Estado da Arte em Auto-sintonia de SGBD Relacionais

Sérgio Lifschitz
sergio@inf.puc-rio.br

Anolan Y. Milanés
anolan@inf.puc-rio.br

Marcos A. Vaz Salles
mvsalles@inf.puc-rio.br

PUC-RioInf.MCC35/04 October, 2004

Abstract: Database tuning is the activity of adjusting parameters, configurations and data structures of a database system to the needs of a given workload. In general, most performance adjustments are done on DBMSs by specialized DBAs. This situation motivates the development of self-tuning database systems, that is, systems that adjust themselves without human intervention in order to obtain more performance to their applications. This report presents a study of database systems self-tuning. We classify the existing proposals in two major research fronts: local self-tuning and global self-tuning. The goal of local self-tuning work is to enhance system performance taking into consideration a specific aspect of the system, such as physical design or memory use. Global self-tuning work, on the other hand, tries to consider the trade-offs involved in tuning different aspects of the system. This report also presents an overview of commercial systems that bring self-tuning features.

Keywords: self-tuning; DBMS; monitoring; tuning.

Resumo: A sintonia de bancos de dados é a atividade de ajustar os parâmetros, configurações e estruturas de um sistema de bancos de dados às necessidades de uma determinada carga de trabalho. Em geral, a maioria dos ajustes de desempenho são realizados sobre os SGBDs por DBAs especializados. Esta situação motiva o desenvolvimento de sistemas de bancos de dados que sejam capazes de auto-sintonia, ou seja, de se ajustarem sem intervenção humana no sentido de obter mais desempenho para suas aplicações. Esta monografia apresenta um estudo sobre auto-sintonia de sistemas de bancos de dados. Classificamos as propostas presentes na literatura em duas grandes frentes de pesquisa: auto-sintonia local e auto-sintonia global. Os trabalhos de auto-sintonia local objetivam aumentar o desempenho do sistema levando em consideração um aspecto específico do sistema, como o projeto físico ou o uso de memória. Os trabalhos de auto-sintonia global, por outro lado, procuram considerar os compromissos envolvidos na sintonia de diferentes aspectos do sistema. Esta monografia apresenta, ainda, um panorama de sistemas comerciais que trazem características de auto-sintonia.

Palavras-chave: auto-sintonia; SGBD; monitoramento; sintonia.

1 Introdução

A sintonia, ou ajuste, de sistemas de bancos de dados (*database tuning*) é a atividade de ajustar os parâmetros, configurações e estruturas de um sistema de banco de dados às necessidades de uma determinada carga de trabalho. Por carga de trabalho entendemos um conjunto de consultas ou atualizações ponderadas por suas freqüências de ocorrência.

É importante diferenciar a sintonia de banco de dados da otimização de consultas. A otimização de consultas é o processo realizado por um SGBD para converter um comando declarativo, escrito por exemplo em SQL, em um plano de execução. Neste, são especificados todos os operadores que serão aplicados aos dados e de que forma eles serão ordenados e compostos. Assim, a otimização de consultas é um processo automático, realizado por componentes do SGBD; a sintonia de banco de dados é realizada, na maioria das vezes, pelo administrador do sistema (DBA) ou por um especialista em desempenho.

Podemos caracterizar o desempenho de um sistema de computação como a eficiência com que este atinge seus objetivos. Três técnicas gerais podem ser utilizadas para avaliar o desempenho de um sistema [Jai91]: modelagem analítica, simulação e monitoramento (ou medição). A primeira permite que avaliações qualitativas e quantitativas sejam realizadas mesmo na ausência de uma implementação real do sistema. Para sistemas de bancos de dados, a criação de modelos analíticos é uma tarefa complexa e, assim, diversas suposições são feitas para simplificação dos modelos. Infelizmente, estas suposições muitas vezes podem comprometer a qualidade ou generalidade dos resultados obtidos.

A técnica de simulação, por outro lado, permite obter resultados mais precisos sobre o aspecto sob estudo do sistema. Entretanto, construir uma simulação exige maior esforço e o conhecimento de ferramentas específicas. Já o monitoramento somente pode ser aplicado em sistemas já existentes e, apesar de poder gerar as medições mais realistas, se constitui em uma carga extra para o sistema tanto em termos de processamento quanto em espaço necessário para armazenar as informações coletadas.

Modificações no desempenho de um sistema já existente podem ser obtidas ajustando as suas alocações de recursos através das variáveis ou parâmetros de controle de sintonia (ou simplesmente parâmetros de controle). A determinação destes parâmetros requer a observação de três classes de métricas [BHJS00]:

1. Métricas de Configuração: são características que não mudam ajustando os parâmetros de controle, como a quantidade de memória e a velocidade da CPU.

2. Medidas de Desempenho ou Métricas de Nível de Serviço: são as que definem o desempenho do sistema, como:

   - a vazão (*throughput*), que é uma medida de produtividade que corresponde à quantidade de trabalho executada pelo sistema durante um dado período de tempo,
   - o tempo de resposta, que é o tempo que demora o processamento das requisições submetidas ao sistema.

3. Métricas de carga de trabalho: designa todo o processamento de requisições submetidas ao sistema pelo usuário durante um dado intervalo de tempo. O desempenho dos sistemas de banco de dados é extremamente dependente da carga. Em [Vic98], é proposto um conjunto de parâmetros para a caracterização da carga de trabalho em bancos de dados. Estes são: a localidade de acesso ou de referência, o comprimento das transações, a proporção de operações de escrita das transações, o nível de multiprogramação e a taxa de chegada das consultas.

1.1 Atividades de Sintonia

Os atuais produtos de bancos de dados oferecem entre dezenas e centenas de parâmetros de controle e configurações que podem ser ajustados para alcançar um funcionamento mais eficiente [WHMZ94]. O ajuste destes parâmetros é complexo, tanto pela necessidade de compreensão dos algoritmos que são usados pelo sistema quanto pela possibilidade de inter-relações entre os ajustes. Pode-se dizer que na tarefa de ajuste de bancos de dados um bom princípio é "Pensar globalmente, consertar localmente" [SB03]. Nos atuais sistemas disponíveis comercialmente, a atividade de sintonia de bancos de dados é feita, em grande parte, de forma manual.

Desta forma, é necessária uma quantidade crescente de profissionais especializados para realizar o ajuste de desempenho das complexas aplicações de bancos de dados hoje existentes. Como podemos perceber, à medida que os sistemas de bancos de dados vão se tornando cada vez mais comuns em nossa sociedade, esta dependência de recursos humanos se torna muito indesejável. A necessidade de especialistas para a operação dos sistemas dificulta a adoção em larga escala de SGBDs [BBC+98].

Além dos ajustes de desempenho, outras atividades também exigem a atenção de administradores de sistemas de bancos de dados. Podemos citar como exemplos a execução periódica de procedimentos de *backup* e a manutenção de privilégios de segurança. Este tipo de atividade pode ser caracterizado como parte integrante da administração do sistema e visa permitir a sua operação continuada e não necessariamente uma melhora de desempenho. As tarefas de administração também exigem a intervenção freqüente de profissionais especializados.

1.2 Auto-sintonia em Sistemas

Estes fatos levaram tanto o meio acadêmico quanto a indústria a colocar como um item de grande prioridade em suas pautas de pesquisa e desenvolvimento a criação de sistemas de bancos de dados que sejam capazes de auto-sintonia e de auto-administração [Hor01, BBC+98]. O estudo da auto-sintonia de sistemas computacionais não é recente, como podemos constatar em [Fer78]. Entretanto, a sua relevância para a área de banco de dados aumentou significativamente nos últimos anos. Espera-se que, no futuro, os sistemas demandem uma quantidade menor de profissionais especializados para a manutenção de seu desempenho e de sua operação. Num cenário ideal, os sistemas conseguiriam alcançar um desempenho adequado sem qualquer necessidade de ajuste manual. Mais do que isto, à medida que as cargas de trabalho submetidas ao sistema mudassem, este iria procurar se adaptar dinamicamente para continuar apresentando um desempenho aceitável.

Esta monografia objetiva obter uma classificação dos principais trabalhos existentes na literatura sobre auto-sintonia de sistemas de bancos de dados relacionais. Para mantermos uma nomenclatura consistente, é importante notar que, dentro da estratégia de *autonomic computing* da IBM [Hor01], o termo utilizado para auto-sintonia é auto-otimização (*self-optimizing*).

Nosso foco reside no ajuste de desempenho do sistema e não nas suas tarefas de administração rotineiras. Por mais que seja difícil cobrir todos os trabalhos publicados, acreditamos ter mapeado os grupos de pesquisa que mais se engajaram em automatizar ou eliminar completamente os ajustes manuais de desempenho de que os sistemas hoje necessitam. Procuramos também apresentar uma descrição resumida das propostas defendidas por cada grupo de forma que o leitor possa se ambientar com os conceitos desenvolvidos e tenha um ponto de partida para estudos aprofundados sobre os temas que mais lhe interessarem.

Podemos distinguir, até o momento, duas grandes frentes de pesquisa nas iniciativas analisadas: criar sistemas que sejam capazes de auto-sintonia por projeto (veja [CW00]) ou atacar problemas de sintonia dos sistemas atualmente disponíveis para torná-los mais autônomos. Na primeira frente de pesquisa, o objetivo é obter sistemas que dosem os conflitos por recursos de seus diversos módulos componentes de forma a obter uma solução globalmente ótima. Já na

segunda frente de pesquisa, vários problemas podem ser considerados em particular, como o ajuste do nível de concorrência do sistema, o posicionamento de dados ou o uso mais adaptativo de memória [WHMZ94]. Cada problema é tratado sem se levar em consideração as interações e conflitos por recursos existentes entre componentes distintos do sistema. Ganha-se, entretanto, compreensão sobre a natureza de cada problema particular.

1.3 Organização

O restante deste documento está organizado conforme descrito a seguir:

- a Seção 2 discute alguns dos príncipios gerais que são encontrados na construção de sistemas ou componentes capazes de auto-sintonia;
- a Seção 3 detalha a nossa classificação dos trabalhos existentes na literatura e mostra os principais grupos de pesquisa dedicados ao estudo de questões de auto-sintonia em sistemas de bancos de dados relacionais;
- a Seção 4 apresenta as propostas realizadas para a construção de sistemas que sejam capazes de auto-sintonia por projeto ou por adaptação;
- a Seção 5 se foca nos trabalhos que endereçam questões específicas de sintonia que não são bem resolvidas nos sistemas atuais;
- a Seção 6 mostra a abordagem adotada pelos fornecedores de SGBDs para incluir técnicas de auto-sintonia nos sistemas. Discute, ainda, técnicas usadas nos produtos oferecidos por fornecedores de ferramentas de análise de desempenho que objetivam automatizar parcialmente tarefas rotineiras de sintonia;
- por fim, a Seção 7 discute algumas das conclusões de nosso estudo e procura sugerir alguns trabalhos futuros.

2 Princípios de Auto-sintonia

A auto-sintonia de bancos de dados refere-se ao ajuste, de forma automática, dos parâmetros, configurações e estruturas de um SGBD com o objetivo de processar mais eficientemente uma determinada carga de trabalho. A auto-sintonia, bem como a atividade de sintonia de bancos de dados, deve respeitar alguns limites quanto ao tipo de ações que podem ser empreendidas para aumentar o desempenho do sistema. Assim, expandir a configuração de hardware, através da adição de processadores, memória e discos, ou mudar a versão do software do SGBD não são ações que estariam no escopo de um componente de auto-sintonia do sistema. Este tipo de componente deve criar controles sobre a forma como o sistema se utiliza do hardware disponível para acessar os dados [1].

Ações válidas incluem, portanto, a criação de estruturas de apoio para acelerar a busca de informações, a coleta de estatísticas sobre os dados armazenados, a alocação dos dados em disco e em memória de forma a acelerar o seu acesso, e o controle de quantas transações serão atendidas pelo sistema e com qual tempo de resposta. Nos sistemas atuais, muitas destas decisões exigem a intervenção de DBAs especializados. O sistema não é capaz de analisar automaticamente qual é a melhor forma de utilizar o hardware em que foi instalado para processar a carga de trabalho que lhe é submetida.

[1] Em [SB03], a melhoria da configuração de hardware é descrita como uma atividade de sintonia válida. Em nossa visão, a sintonia deve se limitar ao uso dos recursos disponíveis. Isto estabelece uma divisão clara entre sintonia e planejamento de capacidade.

Uma das grandes dificuldades da auto-sintonia de bancos de dados reside no fato de que a configuração de melhor desempenho para o sistema depende fundamentalmente do *ambiente* em que este está inserido. A definição do ambiente varia de acordo com o foco de estudo. Para o sistema como um todo, o ambiente refere-se à combinação de hardware, carga de trabalho e outros programas que são executados no mesmo hardware (por exemplo, o sistema operacional). Já para um módulo específico do sistema, o ambiente pode ser definido pelos demais componentes do SGBD em contato com esse módulo e pelas requisições a que este módulo precisa atender (carga de trabalho).

Analisemos um exemplo. O ambiente para o otimizador de consultas envolve o analisador sintático, o catálogo, a máquina de execução e as requisições de comandos a serem otimizados. A qualidade dos planos encontrados pelo otimizador depende, entre outros fatores, da adequação das estatísticas presentes no catálogo aos comandos que estão sendo submetidos.

Tradicionalmente, a escolha de quais estatísticas serão mantidas no catálogo é deixada nas mãos do administrador de banco de dados. Podemos automatizar esta escolha através da criação de um componente de auto-sintonia que interaja com o sistema no lugar do DBA. A Figura 1 mostra o ambiente em que poderia estar inserido um componente de auto-sintonia de estatísticas.

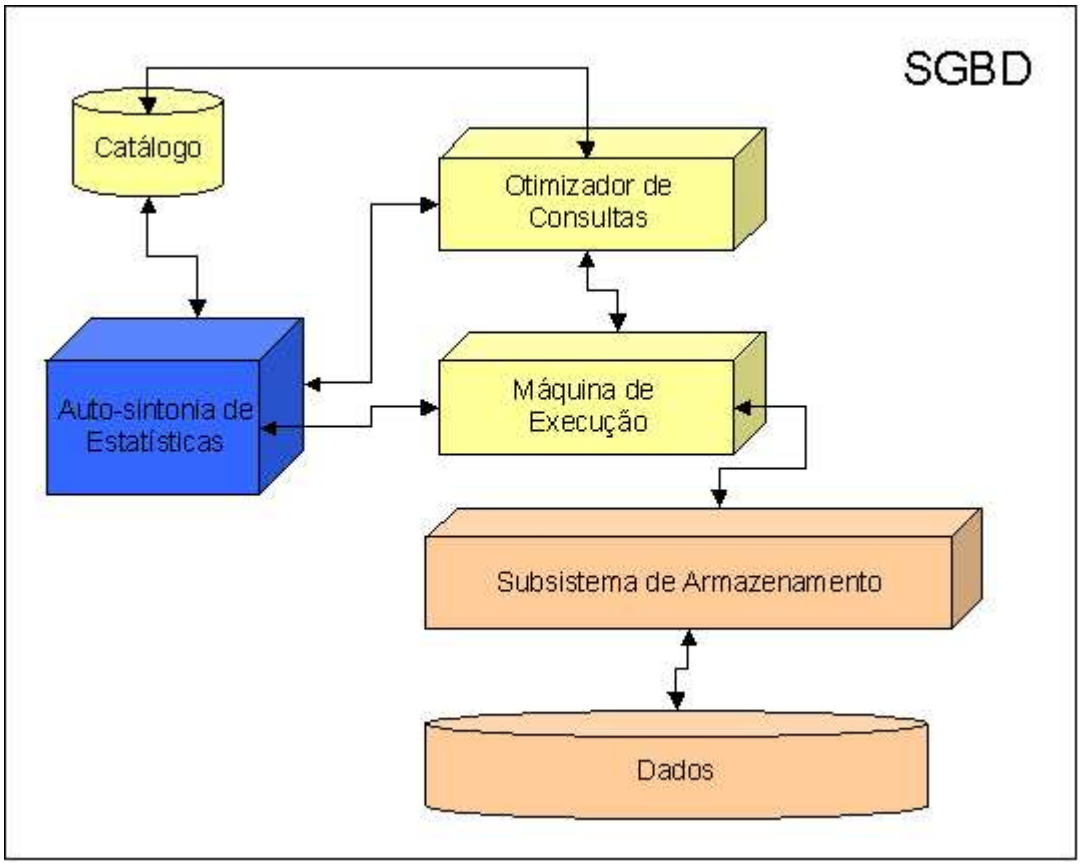

Figura 1: Um componente para auto-sintonia de estatísticas e o seu ambiente

O ambiente do componente de auto-sintonia de estatísticas é composto do otimizador de consultas, da máquina de execução, do catálogo (em especial, das estatísticas presentemente definidas no banco de dados) e das consultas que são submetidas ao otimizador. Este componente poderia avaliar se as estatísticas presentes atualmente no catálogo realmente são as mais adequadas para as consultas dadas e poderia garantir que estas estatísticas sejam automaticamente coletadas e atualizadas.

Para projetarmos os mecanismos de funcionamento do componente de auto-sintonia de estatísticas, podemos aplicar um conjunto de princípios básicos. Acreditamos ter observado a repetida aplicação destes princípios nos trabalhos sobre auto-sintonia que encontramos na literatura e em produtos comerciais. Estes princípios são:

- *Definição do ambiente em que o componente de auto-sintonia está inserido.* Conforme vimos na discussão anterior, o ambiente depende fundamentalmente do escopo do componente de auto-sintonia que está sendo projetado. Nos trabalhos estudados, percebemos que o enfoque adotado pode ser local ou global. No enfoque local, o componente de auto-sintonia possui como ambiente um ou mais módulos do SGBD. O seu objetivo é automatizar uma atividade de sintonia específica, como, por exemplo, a escolha e atualização de estatísticas (veja a seção 5). Nesta automatização, o componente deve levar em conta considerações de custo-benefício para poder assumir compromissos (*trade-offs*). Por exemplo, o componente de escolha de estatísticas deve pesar o benefício de armazenar uma determinada estatística em relação ao custo em termos de espaço que o armazenamento desta estatística incorre [2]. Já no enfoque global, consideramos a sintonia do sistema de banco de dados como um todo. O ambiente consiste do hardware, carga de trabalho e outros programas que são executados no mesmo hardware. A complexidade da abordagem global está no fato de que os diversos componentes do SGBD (gerente de memória, controle de concorrência, controle de recuperação, otimizador de consultas, entre outros) afetam o desempenho uns dos outros de formas ainda não compreendidas [WMHZ02]. Um componente de auto-sintonia global deve procurar alcançar um equilíbrio entre as decisões locais de sintonia pertinentes a cada cada módulo do SGBD de tal forma que o desempenho do sistema seja maximizado (veja a seção 4). Um exemplo seria a decisão de criar um índice para resolver um problema de contenção de bloqueios sobre uma tabela pequena e não para acelerar uma consulta específica.

- *Modificação do sistema através de um ciclo de controle de realimentação (feedback control loop)* [WMHZ02]. Um ciclo de controle de realimentação é um método para controlar o efeito das decisões de sintonia sobre o desempenho do sistema. O método se divide em três fases: observação, predição e reação. Na fase de observação, são coletadas métricas e estatísticas que permitem avaliar se existe algum problema de desempenho num componente específico ou no sistema como um todo. Num componente de auto-sintonia de estatísticas, poderíamos coletar as consultas processadas pelo otimizador, suas informações estimadas de seletividades e também as informações reais de seletividades encontradas durante a execução. Cabe ressaltar que um componente deste gênero poderia analisar a corretude de previsão tanto de seletividades provenientes de seleções quanto de junções [SLMK01]. A fase de predição consiste em estimar o desempenho futuro do sistema com base nas métricas coletadas e tomar decisões sobre que ações podem ser empreendidas caso seja necessário sintonizar o sistema. Para o componente de auto-sintonia de estatísticas, poderíamos verificar a diferença entre as seletividades reais e estimadas e identificar que determinadas colunas de uma tabela precisam ter suas informações estatísticas atualizadas. Por fim, na fase de reação, o componente de auto-sintonia implementa as ações de sintonia do sistema. No nosso exemplo, o componente de auto-sintonia de estatísticas realizaria um refinamento das estatísticas referentes à estimativa de seletividades para a consulta que foi processada.

- *Coleção de estatísticas e informações de forma que haja pouco impacto no desempenho do sistema.* Determinar se há um problema de desempenho no sistema que possa ser resolvido através de ações de sintonia exige que seja realizado um diagnóstico. Este diagnóstico será

[2] Esta é uma aplicação do príncipio conhecido de sintonia de banco de dados de "Estar preparado para compromissos" [SB03].

baseado em um conjunto de métricas e estatísticas coletadas durante a operação do sistema. Se desejamos automatizar a sintonia do sistema, precisamos criar componentes de auto-sintonia que continuamente recuperem informações do sistema e diagnostiquem se há algum problema de desempenho. Como o monitoramento realizado pelo componente de auto-sintonia deve ser contínuo, devemos tornar o seu impacto sobre o desempenho do sistema o menor possível. Isto é necessário para que o próprio diagnóstico não seja comprometido. No caso do componente de auto-sintonia de estatísticas, coletamos informações sobre a consulta submetida ao SGBD e suas seletividades reais e estimadas. Devemos tornar esta coleta o menos intrusiva possível, potencialmente através da instrumentação dos demais módulos do SGBD que estão no ambiente do componente de auto-sintonia (no caso, o otimizador e a máquina de execução).

- *Utilização de modelos matemáticos e estimativas para prever o desempenho futuro do sistema.* Um componente de auto-sintonia precisa prever o desempenho futuro do sistema com base em métricas coletadas periodicamente. Modelos formais sobre o desempenho do sistema podem nos prover valiosas informações sobre como o comportamento de um sistema evolui de acordo com parâmetros de entrada [Jai91]. No componente de auto-sintonia de estatísticas, após a comparação realizada entre as seletividades estimadas e reais, podemos refinar o modelo de custos do otimizador através do cálculo de estatísticas mais apuradas. A determinação de quais estatísticas devem ser refinadas e de que forma isto deve ocorrer exige uma modelagem cuidadosa (veja a seção 5.6). Em geral, precisamos ter algum modelo que correlacione o histórico de desempenho do sistema com o seu desempenho futuro em curto prazo. Este modelo pode ser construído de forma analítica ou através de técnicas estatísticas, como o uso de redes neurais [BHJS00]. Com o uso do modelo construído, selecionamos as ações de sintonia que tendem a produzir melhores resultados de desempenho.

- *Realização de ajustes simples on-line e ajustes complexos off-line.* Tanto a enumeração de ações possíveis para sintonizar o sistema quanto a sua execução podem, em algumas situações, impactar muito negativamente o desempenho do sistema. Realizar este tipo de computação *on-line*, enquanto o sistema processa requisições, pode ser impraticável. Para a auto-sintonia de estatísticas, a ação de refinar as estatísticas sobre uma determinada tabela pode ser muito custosa computacionalmente dependendo do método empregado. Assim, podemos registrar a necessidade de tomar esta ação para execução *off-line*, em um momento em que o processamento de requisições do sistema não será comprometido. Este princípio é uma aplicação de um princípio conhecido de sintonia de bancos de dados de "Usar particionamento para quebrar gargalos" [SB03]. No caso, particionamos a execução de ações complexas no tempo, nos aproveitando de algum momento "quieto" do sistema para a sua execução. Se, por outro lado, as ações de sintonia exigem poucos recursos do sistema para a sua implementação, elas podem ser realizadas *on-line*, desde que respeitando momentos de sobrecarga.

