP ON COOPERATIVE INFORMATION AGENTS, p. 116–125, 1997.

[63] **Veritas software corporation**. http://www.veritas.com, acessado em 11/01/2004.

[64] VICARI, S.. **Proposta de um sistema de regras de sintonia de performance para aplicações de bancos de dados**. Master's thesis, Universidade Federal do Rio Grande do Sul, July 1998.

[65] WORSLEY, J.; DRAKE, J.. **Practical PostgreSQL**. O'Reilly & Associates, 2002.

[66] WEIKUM, G.; HASSE, C.; MÖNKEBERG, A. ; ZABBACK, P.. **The comfort automatic tuning project**. Information Systems, 19(5):381–423, 1994.

[67] WEISS, S. M.; INDURKHYA, N.. **Predictive Data Mining: A Practical Guide.** Morgan Kaufmann Publishers, 1998.

[68] WEIKUM, G.; MÖNKEBERG, A.; HASSE, C. ; ZABBACK, P.. **Self-tuning database technology and information services: from wishful thinking to viable engineering**. Em: PROCEEDINGS OF THE INTERNATIONAL CONFERENCE ON VERY LARGE DATABASES (VLDB), p. 20–31, 2002.

[69] VON WANGENHEIM, C. G.; VON WANGENHEIM, A.. **Raciocínio Baseado em Casos**. Editora Manole Ltda., 2003.

[70] Weiss, G., editor. **Multiagent Systems: A modern Approach to Distributed Artificial Intelligence**. The MIT Press, Cambridge, MA, USA, 1999.

[71] WEISER, M.. **Some computer science issues in ubiquitous computing**. Communications of the ACM, 36(7):75–84, 2003. Special issue on Computer Augmented Environments.

[72] WISETH, K.. **Oracle database 10g, the world's first self-managing, grid-ready database arrives**. Oracle Magazine, p. 28–37, September 2003.

[73] WOOLDRIDGE, M.. **Intelligent agents**. Em Weiss [70], capítulo 1, p. 27–78.

A
Anexo

Apresentamos aqui um grupo de algoritmos para controle de sintonia que foram usados durante a implementação deste trabalho para comprovar as interações entre os agentes.

A.1
Substituição de páginas em cache

As operações sobre os dados são realizadas em memória principal. Dado que seu custo usualmente impede manter o banco de dados em memória RAM, são utilizadas técnicas que gerenciam a disponibilidade das páginas de memória, assim como a integridade e persistência dos dados. A página é a unidade básica de transferência de dados entre memória principal e disco. O propósito fundamental do gerente de *buffer* é manter endereçáveis as páginas em memória principal e coordenar a escrita das páginas a disco com outros gerentes.

O número de acessos a disco deve ser minimizado. O gerente de *buffer* administra um segmento em memória virtual que está particionado em fragmentos de igual tamanho chamados de *frames*, que podem conter exatamente uma página.

O *buffer* de dados está implementado como uma fila, como é mostrado na figura A.1. Essa fila se organiza de forma que em um extremo se colocam as páginas mais recentemente usadas que vão avançando na fila no sentido LRU (*Least Recently Used*- Menos Recentemente Usadas) na medida em que aumenta o tempo sem serem atualizadas. Quando são atualizadas se colocam no extremo MRU (*Most Recently Used* - Mais Recentemente Usadas) novamente. A estratégia LRU consiste em que a página substituída é aquela do extremo LRU, que é então colocada no extremo MRU da fila. Já a estratégia MRU substitui a página no extremo MRU.

Uma requisição de página ao gerente de *buffer* é seguida dos seguintes passos:

