3
Auto-sintonia de Índices através de Agentes

Neste capítulo apresentamos nossa proposta de arquiteturas de agentes para auto-sintonia de índices em sistemas de bancos de dados relacionais. Iniciamos nossa discussão na seção 3.1 abordando mecanismos que o SGBD deve implementar para permitir a escolha automática de índices. Em seguida, apresentamos uma arquitetura com um único agente embutido no SGBD na seção 3.2. Uma arquitetura mais complexa, que envolve múltiplos agentes, é então mostrada na seção 3.3. Concluímos o capítulo na seção 3.4 apresentando alguns comentários sobre as arquiteturas propostas.

3.1
Infra-estrutura no SGBD

A implementação de agentes de auto-sintonia exige a existência de mecanismos no sistema que permitam a avaliação e a realização de ajustes. Nesta seção iremos descrever dois mecanismos que são interessantes para a implementação de agentes que escolhem índices automaticamente. O primeiro mecanismo é a possibilidade de simular configurações hipotéticas de índices no sistema. Como usamos nesta dissertação heurísticas de escolha de índices candidatos baseadas em custos do otimizador, este mecanismo é um pré-requisito. Este tipo de recurso permite estimar, através do próprio otimizador do sistema, o quanto um índice pode trazer de benefício para uma determinada consulta.

Já o segundo mecanismo é a capacidade de criar índices *on-line*, sem comprometer o processamento de transações sobre as tabelas que precisam ser indexadas. Apesar de não ser absolutamente indispensável para a escolha automática de índices, este mecanismo permite que criações de índices não prejudiquem o processamento de transações de escrita sobre as tabelas subjacentes. Sem este mecanismo, as criações de índices podem fazer com que transações de escrita passem por esperas substanciais. Isto pode

significar, por sua vez, a violação de requisitos de tempo de resposta para essas classes de transações.

Simulação de Configurações Hipotéticas de Índices

Conforme descrito na seção 2.1, a seleção do projeto físico de um banco de dados pode ser automatizada através de ferramentas baseadas em regras ou através de ferramentas que simulam configurações físicas hipotéticas e as custeiam com o otimizador. Na abordagem baseada em simulação de configurações, precisamos estender a interface do SGBD para permitir que estruturas físicas hipotéticas sejam definidas. É necessário que o servidor dê suporte a algum mecanismo que permita simular a presença de um índice na base de dados sem que seja necessário materializar fisicamente este índice. Precisamos ainda permitir que o otimizador do sistema produza planos de execução e seus custos levando em conta a existência de estruturas hipotéticas. Esta última capacidade é importante para que possamos reaproveitar o modelo de custos presente no otimizador e evitar a construção de um modelo de custo especializado na ferramenta de seleção de índices.

No trabalho de Finkelstein et al. [25], quando é realizada a seleção de índices para um determinado banco de dados, são criadas réplicas, no catálogo do SGBD, das tabelas envolvidas. Estas réplicas não possuem extensão física, mas recebem todas as informações estatísticas das tabelas originais. Para simular uma configuração de índices, índices reais são criados sobre as réplicas (que são tabelas vazias) e as estatísticas sobre os índices são artificialmente corrigidas pela ferramenta de seleção de índices. Uma desvantagem desta abordagem é o fato de que a ferramenta de seleção de índices precisa ter privilégios suficientes para modificar diretamente as informações presentes no catálogo. À medida que o sistema evolui, é necessário garantir que as manipulações realizadas pela ferramenta não produzam impactos sobre outros módulos do SGBD. O trabalho de [25] recomenda que a ferramenta de seleção de índices seja escrita pelos mesmos profissionais responsáveis pelo desenvolvimento do SGBD.

Finkelstein et al. [25] propõem, ainda, que o SGBD possua um comando para obtenção do custo e do plano de execução que o otimizador gera para uma determinada consulta. Este comando é chamado de `explain`. Para obter uma estimativa de custos de execução de uma consulta sob uma configuração hipotética, a ferramenta de seleção de índices troca as

referências a tabelas na consulta pelas suas respectivas réplicas e utiliza o comando `explain`.

O trabalho de Frank et al. [24] propõe que o otimizador de consultas do SGBD seja estendido para permitir a obtenção de um plano de execução sob a suposição de que um dado conjunto de índices hipotéticos existe na base. Um índice hipotético é um índice que não está materializado no sistema mas que pode ser reconhecido como um índice válido pelo SGBD para a estimativa de custos e planos de acesso. Frank et al. [24] não exploram, entretanto, quais alterações seriam necessárias no SGBD para dar suporte à noção de índices hipotéticos.