- *Substituição de decisões de sintonia por decisões de políticas de uso.* Em algumas situações, pode ser impossível priorizar a forma de alocação de recursos do sistema sem informações que sejam fornecidas pelo administrador [BMCL94]. Nestas situações, os SGBDs atuais deixam a alocação de recursos totalmente a cargo do administrador de banco de dados, gerando mais problemas de sintonia a serem endereçados. Uma alternativa a este cenário é deixar a alocação de recursos totalmente por conta do sistema, mantendo como configurações para o administrador apenas a especificação de políticas de uso. Um tipo de política que um administrador poderia especificar seria a de que uma determinada classe de transações sempre tivesse um tempo de resposta inferior a 1 segundo. Repare que neste

caso o administrador não informa como o sistema deve alcançar o objetivo de desempenho desejado, mas fornece informações suficientes para que uma priorização de recursos seja realizada entre as diversas classes de transações que são submetidas ao sistema. No caso do componente de auto-sintonia de estatísticas, poderíamos desejar especificar que determinadas consultas têm prioridade maior do que as demais. Para estas consultas com maior prioridade, o componente poderia obter estatísticas mais detalhadas para melhorar o plano de execução calculado pelo otimizador. É impossível para o componente decidir quais são as consultas mais prioritárias. Esta informação deve ser fornecida pelo DBA para configurar o comportamento do componente de auto-sintonia.

Mais técnicas relevantes especificamente para a construção de um componente de auto-sintonia de estatísticas podem ser encontradas na seção 5.6 deste documento. Nesta Seção, apenas utilizamos este componente como exemplo para ilustrar princípios que podem ser aplicados à construção de componentes de auto-sintonia em geral.

3 Uma Proposta de Classificação das Pesquisas em Auto-sintonia

Nesta seção, apresentamos uma classificação dos trabalhos encontrados na literatura que estudam questões de auto-sintonia em bancos de dados relacionais. Nossa preocupação, para a criação da classificação, foi a de buscar trabalhos que consigam, de alguma forma, *modelar o ambiente* em que o componente ou o sistema estão incluídos e *ajustar o componente ou o sistema automaticamente* a mudanças no ambiente. Assim, uma proposta de um novo algoritmo para realizar a serialização de uma determinada classe de transações, por exemplo, não necessariamente seria considerado um trabalho sobre auto-sintonia em nossa classificação; já um método que modele a carga submetida ao sistema para ajustar automaticamente o seu nível de multiprogramação (*MPL, multiprogramming level*) seria incluído em nosso estudo.

Mostramos, na Figura 2, a nossa proposta de classificação dos trabalhos estudados. As técnicas de auto-sintonia se enquadram em dois grandes grupos: auto-sintonia global e auto-sintonia local. As propostas de auto-sintonia global procuram estabelecer princípios gerais para a implementação de sistemas que se adaptem automaticamente ao seu ambiente. Além disto, este tipo de técnica também busca obter um *equilíbrio* entre os vários componentes do sistema de forma a alcançar um desempenho que seja globalmente melhor do que o obtido caso cada componente tomasse as decisões de sintonia isoladamente.

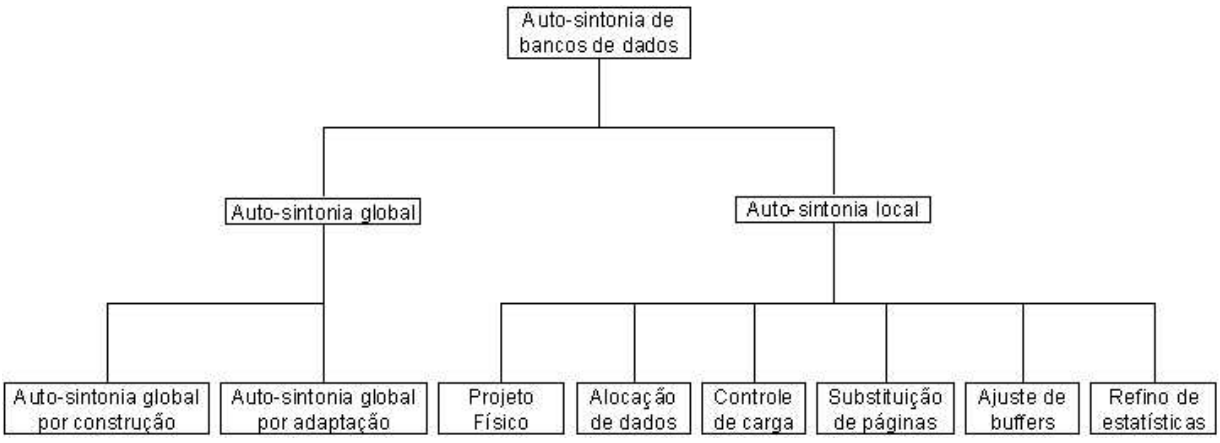

Figura 2: Classificação de propostas de auto-sintonia de bancos de dados

Os trabalhos de auto-sintonia global tendem a seguir uma de duas possibilidades para a sua

criação: por construção e por adaptação. Os sistemas com auto-sintonia global por construção são implementados de tal forma que o próprio sistema possui o conhecimento de modelos sobre os impactos de desempenho de sua interação com o ambiente. Assim, ele se adapta no decorrer de seu funcionamento para manter um desempenho aceitável mesmo diante de mudanças no ambiente. Uma vantagem deste tipo de abordagem é a eliminação dos parâmetros de sintonia do sistema, tornando a sua administração mais simples. Por outro lado, estes sistemas precisam lidar com a complexidade de integrar mecanismos de auto-sintonia com a base de código do próprio sistema. Esta pode ser uma grande dificuldade a se superar, conforme ressaltado em [CW00], e uma estratégia recomendada é já realizar a codificação inicial do sistema com os mecanismos de auto-sintonia global embutidos.

Outra possibilidade é estudar o sistema como se este fosse uma caixa-preta e criar um componente externo que ajusta os parâmetros e configurações de desempenho de acordo com um modelo previamente existente das interações entre o sistema e o seu ambiente. Em nossa proposta de classificação, este tipo de estratégia é denominada de auto-sintonia global por adaptação. Uma vantagem deste tipo de abordagem é a possibilidade de acoplarmos componentes de auto-sintonia global a sistemas já previamente existentes sem necessidade de interferir em seu código.Uma dificuldade é a necessidade de o componente criar algum tipo de modelo sobre o comportamento do sistema analisado [BHJS00]. Além disto, também é importante levar em conta que os mecanismos necessários para o monitoramento do sistema (observação) e para a execução de ações de sintonia (controle) devem ser acessíveis externamente. Os conceitos de observação e controle de sistemas são oriundos da área conhecida como Teoria de Controle [Oga97].

Já as propostas de auto-sintonia local procuram enfocar problemas específicos de sintonia que os sistemas atuais apresentam. Conseguimos encontrar na literatura e nos principais produtos comerciais propostas de auto-sintonia para os seguintes problemas:

- projeto físico, em especial seleção de índices;
- alocação de dados;
- controle de carga;
- substituição de páginas;
- ajuste de buffers;
- refino de estatísticas.

A definição de cada problema, bem como uma discussão sobre os trabalhos estudados, podem ser encontradas na seção 5.

Conforme podemos ver até mesmo pela classificação, existe um grande desequilíbrio na ênfase dada às pesquisas até o presente momento. Poucas são as propostas de auto-sintonia global em comparação com a quantidade de técnicas pesquisadas para auto-sintonia local. Isto se deve à própria complexidade de entender e estudar o comportamento do desempenho de um SGBD como um todo [WMHZ02]. A estratégia seguida na literatura foi a pesquisa das questões que devem ser consideradas para cada componente específico que pode ser sintonizado no sistema. Com isto, é possível ganhar mais conhecimento sobre que facetas do ambiente afetam cada componente do SGBD e, então sim, passar a investigar as relações existentes entre componentes que competem por recursos semelhantes. Pouco trabalho exploratório foi feito para investigar as relações existentes entre os ajustes de componentes distintos. Mesmo assim, a experiência de ajuste manual de bancos de dados indica que compreender estas relações pode ser bastante difícil [Sha96].

O fato de existirem poucas propostas de auto-sintonia global não invalida, entretanto, esta abordagem. Uma possibilidade que poderia ser investigada é a de construção de sistemas mais

simples e que sejam auto-sintonizáveis por projeto [CW00]. Para sistemas com menor escopo funcional, seria factível modelar analiticamente o comportamento de seu desempenho. Com isto, teríamos componentes menores e mais previsíveis, capazes de oferecer garantias quanto aos seus níveis de serviço.

Além de classificar as propostas estudadas, conseguimos, ainda, identificar alguns grupos de pesquisa dedicados ao estudo de questões de auto-sintonia de sistemas de bancos de dados ou de sistemas em geral. Os grupos identificados são:

- Projeto SMART (*Self Managing and Resource Tuning*) do Centro de Pesquisas IBM Almaden em parceria com os Laboratórios de Toronto e do Vale do Silício [SMA]. Faz parte da iniciativa de computação autônoma (*autonomic computing* [Hor01]) da IBM e procura aumentar as capacidades de auto-administração de bancos de dados. O foco é criar tecnologias para o DB2 UDB, conforme discutido em [LLZ02].

- Projeto AutoAdmin da Microsoft Research [AUT]. Objetiva criar sistemas de bancos de dados auto-sintonizáveis e auto-administráveis. Este é um alvo de longo prazo e até o momento o grupo se focou nos problemas de projeto físico (seleção de índices e visões materializadas) e de refino de estatísticas. A abordagem é permitir que o sistema registre os seus padrões de uso e se adapte às necessidades das aplicações. A tecnologia que é desenvolvida por este grupo está sendo integrada no Microsoft SQL Server.

- Laboratório de Sistemas de Bancos de Dados da Universidade de Queen's, Canada [LAB]. Mantém uma linha de pesquisa sobre ajuste dinâmico de SGBDs. O objetivo do projeto é investigar técnicas para que os SGBDs possam lidar com cargas de trabalho que possuam amplo leque de demandas por recursos e tempos de execução. Neste cenário, o grupo acredita que os DBAs deveriam especificar os objetivos de desempenho das diversas classes de transações e o sistema deve procurar atingir, sem intervenção, os objetivos estabelecidos. Este projeto de gerência de recursos orientada a objetivos é tocado em parceria com o grupo do IBM DB2 do Laboratório de Toronto.

- Grupo de Pesquisa em Bancos de Dados e Sistemas de Informação, da Universidade de Saarland, Alemanha [SAA]. Conduziu um projeto de pesquisa com resultados muito relevantes denominado projeto COMFORT (*COMfortable PerFORmance Tuning* [WHMZ94]). O grupo identificou alguns princípios gerais para construção de sistemas de bancos de dados capazes de auto-sintonia, mas seus trabalhos de implementação se focaram em problemas específicos em áreas como controle de carga, alocação de dados e substituição de páginas. Apesar destes trabalhos serem de origem acadêmica e não comercial, a implementação feita não está disponível publicamente. Pesquisas mais recentes deste grupo endereçaram os problemas de refino de estatísticas em bancos de dados relacionais através da criação automática de sinopses de dados [KW02] e de oferecimento de garantias de nível de serviço em sistemas de execução de *workflows*.

- Departamento de Sistemas Adaptáveis (*Adaptive Systems Department*) do Centro de Pesquisas IBM T. J. Watson [TJW]. Desenvolve tecnologias e metodologias para gerenciar mudanças em sistemas de computação, dentro do escopo da iniciativa da IBM de computação *on-demand*. Possui um projeto especialmente voltado para aplicar técnicas formais de teoria de controle e modelagem estatística para a regulação e otimização de sistemas computacionais. O grupo possui alguns trabalhos de auto-sintonia já publicados voltados para sistemas como servidores de email e servidores HTTP. Estudos sobre SGBDs também estão sendo conduzidos [DEF+03].

Além dos grupos de pesquisa citados, ainda temos alguns grupos envolvidos com o estudo de questões de auto-sintonia voltados para a elaboração de produtos comerciais. Alguns exemplos

seriam fornecedores de sistemas de bancos de dados, como a Oracle, e fornecedores de ferramentas para análise de desempenho, como Quest, Embarcadero e Veritas. Estas iniciativas serão melhor descritas na seção 4.

Também procuramos iniciativas de auto-sintonia em bancos de dados de código-fonte aberto, em especial nos sistemas mais populares MySQL e PostgreSQL. Não foi possível encontrar nenhum tipo de iniciativa de auto-sintonia sendo conduzida para melhorar o uso destes sistemas.

Até o momento, muito progresso foi feito no sentido de tornar SGBDs relacionais mais capazes de auto-sintonia. No entanto, ainda existem grandes desafios a serem vencidos [WMHZ02]. No restante deste documento, procuramos apresentar um breve resumo dos trabalhos que foram realizados e indicar alguns desafios que ainda precisam ser superados.

4 Auto-sintonia Global

Os sistemas de bancos de dados oferecem serviços baseados em um grupo de recursos finito, cuja alocação pode ser ajustada dependendo da carga de trabalho. O ajuste de como o sistema faz uso dos recursos disponíveis é feito através da variação de um conjunto de controles e deve objetivar a obtenção de níveis de serviço pré-definidos. Num sistema com auto-sintonia, é o sistema quem deve escolher os valores ótimos de suas variáveis de controle, de forma a satisfazer os requisitos de desempenho previamente estabelecidos pelo administrador ou usuário do sistema. A análise deve ser feita enxergando o sistema como um todo, dado que alterações de um parâmetro podem se refletir em outros.

A auto-sintonia global de sistemas de computação é um problema muito complexo, que apresenta, entre outros desafios:

- A detecção de problemas de desempenho não é trivial, uma vez que um problema detectado em um recurso pode ter sido causado por contenção por outro.
- É difícil predizer o desempenho do sistema após determinada mudança, devido ao fato das inter-relações que existem entre os recursos e as cargas de trabalho serem ainda pouco conhecidas. As inter-relações entre recursos diferentes também podem afetar o desempenho do sistema de formas não compreendidas.
- Um grande grupo de métricas devem ser obtidas e alguma medida de desempenho definida. No caso desta medida ser definida através de objetivos de desempenho, alguma forma simples de interação com o usuário deverá ser criada para que ele os defina para cada classe de carga.
- As modificações devem ser efetuadas sem prejudicar o sistema.
- A obtenção das métricas de desempenho deve minimizar o consumo de recursos de sistema para não comprometer a sua corretude nem afetar o sistema observado.

Vários trabalhos sobre auto-sintonia têm assumido a abordagem global, não somente objetivando controlar sistemas de bancos de dados, mas também voltados para servidores HTTP, sistemas operacionais e sistemas de computação em geral.

Nesta seção descreveremos brevemente algumas das pesquisas relacionadas à auto-sintonia global de sistemas computacionais em geral e de sistemas de bancos de dados em particular. Em primeiro lugar, revisaremos conceitos da área de agentes de *software* que têm sido empregados no contexto de auto-sintonia de sistemas de bancos de dados.

4.1 Agentes

Alguns trabalhos de auto-sintonia têm investigado o uso da tecnologia de agentes de *software* [3][BHJS00, DHPB03, Mac00, CL02], devido a suas características de autonomia e adaptabilidade. Segundo [Woo99], agentes de *software* são sistemas de computação que estão situados em um ambiente, capazes de ações autônomas nesse ambiente para atingir os seus objetivos de projeto.

A questão da construção de SGBDs baseados em agentes foi levantada em [Mac00], em que são propostas três arquiteturas para agentes em SGBDs:

1. Em camadas: Permite implementar agentes sobre SGBDs existentes sem ter praticamente que modificar o SGBD objetivo. A ação dos agentes é limitada pois devem acessar aos componentes do SGBD através das interfaces oferecidas.

2. Embutida: Os agentes são embutidos dentro de SGBDs já construídos para aumentar e estender suas funcionalidades. Esta arquitetura tem limites impostos pelas funcionalidades oferecidas pelo SGBD.

3. Integrada: Os componentes do SGBD são implementados como subsistemas de agentes. A integração dos agentes no sistema aporta mais flexibilidade e controle que nas outras arquiteturas, mas ao mesmo tempo traz mais dificuldades de implementação.

4.2 Auto-sintonia de sistemas

Uma referência obrigatória no tema de auto-sintonia de sistemas, é o manifesto da IBM sobre computação autônoma [Hor01]. Nele, é feito um chamado à construção de sistemas integrados, autônomos, capazes de se adaptar automaticamente às variações do ambiente, em que as operações de gerência foquem o sistema inteiro e não a soma das partes por separado. Soluções para este problema têm sido propostas em trabalhos como [Bar93, BHJS00, Hel97], entre outros.

O trabalho de [Bar93] foi um dos primeiros trabalhos a tratar o tema da auto-sintonia global de sistemas de computação. O trabalho propôs a construção de um sistema baseado em uma arquitetura que separasse as tarefas de definição de expectativas, monitoramento de desempenho, comparação das medições com as expectativas, e execução de ações como resultado desta comparação (modificações no sistema ou novas medições). A atividade de comparação e análise deveria concentrar-se em um agente único, o que permitiria ter uma visão global do sistema e simplificar a sua implementação. Os componentes propostos para a implementação dessa arquitetura são sensores e efetuadores [RN95].

Com o objetivo de diminuir os custos da análise, o agente de análise executa o monitoramento hierarquicamente: se as expectativas das métricas de alto nível são satisfeitas, as métricas de baixo nível não são monitoradas. Os resultados das medições são armazenados num banco de dados para análise posterior.

Em [Hel97] é descrita a tecnologia de Sistemas Automáticos de Sintonia (Automated Tuning Systems - ATS). A arquitetura proposta não somente consegue propror ações de sintonia como também automatiza o processo de ajuste através da adição de uma camada de controle de realimentação sobre o sistema gerenciado, ou sistema objetivo.

As etapas que este trabalho coloca como passos a serem executados por um ATS durante o processo de sintonia são: detecção, diagnose, ação e avaliação. A Figura 3 mostra a operação de um ATS.

O controlador do ATS compara as políticas de sintonia ou expectativas especificadas pelo administrador do sistema com os dados de desempenho observados no sistema controlado (etapa de detecção), determina a causa do problema (etapa de diagnose) e executa ações através dos

[3]que chamaremos aqui simplesmente de agentes

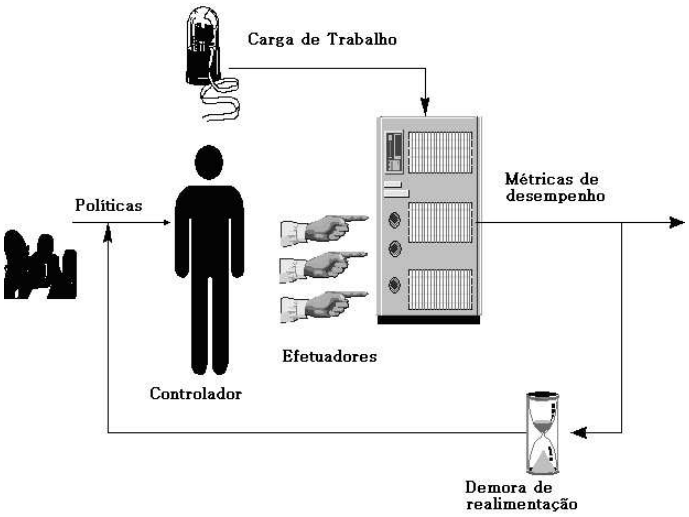

Figura 3: Operação de um ATS [Hel97]

efetuadores no caso das expectativas não serem satisfeitas (etapa de ação). É preciso, para que estas ações possam ser executadas em tempo real, que existam os controles de sintonia correspondentes aos parâmetros a serem ajustados. Posteriormente, é executada uma etapa de avaliação onde se determina se os objetivos foram atingidos.

A implementação de sistemas genéricos para auto-sintonia é abordada em [BHJS00, DHPB03] para os casos do controle de um servidor Lotus Notes e Apache, respectivamente. Em ambos os trabalhos foi utilizado o AutoTune, um *framework* de agentes que permite controlar a alocação de recursos em sistemas distribuídos heterogêneos a partir de determinadas políticas de sintonia definidas pelo administrador do sistema. O fato de que não é preciso o conhecimento prévio do sistema objetivo permite reutilizar o *framework*, além de possibilitar uma maior adaptabilidade às mudanças do sistema objetivo.

Em cada caso, três agentes são construídos usando o ABLE (*Agent Building and Learning Environment*) [BHJS00] para o controle de auto-sintonia:

- Um agente de modelagem, que muda os parâmetros de sintonia do sistema objetivo e utiliza os dados de desempenho coletados e armazenados no repositório AutoTune para construir um modelo que vai refletir as relações dinâmicas entre os parâmetros de sintonia e as métricas de desempenho. É este aprendizado que faz com que o sistema objetivo possa ser genérico. No caso do AutoTune, isto é feito usando redes neurais. Este modelo pode ser modificado no caso de mudanças no sistema objetivo.

- Um agente que aprende o modelo do sistema.

- Um agente que determina os valores apropriados para os parâmetros de controle.

Problemas de demoras no monitoramento e no ciclo de realimentação podem fazer com que seja necessário também um mecanismo para predizer a carga de trabalho. Em [DEF+03] a arquitetura de [BHJS00] é aplicada para a auto-sintonia de parâmetros de um SGBD.

Estes trabalhos são alguns dos resultados publicados pelo grupo do Departamento de Sistemas Adaptáveis do Centro de Pesquisas IBM T. J. Watson (veja a seção 3). Outros trabalhos desse grupo tratam de problemas de detecção automática de padrões de modelos de carga, assim como

de problemas de desempenho, caracterização de carga de trabalho, e navegação por bases de conhecimento, entre outros.

Finalmente, existem também trabalhos que tratam da auto-sintonia de sistemas específicos, como é o caso dos sistemas operacionais. Um exemplo é [FN99], que propõe um mecanismo de auto-sintonia para sistemas operacionais baseado na redução do espaço de busca dos parâmetros de sintonia usando algoritmos genéticos. Este trabalho afirma que os algoritmos genéticos provêem rápida e facilmente um grupo de parâmetros do sistema que oferecem um desempenho razoável (ainda que não garantam que seja o ótimo). Um resultado interessante do trabalho é a comparação entre métodos de geração dos parâmetros combinados para formar os cromossomos. Alguns dos sistemas sintonizados usando cromossomos gerados randomicamente atingiram melhor desempenho do que aqueles em que os parâmetros foram gerados por algoritmos de sintonia individuais. A conclusão é que a combinação dos valores ótimos dos algoritmos individuais pode estar longe da solução ótima global. Outros sistemas objetivo de trabalhos de auto-sintonia são os sistemas de bancos de dados.