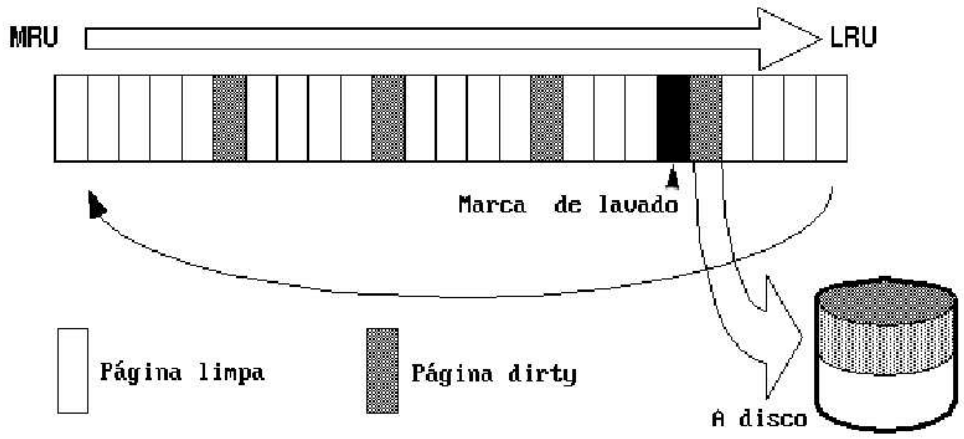

Figura A.1: A estratégia LRU toma uma página limpa do extremo LRU da cache

- Checar se a página solicitada está no *buffer*. Se for encontrada é devolvido o endereço do *frame*. Esse caso é chamado de *cache hit*.
- Caso contrário a página deve ser trazida do disco e colocada no *buffer*. Por esse motivo se procura primeiramente algum *frame* que não contenha uma página válida.
- Se não existem *frames* com páginas não válidas, segue a determinação da página que será retirada. A escolha depende da estratégia de substituição de páginas implementada. Por exemplo, de seguir uma estratégia MRU será selecionada a página mais recentemente utilizada (a que está no extremo MRU da fila). Já utilizando uma estratégia LRU, a escolhida será a página menos recentemente utilizada(extremo LRU).
- Se a página a retirar estiver marcada como *dirty* (modificada), deverá ser escrita em seu bloco do disco.
- O bloco que contém a página solicitada é copiado no *frame* eleito.
- O endereço do *frame* é devolvido.

LRU é usada comumente para páginas que serão usadas mais de uma vez ou que precisam ser atualizadas. Já o MRU é usado para páginas que serão lidas somente uma vez, como longas varreduras de tabelas únicas.

A política de substituição de páginas de PostgreSQL é LRU. Aplicando essa política se obtém bons resultados para determinados padrões de acesso ao banco de dados, como aqueles que implicam atualizações freqüentes e alta localidade de dados [1]. No caso de operações como longas varreduras seqüenciais,

[1] Localidade é a tendência de dados recentemente usados de serem novamente referenciados por operações posteriores.

entretanto, não deve ser usada essa estratégia, pois a cache seria esvaziada de informação útil. Nesses casos, MRU torna-se uma boa escolha.

A.2
Nível de Multiprogramação

O Nível de Multiprogramação (*MultiProgramming Level - MPL*) é o parâmetro que controla o número máximo de processos concorrentes que podem ser atendidos pelo sistema. O MPL pode influir de forma inversa na vazão e no tempo de resposta, sendo que até um certo limite o aumento do MPL aumenta a vazão mas diminui o tempo de resposta por causa da contenção de recursos requisitados por vários processos simultaneamente. A determinação de seu valor ótimo é um processo complexo por causa da dependência que tem com a carga de trabalho, como mostrado na figura 2.2.

Em sistemas de processamento de transações que executam concorrentemente, pode ocorrer um problema de sobrecarga em que excessivos conflitos de bloqueio provoquem que o tempo de resposta de uma grande parte das transações aumente drasticamente, a vazão caia e o sistema deva ser reiniciado. Este problema é chamado de *thrashing* de contenção de dados. Conforme [18, 66], o *thrashing* pode ser evitado através do controle do MPL.

Em [49] se sugere basear a escolha do nível de multiprogramação no método de ajustes incrementais [25], dado que lamentavelmente os sistemas de banco de dados não possuem o algoritmo proposto em [66] que não pode ser usado de outra forma porque requer monitoramento contínuo. A proposta de [66] propõe um método chamado controle de carga orientado a conflitos. Nele se procura o controle automático e dinâmico do nível de bloqueios através da observação de uma métrica chamada de "razão de conflito"(*conflict ratio*) que se calcula como a razão entre o total de bloqueios existentes no sistema e o total que pertencem a transações ativas, refletindo o grau de contenção de dados presente no sistema. A razão de conflito tem um valor crítico quase constante com independência da carga bem definido tanto experimental como analiticamente em torno de 1.3 [59, 66]. Caso esta métrica alcance seu valor crítico, existem grandes probabilidades de thrashing e devem ser tomadas ações de sintonia. Estas podem ser o controle de admissão e o controle de cancelamento. O princípio de trabalho do controle de carga orientado a conflitos é mostrado na figura A.2.

O controle de admissão determina se uma nova transação deve ser processada ou se deve esperar em fila na entrada do sistema. O algoritmo para o controle de admissão proposto em [66] é o seguinte:

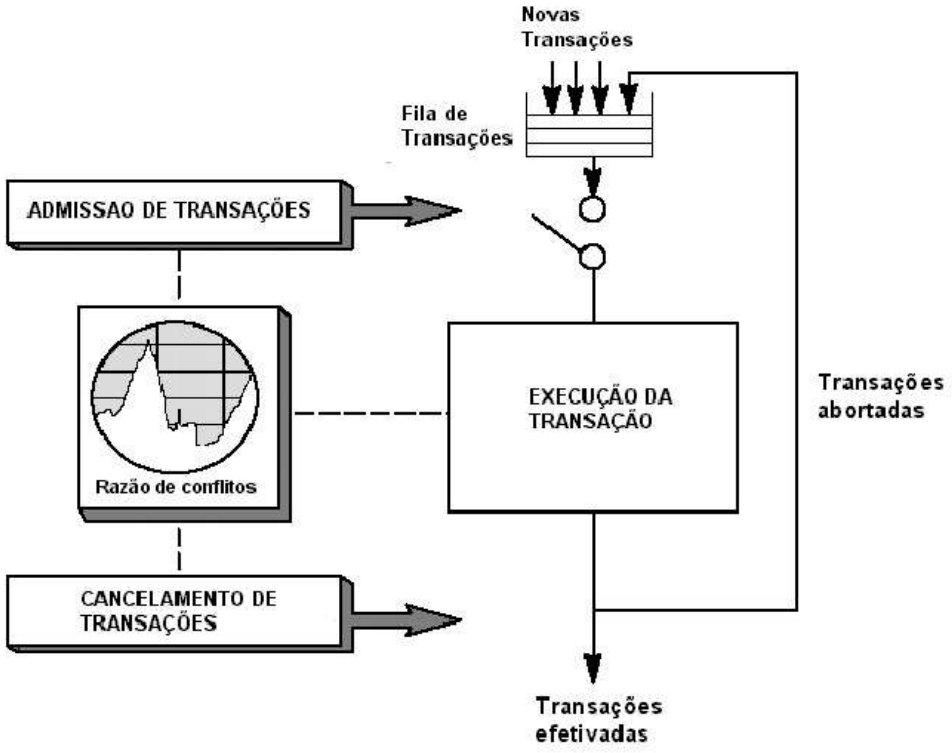

Figura A.2: O princípio de trabalho do controle de carga orientado a conflitos [66].

```
upon a BOT request for transaction T:
   if conflict ratio < critical conflict ratio
   then admit T
   else put T in the transaction queue
   fi

upon the EOT of transaction T:
   update conflict ratio
   if conflict ratio < critical conflict ratio then
     foreach transaction Tqueued in the transaction queue do
       if (Tqueued will be started the first time) or
         Tqueued has been a deadlock victim or cancelation
         victim before and all transactions that Tqueued was
         waiting for in its previous execution have been
         succesfully completed)
       then admit Tqueued
       fi
     od
```

```
   fi
```

Note que o algoritmo é executado ao ser requisitado um inicio de transação (BOT - *Begin Of Transaction*) e ao terminar uma transação. Uma nova transação será admitida no sistema se o valor da razão de conflitos não ultrapassar o nível crítico. Caso contrário a transação é colocada na fila. Ao terminar de executar uma transação é checado o estado da razão de conflitos. Se o valor da variável estiver dentro do intervalo admissível, uma heurística é executada para liberar transações da fila. Esta prioriza transações novas e aquelas que foram canceladas e já podem ser completadas.

Uma melhora ao algoritmo pode consistir na determinação dos *locks* que cada nova transação vai solicitar, de forma que não sejam colocadas na fila aquelas cujas requisições não estão relacionadas com o problema presente de *thrashing*. Da mesma forma podem ser escolhidas as transações a serem liberadas da fila quando o valor da razão de conflito sair da região crítica. Isso implica, por outro lado, na necessidade de algoritmos para controlar a possibilidade de *deadlocks* entre as transações que ficam na fila. Nessa dissertação não foi usada uma implementação aprimorada deste controle pois não se trata de um objetivo fundamental da pesquisa e também devido as limitações de tempo para desenvolvimento.

Dado que as transações que estão sendo executadas pelo sistema podem continuar a solicitar novos bloqueios, o controle de admissão é complementado com o controle de cancelamento. O controle de cancelamento pode abortar transações que presentemente estão no sistema, colocando-as na fila de transações até serem reiniciadas posteriormente pelo mecanismo de controle de admissão. O algoritmo para o controle de cancelamento proposto é apresentado a seguir:

```
Upon a lock wait of transaction T:
   update conflict ratio
   while conflict ratio >= critical conflict ratio do
      among the transactions that are blocked and block
      other transactions
         determine the transaction Tvictime with the smallest product
         number of locks held * number of previous restarts
         (with the transaction with the highest product being exempt)
      abort Tvictim and put it in the transaction queue
      if no elegible transaction found then exit loop fi
   od
```

O algoritmo mostrado cancela a transação com o menor produto número de bloqueios possuídos*número de prévios reinícios. A idéia por trás a heurística para a escolha da vítima de cancelamento está em um compromisso entre a minimização de esforço perdido e a tentativa de evitar *starvation*. A transação cancelada é colocada na fila de transações até ser readmitida no sistema pelo controle de admissão.