A adição da noção de índices hipotéticos ao sistema permite que uma ferramenta de seleção de índices faça simulações de configurações de forma mais elegante do que através da criação de réplicas de tabelas no catálogo. A criação de réplicas exige que uma ferramenta externa tenha acesso a informações críticas do catálogo sobre estatísticas de otimização e as altere diretamente. Na abordagem com índices hipotéticos, as estimativas de estatísticas para as estruturas simuladas são feitas pelo próprio código (estendido) do SGBD, evitando manipulações externas do catálogo.

O trabalho de Chaudhuri e Narasayya [11] explora em detalhe que tipo de alterações seriam necessárias para possibilitar a simulação de configurações físicas hipotéticas no sistema. De fato, as alterações propostas foram implementadas no SGBD Microsoft SQL Server [41]. É proposta uma infra-estrutura para análise de configurações hipotéticas que considera não somente índices hipotéticos como também fatores de escala sobre as tabelas. Assim, é possível simular quais índices seriam utilizados caso as tabelas do banco de dados fossem maiores do que são atualmente. Mais do que isto, é possível simular até mesmo como seria a escolha de índices caso a tabela possuísse um índice primário[1] diferente do atualmente definido.

Este tipo de simulação exige que diversas estatísticas do catálogo sobre tabelas, índices e ordenação dos dados sejam afetadas. Desta forma, Chaudhuri e Narasayya, em [11], recomendam criar um conjunto de tabelas separadas do catálogo para registrar as informações sobre índices hipotéticos, fatores de escala e estatísticas estimadas sobre índices e tabelas. Isto é feito para que, no otimizador de consultas, seja configurável a escolha de quais tabelas serão usadas para recuperar informações sobre estatísticas,

[1]Lembramos que um índice primário é um índice cujas chaves no nível folha estão dispostas fisicamente na mesma ordem que os registros correspondentes na tabela. Em muitos sistemas, o nível folha do índice primário já possui as informações dos registros da tabela, não sendo necessária um estrutura adicional para a mesma. Para uma discussão sobre o uso de índices primários e secundários, ver [53].

organização física e índices definidos. Estas podem ser as tabelas do próprio catálogo (no uso da configuração real) ou as tabelas criadas para registro de informações hipotéticas (no uso da configuração hipotética).

Em [11] foi criado um modo específico por conexão que determina como será a operação do otimizador. Cada conexão de usuário deve determinar se seus comandos serão otimizados enxergando as configurações hipotéticas (e quais delas) ou enxergando as configurações reais do sistema. A vantagem deste tipo de implementação é a de que diferentes usuários podem simular, ao mesmo tempo, distintas configurações hipotéticas para o sistema. O comando para exibição do plano de execução (`explain`) não sofre nenhuma extensão sintática, mas se torna sensível ao modo determinado para a conexão estabelecida pelo usuário.

A idéia de criar um índice, ainda que hipotético, que só seja visto por uma determinada conexão ao banco de dados não se adequa exatamente ao paradigma hoje existente para a criação de índices em sistemas relacionais. Quando um índice real é criado, sua existência se torna manifesta para todos os usuários do sistema. Assim, pode haver uma dificuldade de compreensão de como utilizar o mecanismo de criação de índices hipotéticos quando o DBA for fazer uma simulação de forma iterativa no sistema. Para contornar esta desvantagem, Chaudhuri e Narasayya [11] sugerem que a funcionalidade de criação de configurações hipotéticas seja utilizada por ferramentas de análise e seleção de índices que ofereçam uma interface mais facilmente compreensível para o DBA.

Como as possibilidades de simulações propostas em [11] são diversas, separar as informações sobre estruturas hipotéticas das informações do catálogo torna o impacto das mudanças sobre o sistema mais gerenciável. Esta, entretanto, não é a única abordagem possível de implementação. Lohman et al. [37] sugerem que o esquema do sistema de banco de dados seja temporariamente populado com definições hipotéticas de índices. Esta abordagem foi implementada no SGBD IBM DB2 UDB [34]. O trabalho propõe que a escolha de índices para uma determinada consulta seja realizada pelo próprio otimizador do sistema. Para isto, quando a consulta é submetida para otimização, é utilizada uma heurística para enumerar índices hipotéticos e estes são injetados no esquema do sistema[2]. O processo de otimização da consulta prossegue, então, normalmente. O otimizador poderá escolher um plano que utiliza um ou vários dos índices hipotéticos registrados no catálogo. Ao final da otimização os índices hipotéticos são removidos do esquema.