4.3 Auto-Sintonia Global de Sistemas de Bancos de Dados

A otimização do desempenho de um sistema deveria focar o sistema como um todo ([SB03, Moh92, WMHZ02, CL02]) e não simplesmente os diferentes módulos funcionais em separado. As interações entre os componentes podem provocar uma desestabilização do sistema a partir de ações de sintonia aplicadas a um recurso particular. Por exemplo, a otimização de consultas deve considerar os custos de CPU associados com o controle de concorrência [Moh92].

A abordagem global do processo de sintonia de SGBDs não é muito comum entre as pesquisas relacionadas ao tema de auto-sintonia. Pelo contrário, a grande maioria das propostas trata de mecanismos para resolver problemas localizados de sintonia (veja a seção 5), em sua grande maioria não situados dentro do ambiente real do SGBD onde existem interações entre os recursos. Exceções notáveis neste sentido são os trabalhos de [WHMZ94], em que se discutem questões arquiteturais da implementação de um sistema com auto-sintonia, [CL02], em que é descrito um mecanismo para gerência de índices dentro de uma arquitetura global de auto-sintonia utilizando agentes de *software* (veja a seção 5.1), e [Ben03], que propõe um *framework* para o diagnóstico de problemas de desempenho no sistema.

Várias empresas oferecem ferramentas de diagnóstico de desempenho para bancos de dados, tanto as produtoras de SGBDs como empresas independentes (veja a seção 6). Todas estas têm em comum o fato de que suas ferramentas se limitam a sugerir ações e não as executam. Como foi apontado por [Hel97], as sugestões admitem imprecisões que não são admissíveis na hora de tomar ações concretas. Além disso, a implementação de sistemas de auto-sintonia precisa do acesso às métricas do sistema objetivo, assim como da possibilidade de modificação dinâmica dos parâmetros de sintonia, seja através de APIs (*Application Program Interface*) ou de variáveis e funções se o mecanismo estivesse embutido dentro do sistema a controlar.

Embora existam algumas outras publicações relativas ao tema de auto-sintonia global de SGBDs, a maioria dos trabalhos sobre o tema vêm dos grupos do projeto AutoAdmin da Microsoft Research [CW00, CN98a, CN98b], do grupo da Universidade de Saarland [WMHZ02, WHMZ94], e do Laboratório de Sistemas de Bancos de Dados da Universidade de Queen's, Canada [Ben03, MPLR02].

As propostas de auto-sintonia global de bancos de dados podem dividir-se em duas vertentes: os sistemas que são projetados para satisfazer requisitos de desempenho e os sistemas aos que são acrescentados módulos para efetuar tarefas de auto-sintonia.

Auto-sintonia por Projeto

Representando a primeira das vertentes, no *position paper* [CW00], os autores afirmam que os SGBDs atuais sofrem de uma complexidade inerente que é muito difícil de ser atacada. A quantidade excessiva de características e modos que o sistema oferece para o seu uso faz com que a tarefa de sintonia exija um conhecimento muito abrangente e ao mesmo tempo especializado sobre o sistema. Assim, é proposto que passemos a *construir sistemas de bancos de dados mais simples*, para os quais seja possível modelar formalmente o seu comportamento e a sua necessidade de ajuste. Idealmente, estes sistemas seriam especializados em determinadas funções de gerência de dados e seriam completamente capazes de auto-sintonia por projeto.

A idéia de quebrar os SGBDs em módulos para facilitar a gerência é proposta também em [HA03, McC03]. O primeiro apresenta o desenho de um SGBD de alto desempenho que ao mesmo tempo é escalonável e fácil de gerenciar. A proposta é quebrar o SGBD em módulos e encapsulá-los em etapas auto-contidas conectadas por filas. Cada etapa é um servidor independente com seu próprio contexto formado por sua fila, suporte a *threads* e gerência de recursos. Os servidores se comunicam e interagem com as outras etapas através de uma interface bem definida. As etapas aceitam pacotes que contém requisições e dados privados, efetuam operações sobre essa informação e podem enviar este ou novos pacotes a outras etapas, simulando uma fila de produção.

Os autores afirmam que este tipo de arquitetura promove a autonomia das etapas, a localidade dos dados e das instruções, minimiza o uso de variáveis globais e facilita a sintonia do SGBD. Os resultados deste trabalho estão baseados em simulações, mas um protótipo está sendo implementado.

Na mesma linha de criação de SGBDs baseados em componentes, em [McC03] se propõe a construção de SGBDs não necessariamente relacionais baseados em componentes de baixa granularidade, sendo que os componentes são entidades concretas que consistem em implementação e interfaces. O trabalho afirma que a componentização deve abarcar também o sistema operacional. Os blocos resultantes são configurados ou reconfigurados automaticamente, o que resulta em uma gerência de recursos realmente adaptativa. O interesse específico deste trabalho está na computação ubíqua, que possui altos requisitos de flexibilidade e eficiência de tamanho das aplicações.

Uma preocupação ante este tipo de propostas é se a eficiência realmente conseguirá ultrapassar as de implementações atuais de SGBDs. A factibilidade de uma implementação eficiente do trabalho de [McC03] é inferida a partir de experiências anteriores no desenvolvimento de núcleos de sistemas operacionais baseados nos mesmos tipos de componentes, em que a velocidade e os requerimentos de espaço melhoraram.

Até onde conhecemos, não existem ainda sistemas de bancos de dados implementados seguindo a abordagem de auto-sintonia por projeto.

Auto-sintonia por Adaptação

A segunda vertente tem sido mais bem sucedida. A equipe de Universidade de Saarland fez uma pesquisa como parte do projeto Comfort [WHMZ94] para a determinação de princípios arquiteturais da auto-sintonia de bancos de dados e sistemas de transações. As conclusões deste trabalho são discutidas em [WMHZ02], em que se expõe como princípio fundamental o conceito de ciclo de controle de realimentação (*feedback control loop*) como método de controle do efeito sobre o desempenho das mudanças executadas durante ações de auto-sintonia. Este ciclo é composto pelas fases de observação, predição e reação. A etapa de observação consiste em monitorar as métricas de desempenho e detectar problemas. Nessa etapa, o ponto fundamental é a determinação das métricas que caracterizem o desempenho do sistema. A predição deve calcular as vantagens da execução de cada possível ação de sintonia, escolher a melhor e determinar

a intensidade da mudança. A reação executa a ação escolhida de acordo com a intensidade determinada. O objetivo do trabalho foi fundamentalmente a determinação de um conjunto de políticas que permitam decidir como ajustar os parâmetros do sistema.

A proposta de [WHMZ94] apresenta resultados experimentais para a aplicação destes conceitos a vários problemas de auto-sintonia local, como controle de carga (veja a seção 5.3), substituição de páginas de memória (veja a seção 5.4), e alocação de dados (veja a seção 5.2). Apesar de apresentar princípios gerais de construção, o trabalho não chegou a uma implementação de um sistema que realmente realize auto-sintonia global. Foram discutidos apenas os princípios relacionados com este tema, aderindo ao princípio já mencionado anteriormente de "Pensar globalmente, consertar localmente"[SB03].

O objetivo de pesquisa do projeto AutoAdmin da Microsoft Research é a construção de SGBDs auto-sintonizáveis e auto-gerenciáveis. Como resultado dos trabalhos, até agora na área de projeto físico e refino de estatísticas, foi incluído no SQL Server o *Index Tuning Wizard*, uma ferramenta que pode sugerir os índices e visões materializadas apropriados para determinada carga de trabalho. No entanto, o grupo ainda não publicou informações referentes à abordagem que vai ser usada para a auto-sintonia do sistema.

Poucos trabalhos propõem a implementação de um sistema específico para a auto-sintonia global de SGBDs. Podemos destacar [CL02], que descreve a única arquitetura global que conhecemos para a integração de mecanismos locais de auto-sintonia, e [MPLR02, Ben03], onde se propõe um sistema para auto-sintonia global chamado Quartermaster.

Em [CL02] é proposto um sistema local e global de auto-sintonia baseado em agentes, dentro do contexto de uma arquitetura baseada em agentes de *software* para auto-sintonia de índices (veja a seção 5.1). A introdução de agentes no sistema permite tomar tanto posturas reativas, como também pró-ativas, e não se limita a sugerir a criação ou destruição de índices, mas executa as ações de forma automática. Por exemplo, um agente poderia monitorar o fato de que o otimizador está escolhendo realizar junções preferencialmente usando técnicas de *hash* e isto poderia ser comunicado para um outro agente responsável pela gerência de memória do sistema.

O modelo de auto-sintonia global proposto no trabalho inclui um repositório que garante a persistência das experiências passadas, de forma que uma decisão que afetou negativamente o desempenho do sistema não será executada novamente. O modelo prevê também a possibilidade de comunicação entre os componentes de auto-sintonia, o que poderia ser útil em caso de ser conhecida a inter-relação entre os componentes. Assim, alguma ação poderia ser executada de modo preventivo para que mudanças em um componente não afetassem outros.

A interação entre os componentes do modelo é apresentada na Figura 4.

Este tipo de abordagem implica, no entanto, que cada agente pode necessitar manter crenças não apenas sobre si mesmo, mas, também, sobre os estados e ações dos outros agentes [BGK+95].

Em [MPLR02] se descreve um sistema chamado Quartermaster (veja a Figura 5) para a alocação dinâmica de recursos em um sistema de banco de dados, de forma a satisfazer determinados níveis de serviço no âmbito de um sistema de comércio eletrônico. A carga de trabalho é dividida em classes de transações, cada uma com seus próprios objetivos de desempenho que devem ser satisfeitos.

A gerência dinâmica de recursos orientada a objetivos permite ao DBA especificar seus níveis de desempenho desejados e abstrair quais são as configurações que permitem atingir esses objetivos. Isto implica que o sistema tem que traduzir esses objetivos de desempenho para ajustes de recursos.

O Monitor coleta os dados de desempenho para cada classe de transação e os armazena no repositório de gerência. Este repositório contém também descrições das classes de desempenho e regras de sintonia fornecidas pelo administrador. O analisador examina periodicamente os dados coletados procurando violações dos objetivos de desempenho. No caso de existir alguma, a informação é transferida ao Planejador, que usa essa informação e os dados no repositório para

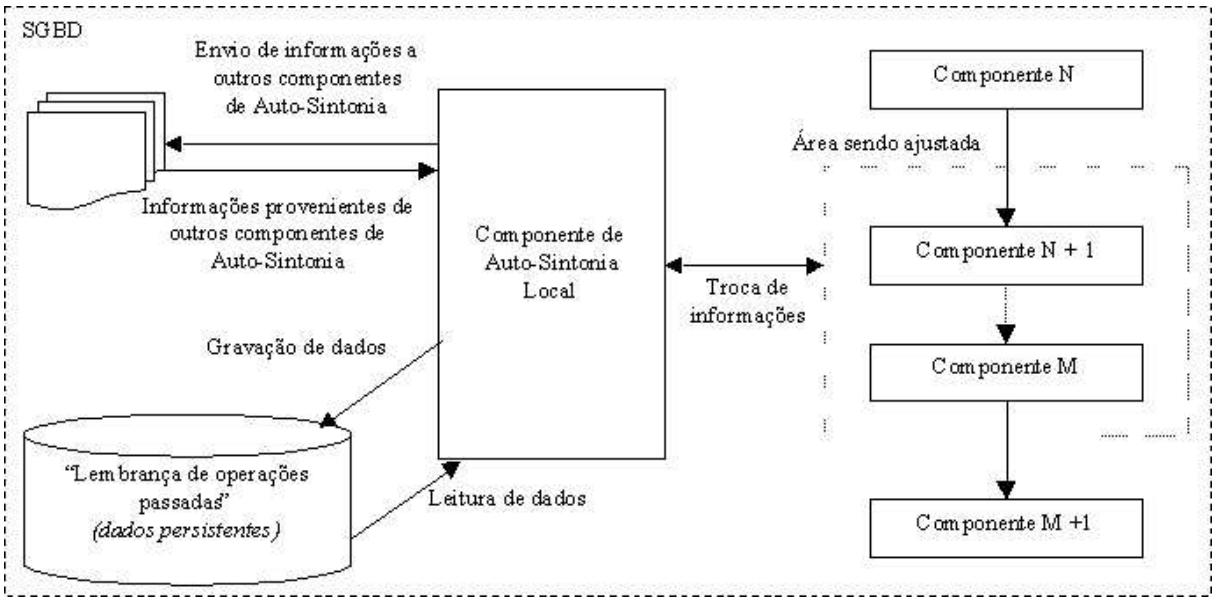

Figura 4: Esquema global de Auto-Sintonia com comunicação entre componentes e armazenamento persistente de dados [CL02]

propor um ou mais ajustes dos parâmetros de sintonia do servidor. O processo é registrado no Log de Eventos para explicar a decisão posteriormente. O Planejador apresenta as propostas de sintonia ao DBA, este escolhe uma e o Controlador a executa.

Um protótipo desse sistema foi implementado para o DB2 UDB.

O *framework* e o algoritmo de diagnóstico implementado no módulo chamado de Planejador na Figura 5, foi descrito em [Ben03]. Como parte do trabalho, foi construído um modelo de recursos, um modelo de carga de trabalho, regras de diagnose e uma árvore de diagnose baseada tanto no modelo de carga de trabalho quanto nas regras de diagnose. Esta informação é extraída da documentação do SGBD e da experiência dos DBAs. No processo de diagnose do sistema a árvore é percorrida e são avaliados os recursos cuja sintonia pode ser a solução do problema. Estes recursos são inseridos em um lista de ajustes possíveis.

A determinação de qual recurso será ajustado é feita por algoritmos de sintonia que escolhem qual ajuste oferecerá o melhor desempenho para o sistema. O modelo de recursos contém informações sobre os recursos do SGBD e as relações entre eles. Assim, podemos utilizá-lo para determinar os impactos que serão gerados pelo ajuste de qualquer um dos recursos da lista de ajustes possíveis.

Como resultado do trabalho, o sistema conseguiu reagir a mudanças de uma carga de trabalho OLTP. O ajuste dos recursos sugerido pelo sistema de diagnose foi feito manualmente devido ao fato de que o DB2 não possui ainda os mecanismos necessários para efetuá-los dinamicamente. Alguns trabalhos futuros a serem realizados neste projeto são o estudo da convergência do diagnóstico de modo que o sistema chegue a um estado de desempenho ótimo, a geração automática da árvore de diagnóstico, a integração do sistema de diagnóstico com o SGBD, e a aplicação do Quartermaster em outros domínios de aplicação.

5 Auto-sintonia Local

Esta seção apresenta descrições das propostas classificadas como de auto-sintonia local. Cada trabalho foca seus esforços na automatização dos ajustes do sistema para resolver um entre os seguintes problemas:

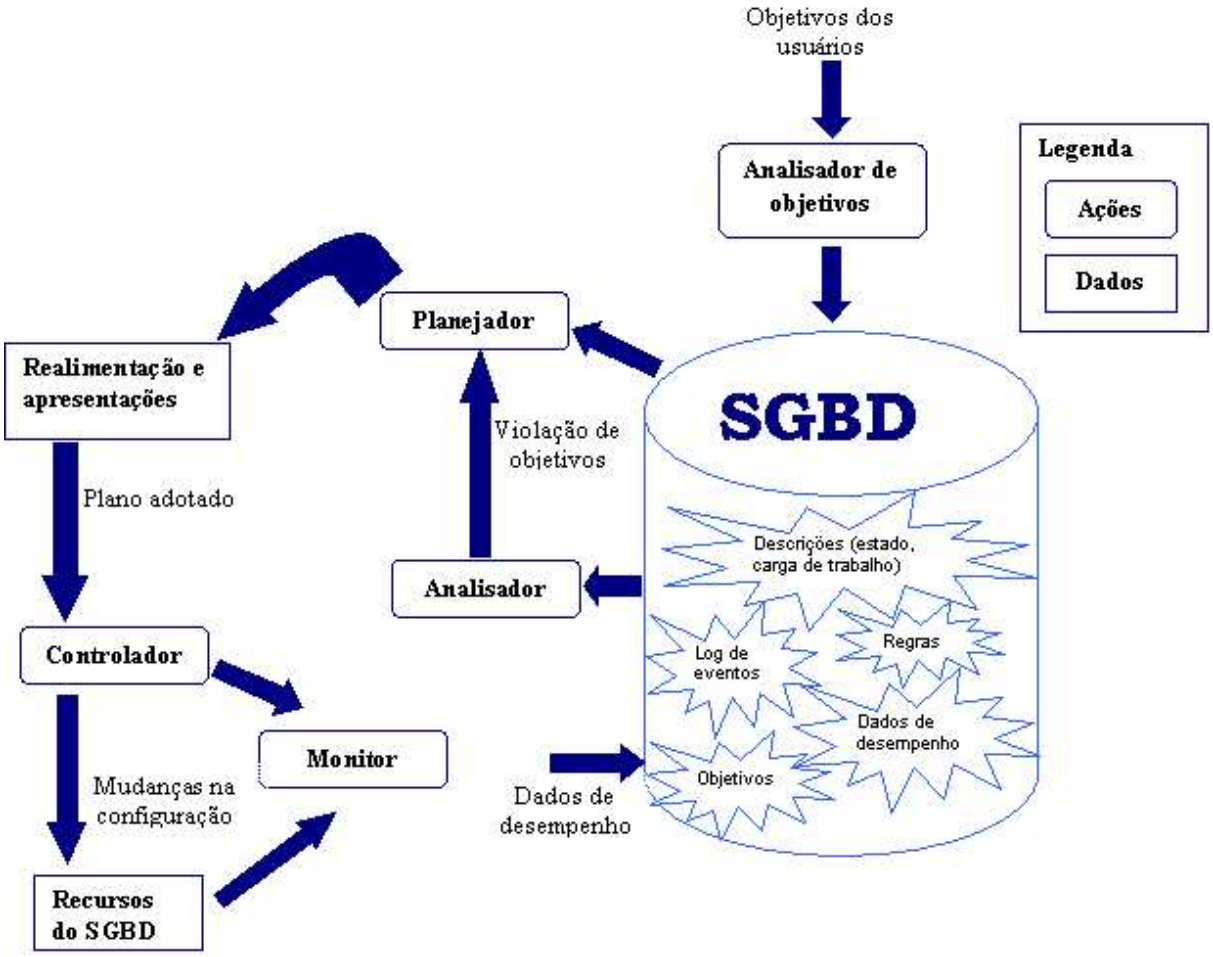

Figura 5: Arquitetura do Quartermaster [Ben03]

- projeto físico: o problema consiste em escolher a melhor organização física para um dado esquema e uma carga de trabalho específica. Muito trabalho foi investido nesta área para prover ferramentas que auxiliam na seleção de índices para o banco de dados.
- alocação de dados: dados N elementos de armazenamento, devemos determinar como podemos alocar e realocar dinamicamente fragmentos de arquivos ou relações nestes elementos de tal forma que o equilíbrio de carga seja o melhor possível mesmo diante de mudanças nos padrões de acesso da carga de trabalho.
- controle de carga: em um sistema que processa múltiplas transações de forma concorrente e que se utiliza de bloqueios (*locks*) para garantia das propriedades ACID, o objetivo é regular o nível de multiprogramação do sistema de forma a evitar que a sua vazão (*throughput*) caia devido a conflitos de bloqueio (*thrashing* de contenção de dados).
- substituição de páginas: o problema é manter em memória o conjunto de páginas mais populares do banco de dados, mesmo quando a carga de trabalho a que o sistema é submetido muda.
- ajuste de buffers: procura-se estudar quantos buffers distintos devem ser configurados no banco de dados e de que forma devem ser distribuídos os objetos (tabelas e índices) para estes buffers para que o desempenho aumente.

- refino de estatísticas: o objetivo é escolher quais estatísticas devem ser criadas no banco de dados e refinar estas estatísticas sem necessidade de executar comandos específicos para coleta no sistema.

É importante deixar claro que os problemas listados acima não são as únicas questões de sintonia que poderiam ser estudadas seguindo o enfoque de auto-sintonia local. Estes são problemas que foram pesquisados na literatura até o presente momento. Exploramos, nas próximas subseções, cada um destes problemas em maior detalhe.

5.1 Projeto Físico

O projeto físico de bancos de dados relacionais consiste na seleção de um conjunto de estruturas de dados que irão materializar e dar suporte às relações definidas no modelo lógico de forma a acelerar o processamento da carga de trabalho submetida ao sistema. Assim, diversas organizações de arquivo devem ser consideradas, como por exemplo arquivos seqüenciais, índices, estruturas particionadas e visões materializadas, entre outros. O administrador de banco de dados tipicamente precisa escolher quais estruturas físicas utilizar para permitir que o sistema apresente um desempenho adequado. Automatizar estas escolhas representaria um grande passo em direção a sistemas capazes de auto-sintonia.

Em [RS91] é apresentado um arcabouço (*framework*) para a seleção de estruturas de arquivos aplicáveis a uma determinada carga de trabalho. O trabalho propõe que modelemos cada estrutura de arquivo possível como uma *característica* que pode ser aplicada ao esquema do banco de dados. É proposta uma estratégia de busca que enumera as *características interessantes* para as consultas presentes na carga de trabalho e avalia se o uso destas características produz planos de menor custo para a execução conjunta destas consultas. Uma limitação da proposta é a de que a estratégia empregada para busca em [RS91] é baseada em regras especializadas, o que pode representar uma dificuldade de manutenção para o sistema à medida que os otimizadores de consultas evoluem e estas regras precisam ser adaptadas.

Nas próximas subseções, iremos enfocar alguns aspectos do problema de projeto físico. Estes aspectos foram estudados em diversos trabalhos na literatura e se dividem em:

- seleção e auto-sintonia de índices;
- escolha de visões materializadas;
- determinação do particionamento de relações em sistemas de bancos de dados paralelos.

Seleção de Índices

Uma das tarefas que DBAs precisam realizar é a escolha de índices que possam melhorar o desempenho das consultas submetidas ao SGBD. O problema de seleção de índices em bancos de dados relacionais consiste em, dada uma carga de trabalho $W = \{(Q_i, f_i), i = 1, \ldots, n\}$ em que Q_i é uma consulta SQL e f_i é a sua freqüência de submissão, minimizar o custo total de processamento de W pelo sistema respeitando um limite de espaço S disponível para a criação de índices. O custo total de processamento é a soma dos custos de acesso, atualização e manutenção dos índices. Já foi demonstrado que uma versão restrita do problema de seleção de índices é NP-difícil [Com78].

Existem várias propostas na literatura que atacam o problema de seleção de índices em bancos de dados. As primeiras tentativas de solução procuraram modelar formalmente o problema a fim de caracterizar as estatísticas a serem coletadas pelo sistema e os critérios de decisão para criação dos índices. Conforme pode ser visto em [HC76], a modelagem formal do problema mesmo com o escopo reduzido a consultas e atualizações envolvendo uma única relação indexada com árvores

B+ encontra diversas dificuldades, tanto na qualidade das estatísticas a serem acumuladas quanto na própria complexidade do modelo criado. O artigo propõe que o próprio sistema de banco de dados possua um intervalo fixo, ajustável pelo DBA, de observação das estatísticas sugeridas e que, ao final de cada intervalo, ocorra um processo de reorganização com base no que foi coletado nos últimos intervalos. Entre outros trabalhos que seguem a mesma linha de atuação, podemos citar [AB77], [Kin74], [MS78] e [Sch75].

O trabalho de [Wha85] define a propriedade de separabilidade de custos de métodos de junção para permitir que resultados de escolhas de índices para tabelas individuais possam ser combinados para a determinação de índices sobre múltiplas tabelas. Infelizmente, nem todos os métodos de junção apresentam a propriedade de separabilidade, uma vez que a escolha de índices sobre uma das tabelas envolvidas na junção pode afetar quais índices devem ser criados sobre a outra tabela. Em [FST88], algumas observações fundamentais para lidar com o problema de seleção de índices são realizadas. O trabalho propõe que seja criada uma ferramenta de sugestão de índices que possua as seguintes características:

- consiga encontrar soluções para o problema envolvendo múltiplas tabelas simultaneamente;
- não imponha restrições sobre os comandos SQL da carga de trabalho, como, por exemplo, que os critérios para restrição de consultas sejam apenas comparações de colunas com constantes;
- permita a seleção de índices primários (*clusterizados*), bem como de índices secundários (*não-clusterizados*);
- não apresente discrepâncias entre o modelo de custos empregado pela ferramenta e o modelo utilizado pelo sistema para a escolha de quais métodos de acesso serão empregados. Em alguns trabalhos, como por exemplo [CBC93], são desenvolvidos modelos de custos detalhados para a ferramenta de seleção de índices. Estes modelos não estão em sincronia com o modelo utilizado pelo otimizador de consultas do sistema. Isto não é desejável, uma vez que o otimizador pode não realizar as mesmas suposições sobre custos que a ferramenta. Esta discrepância pode causar, até mesmo, a escolha de métodos de acesso distintos dos previstos pela ferramenta para o processamento das consultas.

O trabalho mostra a implementação, junto ao Sistema R, da IBM, de uma ferramenta que possui as características acima mencionadas. Vale dizer que a estratégia empregada para avaliar uma configuração física candidata com o otimizador de consultas é a criação, no catálogo do sistema, de um conjunto de réplicas das tabelas envolvidas na configuração. Estas réplicas contêm exatamente as mesmas estatísticas que as tabelas originais e a criação de índices sobre estas réplicas possui custos baixos uma vez que elas não possuem extensão física. Os custos de execução das consultas são estimados através do comando EXPLAIN do sistema [FST88].

Em [FON92], é seguido o mesmo conceito de utilizar o modelo de custos do próprio otimizador do sistema para guiar as decisões sobre quais são as melhores configurações de índices. O trabalho propõe que a interface de chamada ao otimizador seja estendida com a capacidade de considerar um conjunto de *índices hipotéticos* durante a otimização. Um índice hipotético (ou virtual) é um índice considerado pelo otimizador para a geração de um plano de execução, mas que existe apenas no esquema do SGBD e não fisicamente. É proposta uma estratégia de busca baseada na criação de um grafo de benefícios que registra quais ganhos de custos, conforme medido pelo otimizador, são obtidos em diferentes configurações de índices hipotéticos e reais. A melhor configuração hipotética encontrada é recomendada para o usuário da ferramenta ou criada após um período de observação fixo configurado no SGBD. Outras propostas que apresentam algoritmos para busca de configurações de índices são [ISR83], [BPS90] e [CFM95].

Uma dificuldade presente nestes trabalhos é a suposição de que os índices a serem criados para o próximo período de operação do SGBD podem ser derivados diretamente a partir do último período de observação fixo realizado. Esta suposição não é verdadeira para, por exemplo, aplicações OLTP com características de sazonalidade. Outra limitação, esta de natureza mais algorítmica, é a enumeração de índices candidatos com múltiplas colunas de uma forma eficiente. Muitos trabalhos enfocam apenas a enumeração de índices com uma coluna.

Como o problema de seleção de índices é relevante tanto do ponto de vista acadêmico quanto do ponto de vista da indústria, alguns dos principais fornecedores de bancos de dados investiram recursos no aprimoramento de ferramentas para a recomendação de índices. Em [ACN00, CN98a, CN98b, CN97] é descrita a ferramenta de seleção de índices implementada para o Microsoft SQL Server, o *Index Tuning Wizard*. Podemos ver a sua arquitetura na Figura 6.

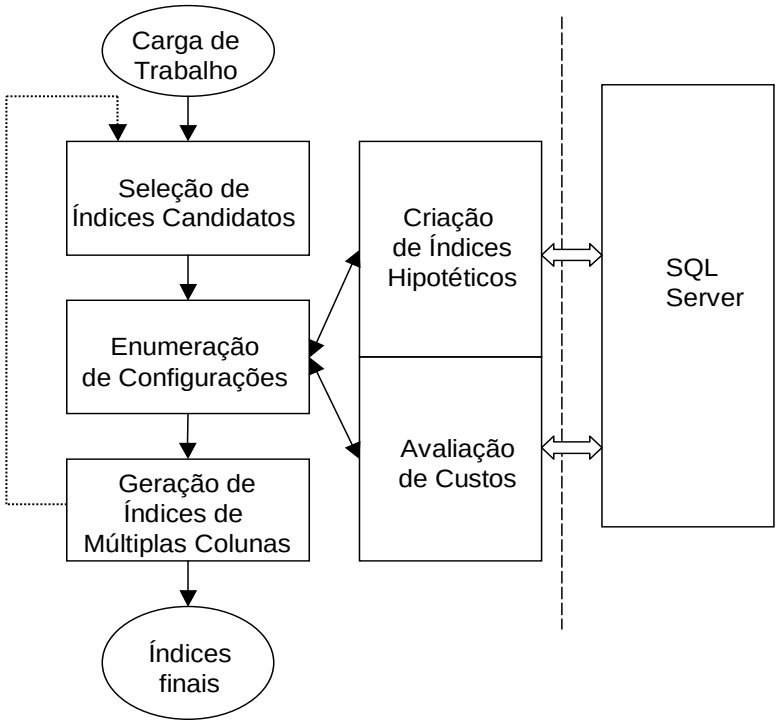

Figura 6: Arquitetura do Index Tuning Wizard [CN97]

É dada uma carga de trabalho como entrada para o Index Tuning Wizard com n consultas SQL. A ferramenta realiza, então, um conjunto de iterações que procuram derivar índices com um número crescente de colunas. A vantagem deste tipo de abordagem é a possibilidade de se basear em índices úteis com menos colunas para poder enumerar índices de múltiplas colunas de forma mais eficiente. A cada iteração, a ferramenta divide a carga de trabalho inicial em suas n consultas, que são analisadas independentemente. O resultado de cada uma destas análises é um conjunto de índices candidatos. O conjunto de todos os índices candidatos é usado como entrada para selecionar configurações de índices que sejam adequadas para a carga de trabalho inicial como um todo. Estas configurações de índices somente levam em conta índices de múltiplas colunas até o número de colunas estabelecido como limite para a iteração corrente.

Reforçamos que as únicas modificações feitas no SGBD nesta proposta são a possibilidade de criar índices hipotéticos e a capacidade do otimizador de consultas levar em consideração os mesmos no processo de otimização. O processo da ferramenta de seleção acessa de forma remota estes serviços que são oferecidos pelo processo do SGBD. Em [ACN00] esta ferramenta foi estendida para sugerir tanto a criação de índices quanto de visões materializadas. O mesmo grupo de pesquisa, em [CN99], estudou, ainda, a possibilidade de realizar a fusão dos índices

encontrados como resultado da ferramenta anteriormente construída. A idéia de realizar uma fusão de índices (*index merging*) é encontrar um conjunto de índices que tenha menor custo de atualização e manutenção dos que o conjunto original, mas que preserve a qualidade no tempo de resposta obtido para a resolução dos comandos da carga de trabalho.

As propostas anteriores utilizam o otimizador para avaliar possíveis conjuntos de índices, mas nunca para recomendar os índices que devem ser empregados. Em [LVZ+00], temos a proposta de uma ferramenta para seleção de índices para o IBM DB2. Este trabalho sugere uma integração muito mais próxima entre o otimizador de consultas do banco de dados e a funcionalidade de seleção de índices. É sugerido que o próprio otimizador do SGBD tenha a capacidade de enumerar os melhores índices para uma dada consulta SQL. Para isto, todos os índices hipotéticos relevantes são enumerados antes do processo de otimização e fica totalmente a cargo do otimizador selecionar quais índices serão efetivamente selecionados. A arquitetura desta ferramenta é mostrada na Figura 7.

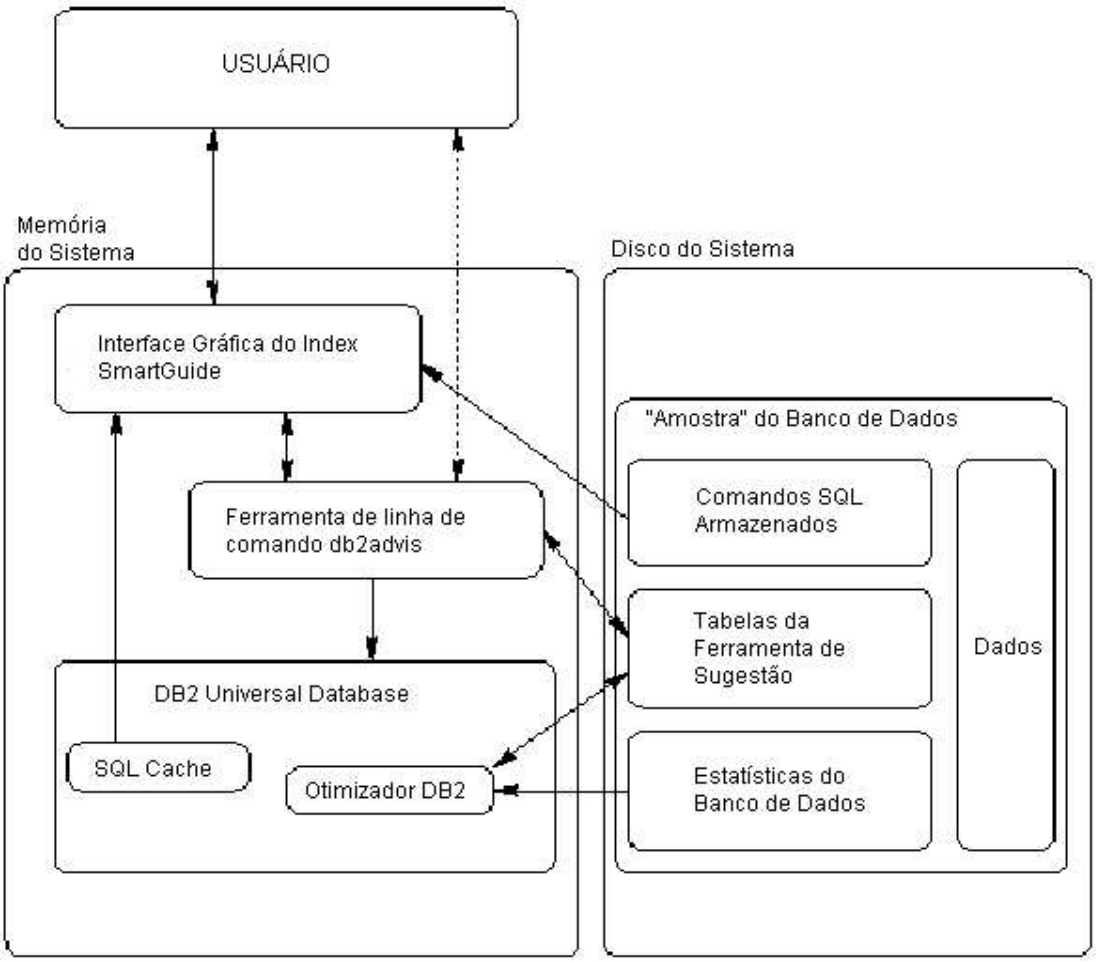

Figura 7: Arquitetura da Ferramenta de Seleção de Índices do DB2 [LVZ+00]

Para que o processo seja eficiente, o importante é limitar de forma inteligente a enumeração de índices hipotéticos realizada antes da otimização. É proposta uma heurística que leva em conta os principais usos de acessos indexados aos dados e suas combinações para criar índices hipotéticos com múltiplas colunas por construção. Como a seleção de índices para uma dada consulta é então implementada pelo próprio otimizador, precisamos de apenas uma chamada ao otimizador para determinar os índices que melhor se adequam à consulta sendo analisada.

Após termos em mãos os índices candidatos para cada consulta e os seus benefícios estimados, somente resta o problema de achar os melhores índices para uma dada carga de trabalho (conjunto de comandos SQL). Este problema é encarado como uma variante do Problema da Mochila e é proposta uma heurística gulosa para a sua resolução.

O Oracle também possui uma ferramenta para recomendação de índices, que faz parte do Oracle Enterprise Manager 9i Tuning Pack [Orab]. A carga de trabalho submetida para a ferramenta pode ser especificada manualmente, obtida a partir das consultas mais pesadas presentes no cache atual de SQL do banco de dados ou monitorada durante um período de observação fixo envolvendo algumas tabelas dadas. Não foi possível encontrar uma publicação detalhando a arquitetura ou a estratégia de busca empregadas por esta ferramenta para calcular quais índices devem ser criados ou removidos.

Auto-sintonia de Índices

Todas as propostas discutidas acima têm como uma de suas principais dificuldades a definição de uma carga de trabalho relevante (ou, de forma equivalente, a definição de um período fixo de observação do sistema). Desta forma, a qualidade do projeto físico obtido depende fundamentalmente da carga de trabalho submetida pelo DBA para a ferramenta. Isto se distancia do ideal de um sistema que selecione os seus próprios índices automaticamente.

É importante notar, ainda, que uma dificuldade presente em todas as propostas que envolvem produtos comerciais é a impossibilidade de analisarmos o código produzido para a integração da estratégia de busca com o SGBD.

A proposta de [CL02] procura atacar estas deficiências. Os autores investigam como o uso da tecnologia de agentes de *software* poderia ser útil para a construção de SGBDs que são capazes de auto-sintonia e de auto-administração. É analisada em detalhes a aplicação deste tipo de tecnologia ao problema de auto-sintonia de índices. É proposto um agente de software integrado ao código do SGBD e construído em camadas. Este agente coleta informações sobre as consultas submetidas ao SGBD e, com base em uma heurística, decide quando deve criar ou destruir índices em benefício do sistema. Podemos ver a arquitetura usada para a integração entre o agente e o SGBD na Figura 8.

A heurística proposta registra os ganhos ou perdas acumulados de cada índice candidato para as consultas submetidas iterativamente ao sistema. Quando o ganho acumulado supera o custo de criação do índice, o índice é criado. De forma análoga, quando a perda acumulada supera o custo de destruição do índice, o índice é destruído. Neste tipo de heurística, não é necessário definir um período de observação para o SGBD, uma vez que a decisão de reorganização não é disparada por um limite de tempo. Isto traz o benefício de eliminar da agenda do DBA a necessidade de estabelecer um limite fixo de tempo pouco intuitivo quando comparado às aplicações que rodam sobre o sistema.

Um ponto forte da proposta é o fato de que o agente é integrado com o SGBD de forma embutida. Isto permite que o método de coleta das consultas SQL submetidas seja implementado de forma mais eficiente do que os mecanismos tradicionais de rastreamento (*tracing*) do sistema. Além disto, também é importante ressaltar que o agente realiza a criação e destruição de índices de forma totalmente automatizada, sem intervenção de um administrador. A proposta de [CL02] está sendo implementada para que seja possível avaliar a qualidade dos índices gerados e a penalidade sobre o desempenho do sistema. Trabalhos iniciais que mostram o desenrolar desta implementação são [Nor03, Sal03].

Escolha de Visões Materializadas

Os atuais sistemas comerciais oferecem uma série de facilidades para a criação de armazéns de dados e a implementação de aplicações OLAP sobre estes armazéns. Uma das facilidades

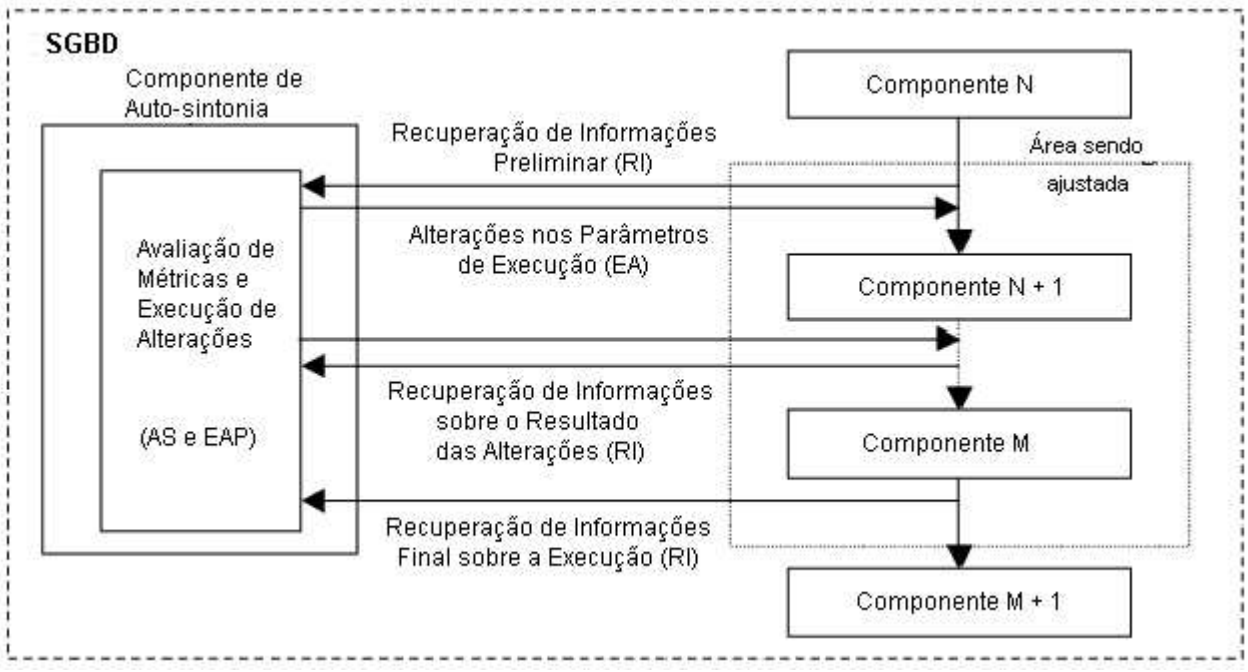

Figura 8: Arquitetura de Agente Embutido para Auto-sintonia de Índices de [CL02]

disponíveis nos servidores é a possibilidade de materializar visões. Uma visão materializada pré-computa uma determinada visão dos dados e, assim, pode ser usada para acelerar o resultado de consultas, principalmente quando são empregadas técnicas de reescrita. À medida que os dados são atualizados, entretanto, é necessário realizar manutenções nas visões materializadas para que estas se mantenham consistentes.

Em [GHRU97, GM99, Gup97, HRU96], os autores investigam algoritmos para automatizar a escolha de visões materializadas que dão suporte a aplicações OLAP. Há uma suposição de que a carga de trabalho a ser processada já é conhecida e a pesquisa enfoca a criação de modelos de custos e algoritmos para permitir uma busca pelo espaço de possíveis visões materializadas. Os algoritmos recebem como entrada as visões que podem ser definidas sobre um cubo de dados e apresentam como saída quais destas visões devem ser materializadas.

Uma faceta interessante de [GHRU97] é a de que, à medida que as visões materializadas são escolhidas, são também considerados na busca índices sobre estas visões. O uso de índices sobre as visões construídas pode acelerar significativamente o processamento de consultas da carga de trabalho. Em outros trabalhos, como [BPT97], [SDN98], [TS97], [ZY99] e [YKL97], os algoritmos propostos enfocam apenas a seleção das visões materializadas mais adequadas em um tempo de computação aceitável, sem considerar a possibilidade de criar índices sobre as visões consideradas. A adequação de cada visão recomendada pode ser julgada em termos da aceleração que é esperada para a carga de trabalho ou da redução nos custos de atualização do banco de dados.

Em [LQA97], é abordado o problema de materializar visões e índices que permitam que o tempo gasto no processo de carga de um armazém de dados seja minimizado. Dados um conjunto de relações base e uma visão materializada primária já existentes no armazém, é mostrado um algoritmo baseado no método A* para obter visões materializadas secundárias e índices que permitam acelerar a atualização das estruturas primárias. O algoritmo apresentado computa a solução ótima para o problema através de uma busca pelo espaço de soluções que possui complexidade exponencial. Por conta da limitação de desempenho do método de solução automático, os autores propõem um conjunto de heurísticas que poderiam ser empregadas pelo administrador

do sistema para selecionar visões materializadas e índices de forma a encontrar uma solução de boa qualidade. Outro trabalho numa linha similar de atuação é [Rou82], que propõe a seleção de índices para acelerar o processamento de consultas envolvendo visões. No trabalho de [TB00], procura-se obter uma formulação geral do problema de seleção de visões materializadas em que possam ser considerados tanto os objetivos de acelerar o processamento da carga de trabalho quanto de minimizar o custo de carga ou atualização das visões materializadas no sistema.

Em nenhum dos trabalhos discutidos até este ponto há um estudo sobre como aconteceria a integração dos algoritmos propostos a um SGBD real. O trabalho de [ACN00] aborda a seleção conjunta de visões materializadas, índices e índices sobre visões materializadas para cargas de trabalho SQL em geral. A pesquisa foi desenvolvida dentro do contexto do projeto AutoAdmin da Microsoft (veja a seção 3) e seu resultado foi incorporado na ferramenta *Index Tuning Wizard* do Microsoft SQL Server 2000. Os autores propõem uma arquitetura em que a seleção de índices e visões materializadas é realizada em dois passos: a escolha de candidatos e a enumeração de configurações. Inicialmente, um conjunto de visões materializadas candidatas para a carga de trabalho SQL considerada é escolhida. A escolha das visões candidatas utiliza diversos algoritmos para selecionar candidatos que sejam sintaticamente relevantes para a carga de trabalho e para reduzir o conjunto de candidatos a aqueles que têm o potencial de trazer maiores benefícios para pelo menos um dos comandos presentes na carga de trabalho. A idéia é permitir que apenas candidatos relevantes entrem na fase de enumeração de configurações.

Nesta segunda fase do processo, é utilizado um algoritmo que realiza combinações das estruturas candidatas provenientes da primeira fase para gerar configurações físicas possíveis. Estas configurações físicas são custeadas pelo otimizador do sistema e a configuração encontrada que apresenta menor custo é retornada pela ferramenta. É importante frisar que o uso do otimizador para custear configurações é um diferencial deste trabalho. Outro ponto fundamental é o de que, na montagem das configurações físicas, índices e visões materializadas são considerados ao mesmo tempo na enumeração, o que permite alcançar soluções de maior qualidade para o projeto físico recomendado.

Particionamento de Relações em Bancos de Dados Paralelos

Outra questão estudada na área de projeto físico foi a seleção de atributos para particionamento de relações em bancos de dados paralelos. Em [Zil98], são propostos algoritmos para realizar o particionamento de forma estática (alocação inicial) e dinâmica (permitindo reorganização de acordo com a carga de trabalho). Os algoritmos de seleção não interagem diretamente com o otimizador de consultas, entretanto, e o trabalho utiliza uma métrica de ganho líquido própria para avaliar as decisões de particionamento.