[2]Como o trabalho de [37] foi feito sobre um sistema comercial, não foi possível

Este trabalho também ilustra como estatísticas para os índices hipotéticos podem ser construídas, apresentando fórmulas para estimar estatísticas dos índices a partir das estatísticas existentes para as tabelas sobre as quais os índices seriam criados. É importante observar que as estatísticas necessárias variam de sistema para sistema, sendo este tipo de formulação dependente dos parâmetros necessários para o modelo de custo do otimizador do SGBD. Em alguns sistemas, como é o caso do PostgreSQL [46], apenas estatísticas sobre as colunas das tabelas são registradas. No IBM DB2 UDB, todo índice organizado como uma árvore B+ precisa de estatísticas adicionais referentes ao número de níveis e de folhas do índice, à unicidade das chaves armazenadas e ao grau de correlação[3] entre as chaves do índice e as tuplas da tabela subjacente.

Ainda assim cabe ressaltar que alguns parâmetros estatísticos que não podem ser deduzidos facilmente recebem valores pessimistas [37], de tal forma que as estimativas de custos produzidas para índices hipotéticos gerem valores potencialmente maiores do que para os índices reais correspondentes. Um destes parâmetros é a correlação do índice. Quando se assume que a ordenação da tabela e do índice não possuem correlação, sempre será feita uma estimativa de custos conservadora para o uso do índice hipotético.

Índices Hipotéticos no PostgreSQL

Nesta dissertação, implementamos extensões para simulação de índices hipotéticos no SGBD objeto-relacional PostgreSQL [46]. Para isto, foi necessário implementar um mecanismo para registro dos índices hipotéticos no catálogo e alterar o otimizador de consultas para reconher as configurações hipotéticas registradas. Estas alterações necessárias nos componentes do SGBD são descritas em detalhe na seção 4.4.

Em nossa implementação de índices hipotéticos, restringimos nossa atenção a índices completos organizados como árvores B+ (veja o Capítulo 1). O PostgreSQL permite que várias organizações de arquivo sejam utilizadas para índices. O sistema traz como padrão organizações em *hash*, árvores B+ e árvores R. A exemplo de outros sistemas objeto-relacionais, o PostgreSQL também oferece a possibilidade de criar organizações de índices definidas pelo usuário.

determinar se os índices são inseridos apenas nas estruturas compartilhadas de memória do SGBD para representação do catálogo ou se estes são de fato persistidos na metabase.

[3] A correlação, no caso, é um fator que mede o quanto as linhas da tabela estão ordenadas de acordo com as colunas do índice (*clustering*).

Restringir o uso de índices hipotéticos a árvores B+ tem pouco impacto em nosso estudo. Em princípio, não estamos interessados em avaliar a indexação de dados espaciais, o que faz com que não consideremos árvores R. Quanto a organizações *hash*, o uso de deste tipo de índice é desencorajado pelos próprios desenvolvedores do SGBD [46], uma vez que a implementação da estrutura de *hash* realizada limita gravemente o grau de concorrência que pode ser alcançado no acesso à tabela.

A restrição de uso de apenas índices completos tampouco tem grande impacto para nossas considerações. Para todo índice parcial, há um índice completo equivalente que oferece os mesmos benefícios dado o custo extra de gastar mais espaço em disco para sua materialização. Não consideramos esta desvantagem suficientemente significativa para justificar um trabalho extra de implementação em nosso protótipo.

É importante observar também que o PostgreSQL não dá suporte a índices primários. Apesar disto, é possível alterar a ordenação da tabela base através de procedimentos de exportação da tabela, reordenação utilizando utilitários externos e reimportação da tabela. O SGBD não oferece garantias quanto à manutenção da ordenação realizada quando ocorrem atualizações, uma vez que cada tabela é mantida num arquivo seqüencial simples (*heap file*). Não consideramos a possibilidade de alterar a ordenação física da tabela em nosso mecanismo de simulação de configurações hipotéticas de índices. Este tipo de capacidade de simulação poderia ser interessante para arquiteturas que tratassem do problema de projeto físico como um todo.

Propomos aqui estender a linguagem de definição de dados do SGBD para conter os seguintes novos comandos:

1. `create hypothetical index`;
2. `drop hypothetical index`;
3. `explain hypothetical`;
4. `explain noindices`.

Os primeiros dois comandos têm como finalidade possibilitar o registro e a remoção de índices hipotéticos. A exemplo de Lohman et al. [37], estas definições são armazenadas no próprio catálogo, que foi estendido para permitir diferenciar índices hipotéticos de índices reais.

O ajuste de informações estatísticas para índices hipotéticos no PostgreSQL exigiu a análise de quatro fatores: seletividade, correlação, número de tuplas e número de páginas de um índice. Os parâmetros de seletividade

e de correlação são calculados no sistema com base em informações sobre a tabela e suas colunas. Assim, não é necessário fazer nenhum ajuste para calcular estes fatores para índices hipotéticos.