O fato de não usarmos o otimizador de consultas para avaliar as decisões de reorganização física gera a necessidade de criar um modelo de custos alternativo. Isto não é desejável, tanto pela necessidade de evolução deste novo modelo quanto pelo volume de trabalho que já foi investido para refinar os modelos de custos dos otimizadores de consultas modernos.

No trabalho de [RZML02], toda a avaliação dos benefícios de um determinado particionamento é feita através da interação do módulo de seleção com o otimizador de consultas. O trabalho foi implementado com o banco de dados DB2 UDB, que oferece paralelismo para uma máquina de memória distribuída (*shared-nothing*).

O otimizador é estendido com dois novos modos de operação para a seleção de partições: recomendação e avaliação. No modo de recomendação, dada uma consulta, o otimizador enumera um conjunto de partições candidatas para as tabelas base e escreve em uma tabela auxiliar as partições de cada tabela base que trazem o maior benefício para a consulta. No modo de avaliação, podemos estabelecer um conjunto de partições hipotéticas para as tabelas base de uma consulta e avaliar o custo do plano resultante escolhido pelo otimizador supondo o uso destas partições.

O trabalho propõe uma ferramenta que utiliza o otimizador em modo de recomendação para sugerir um conjunto de partições candidatas para uma carga de trabalho. A partir destas partições candidatas, estratégias heurísticas de busca das melhores combinações de partições para a carga de trabalho são empregadas. Estas estratégias utilizam o otimizador em modo de avaliação para determinar o custo de cada combinação de partições para as tabelas base. Ao final, a ferramenta sugere um particionamento para as tabelas base da carga de trabalho.

5.2 Alocação de Dados

Em um sistema em que múltiplos elementos de armazenamento estão disponíveis, precisamos decidir como distribuir os arquivos ou relações pelos elementos. O problema de alocação de dados já é considerado na literatura há alguns anos. Em [CABK88], estuda-se a alocação estática de fragmentos de uma relação em uma máquina de memória distribuída (*shared-nothing*). Por alocação estática, entendemos que não são levados em conta a evolução da carga de trabalho e a adaptação automática do sistema. Desta forma, esta proposta não pode ser vista como uma proposta de auto-sintonia em si, mas ela traz alguns conceitos importantes para a compreensão de propostas mais recentes.

Em especial, são importantes as definições de *calor* e de *temperatura* de um objeto (fragmento, bloco, registro, relação, etc). O calor de um objeto é a freqüência de acesso deste objeto por unidade de tempo. Ele representa o peso que este objeto tem no balanceamento da carga do sistema. Já a sua temperatura é a razão entre o seu calor e seu tamanho. Ela mede a relação custo-benefício de mover este objeto para outro elemento de armazenamento.

A alocação dos dados é feita em dois grandes passos: particionamento (*declustering* ou *partitioning*) e atribuição (*relation assignment* ou *allocation*). Durante o particionamento, decidimos em quantos e quais fragmentos uma relação será quebrada; na atribuição, decidimos em que elemento de armazenamento posicionar cada fragmento. A proposta de [CABK88] utiliza heurísticas para resolver os dois subproblemas.

As propostas de auto-sintonia para alocação de dados estudadas se diferenciam primordialmente pela tecnologia do sistema paralelo considerado. Abaixo, veremos propostas que endereçam *arrays* de discos, máquinas de memória distribuída (*shared-nothing*) e redes de estações de trabalho (*NOW, network of workstations*).

Alocação de dados em *Arrays* de Discos

Em [SWZ93, SWZ98], são consideradas estratégias para a migração de dados e balanceamento de carga em *arrays* de discos. O trabalho leva em consideração os problemas de particionamento e atribuição em separado. O particionamento, devido ao foco em *arrays* de discos, é realizado no nível físico, através de técnicas de *striping*. Este tipo de método divide um arquivo em segmentos de unidades de dados de tamanho fixo que são atribuídos aos discos em *round-robin*. A unidade de *striping* é o número de bytes ou blocos consecutivos armazenados em um dado disco.

Determinar qual deve ser a unidade de *striping* para o sistema (ou para arquivos específicos) é o principal problema de ajuste no momento do particionamento dos dados. A escolha da unidade de *striping* tem um efeito importante sobre a migração de dados para balanceamento de carga. Unidades menores permitem que a migração alcance um balanceamento mais fino, mais preciso. Entretanto, unidades pequenas demais podem limitar muito a vazão do sistema, uma vez que servir uma única requisição pode implicar em esperas em muitos discos para processamento quando a carga sobre o sistema de discos aumenta.

Em [SWZ98], são propostos modelos formais para a escolha de uma unidade de *striping* global para os arquivos do sistema e também para a escolha de unidades de *striping* independentes para cada arquivo, se necessário. Resultados experimentais indicam que a escolha de uma unidade de

striping global permite uma computação muito mais rápida e pouca perda de desempenho no processamento das cargas de trabalho submetidas ao sistema.

No problema de atribuição, o trabalho traz interessantes contribuições de auto-sintonia. É proposta uma heurística, denominada de "resfriamento de discos" (*disk cooling*), que permite a migração de dados para ajustar o desbalanceamento de carga causado por tendências ou mudanças nos padrões de acesso. Os dados são migrados dos discos "quentes" para os discos "frios" do sistema, conforme mostrado na Figura 9.

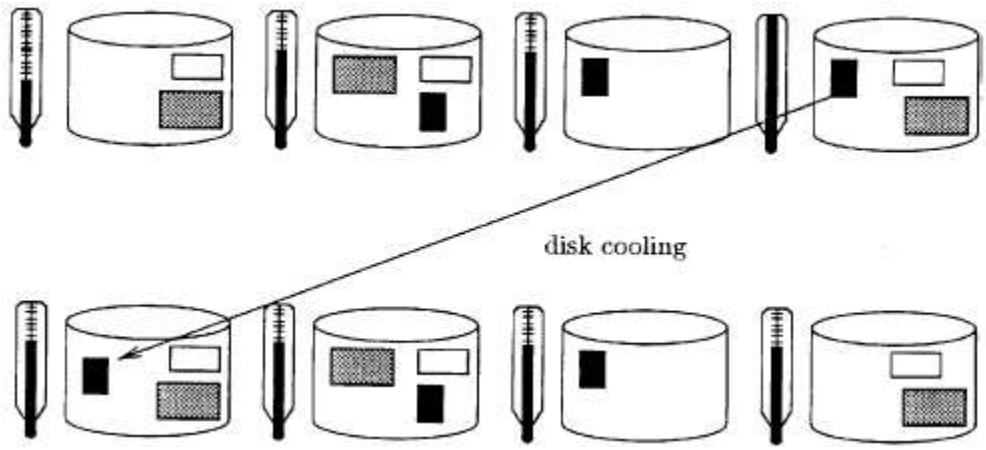

Figura 9: Resfriamento de discos [SWZ98]

A heurística de resfriamento é invocada em intervalos fixos de tempo e somente realiza realocações de dados caso as estatísticas coletadas sobre o sistema indiquem desbalanceamento. As realocações também podem não ser executadas caso a carga sobre o disco mais quente já seja excessiva. Numa situação como esta, iniciar a migração de dados colocaria uma carga adicional sobre os discos que poderia degradar ainda mais o desempenho do sistema. Assim, conforme também verificado experimentalmente pelos autores, a heurística possui um comportamento próativo, realizando as migrações em momentos em que o sistema possui carga ainda administrável e não tomando ações que poderiam piorar as situações de carga mais agressiva.

Em [ACDN03], é observado que a estratégia de espalhar uniformemente todos os objetos do banco de dados entre os discos disponíveis (*full striping*) não oferece o melhor desempenho para o processamento de cargas de trabalho que envolvam uma quantidade significativa de co-acesso entre objetos. Um exemplo seria a execução de um *merge join* entre duas tabelas grandes. Para processar a junção, é mais eficiente que cada uma das tabela resida em um conjunto de discos distinto.

Assim, o trabalho modela o problema de distribuir os objetos entre os discos existentes como um problema de otimização. O objetivo é encontrar um equilíbrio entre paralelismo de I/O e acessos randômicos de forma a maximizar o desempenho de processamento de uma carga de trabalho dada. O trabalho também considera na modelagem do problema características de disponibilidade tipicamente encontradas em configurações de *arrays* de disco (RAID 0, 1 e 5) e características de gerenciamento, como a necessidade de posicionar dois objetos relacionados nos mesmos discos.

É proposta, então, uma arquitetura para a ferramenta de escolha de alocação dos objetos nos discos. Nesta arquitetura, se destacam os módulos de pré-processamento da carga de trabalho, busca de soluções e modelo de custos. O pré-processamento da carga de trabalho envolve interação com o otimizador de consultas para determinar os planos de execução de cada comando. A partir dos planos, são determinados os objetos co-acessados pela carga de trabalho.

O módulo de modelo de custo permite estimar a quantidade de I/O e o tempo de resposta

que será observado para uma determinada consulta quando executada sobre uma alocação de objetos dada. Foi necessário incluir um modelo de custos independente do otimizador para este tipo de estimativa uma vez que os otimizadores hoje comumente disponíveis são insensíveis a alocação física dos objetos entre os discos. Já o módulo de busca de soluções usa uma heurística gulosa em dois passos para encontrar uma alocação de objetos válida.

Com o uso dos módulos descritos, é possível encontrar alocações de objetos válidas e satisfatórias sem necessariamente materializar a alocação ou executar os comandos da carga de trabalho. É importante observar que, ainda assim, uma limitação do trabalho é a de que a carga de trabalho de entrada deve ser fornecida por um DBA. O sistema não é capaz de automaticamente determinar uma carga de trabalho relevante.

Alocação de dados em uma Máquina de Memória Distribuída

A alocação dinâmica de dados em uma máquina de memória distribuída (*shared-nothing*) é analisada em [LKO+00]. Repare que, em relação ao caso de *arrays* de discos, temos a complexidade maior de que o sistema possui elementos de processamento totalmente independentes. O método proposto é baseado em uma estrutura de indexação de altura globalmente balanceada e que realiza o registro da carga submetida a uma dada relação em vários níveis de granularidade. O trabalho se diferencia de propostas anteriores para estruturas de dados distribuídas e migração de fragmentos, como [LNS93] e [HLY93], por propor uma estrutura que consegue se adaptar automaticamente a modificações nos padrões de acesso aos dados de forma a obter um melhor balanceamento de carga.

A mecanismo de indexação é composto por uma estrutura em dois níveis. O primeiro nível direciona a busca para o nó em que os dados estão armazenados. O segundo nível é uma coleção de árvores B+, uma em cada nó, que indexam os dados nos nós de forma independente. O mecanismo é denominado árvore aB+ (*aB+-tree*, de *adaptive B+-tree*), pois mantém a estrutura com uma altura balanceada globalmente de forma adaptativa. São propostos algoritmos para inserção, remoção e busca neste novo índice.

A migração de dados ocorre através do uso inteligente da estrutura descrita acima. A migração se inicia quando um nó central detecta que as quantidades de acessos a um determinado nó começa a causar um desbalanceamento do sistema. A quantidade de dados a ser migrada deste nó para um menos utilizado é obtida através de uma estratégia *top-down*. Na raiz, determina-se o número de subárvores a serem migradas. Se o número de acessos de uma subárvore é muito grande, podemos avaliar recursivamente as subárvores do nível inferior. O processo é repetido até determinar-se uma quantidade adequada de dados a serem migrados. Para diminuir a sobrecarga sobre o sistema, são mantidas estatísticas apenas sobre os números de acessos a cada nó e os números de acessos a níveis inferiores na estrutura do nó são estimados supondo uma distribuição uniforme dos acessos.

Como a migração é feita através de movimentações de subárvores inteiras, a estratégia procura adicionar uma nova subárvore separada no nó de destino. Isto faz com que a migração ocorra de forma muito mais rápida do que a estratégia convencional de inserir os registros no nó de destino um de cada vez. Este tipo de migração "em massa", entretanto, somente pode ser aplicada ao índice primário para acesso da relação.

Alocação de dados em uma Rede de Estações de Trabalho

A proposta de [VBW98] considera a migração de dados para balanceamento de carga em uma rede de estações de trabalho (*NOW*). A proposta se limita a considerar acessos baseados em chaves únicas aos dados. Assim, os autores acreditam que sua abordagem é bastante adequada para servidores de documentos HTML, por exemplo.

Novamente, como na proposta de [LKO+00], existe um registro da carga submetida ao sistema em diferentes níveis de granularidades para permitir que migrações dinâmicas de dados sejam realizadas quando registros frios se tornam quentes e vice-versa. As estatísticas coletadas em cada nó consideram o efeito da carga tanto sobre os discos quanto sobre os elementos de processamento.

Uma consideração especial desta proposta é a possibilidade de adicionar ou remover nós (servidores) do sistema à medida que a carga é dinamicamente rebalanceada. O objetivo é atender ao requisito de escalabilidade de custo/desempenho [DG92]: quando o volume total da carga é multiplicado por um fator N, então multiplicar o tamanho da rede de estações pelo fator N melhora a vazão pelo fator N enquanto mantém o tempo de resposta aproximadamente constante.

A decisão de adição ou remoção de servidores se baseia no acompanhamento de uma métrica de carga global do sistema. A carga global considera a média das cargas locais de cada nó de processamento usado. O administrador estabelece limites que indicam a carga global desejada no sistema e a carga máxima que pode ser submetida a um nó individualmente. A partir destes parâmetros, o sistema decide quando deve incluir um novo servidor, remover um servidor ou simplesmente realizar migrações de dados que permitam um melhor balanceamento entre os servidores já alocados.

A migração entre os servidores se dá em fragmentos. O tamanho dos fragmentos utilizados é adaptável e selecionado pelo sistema automaticamente de acordo com a precisão necessária para realizar o balanceamento de carga. Mais do que isto, o sistema também toma o cuidado de realizar operações de migração com prioridade mais baixa do que as operações convencionais, de forma a evitar que a vazão do sistema seja prejudicada. A única exceção a esta regra ocorre quando as requisições convencionais já não conseguem ser processadas pelo nó de origem. Neste caso, a migração ocorre com prioridade máxima para permitir que o nó volte a novamente processar requisições.

5.3 Controle de Carga (MPL)

Nesta seção tratamos de um problema que ocorre em sistemas que processam transações utilizando como controle de concorrência o método 2PL (*two-phase locking*). Este é o método comumente implementado na maior parte dos sistemas comerciais. Uma descrição do 2PL pode ser encontrada em [GUW00].

Neste tipo de sistema, as transações obtêm bloqueios (*locks*) sobre os itens de dados para evitar que estes itens sejam concorrentemente acessados por outras transações. Quando uma transação tenta obter acesso a um item que já foi bloqueado em modo não-compatível por outra, esta transação entra em um estado de espera. Esta situação é denominada de conflito de bloqueio (*lock conflict*). Em sistemas que utilizam o 2PL, a principal causa de degradação do desempenho é referente aos tempos de espera das transações que sofrem conflitos de bloqueio [Tho98].

Como múltiplas classes de transações obtêm bloqueios que podem causar conflitos, existe um limite para o número de transações simultâneas que um sistema pode executar. Isto ocorre pois quanto mais conflitos existem, maior é o tempo de espera de transações e menor é o tempo de processamento efetivo. A partir de um certo ponto, o tempo de espera passa a dominar o tempo de processamento e vazão do sistema cai de forma drástica. Se o sistema continua aceitando que novas transações sejam executadas de forma concorrente, este comportamento pode se agravar até um ponto em que o sistema colapsa (ou precisa ser reiniciado). Este problema é denominado de *thrashing* de contenção de dados. Repare que a tendência deste problema ocorrer é maior nos momentos de pico de uso do sistema, o que o torna ainda mais sério.

Um forma de atacar o problema de *thrashing* de contenção de dados é através de uma efetiva limitação do nível de multiprogramação do sistema. Alguns métodos foram propostos na literatura para realizar esta limitação. Uma discussão com referências pode ser encontrada em [Tho98].

Nossa atenção, entretanto, deve se limitar às técnicas que apresentam características de auto-sintonia. Neste contexto, enfocamos técnicas que procuram eliminar os parâmetros de ajuste do sistema. Nos sistemas comercialmente disponíveis, é comum que o DBA precise especificar justamente qual será o MPL do sistema. Este parâmetro é de difícil determinação e ajuste.

O trabalho de [WHMZ94] apresenta uma técnica de auto-sintonia para o problema de *thrashing* de contenção de dados. O método é denominado de controle de carga orientado a conflitos. Foi aplicado o conceito de *ciclo de controle de realimentação* no projeto deste método, que segue, portanto, três fases: observação, predição e reação.

Na fase de observação, o sistema monitora uma métrica, denominada razão de conflitos (*conflict ratio*), que mede a razão entre a quantidade de bloqueios total existente e a quantidade de bloqueios pertencentes a transações ativas. O sistema começa a exibir um comportamento de *thrashing* de contenção de dados quando a razão de conflitos alcança um valor limite fixo e estável [4], que não precisa ser dado como entrada para o sistema por um DBA. A obtenção deste valor foi feita tanto de forma experimental como também através da resolução de um modelo analítico sobre o comportamento do 2PL [WHMZ94, Tho98].

Assim, se o sistema monitora a razão de conflitos, a etapa de predição é trivial. Caso ela alcance o seu valor crítico, existem grandes chances de um comportamento de *thrashing* ser iniciado e, assim, ações devem ser tomadas. Estas ações compõem a etapa de reação e podem ser de dois tipos: controle de admissão e controle de cancelamento.

O controle de admissão determina se uma nova transação deve ser processada ou se deve esperar em uma fila na entrada do sistema. Já o controle de cancelamento atua abortando transações que presentemente estão no sistema. Assim, estas técnicas combinadas podem causar reduções ou controlar o MPL do sistema. De fato, resultados experimentais indicam que o método como um todo é efetivo.

Outro trabalho que propõe técnicas para o ajuste do MPL do sistema é [BMCL94]. Há um ajuste do MPL por classe de transação submetida ao sistema. Cada classe de transação possui objetivos de desempenho especificados através de seu tempo de resposta desejado. O trabalho realiza um ajuste simultâneo do MPL e dos buffers de memória do sistema. Desta forma, iremos apresentá-lo em maior detalhe na seção 5.5.

Ainda na linha de ajuste do MPL por classe de transação, podemos citar o trabalho de [Rah97]. É criado um módulo, denominado de *controlador local de desempenho*, que é embutido no SGBD e interage com os componentes de gerência de transações, gerência de buffers e controle de concorrência. O controlador local de desempenho é um processo invocado periodicamente que obtém métricas de desempenho provenientes do SGBD. A partir destes dados, determina se existem gargalos de recursos que impedem que os objetivos de desempenho das diversas classes de transações sejam alcançados. Caso algum gargalo tenha sido detectado, o controlador procura resolver o problema através de modificações incrementais nos valores do MPL por classe de transação. Apesar de o controlador ter certamente sido elaborado a partir da idéia de um *ciclo de controle de realimentação*, o trabalho não discute qual seria uma arquitetura de software interessante para a sua construção.

5.4 Substituição de Páginas em Memória

Substituir páginas de memória de forma inteligente é fundamental para o desempenho de sistemas de bancos de dados. Várias políticas consagradas são conhecidas para este problema, sendo que a maior parte dos sistemas comercialmente disponíveis implementam a política LRU (*Least Recently Used*). A principal característica de propostas de substituição de páginas que

[4] Em [Tho98], é desenvolvido um modelo analítico que indica que o valor limite para a razão de conflitos está na faixa de 1.25 a 1.55. Foi confirmado experimentalmente que de fato o comportamento de *thrashing* de contenção de dados começa a ocorrer quando a razão de conflitos alcança um valor em torno de 1.3 [WHMZ94]. Este valor limite é denominado de razão de conflitos crítica.

podem ser vistas como técnicas de auto-sintonia é a sua capacidade de se adaptar a *mudanças no conjunto de páginas populares* do banco de dados. A clássica estratégia LRU manda para disco a página que foi referenciada menos recentemente quando há a necessidade de uma substituição, mas, em algumas situações, incorre em sobrecargas para trazer de volta à memória uma página recém-enviada para disco ou para manter em memória páginas que não serão referenciadas novamente tão cedo.

Mostramos um exemplo desta situação a seguir. Suponha que tenhamos uma aplicação de bancos de dados com múltiplas transações com boa localidade nas referências a páginas compartilhadas, de forma que 5000 páginas em buffer de um total de 1 milhão de páginas de disco recebem 95% das referências das transações concorrentes. Se um processo *batch* pouco freqüente inicia uma varredura seqüencial por todas as páginas do banco de dados, esta varredura irá substituir as páginas comumente referenciadas presentes no buffer por páginas com pequenas chances de serem referenciadas novamente. A tendência nesta situação é que o tempo de resposta se degrade consideravelmente para as transações concorrentes, uma vez que haverá um aumento de I/O (com conseqüente formação de filas de espera) por páginas que costumam ficar residentes no buffer.

Algumas estratégias de substituição de páginas foram propostas para endereçar este tipo de problema com o algoritmo LRU. Uma delas é a possibilidade, existente em alguns sistemas comerciais, de utilizar múltiplos buffers, com objetos especificamente alocados (veja a seção 5.5). Esta estratégia, bem como outras que foram propostas, exige uma quantidade razoável de ajustes e especificações de parâmetros sobre a carga de trabalho a ser processada pelo SGBD. Estes ajustes precisam ser feitos manualmente por um DBA. Assim, não podemos considerar este tipo de método propriamente como possuindo características de auto-sintonia.

Métodos baseados em k-distância

Uma proposta de substituição de páginas que apresenta verdadeiras características de auto-sintonia é a da estratégia LRU-k, descrita em [OOW93, WHMZ94]. A política LRU-k se ajusta automaticamente à evolução da popularidade das páginas do banco de dados e não requer a realização de configurações de parâmetros manualmente.

A política LRU-k procura estimar o tempo entre chegadas de referências às páginas do buffer utilizando uma abordagem estatística. Para cada página, são registrados os tempos das últimas k referências realizadas. Com base nestes tempos é calculada uma medida para cada página p, denominada a k-distância para trás, que é a distância para a k-ésima mais recente referência a p.

O algoritmo consiste em, no momento de substituir uma página, escolher aquela cuja k-distância para trás é máxima no buffer. As condições para otimalidade desta heurística são discutidas em [OOW93] para um modelo em que é assumida independência entre as referências a páginas e conhecimento de apenas as últimas k referências a cada página no buffer.

No caso de $k = 2$, temos a política denominada LRU-2. Para $k = 1$, temos LRU-1, que é exatamente a estratégia LRU clássica. Resultados experimentais indicam que LRU-2 realmente melhora as decisões de substituição em relação a política LRU clássica. Esta melhora é causada por uma melhor capacidade em estimar o tempo entre chegadas de referências a cada página do buffer. Mais do que isto, os resultados indicam que a maior parte dos benefícios da detecção automática de mudanças nos padrões de acesso da carga de trabalho já são obtidos com a estratégia LRU-2. Para $k > 2$, maior precisão é alcançada, mas também com maior custo de processamento.

O custo de implementação da estratégia LRU-2 é superior ao da estratégia LRU clássica, uma vez que é necessário armazenar dois valores de últimas referências por página. Entretanto, seu desempenho na determinação das páginas populares do banco de dados, mesmo na presença de mudanças na carga de trabalho submetida, compensa o seu custo adicional de processamento

[OOW93].

Um algoritmo com custo de processamento inferior ao de LRU-2 é proposto em [JS94]. Neste artigo, é discutida a estratégia 2Q. O método 2Q, a exemplo de LRU-2, também consegue se adaptar a mudanças na popularidade das páginas do banco de dados e não requer ajuste de parâmetros de execução por um DBA especializado.

O algoritmo LRU-2 procura remover páginas frias do buffer quando uma substituição é necessária. A essência da estratégia 2Q é somente admitir páginas quentes ao buffer principal. Da mesma forma que LRU-2, 2Q testa as páginas com base no tempo de sua segunda última referência. Entretanto, na estratégia 2Q, o buffer é dividido em duas grandes regiões: A1 e Am. A região Am é uma fila convencional em que são admitidas apenas as páginas consideradas quentes pelo algoritmo. A região A1 é dividida em duas filas, A1in e A1out, que mantém referências para páginas. Em A1in, são guardadas as páginas recém-entradas no buffer; em A1out, são guardadas as referências das páginas de A1in que foram removidas do buffer.

Com esta política, o algoritmo consegue identificar páginas que sofreram sua primeira referência e foram referenciadas novamente após um tempo suficientemente longo. Estas páginas são consideradas realmente quentes. Isto está em sintonia com a idéia de que uma página pode sofrer muitas referências seguidas em um breve intervalo de tempo por serem referências correlacionadas. Reparar que uma página com muitas referências correlacionadas não é realmente popular, uma vez que outras transações do banco de dados não estão necessariamente nela interessadas.

Métodos baseados em Padrões de Acesso

Alguns métodos para ajuste dinâmico da política de substituição de páginas empregada no SGBD foram propostos com base na noção de padrões de acesso [CD85]. Sistemas de bancos de dados relacionais dão suporte a um conjunto determinado de operações físicas para acesso e manipulação dos dados. Uma análise sistemática revela que o padrão de acesso de uma dada operação pode ser decomposto em um conjunto de padrões de acesso simples [CD85].

Entre estes padrões simples de acesso, podemos destacar:

- acesso seqüencial: executado, por exemplo, quando uma relação é varrida página a página.
- acesso repetitivo (*looping*): executado, por exemplo, em uma junção de *loops* aninhados entre duas relações.
- acesso randômico: executado, por exemplo, em um seleção usando um índice não-clusterizado [5].

Cada padrão de acesso exige um tratamento diferenciado por parte do gerente de buffer do sistema. Para um padrão seqüencial, podemos alocar uma única página no buffer para o processamento, uma vez que as páginas trazidas de disco não serão referenciadas novamente neste tipo de acesso. Já para um padrão repetitivo, é necessário um conjunto de páginas que contém o número de páginas referenciadas no *loop*. É interessante empregar uma política de substituição como MRU (*Most Recently Used*), uma vez que voltaremos a referenciar mais cedo as páginas menos recentemente usadas [LM03]. Enfim, para um padrão randômico, é interessante alocar múltiplas páginas de buffer e podemos empregar uma política de substituição clássica, como LRU.

O trabalho de [CR93] propõe o uso de *feedback* proveniente das consultas executadas pelo SGBD para melhorar a política de substituição de páginas. É usado um modelo analítico para estimar a curva de faltas de página (*page faults*) esperadas por tamanho alocado do buffer. A curva de faltas de página pode ser utilizada para detectar qual padrão de acesso está sendo

[5] Para uma definição dos conceitos de índice clusterizado e não-clusterizado, ver [SB03].

submetido ao gerente de buffer. O mecanismo de *feedback* registra para uma dada seqüência de referências a páginas quais são os parâmetros estimados para a sua curva de faltas de páginas. Com base nestas informações, quando a mesma seqüência de referências a páginas se repete, o método consegue prever que tipos de padrão de acesso serão realizados pela consulta e ajustar a política de substituição de páginas empregada e o número de buffers alocados.

Em [FNS95] são propostos métodos adaptáveis para a alocação de páginas em memória. Os autores observam que as políticas anteriormente propostas baseadas em padrões de acesso sempre procuram reservar uma quantidade fixa de páginas em memória para cada padrão de acesso empregado por uma consulta. O trabalho propõe um algoritmo em que o número de buffers reservados para cada padrão pode ser determinado dinamicamente. Esta determinação ocorre através de um modelo analítico de filas que prevê as necessidades futuras da carga de trabalho. As decisões de admissão de novas consultas e de alocação de buffers para cada consulta são tomadas objetivando maximizar a vazão do sistema ou minimizar a utilização efetiva do disco.

5.5 Ajuste de Buffers de Memória

Além das técnicas debatidas na seção 5.4, alguns outros métodos foram desenvolvidos para tornar a gerência de memória em servidores de dados mais auto-sintonizável. Nesta seção, discutimos trabalhos em duas linhas diferentes de pesquisa.

Configuração de Múltiplos Buffers

A primeira linha se foca em automatizar a alocação de objetos do banco de dados (tabelas e índices) a múltiplas áreas de buffer de forma a aumentar o desempenho do sistema. Alguns sistemas comerciais, como o IBM DB2, permitem que este tipo de configuração seja realizada. Na prática, entretanto, é muito difícil para o DBA separar os objetos entre várias áreas de buffer de forma a otimizar o desempenho, uma vez que existem muitas possibilidades de alocação.

Em [XMP02] é proposto um algoritmo para a alocação dos objetos do banco de dados a um conjunto de buffers dados. São usadas técnicas de *data mining* para agrupar os objetos com características semelhantes nos mesmos buffers. Entre as características consideradas, estão propriedades inerentes dos objetos (por exemplo, se são tabelas ou índices) e padrões de acesso da carga de trabalho que é submetida ao sistema. O objetivo é minimizar a quantidade de acessos a discos que a carga de trabalho deve realizar.

Para o algoritmo de [XMP02], é necessário que cada objeto seja associado a um vetor de características. Em seguida, são usadas técnicas de agrupamento (*clustering*) para selecionar quais grupos de objetos serão alocados em cada um dos buffers disponíveis. Por fim, são escolhidos tamanhos para cada um dos buffers. Os tamanhos se baseiam em uma estimativa de tamanhos usados pelos objetos quando a carga de trabalho é submetida ao sistema com um único buffer configurado.

A proposta foi implementada no IBM DB2 UDB e resultados experimentais foram gerados usando como carga de trabalho o *benchmark* TPC-C [TPC]. A alocação de objetos gerada pela ferramenta foi pelo menos tão boa quanto alocações feitas por DBAs especializados. O sistema, entretanto, não consegue adaptar a alocação realizada a mudanças na carga de trabalho de forma automática.

Gerência de Memória baseada em Classes de Transações

A segunda linha de pesquisa trabalha na gerência da memória de trabalho do sistema. Além dos buffers de memória que acomodam páginas de dados e índice lidas do banco de dados, existem áreas de memória do sistema que são reservadas para uso por *classes de transações* específicas [WKKS99]. Exemplos seriam transações que realizam atualizações com acesso baseado em chave

primária e transações somente de leitura que percorrem uma relação inteira para calcular uma agregação. Diferentes classes de transações podem possuir diferentes padrões de acesso ao buffer de páginas compartilhado do SGBD, bem como ter distintas necessidades de uso de memória de trabalho. Como exemplos de memória de trabalho, podemos citar áreas alocadas para *hash* e para ordenações.

Nos sistemas atuais existem parâmetros que realizam uma alocação estática destas áreas de trabalho. Esta alocação pode causar desperdícios de memória durante a execução de transações concorrentes. É possível que determinada classe de transações utilize a memória para si alocada apenas parcialmente enquanto outra classe precisaria de mais memória para ter um desempenho ótimo. Estas necessidades de memória variam dinamicamente à medida que as transações do sistema são executadas.

Como o sistema não tem como priorizar a importância de uma classe de transações sobre outra, as propostas que atacam este problema preconizam que um administrador especifique *objetivos de desempenho* por classe de transação. Com base nestes objetivos, o sistema realiza o monitoramento e a alocação de memória dinamicamente. Assim, podemos considerar este tipo de proposta como sendo de auto-sintonia, uma vez que o sistema passa a se adaptar automaticamente a objetivos de desempenho estabelecidos. Esta é uma aplicação do princípio de auto-sintonia de "substituição de decisões de sintonia por decisões de política de uso".

Em [BCL93a, BCL93b, BMCL94] temos um trabalho que realiza este tipo de ajuste de memória por classes de transações. Os objetivos de desempenho de cada classe de transações são especificados em termos de seus tempos médios de resposta esperados. O sistema procura encontrar uma solução para a alocação de recursos de cada classe de transação de forma independente, evitando, porém, soluções que inviabilizam soluções para outras classes. Este tipo de estratégia permite que calculemos uma solução viável mais rapidamente do que quando consideramos todas as classes de uma única vez. Por outro lado, não deixamos de analisar as possíveis interações entre as classes de transações que, na prática, competem pelos recursos compartilhados do sistema.

A proposta de [BMCL94] não apenas se restringe à gerência de memória e também realiza o ajuste do nível de multiprogramação do sistema de forma dinâmica. Propostas para controle de carga do sistema são discutidas na seção 5.3. O trabalho supõe, de forma distinta de outras propostas, que tanto o MPL quanto a alocação de memória podem ser especificados de forma independente para cada classe de transações submetida ao sistema.

A procura de alocações de recursos para o sistema é feita em um *ciclo de controle com realimentação*, de forma que há um monitoramento contínuo do tempo de resposta efetivamente alcançado para cada classe de transação e, portanto, do impacto das decisões tomadas pelo sistema. As decisões em si são geradas por um conjunto de heurísticas e estimativas simples do comportamento futuro do sistema. Como existe *feedback*, mesmo que a aplicação de determinada heurística falhe em produzir o resultado desejado, o sistema continuará a gradativamente se ajustar até alcançar uma solução em que os objetivos das classes sejam aproximados da forma mais fiel possível.

Uma extensão deste trabalho é proposta em [BCL96] com foco no problema de alocação de memória apenas do buffer compartilhado de páginas do SGBD para múltiplas classes de transações. É proposto um método mais especializado para a predição da quantidade de memória necessária para uma dada transação alcançar um taxa de acertos adequada no buffer (*hit rate*). Com base na taxa de acertos em buffer e na quantidade de memória alocada, o sistema pode calcular qual é o tempo de resposta esperado para uma transação. O trabalho ainda permite que múltiplas classes de transações compartilhem páginas no buffer, não realizando uma alocação estática de pedaços do buffer disponível entre as várias classes que o sistema processa. O trabalho de [CFW+95] também trabalha com a noção de reservar regiões do buffer para diferentes classes de transações. Entretanto, neste trabalho, não é permitido o compartilhamento de páginas do

buffer entre as classes tratadas.

Outro trabalho na linha de gerência de memória para múltiplas classes de transações é [SK99a]. Neste trabalho, é implementado um mecanismo de realimentação que dinamicamente altera os tamanhos das áreas de buffer dedicadas (e portanto o *hit rate*) das diversas classes de transações para satisfazer objetivos de tempo de resposta especificados pelo usuário do sistema. O grande diferencial do método ilustrado é o fato de que este foi desenvolvido para a gerência da memória distribuída de uma rede de estações de trabalho. Neste cenário, o sistema deve considerar que é possível fazer acesso a uma página de dados presente no buffer de um nó remoto. Tipicamente, o acesso a um *cache* remoto é mais eficiente do que o acesso a um disco local.

É realizada uma formulação do problema de ajuste dos tamanhos das áreas de buffer nos múltiplos nós como um problema de programação linear. A implementação do sistema prevê que cada nó possua um agente de coleta de informações por classe de transação e determina, ainda, que exista apenas um agente coordenador por classe de transação presente na rede de estações de trabalho. Cada agente coordenador continuamente recebe as informações dos agentes de coleta e realiza a solução do problema linear de ajuste dos tamanhos das áreas de buffer para a sua classe. Quando ajustes são necessários para respeitar os objetivos de tempo de resposta de alguma classe de transação, o agente coordenador desta classe envia comandos de ajuste para os agentes de coleta presentes em cada nó. Este processo é repetido em um *ciclo de controle de realimentação*, de tal forma que a alocação de memória do sistema como um todo caminha no sentido de respeitar os limites de tempo de resposta para todas as classes de transação.

Por fim, indicamos [WKKS99] para a discussão de algumas outras técnicas para problemas mais específicos de ajuste de buffers, como o uso seletivo de *prefetching* em conjunto com estratégias convencionais de substituição de páginas e o uso de *caches* distribuídos em sistemas paralelos.

5.6 Refino Automático de Estatísticas

Outra atividade que é comumente realizada por DBAs para a administração de SGBDs é a coleta periódica de estatísticas. Estatísticas são fundamentais para o correto funcionamento de otimizadores de consultas baseados em custos [GUW00]. Para que o sistema obtenha estatísticas sobre uma tabela, os métodos tradicionais indicam que ou a tabela seja percorrida para a análise dos dados ou que seja feita uma amostragem das linhas da tabela. A segunda técnica se adequa bem para tabelas maiores, com quantidade significativa de linhas.

Devemos ter como objetivo a idéia de tornar a coleta de estatísticas um processo que não mais exija (ou exija muito menos) a intervenção do DBA. Isto segue a idéia de tornar o sistema mais auto-sintonizável.

Manutenção Automática de Estatísticas

Em [GMP97], é proposto um método para a manutenção incremental de histogramas. Uma das percepções fundamentais deste trabalho é a de que os histogramas utilizados por um sistema somente precisam disponibilizar estimativas razoavelmente acuradas da distribuição dos dados. Assim, é criada uma amostra de apoio (*backing sample*) formada pela materialização de um conjunto de tuplas aleatoriamente escolhidas a partir da relação original. As estimativas de distribuição dos dados são derivadas a partir da amostra de apoio. Como a amostra de apoio é de tamanho controlável, as atualizações realizadas no banco de dados podem ser propagadas para uma manutenção coerente da amostra. Com isto, o resultado esperado é o de que a atividade administrativa de reconstruir os histogramas sobre as tabelas do banco de dados seja muito diminuída. Outro trabalho que se aproveita das atualizações realizadas no banco de dados para refinar histogramas é [DIR99]. Este trabalho, entretanto, faz o refinamento do histograma em memória, sem se utilizar de uma amostra de apoio.

Muitos outros trabalhos da literatura se focam em tornar a manutenção de estatísticas mais simples ou mais computacionalmente eficiente, usualmente através do uso de técnicas de amostragem. Não consideramos estes trabalhos, entretanto, como especialmente voltados para tornar o sistema auto-sintonizável, uma vez que seu objetivo não é o de eliminar, ou automatizar em grande medida, o trabalho de administração relacionado à manutenção de estatísticas atualizadas e de boa qualidade.

Alguns trabalhos, que seguem uma outra vertente, procuram se aproveitar da análise dos resultados das consultas submetidas ao próprio SGBD para refinar as estatísticas disponíveis. Este tipos de estratégia está em consonância com o princípio de auto-sintonia de criar um *ciclo de controle de realimentação*. Mais do que isto, estes trabalhos procuram automatizar a coleta de estatísticas que se adequam melhor às cargas de trabalho processadas pelo SGBD.

Em [CR94], é proposto um método que leva em consideração o resultado das consultas processadas pelo SGBD para adaptar as estimativas de seletividades realizadas pelo sistema. As seletividades são estimadas através de um polinômio que aproxima a distribuição real dos dados. À medida que o sistema processa consultas e atualizações, os coeficientes deste polinômio são adaptados para melhor refletir a distribuição de dados. O uso dos resultados de consultas e atualizações têm como conseqüência o fato de que a região mais utilizada da distribuição dos dados será a que terá a seletividade melhor aproximada. Com isto, o sistema adapta as suas estatísticas de forma a processar mais eficientemente as consultas de fato a ele submetidas. Outra conseqüência relevante é o fato de que a coleta das estatísticas se aproveita dos próprios comandos submetidos ao sistema, tornando, portanto, dispensável que seja realizado um procedimento operacional por parte do DBA para a coleta periódica.

Em [AC99], é proposta uma arquitetura em que o otimizador possui um *ciclo de controle de realimentação* para o refino incremental de histogramas. A cada consulta que é executada, são monitorados os resultados que esta consulta retorna. Estes resultados servem para aperfeiçoar histogramas criados sobre as colunas das tabelas base do banco de dados (veja a Figura 10). Desta forma, a construção de histogramas para melhor estimativa da distribuição de valores em colunas ou grupos de colunas (histograma multidimensional) pode ser realizada com custos muito baixos. O refino dos histogramas pode ser feito enquanto o sistema processa consultas (*on-line*) ou as informações da carga de trabalho podem ser armazenadas para serem processadas posteriormente em um momento "quieto" do banco de dados (*off-line*).

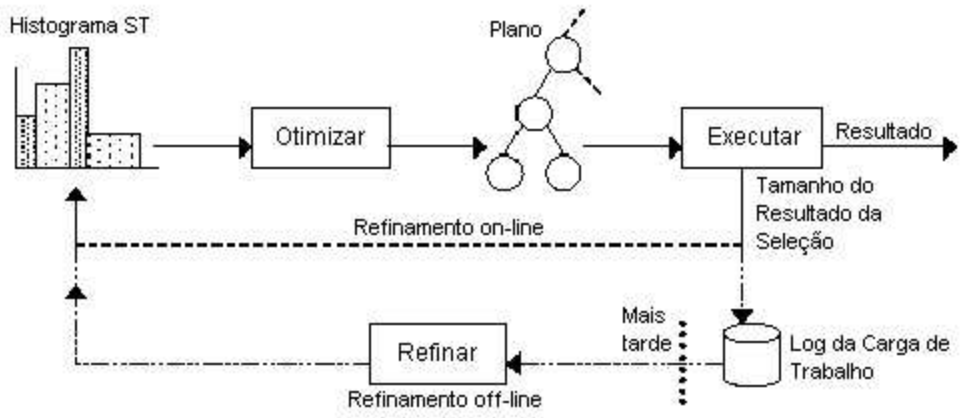

Figura 10: Arquitetura para refino de estatísticas [AC99]

O trabalho de [SLMK01] propõe que o otimizador deve aprender com as consultas que são executadas. Este aprendizado se dá justamente através do refino das estatísticas usadas pelo otimizador. Temos, assim, novamente o conceito de um *ciclo de controle de realimentação*,

em que o monitoramento da operação do sistema permite que ele seja ajustado para aumentar o desempenho. O trabalho segue uma arquitetura muito semelhante à de [AC99], mas traz inovações na amplitude em que o refino é realizado. A principal métrica que é refinada é a de cardinalidade. Tanto cardinalidades referentes a tabelas base como também a aplicações de operadores, como junções, são levadas em consideração. Desta forma, o sistema consegue confrontar as estimativas de cardinalidade realizadas para operações com o verdadeiro número de linhas retornado. Isto permite que o otimizador aprenda, portanto, com erros nas estimativas feitas anteriormente e melhore o seu modelo de custos sobre o banco de dados.

Seleção de Sinopses de Dados

Outra questão importante para a gerência de estatísticas é a escolha de sobre quais tabelas e colunas serão criadas *sinopses de dados* (*data synopses*). Em sistemas comerciais, precisamos decidir, por exemplo, sobre quais colunas devem ser criados histogramas. Este tipo de decisão é hoje inteiramente tomada pelo administrador do sistema.

Isto cria um *problema de projeto físico de sinopses de dados*. Uma solução para este problema é discutida em [KW02]. Dado um conjunto de relações e um conjunto de consultas do tipo seleção-projeção-junção (*SPJ queries*), o trabalho objetiva encontrar a melhor combinação de sinopses tal que o erro de estimativa cometido no processamento das consultas seja minimizado e que um determinado espaço físico para a criação de sinopses não seja excedido. São consideradas tanto sinopses que podem ser criadas sobre relações individuais quanto sinopses sobre junções. É mostrado um método que permite derivar sinopses de tal forma que as estimativas necessárias para cada consulta analisada sejam realizadas com apenas uma sinopse.

O trabalho de [CN00] realiza uma análise de quais estatísticas são essenciais para um determinado conjunto de consultas. O objetivo é conseguir derivar qual é o conjunto mínimo de estatísticas necessário para que o plano de execução encontrado para a consulta seja eficiente. Repare que este trabalho também se foca em determinar quais estatísticas devem ser coletadas em oposição a tornar a manutenção de estatísticas previamente escolhidas mais simples.

Os autores então propõem que a atividade de coleta de estatísticas por DBAs pode ser, em algumas situações, completamente automatizada fazendo-se com que o próprio sistema complemente, no momento da compilação de uma consulta, as estatísticas disponíveis com as estatísticas que são essenciais para a consulta dada. Com isto, o sistema conseguiria tanto decidir quais sinopses de dados são importantes quanto criá-las no momento adequado para o seu uso. A criação de estatísticas *on-line* pode representar um custo extra para o sistema e em ambientes com forte necessidade de processamento de transações estes custos podem não ser desprezíveis. Desta forma, é proposto que sejam disponibilizadas políticas, selecionáveis pelo administrador, para a aplicação da técnica de criação de estatísticas essencias de forma *on-line* ou *off-line*.

O trabalho de [JJOT01] enfoca o problema de criar um conjunto de histogramas que seja globalmente ótimo, no sentido de minimizar o erro no processamento de um conjunto de consultas e gastar um espaço físico fixo disponível no catálogo para armazenamento dos histogramas. Leva-se em consideração que distribuições de dados uniformes requerem menos *buckets* para serem representadas do que distribuições com grandes desvios (*skew*). Assim, são propostos três algoritmos para calcular como distribuir os *buckets* entre os histogramas presentes no catálogo de forma que o desvio nas estimativas geradas para o processamento da carga de trabalho submetida ao sistema seja o menor possível. Outro trabalho que também se aproveita dos resultados de consultas para determinar quais regiões de um histograma merecem mais *buckets* é [BCG01]. Neste trabalho, são propostas técnicas para aumentar a acuidade de histogramas multidimensionais.

Em [GLR00], vemos que um uso relevante de sinopses de dados é para fornecer respostas aproximadas a consultas. Em muitas aplicações, como por exemplo aplicações OLAP, a acurácia do resultado de determinada consulta não precisa ser total. Isto pode ser explorado para responder a consultas de forma mais rápida e mais interativa. O trabalho propõe que seja mantido uma

amostra não uniforme sobre os dados de uma determinada relação. Esta amostra é construída de forma que a probabilidade de uma tupla estar na amostra é proporcional à freqüência com que esta tupla é requisitada para responder às consultas submetidas ao sistema. Esta estratégia advém do fato de que, na prática, aplicações OLAP apresentam consultas com grande localidade no seu acesso a dados. Desta forma, algumas tuplas de uma determinada relação são mais significativas para obter respostas precisas do que as demais. O trabalho apresenta técnicas para que a amostra construída possua fundamentalmente tuplas que pertencem a este conjunto relevante e seja eficiente para a resposta a consultas de agregação sobre junções de chave estrangeira.

6 Sistemas comerciais

Várias empresas oferecem ferramentas de gerência de desempenho de bancos de dados. Este é o caso de fornecedores de SGBDs, como a Oracle [Oraa], Microsoft [Mic] e IBM [IBM], e de produtores independentes, entre os quais se distinguem Veritas [Ver], BMC [BMC], Teamquest [Tea], Embarcadero [Emb] e Quest [Que].

Nesta seção, analisamos a integração de técnicas de auto-sintonia em alguns dos produtos fornecidos pela indústria. Esta análise está baseada na informação disponível nos sites dos fornecedores e não em uma avaliação sistemática.

Devemos reforçar que nosso foco de estudo são ferramentas voltadas para desempenho de sistemas de bancos de dados. Existem outras ferramentas relevantes para a gerência de recursos de tecnologia de informação como um todo. Entre elas, podemos citar o IBM Tivoli [Tiv], o HP OpenView [OV] e o CA Unicenter TNG [CA]. Estas ferramentas procuram tornar toda a infra-estrutura de computação (*hardware* e *software*) de uma organização mais autônoma. Também existem componentes nestas plataformas voltados para a gerência e administração de sistemas de bancos de dados. Entretanto, o objetivo destas plataformas é a gerência integral de serviços de aplicação e não a análise detalhada de componentes específicos.

6.1 Fornecedores de Sistemas de Bancos de Dados

No artigo [EPBM03] se enumeram as características que um SGBD deveria ter para ser considerado autônomo a partir da análise dos três maiores fornecedores de sistemas de bancos de dados: IBM (DB2 UDB), Microsoft (SQL Server), e Oracle. O trabalho conclui que estes sistemas precisam diminuir a necessidade de entrada de dados por parte do DBA, ser capazes de se adaptar dinamicamente à mudanças da carga de trabalho ou do ambiente, permitir ajustes on-line e integrar a determinação e solução dos problemas com ferramentas analíticas. Para concretizar estas evoluções, duas estratégias são possíveis: reescrever os sistemas incluindo técnicas para torná-los autônomo por projeto ou embutir mecanismos de autonomia gradualmente nos sistemas atualmente existentes. Os autores opinam que a tendência da indústria seja adotar a segunda estratégia e introduzir modificações progressivamente como sugere a Figura 11.

De fato, mudanças no sentido de incluir autonomia nos sistemas são introduzidas a cada nova versão liberada. Assim, as últimas versões do DB2, Oracle e SQL Server permitem o ajuste on-line de alguns parâmetros, ainda que existam alguns que exigem a reinicialização do servidor. Todos estes SGBDs oferecem uma ferramenta para seleção de índices (DB2: Design Advisor, SQL Server: Index Wizard, e Oracle: Index Tuning Wizard) que recomenda um conjunto de índices para uma dada carga de trabalho. Nenhum deles, no entanto, cria ou destroi índices sem a autorização do DBA. Os sistemas permitem, ainda, a reorganização on-line de índices e possuem um grupo de ferramentas que coletam informações de desempenho para uso do administrador. O DB2 Health Center e o Oracle Enterprise Manager Console permitem ao DBA especificar quais partes do sistema ele deseja monitorar.

Básico	Gerenciado	Preditivo	Adaptativo	Autônomo
Nível 1	Nível 2	Nível 3	Nível 4	Nível 5
Múltiplas fontes de dados gerados pelo sistema. Requer um extenso grupo de administradores altamente preparados	Dados consolidados através de ferramentas de gerenciamento. O administrador analisa e executa ações.	O sistema monitora, correlaciona e recomenda ações. O administrador aprova e começa a executar ações.	O sistema monitora, correlaciona e recomenda ações. Os administradores gerenciam o desempenho controlando SLAs.	Componentes integrados dinamicamente gerenciados por regras de negócio. Os administradores se concentram em tornar reais as necessidades do negócio.
	Maior reatividade do sistema. Melhoria da produtividade.	Menor dependência de habilidades pessoais. Melhor e mais rápida tomada de decisões.	Administração rápida e robusta com mínima intervenção humana.	A gerência é dirigida por regras de negócio. Agilidade e robustez nos negócios.
Manual				Autônomo

Figura 11: Evolução dos sistemas atuais em direção a sistemas autônomos.
Fonte: IBM Global Services and Autonomic Computing, IBM White Paper, October 2002, http://www-3.ibm.com/autonomic/pdfs/wp-igs-autonomic.pdf

A seguir descrevemos brevemente características e ferramentas interessantes de cada um destes sistemas com respeito a auto-sintonia.

6.1.1 Oracle 10g

Um requisito indispensável para a implementação de sistemas de auto-sintonia é a disponibilidade de mecanismos que permitam mudar dinamicamente a alocação de recursos no sistema. Algumas mudanças nesse sentido vêm sendo introduzidas nos bancos de dados comerciais. Por exemplo, já na versão 9i [NBT03], o SGBD Oracle tinha integrado o gerenciamento dinâmico de memória, que permite ajustar a configuração do buffer compartilhado ou do buffer do sistema sem necessidade de reinicializar o banco antes que as modificações tenham efeito.

A documentação disponível sobre a versão 10g [Wis03] promete, entretanto, toda uma infraestrutura para facilitar a gerência do sistema, mesmo que este tenha sido instalado em um conjunto de servidores distintos. A arquitetura apresentada inclui a instrumentação de todas as camadas para capturar dados que são armazenados no Repositorio Automático de Carga de Trabalho (*Automatic Workload Repository - AWR*). Ao armazenar o histórico de desempenho, o AWR permite tomar ações pró-ativas, pois é possível extrair padrões que sinalizem potenciais problemas de desempenho. Desta forma, o AWR se torna um componente fundamental para viabilizar o diagnóstico de situações problemáticas no sistema.

Integrado no núcleo do banco está o Monitor Automático de Diagnóstico do Banco de Dados (*Automatic Database Diagnostic Monitor - ADDM*), uma ferramenta para o diagnóstico integral de problemas com o sistema de banco de dados. O ADDM monitora o estado do banco em intervalos regulares a partir dos dados coletados no AWR e efetua análises utilizando um meca-

nismo de diagnóstico de desempenho baseado no conhecimento dos especialistas em sintonia da Oracle. Este mecanismo realiza uma análise *top-down* procurando gargalos potenciais e retorna um conjunto de possíveis causas. O ADDM consegue valorar o impacto de cada problema no sistema e o ganho caso o problema seja resolvido. Esta infra-estrutura permite não somente alertar o DBA de problemas no sistema como também oferecer recomendações das ações necessárias para corrigi-los. As recomendações de ações corretivas são criadas com a ajuda de um grupo de ferramentas conselheiras (*advisors*). Os ajustes, no entanto, não são executados de forma autônoma e, portanto, consideramos que este SGBD não pode ser classificado como um sistema com auto-sintonia global.

O Oracle 10g fornece também vários mecanismos de sintonia local, como a Gerência Automática de Memória (*Automatic Memory Management - AMM*). Esta é uma ferramenta para a alocação automática de memória que monitora automaticamente as demandas das áreas compartilhadas e realoca memória RAM baseada na carga de trabalho atual.

6.1.2 Microsoft SQL Server

Desde a versão 7.0, a Microsoft [Mic] tem reiterado a intenção de que o SQL Server seja um sistema capaz de auto-sintonia e de auto-administração. O número de ferramentas para sintonia tem aumentado, assim como a capacidade do banco para efetuar mudanças dinâmicas no sistema. Um exemplo é a introdução desde a versão 7.0 do Index Tuning Wizard, uma ferramenta desenvolvida como parte do projeto AutoAdmin da Microsoft Research (veja a seção 5.1) que recomenda um grupo de índices e visões materializadas apropriadas para uma carga de trabalho dada. Esta ferramenta não é autônoma, uma vez que só no caso da recomendação ser aceita pelo administrador esta é efetivamente materializada.

O SQL Server 2000 possui um grupo de mecanismos de auto-sintonia local integrados no sistema. Assim, é capaz de executar automaticamente tarefas como a atualização das estatísticas e a gerência do tamanho do arquivo físico do banco de dados. Alguns parâmetros, como a alocação de memória para operações de banco de dados, não precisam mais da intervenção do administrador para serem ajustados. A versão Enterprise Edition ajusta dinamicamente o número máximo de *read ahead pages* baseado na quantidade de memória disponível. O SQL Server admite, ainda, a configuração de alarmes no caso dos valores dos contadores de desempenho (*performance counters*) ultrapassarem limites pré-definidos. Os alarmes são configurados usando o SQL Server Agent.

A próxima versão do SQL Server (que tem o codinome Yukon), segundo foi anunciado, oferecerá versões melhoradas das ferramentas de auto-sintonia existentes e introduzirá um conjunto de novas ferramentas. Entre estas, podemos destacar uma nova ferramenta de sintonia de índices denominada *Database Tuning Advisor (DTA)*, desenvolvida pelo grupo da Microsoft Research em colaboração com o grupo de desenvolvimento do SQL Server. O DTA, além de sugerir índices, também pode fazer sugestões de particionamento de tabelas do banco de dados. Além desta ferramenta, novas características interessantes incluem a possibilidade de realizar a criação e gerência de índices on-line e a redução do número de parâmetros que requerem reiniciar o servidor de banco de dados após mudanças.

Apesar de todas estas iniciativas, ainda não encontramos indícios de implementação de auto-sintonia global no SQL Server.

6.1.3 IBM DB2 UDB

O DB2 UDB [DB2] oferece mecanismos de auto-sintonia como, por exemplo, o *expert cache*. Quando este mecanismo detecta uma leitura seqüencial em uma tabela, ele aumenta automaticamente o buffer reservado para *prefetching* com o objetivo de reduzir o número de operações de entrada/saída.

Além dos mecanismos de auto-sintonia embutidos no sistema, foram criadas ferramentas para auxiliar o trabalho do DBA. Um exemplo é o *Performance Expert*, que integra produtos de monitoramento, geração de relatórios, gerenciamento de buffer pools e um *Performance Warehouse* em que são armazenados dados para pesquisa e análise de tendências. Assim, a ferramenta oferece uma visão global de todos os subsistemas e instâncias do SGBD que estão sendo executadas, mesmo estando em diferentes plataformas. A ferramenta é capaz, ainda, de sugerir recomendações de sintonia. Outras ferramentas conselheiras também foram incluídas no sistema, como, por exemplo, o DB2 Advisor (ver seção 5.1).

6.2 Fornecedores de ferramentas para análise de desempenho

No grupo de fornecedores de ferramentas de diagnóstico e análise de desempenho para bancos de dados estão: VERITAS Software, BMC Software, Quest Software, Teamquest Corporation e Embarcadero Technologies. Estes em geral oferecem suporte para os principais SGBDs comerciais, que são Oracle, Microsoft SQL Server, IBM DB2 UDB, Sybase e Informix.

As ferramentas de análise de desempenho oferecidas possuem os seguintes aspectos em comum:

1. Oferecem monitoramento em tempo real através da coleta de amostras das métricas de desempenho em intervalos pré-definidos. Este monitoramento tende a ser pouco intrusivo.

2. Mantêm um repositório com os dados coletados para permitir análises de problemas acontecidos e de tendências. Para apoiar estas análises, as ferramentas oferecem componentes para geração de relatórios que ajudam o administrador na tomada de decisões.

3. Defendem que a gerência de desempenho deve ser um processo em que o administrador se envolve de forma pró-ativa. Assim, contêm sistemas de alertas pró-ativos para avisar ao administrador sobre problemas de desempenho existentes ou potenciais. Isto permite que ações corretivas sejam empreendidas antes que o usuário final seja afetado. O administrador pode ser notificado com alertas personalizados através de *pagers*, fax, e-mails ou, até mesmo, telefone. Os alertas podem ter diferentes severidades, como simples avisos ou alarmes, o que permite indicar se estamos enfrentando problemas potenciais ou existentes.

4. Oferecem em geral três tipos de análises automatizadas: limite, análise de correlação e de tendências. Quando um problema é detectado a partir destas análises, é possível analisar o sistema fim-a-fim (*end-to-end*), detectar a fonte do problema e, então, executar uma análise em profundidade para descobrir a verdadeira causa.

5. Baseadas em sistemas especialistas, sugerem recomendações para resolver os problemas de desempenho. Podem incluir ferramentas para sintonia de SQL, gerência e recuperação de espaço em disco, serviços de escalonamento, etc., conectadas com a ferramenta que sugere as recomendações.

6. Oferecem consoles gráficos centralizados. A tendência é que estes sejam multiplataforma e admitam o gerenciamento de sistemas de bancos de dados heterogêneos.

7. Simulam a resposta do sistema ante mudanças nos parâmetros de sintonia e geram estimativas de ganho no caso das mudanças serem efetivadas.

Em nenhum caso estes fornecedores oferecem ferramentas que executem auto-sintonia. Entretanto, é interessante observar nos produtos por eles oferecidos as metodologias assumidas para o processo de sintonia, os mecanismos de monitoramento, e os métodos de diagnóstico e predição da resposta do sistema a ajustes, entre outros. A seguir, enumeramos alguns dos produtos oferecidos por estes fornecedores.

6.2.1 Embarcadero Technologies

O *Performance Center* é uma ferramenta de monitoramento e gerência de desempenho que tem a particularidade de que consegue gerenciar o desempenho de bancos de dados Oracle, Sybase e Microsoft SQL Server em um console único. Ele oferece a capacidade de navegação em profundidade e monitoramento em tempo real, e pode ser integrado com outras ferramentas, como o *DBArtisan* e o *Embarcadero SQL Tuner* (somente para Oracle) para resolver problemas de código ou alocação de recursos em consultas SQL, respectivamente.

6.2.2 Quest Software

A Quest Software oferece uma suíte que permite gerenciar sistemas de bancos de dados heterogêneos a partir de uma única console chamada *Quest Central*. Esta traduz a terminologia e outros detalhes que são diferentes nos SGBDs suportados (Oracle, DB2, SQL Server ou uma mistura deles) para um modelo comum. Assim, o administrador somente precisa ter o conhecimento deste modelo de comportamento do desempenho de SGBDs para poder administrar sistemas provenientes de fornecedores distintos. A suíte contem as ferramentas *Quest Central Performance Diagnostics*, *Quest Central SQL Tuning*, *Quest Central SQL Analysis*, *Quest Central Database Administration*, and *Quest Central Space Management*.

6.2.3 Veritas Software

Com a compra da Precise Software, a Veritas Software extendeu a sua oferta com soluções de gerência de desempenho para vários ambientes, incluindo aplicações baseadas em bancos de dados Oracle, DB2 UDB e SQL Server. O enfoque das ferramentas está orientado a uma gerência fim-a-fim da aplicação. Isto segue a idéia de que uma verdadeira gerência de desempenho da aplicação precisa integrar todos os componentes da sua infra-estrutura (sistemas de banco de dados, armazenamento, servidores de aplicações). A Veritas oferece uma suíte formada pelas ferramentas *Insight*, *Indepth* e *Alarm*. O *Veritas Insight* oferece informação sobre os tempos de resposta fim-a-fim de qualquer aplicação que use a família de protocolos TCP/IP, assim como dos tempos de serviço segmentados. O *Veritas Indepth* permite que o administrador aprofunde sua análise pelas várias camadas sub-componentes do sistema para diagnosticar a verdadeira causa do problema de desempenho. Já o *Veritas Alarm* oferece um serviço de notificações.

A suíte se baseia no monitoramento contínuo das aplicações em tempo real de todas as camadas da infra-estrutura da aplicação, o que permite que o sistema gere ações pró-ativas que evitem uma deterioração notável do desempenho. Um detalhe importante é o fato de que, apesar da amostragem ser freqüente (1s como valor pré-determinado), a ferramenta impõe uma sobrecarga mínima sobre o sistema. O *Veritas Indepth* para Oracle, por exemplo, consegue minimizar esta sobrecarga coletando a informação diretamente da área de memória compartilhada (SGA) do Oracle. A informação é armazenada em arquivos locais que são enviados em intervalos regulares para um repositório chamado *Performance Warehouse*.

Durante a carga do repositório, os dados são analisados e consolidados. O *Performance Warehouse* permite realizar análises históricas e de tendências sobre os dados registrados e identificar zonas com conflito.

6.2.4 Teamquest Corporation

A Teamquest Corporation oferece produtos para gerência de desempenho e planejamento de capacidade. As métricas de desempenho são obtidas por um conjunto de agentes coletores de dados compartilhados por todas as ferramentas. Os *probes*, como são chamados na terminologia da Teamquest, formam o *Teamquest Performance Framework*, um arcabouço para monitoramento dos sistemas objetivo. Eles são responsáveis pelas funções de alarmes e notificação de eventos,

além da coleta automática e armazenamento dos dados de desempenho provenientes dos vários componentes dos sistemas monitorados (SO, servidor de aplicação). Os *probes* podem ser configurados para monitorar novas métricas definidas pelo usuário. A Teamquest também oferece produtos que utilizam os dados coletados para gerar alarmes em caso de problemas de desempenho, efetuar vários tipos de análises sobre os dados, planejamento de capacidade, publicação de relatórios e predição dos efeitos de mudanças sobre o sistema. Em especial, o *Teamquest Model* oferece suporte à construção de modelos de filas (*queueing network models*) do servidor, o que o diferencia de outras soluções baseadas em sistemas especialistas.

6.2.5 BMC Software

A BMC Software possui uma extensa linha de ferramentas para gerência de desempenho. Entre as ferramentas disponíveis, podemos destacar o *Space Expert for Oracle*, que permite reorganizar o banco de dados mantendo-o disponível aos usuários durante o processo. Isto é distinto do método tradicional que consiste em fazer a reorganização off-line. A ferramenta também fornece recomendações de parâmetros relativos ao armazenamento, baseada nos dados de atividade coletados.

Outra ferramenta interessante é o *SQL Explorer* (para DB2 UDB e Oracle), que permite identificar com mínima sobrecarga as consultas mais consumidoras de recursos e de pior desempenho. O *SQL Explorer* consegue otimizar as consultas usando uma tecnologia de simulação não intrusiva chamada *Virtual Instance*. A ferramenta é capaz de reportar o tempo estimado de execução de uma consulta e contém uma base de conhecimento que sugere dicas de otimização. As consultas são extraídas do SGBD por um processo independente chamado *High Speed Data Collector*. A coleta das consultas acessa diretamente a memória do servidor, o que implica um impacto mínimo no desempenho do sistema de banco de dados.

O Patrol 7 [BMC] é primariamente uma arquitetura de monitoramento e gerência de desempenho de sistemas e aplicações totalmente componentizada e distribuída. De uma console única multiplataforma (*Patrol Central*) é possível controlar todas as camadas da infra-estrutura da aplicação. O *Patrol Central* recebe e consolida as informações coletadas pelos agentes remotos, para que os administradores possam consultá-las e tomar decisões. Os outros componentes da arquitetura são os *Patrol Knowledge Modules* e os *Patrol Agents*.

Cada *Patrol Knowledge Module* (*Patrol KM*) fornece instruções aos agentes sobre como gerenciar a aplicação. Os módulos *Patrol Knowledge* são específicos para cada tecnologia monitorada. Os dados sempre devem ser coletados com a mínima sobrecarga possível para o sistema. Assim, o *Performance Collector* da versão Unix, por exemplo, coleta os dados acessando diretamente o *kernel* [Pat].

Os agentes Patrol executam autônoma e remotamente em cada servidor gerenciado, o que significa que não há necessidade de uma conexão contínua com a console central. Cada agente recebe instruções dos módulos *Patrol KM* para executar ações de descoberta de aplicações e objetos, monitoramento automático, detecção de eventos, agregação de informações, início de ações corretivas e notificação aos administradores. Cada servidor contém um agente e um ou vários *Patrol KMs*. Cada agente tem seu próprio repositório local de dados e eventos. Estes podem ser usados para resolução de problemas. A arquitetura oferece suporte para a execução de ações de auto-sintonia. Por exemplo, o *Space Expert Execution Agent* monitora e extrai informações sobre problemas de espaço de armazenamento. No caso de existirem tais problemas, o agente é capaz de agendar e executar um processo de reorganização para recuperar espaço.

Das arquiteturas revisadas, esta é a de maior potencial para a execução de tarefas de auto-sintonia.

7 Conclusão

Existe um crescente interesse no desenvolvimento de sistemas de banco de dados capazes de se auto-adaptar como um todo às condições do ambiente. Isso é notável tanto em trabalhos acadêmicos como em sistemas comerciais.

Neste trabalho, apresentamos uma classificação de propostas de auto-sintonia de bancos de dados presentes na literatura. Dividimos as propostas em dois grupos básicos, enfocando sistemas que realizam auto-sintonia global ou auto-sintonia local. Para cada grupo, realizamos uma descrição mais aprofundada sobre as suas diversas ramificações componentes. Esperamos que o leitor tenha se familiarizado com as diversas propostas existentes e que tenha compreendido os princípios empregados até o momento no projeto de sistemas capazes de auto-sintonia. O princípio mais importante, que permeia praticamente todas as propostas, é a idéia de um *ciclo de controle com realimentação*, em que *métricas* são continuamente coletadas sobre o estado atual do sistema e sua análise permite a *execução de mudanças* e o *acompanhamento do desempenho* resultante.

Existe uma miríade de direções de pesquisa em que as propostas presentes na literatura podem ser estendidas. Abaixo, comentamos algumas possibilidades que são de interesse do nosso grupo de estudo em particular.

Para os problemas de seleção de índices e de ajuste de buffers, podemos notar que a maior parte das propostas procura criar um mecanismo de decisão que atua sobre uma dada carga de trabalho de entrada conhecida. Esta carga de trabalho precisa, entretanto, ser obtida por um administrador do sistema. Acreditamos que trabalhos futuros podem estender estas propostas tornando a obtenção da carga de trabalho e a realização dos ajustes sobre o SGBD totalmente automáticos. Para isto, se faz necessário criar componentes para uma coleta eficiente de estatísticas sobre o desempenho do sistema e estabelecer heurísticas para disparar as ações de ajuste.

Podemos observar, ainda, que as propostas de auto-sintonia global se encontram bem menos desenvolvidas do que as propostas de auto-sintonia local. Isto motiva a sugestão de novos trabalhos que efetivamente implementem SGBDs mais simples modelados para serem auto-sintonizáveis por projeto. O estudo de arquiteturas para a construção de sistemas mais adaptáveis ao seu ambiente é muito importante neste contexto. Agentes de *software*, por exemplo, podem ser utilizados em conjunto com SGBDs para perseguir o objetivo de construção de sistemas auto-sintonizáveis [Mac00].

Outra área de pesquisa relevante é o estudo das relações e conflitos de recursos existentes entre os diversos módulos de SGBDs. Uma compreensão mais aprofundada destes relacionamentos e interações ainda não está bem fundamentada e é fundamental para a construção de sistemas robustos e em que garantias de níveis de serviço possam ser dadas [WMHZ02].

Ainda assim, aspectos comuns nas arquiteturas para auto-sintonia global até agora propostas podem ser identificados. O método de controle tipicamente empregado é o de um ciclo de controle de realimentação. As etapas de auto-sintonia são, em geral, observação (que pode se dividir em monitoramento e análise de situação), previsão e ação. Políticas de sintonia podem ser definidas pelo administrador do sistema para orientar o sistema em seu processo de ajuste. Se os dados coletados na etapa de observação forem armazenados numa base de conhecimento, é possível verificar, numa etapa posterior de análise, se as métricas de desempenho desejadas foram ou não violadas. A coleta de dados deve ser pouco intrusiva de forma a não afetar o comportamento do sistema e, portanto, gerar dados válidos. O sistema de banco de dados deverá possuir mecanismos que permitam tanto o acesso às métricas de desempenho como a modificação em tempo real dos parâmetros que precisem ser corrigidos interativamente.

Todas estas técnicas deverão permitir aos administradores de bancos de dados minimizar o tempo dedicado às tarefas rotineiras inerentes à gerência de desempenho do sistema. Acreditamos que isto poderá torná-los mais livres para se focar em tarefas que permitam realmente derivar valor a partir dos dados.

Referências

[AB77] H. D. Anderson e P. B. Berra. Mininum cost selection of secondary indexes. *ACM Transactions on Database Systems (TODS)*, 2(1):68–90, 1977.

[AC99] A. Aboulnaga e S. Chaudhuri. Self-tuning histograms: building histograms without looking at data. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, páginas 181–192. 1999.

[ACDN03] R. Agrawal, S. Chaudhuri, A. Das, e V. Narasayya. Automating layout of relational databases. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*, páginas 607–618. 2003.

[ACN00] S. Agrawal, S. Chaudhuri, e V. Narasayya. Automated selection of materialized views and indexes for sql databases. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, páginas 496–505. 2000.

[AUT] Projeto AutoAdmin. http://research.microsoft.com/research/DMX/autoadmin/default.asp, acessado em 11/01/2004.

[Bar93] J. S. Barrera III. Self-tuning systems software. In *Proceedings of the Workshop on Workstation Operating Systems, IEEE Computer Society*, páginas 194–197. 1993.

[BBC+98] P. Bernstein, M. Brodie, S. Ceri, D. Dewitt, M. Franklin, H. Garcia-Molina, J. Gray, J. Held, J. Hellerstein, H. Jagadish, M. Lesk, D. Maier, J. Naughton, H. Pirahesh, M. Stonebraker, e J. Ullman. The Asilomar report on database research. *ACM SIGMOD Record*, 27(4):74–80, 1998.

[BCG01] N. Bruno, S. Chaudhuri, e L. Gravano. Stholes: A multidimensional workload-aware histogram. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, páginas 211–222. May 2001.

[BCL93a] K. Brown, M. Carey, e M. Livny. Managing memory to meet multiclass workload response time goals. In *Proceedings of the International Conference on Very Large Databases (VLDB)*. 1993.

[BCL93b] K. Brown, M. Carey, e M. Livny. Towards an autopilot in the dbms performance cockpit. In *Proceedings of the International High Performance Transaction Systems Workshop*. 1993.

[BCL96] K. Brown, M. Carey, e M. Livny. Goal oriented buffer management revisited. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, páginas 353–364. 1996.

[Ben03] D. Benoit. *Automatic Diagnosis of Performance Problems in Database Management Systems*. Tese de Doutorado, School of Computing, Queen's University, 2003.

[BGK+95] J. Bailey, M. Georgeff, D. Kemp, D. Kinny, e K. Ramamohanarao. Active databases and agent systems - a comparison. In *Proceedings of the International Workshop on Rules in Database Systems, Lecture Notes in Computer Science 985*, páginas 342–356. 1995.

[BHJS00] J. P. Bigus, J. L. Hellerstein, T. S. Jayram, e M. S. Squillante. Auto tune: A generic agent for automated performance tuning. In *Proceedings of the International*

Conference and Exhibition on the Practical Application of Intelligent Agents and Multi-Agent Systems (PAAM), páginas 33–52. 2000.

[BMC] BMC Software. http://www.bmc.com, acessado em 11/01/2004.

[BMCL94] K. P. Brown, M. Mehta, M. J. Carey, e M. Livny. Towards automated performance tuning for complex workloads. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, páginas 72–84. 1994.

[BPS90] E. Barucci, R. Pinzani, e R. Sprugnoli. Optimal selection of secondary indexes. *IEEE Transactions on Software Engineering*, 16(1):32–38, 1990.

[BPT97] E. Baralis, S. Paraboschi, e E. Teniente. Materialized views selection in a multidimensional database. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, páginas 156–165. 1997.

[CA] Ca unicenter tng platform. Http://www.ca.com/unicenter.

[CABK88] George P. Copeland, William Alexander, Ellen E. Boughter, e Tom W. Keller. Data placement in bubba. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, páginas 99–108. 1988.

[CBC93] S. Choenni, H. Blanken, e T. Chang. On the selection of secondary indices in relational databases. *IEEE Data and Knowledge Engineering*, 11(3):207–233, 1993.

[CCG+99] S. Chaudhuri, E. Christensen, G. Graefe, V. Narasayya, e M. Zwilling. Self-tuning technology in microsoft sql server. *IEEE Data Engineering Bulletin, Special Issue on Self-Tuning Databases and Application Tuning*, 22(2):20–26, 1999.

[CD85] H. Chou e D. J. DeWitt:. An evaluation of buffer management strategies for relational database systems. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, páginas 127–141. 1985.

[CFM95] A. Capara, M. Fischetti, e D. Maio. Exact and approximate algorithms for the index selection problem in physical database design. *IEEE Transactions on Knowledge and Data Engineering*, 7(6):955–967, 1995.

[CFW+95] J.Y. Chung, D. Ferguson, G. Wang, C. Nikolaou, e J. Teng. Goal oriented dynamic buffer pool management for data base systems. In *Proceedings of the IEEE International Conference on Engineering of Complex Computer Systems*, páginas 191–198. 1995.

[CL02] R. L. C. Costa e S. Lifschitz. Index self-tuning and agent-based databases. In *Proceedings of the Latin-American Conference on Informatics (CLEI)*, páginas 76, Abstracts Proceedings; 12 pp. CD–ROM Proceedings. 2002.

[CN97] S. Chaudhuri e V. Narasayya. An efficient, cost-driven index selection tool for microsoft sqlserver. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, páginas 146–155. 1997.

[CN98a] S. Chaudhuri e V. Narasayya. Autoadmin "what-if" index analysis utility. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, páginas 367–377. 1998.

[CN98b] S. Chaudhuri e V. Narasayya. Microsoft index tuning wizard for sql server 7.0. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, páginas 553–554. 1998.

[CN99] S. Chaudhuri e V. Narasayya. Index merging. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*, páginas 296–303. 1999.

[CN00] S. Chaudhuri e V. Narasayya. Automating statistics management for query optimizers. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*, páginas 339–348. 2000.

[Com78] D. Comer. The difficulty of optimum index selection. *ACM Transactions on Database Systems (TODS)*, 3(4):440–445, 1978.

[CR93] C. M. Chen e N. Roussopoulos. Adaptive database buffer allocation using query feedback. In *Proceedings of the 19th International Conference on Very Large Databases (VLDB)*, páginas 342–353. August 1993.

[CR94] C. M. Chen e N. Roussopoulos. Adaptive selectivity estimation using query feedback. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, páginas 161–172. 1994.

[CW00] S. Chaudhuri e G. Weikum. Rethinking database system architecture: Towards a self-tuning risc-style database system. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, páginas 1–10. 2000.

[DB2] DB2 universal database. http://www-3.ibm.com/software/data/db2imstools/performmgt.html, acessado em 11/01/2004.

[DEF+03] Y. Diao, F. Eskesen, S. Froehlich, J. L. Hellerstein, L. Spainhower, e M. Surendra. Generic, on-line optimization of multiple configuration parameters with application to a database server. In *Proceedings of the IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM), LNCS 2867*, páginas 3–15. 2003.

[DG92] D. DeWitt e J. Gray. Parallel database systems: The future of high performance database systems. *Communications of the ACM*, 35(6):85–98, 1992.

[DHPB03] Y Diao, J. L. Hellerstein, S. Parekh, e J. P. Bigus. Managing web server performance with autotune agents. *IBM Systems Journal*, 42(1):136 – 149, 2003.

[DIR99] D. Donjerkovic, Y. Ioannidis, e R. Ramakrishnan. Dynamic histograms: Capturing evolving data sets. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*, páginas 86–105. 1999.

[Emb] Embarcadero technologies. http://www.embarcadero.com, acessado em 11/01/2004.

[EPBM03] S. Elnaffar, W. Powley, D. Benoit, e P. Martin. Today's DBMSs: How autonomic are they? In *Proceedings of the International Workshop on Database and Expert Systems Applications (DEXA)*, páginas 651–655. IEEE Computer Society, 2003.

[Fer78] Domenico Ferrari. *Computer Systems Performance Evaluation*. Prentice Hall, 1978.

[FN99] D. G. Feitelson e M. Naaman. Self-tuning systems. *IEEE Software*, 16(2):52–60, 1999.

[FNS95] C. Faloutsos, R. T. Ng, e T. K. Sellis. Flexible and adaptable buffer management techniques for database management systems. *IEEE Transactions on Computers*, 44(4):546–560, 1995.

[FON92] M. Frank, E. Omiecinski, e S. Navathe. Adaptive and automated index selection in rdbms. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*, páginas 277–292. 1992.

[FST88] S. Finkelstein, M. Schkolnick, e P. Tiberio. Physical database design for relational databases. *ACM Transactions on Database Systems (TODS)*, 13(1):91–128, 1988.

[GHRU97] H. Gupta, V. Harinarayan, A. Rajaraman, e J. D. Ullman. Index selection for olap. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*, páginas 208–219. 1997.

[GLR00] V. Ganti, M. Lee, e R. Ramakrishnan. Icicles: Self-tuning samples for approximate query answering. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, páginas 176–187. 2000.

[GM99] H. Gupta e I. S. Mumick. Selection of views to materialize under a maintenance cost constraint. In *Proceedings of the International Conference on Database Theory (ICDT)*, páginas 453–470. 1999.

[GMP97] P. B. Gibbons, Y. Matias, e V. Poosala. Fast incremental maintenance of approximate histograms. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, páginas 466–475. 1997.

[Gup97] H. Gupta. Selection of views to materialize in a data warehouse. In *Proceedings of the International Conference on Database Theory (ICDT)*, páginas 98–112. 1997.

[GUW00] H. Garcia-Molina, J. Ullman, e J. Widom. *Database System Implementation*. Prentice-Hall, 2000.

[HA03] S. Harizopoulos e A. Ailamaki. A case for staged database systems. In *Proceedings of the Biennial Conference on Innovative Data Systems Research (CIDR)*. 2003. http://www.pdl.cmu.edu/PDL-FTP/Database/p3.pdf.

[HC76] M. Hammer e A. Chan. Index selection in a self-adaptive data base management system. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, páginas 1–8. 1976.

[Hel97] J. Hellerstein. Automated tuning systems: Beyond decision support. In *Proceedings of the Computer Measurement Group*, páginas 277–292. 1997.

[HLY93] K. A. Hua, C. Lee, e H. C. Young. Data partitioning for multicomputer database systems: A cell-based approach. *Information Systems*, 18(5):329–342, 1993.

[Hor01] P. Horn. Autonomic computing: Ibm's perspective on the state of information technology, 2001. http://www.research.ibm.com/autonomic/manifesto, acessado em 11/01/2004.

[HRU96] V. Harinarayan, A. Rajaraman, e J. D. Ullman. Implementing data cubes efficiently. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, páginas 205–216. 1996.

[HZS99] J. Hellerstein, F. Zhang, e P. Shahabuddin. An approach to predictive detection for service management. In *Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management (IM)*, páginas 309–322. 1999.

[IBM] International Bussiness Machines Corporation - IBM. http://www.ibm.com, acessado em 11/01/2004.

[ISR83] M. Y. L. Ip, L. V. Saxton, e V. V. Raghavan. On the selection of an optimal set of indexes. *IEEE Transactions on Software Engineering*, 9(2):135–143, 1983.

[Jai91] R. K. Jain. *The Art of Computer Systems Performance Analysis : Techniques for Experimental Design, Measurement, Simulation, and Modeling*. John Wiley & Sons, 1991.

[JJOT01] H. V. Jagadish, Hui Jin, Beng Chin Ooi, e Kian-Lee Tan. Global optimization of histograms. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, páginas 223–234. 2001.

[JS94] T. Johnson e D. Shasha. 2q: A low overhead high performance buffer management replacement algorithm. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, páginas 439–450. 1994.

[Kin74] W. F. King. On the selection of indices for a file. Relatório técnico, IBM Research, 1974. Rep. RJ1341.

[KKZ99] A. Kemper, D. Kossmann, e B. Zeller. Performance tuning for sap r/3. *IEEE Data Engineering Bulletin, Special Issue on Self-Tuning Databases and Application Tuning*, 22(2):32–39, 1999.

[KMKPS99] E. Kendall, P. V. Murali Krishna, C. Pathak, e C.B. Suresh. A framework for agent systems. In M. Fayad, D.C. Schmidt, e R. Johnson, editores, *Implementing Application Frameworks: Object-Oriented Frameworks*, páginas 113–154. John Wiley & Sons, 1999.

[KW02] A. C. König e G. Weikum. A framework for the physical design problem for data synopses. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*, páginas 627–645. 2002.

[LAB] Laboratório de Sistemas de Bancos de Dados da Universidade de Queen's. http://www.cs.queensu.ca/home/cords/, acessado em 11/01/2004.

[LKO+00] Mong Li Lee, M. Kitsuregawa, Beng Chin Ooi, Kian-Lee Tan, e A. Mondal. Towards self-tuning data placement in parallel database systems. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, páginas 225–236. 2000.

[LLZ02] S. Lightstone, G. Lohman, e D. Zilio. Toward autonomic computing with DB2 universal database. *ACM SIGMOD Record*, 31(3):55–61, 2002.

[LM03] A. Lerner e I. Manolescu. The nuts and bolts of dbms construction: Building your own prototype. In *Proceedings of the Simpósio Brasileiro de Bancos de Dados (SBBD)*, página 4. 2003. Tutorial.

[LNS93] W. Litwin, M. A. Neimat, e D. A. Schneider. Lh* - linear hashing for distributed files. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, páginas 327–336. 1993.

[LQA97] W. Labio, D. Quass, e B. Adelberg. Physical database design for data warehouses. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*, páginas 277–288. 1997.

[LVZ+00] G. Lohman, G. Valentin, D. Zilio, M. Zuliani, e A. Skelley. DB2 advisor: An optimizer smart enough to recommend its own indexes. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*, páginas 101–110. 2000.

[Mac00] J. A. F. Macêdo. *Agent-based DBMSs study (in portuguese).* Tese de Mestrado, Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio), 2000.

[McC03] J. A. McCann. The database machine: Old story, new slant? In *Proceedings of the Biennial Conference on Innovative Data Systems Research (CIDR)*. 2003. http://www-db.cs.wisc.edu/cidr/program/p6.pdf.

[Mic] Microsoft corporation. http://microsoft.com.

[Moh92] C. Mohan. Interactions between query optimizations and concurrency control. In *Proceedings of the International Workshop on Research Issues on Data Engineering: Transaction and Query Processing (RIDE-TQP)*, páginas 26–35. 1992.

[MPLR02] P. Martin, W. Powley, H. Li, e K. Romanufa. Managing database server performance to meet QoS requirements in electronic commerce systems. *International Journal on Digital Libraries*, 3(4):316–324, 2002.

[MS78] S. T. March e D. G. Severance. A mathematical modelling approach to the automatic selection of database design. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, páginas 52–65. 1978.

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

[NBT03] R. Niemiec, B. Brown, e J. Trezzo. *Oracle 9i Performance Tuning: dicas e técnicas.* Campus, 2003.

[Nor03] M. F. Noronha. Implementação de um agente de auto-sintonia de Índices no postgresql, 2003. Relatório final de Projeto de Programação, Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio).

[Oga97] K. Ogata. *Modern Control Engineering, 3rd edition.* Prentice Hall, 1997.

[OOW93] E. O'Neil, P. O'Neil, e G. Weikum. The lru-k page replacement algorithm for database disk buffering. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, páginas 297–306. 1993.

[Oraa] Oracle corporation. Http://www.oracle.com/.

[Orab] Oracle enterprise manager 9i tuning pack. http://otn.oracle.com/products/oem/files/tp.html, acessado em 11/01/2004.

[OV] Hp openview platform. Http://www.openview.hp.com.

[Pat] Patrol(r) knowledge module (tm) for unix v8.3 - features and faqs. Http://www.softwareinnovations.co.uk/downloads/patrol/9066.pdf.

[Que] Quest software. Http://www.quest.com/.

[Rah97] E. Rahm. Goal-oriented performance control for transaction processing. In *Proceedings of the ITG/GI MMB Conference (Messung, Modellierung und Bewertung von Rechen- und Kommunikationssystemen)*, páginas 285–300. 1997.

[RN95] S. Russel e P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 1995.

[Rou82] N. Roussopoulos. View indexing in relational databases. *ACM Transactions on Database Systems (TODS)*, 7(2):258–290, 1982.

[RS91] S. Rozen e D. Shasha. A framework for automating physical database design. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, páginas 401–411. 1991.

[RZML02] J. Rao, C. Zhang, N. Megiddo, e G. Lohman. Automating physical database design in a parallel database. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, páginas 558–569. 2002.

[SAA] Bancos de dados e sistemas de informação, Universidade de Saarland. http://www-dbs.cs.uni-sb.de/, acessado em 11/01/2004.

[SAC+79] P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, e T. G. Price. Access path selection in a relational database management system. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, páginas 23–34. 1979.

[Sal03] M. V. Salles. Agente de auto-sintonia de Índices baseado em diferenças, 2003. Projeto Final de Programação, Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio).

[SB03] D. Shasha e P. Bonnet. *Database Tuning: Principles, Experiments and Troubleshooting Techniques*. Morgan Kaufmann, 2003.

[Sch75] M. Schkolnick. Secondary index optimization. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, páginas 186–192. 1975.

[SDN98] A. Shukla, P. Deshpande, e J. F. Naughton. Materialized view selection for multidimensional datasets. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, páginas 488–499. 1998.

[Sha96] D. Shasha. Tuning databases for high performance. *ACM Computing Surveys*, 28(1):113–115, 1996.

[Sha97] D. Shasha. Lessons from Wall Street: case studies in configuration, tuning, and distribution. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, páginas 498–501. 1997.

[Sha99] D. Shasha. Tuning time series queries in finance: Case studies and recommendations. *IEEE Data Engineering Bulletin, Special Issue on Self-Tuning Databases and Application Tuning*, 22(2):40–46, 1999.

[SK99a] M. Sinnwell e A. C. König. Managing distributed memory to meet multiclass workload response time goals. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*, páginas 87–94. 1999.

[SK99b] M. Subramanian e V. Krishnamurthy. Performance challenges in object-relational dbmss. *IEEE Data Engineering Bulletin, Special Issue on Self-Tuning Databases and Application Tuning*, 22(2):27–31, 1999.

[SLMK01] M. Stillger, G. Lohman, V. Markl, e M. Kandil. LEO - DB2's LEarning Optimizer. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, páginas 19–28. 2001.

[SMA] Projeto SMART - self managing and resource tuning. http://www.almaden.ibm.com/software/dm/SMART/index.shtml, acessado em 11/01/2004.

[SV99] K. B. Schiefer e G. Valentin. DB2 universal database performance tuning. *IEEE Data Engineering Bulletin, Special Issue on Self-Tuning Databases and Application Tuning*, 22(2):12–19, 1999.

[SWZ93] P. Scheuermann, G. Weikum, e P. Zabback. Adaptive load balancing in disk arrays. In *Proceedings of the International Conference on Foundations of Data Organization and Algorithms (FODO)*, páginas 345–360. 1993.

[SWZ98] P. Scheuermann, G. Weikum, e P. Zabback. Data partitioning and load balancing in parallel disk systems. *The VLDB Journal*, 7:48–66, 1998.

[TB00] D. Theodoratos e M. Bouzeghoub. A general framework for the view selection problem for data warehouse design and evolution. In *Proceedings of the ACM International Workshop on Data Warehousing and OLAP*, páginas 1–8. 2000.

[Tea] Teamquest corporation. http://www.teamquest.com, acessado em 11/01/2004.

[Tho98] A. Thomasian. Concurrency control: Methods, performance, and analysis. *ACM Computing Surveys*, 30(1):70–119, 1998.

[Tiv] Ibm tivoli platform. Http://www.tivoli.com, acessado em 23/01/2004.

[TJW] Departamento de Sistemas Adaptáveis, IBM T. J. Watson. http://www.research.ibm.com/PM/, acessado em 11/01/2004.

[TPC] Transaction processing council. http://www.tpc.org, acessado em 11/01/2004.

[TS97] D. Theodoratos e T. Sellis. Data warehouse configuration. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, páginas 126–135. 1997.

[VBW98] R. Vingralek, Y. Breitbart, e G. Weikum. Snowball: Scalable storage on networks of workstations with balanced load. *Distributed and Parallel Databases*, 6(2):117–156, 1998.

[Ver] Veritas application performance management. http://www.veritas.com/products/category/ProductFamily.jhtml?baseId=4772, acessado em 11/01/2004.

[Vic98] S. Vicari. *Proposta de um sistema de regras de sintonia de performance para aplicações de bancos de dados*. Tese de Mestrado, Universidade Federal do Rio Grande do Sul, 1998.

[Wei99] Gerhard Weiss, editor. *Multiagent Systems: A modern Approach to Distributed Artificial Intelligence*. The MIT Press, Cambridge, MA, USA, 1999.

[Wha85] K. Y. Whang. Index selection in relational databases. In *Proceedings of the International Conference on Foundations of Data Organization and Algorithms (FODO)*, páginas 487–500. 1985.

[WHMZ94] G. Weikum, C. Hasse, A. Mönkeberg, e P. Zabback. The comfort automatic tuning project. *Information Systems*, 19(5):381–423, 1994.

[Wis03] K. Wiseth. Oracle database 10g: The world's first self-managing, grid-ready database arrives. *Oracle Magazine*, September/October:28–37, 2003.

[WKKS99] G. Weikum, A. C. König, A. Kraiss, e M. Sinnwell. Towards self-tuning memory management for data servers. *IEEE Data Engineering Bulletin, Special Issue on Self-Tuning Databases and Application Tuning*, 22(2):3–11, 1999.

[WMHZ02] G. Weikum, A. Mönkeberg, C. Hasse, e P. Zabback. Self-tuning database technology and information services: from wishful thinking to viable engineering. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, páginas 20–31. 2002.

[Woo99] M. Wooldridge. *Intelligent Agents*, capítulo 1, páginas 27–78. In Weiss [Wei99], 1999.

[XMP02] X. Xu, P. Martin, e W. Powley. Configuring buffer pools in DB2 udb. In *Proceedings of the IBM Centers for Advanced Studies Conference (CASCON)*. 2002. CDROM/On-line Proceedings, 12 pp.

[YKL97] J. Yang, K. Karlapalem, e Q. Li. Algorithms for materialized view design in data warehousing environment. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, páginas 136–145. 1997.

[Zil98] D. Zilio. *Physical Database Design Decision Algorithms and Concurrent Reorganization for Parallel Database Systems*. Tese de Doutorado, Department of Computer Science, University of Toronto, 1998.

[ZY99] C. Zhang e J. Yang. Genetic algorithm for materialized view selection in data warehouse environments. In *Proceedings of the International Conference on Data Warehousing and Knowledge Discovery, Springer-Verlag, LNCS 1676*, páginas 116–125. 1999.