Já o número de tuplas e o número de páginas variam de acordo com o índice criado. Assumimos em nossa implementação que estes fatores para índices hipotéticos serão definidos com os mesmos valores de número de tuplas e de páginas da tabela sobre a qual o índice está sendo definido. No caso do número de tuplas, esta suposição nos leva a uma estimativa exata da quantidade de linhas que realmente existiria no índice (uma vez que os índices completos no PostgreSQL são também densos [53]). Já para o número de páginas, normalmente estaremos superestimando a quantidade de páginas real do índice. Esta decisão permite que sejamos pessimistas nas estimativas realizadas com índices hipotéticos. Na seção 4.4, fazemos uma análise da qualidade das estimativas geradas com o uso de índices hipotéticos.

O terceiro comando listado acima, também adicionado na linguagem de definição de dados do sistema, é uma extensão do comando `explain` do SGBD. Ele permite que sejam obtidos planos de execução e custos de consultas levando em consideração os índices hipotéticos definidos. Assim, é possível, na sessão de um dado usuário, tanto executar comandos de acordo com as configurações reais quanto simular quais seriam as escolhas do otimizador caso as configurações hipotéticas fossem utilizadas. Nesta dissertação, índices hipotéticos criados por um usuário do SGBD são vistos pelos demais usuários, o que está de acordo com o paradigma de criação de índices reais.

O quarto comando apresentado é outra extensão do comando `explain` do SGBD. Esse comando permite que geremos um plano de execução ignorando completamente a existência dos índices (reais ou hipotéticos) definidos no sistema. Conforme veremos na seção 3.2, isto permite que estimemos o benefício decorrente da manutenção de um índice real no sistema. As propostas existentes na literatura, como [10, 37], não deixam claro como pode ser feita a avaliação do benefício de índices reais. Este tipo de avaliação é importante para permitir a recomendação de índices a serem removidos. Na implementação realizada, não avaliamos a remoção de índices. Assim, este comando não foi incluído no PostgreSQL.

Adicionamos os demais comandos para criação e uso de índices hipotéticos ao PostgreSQL. Estes comandos podem ser utilizados por um DBA (como em [11]), por uma ferramenta de seleção de índices, ou, como veremos nas próximas seções, por agentes que automatizam a escolha e a

criação de índices.

Criação On-line de Índices

Um outro mecanismo importante para a automatização da tarefa de criação de índices é o de permitir que esta operação seja feita *on-line*, ou seja, enquanto atualizações e consultas são concorrentemente processadas sobre a tabela base. Em uma arquitetura autônoma, em que as decisões sobre criação de índices não estão mais concentradas nas mãos dos administradores de bancos de dados, é benéfico que o sistema possa proceder com operações de reorganização física sem causar impactos de disponibilidade no processamento de transações concorrentes. Este tipo de técnica também se revela de grande utilidade quando é um requisito que o sistema de banco de dados esteja permanentemente disponível (em um regime 24 x 7), como por exemplo em aplicações de venda de varejo via *web*.

Diversas técnicas são descritas na literatura para a criação *on-line* de índices. Em [38], dois algoritmos são propostos para permitir a execução concorrente da construção de um índice e de inserções e remoções. No primeiro, as transações atualizam diretamente a tabela e o índice sendo construído. Ações são tomadas para permitir que as alterações feitas pelas transações, tanto na tabela quanto no índice, possam ser desfeitas utilizando o *log* do sistema (em caso de *rollbacks*). Já no segundo método, um arquivo extra é mantido para o índice em construção. Todas as operações de atualização são registradas no arquivo extra enquanto o índice é construído a partir das chaves presentes na tabela. Ao final, as operações registradas no arquivo extra são aplicadas ao índice. Enquanto isto, novas transações continuam escrevendo suas operações no arquivo extra. Quando todas as atualizações estão presentes no índice, as transações são sinalizadas para que as alterações passem a ser diretamente propagadas para o novo índice construído.

Alguns outros trabalhos, como [23] e [54], empregam uma técnica comum para a construção *on-line* de índices e a realização de outros tipos de reorganização física *on-line*. A criação do índice ocorre enquanto atualizações são realizadas na tabela base. São registradas informações de número de seqüência no *log* do sistema antes e depois da criação e, então, as modificações relevantes presentes no *log* são aplicadas sobre a nova estrutura criada até que esta esteja atualizada. Repare que novas atualizações podem ser realizadas durante esta etapa de leitura do *log*. Neste momento, atualizações na tabela são brevemente inibidas e o índice é

sincronizado com a versão corrente da tabela aplicando apenas as diferenças restantes no *log*.

Uma avaliação de desempenho de algoritmos para a criação *on-line* de índices pode ser encontrada em [55]. Os autores implementaram algoritmos que possuem o seguinte funcionamento geral: um processo de construção varre os dados da tabela, copiando as informações necessárias para as entradas do índice, de forma concorrente com escritores que modificam os mesmos dados. O sistema registra as atualizações que ocorrem durante esta varredura. Posteriormente, o processo de construção combina estas atualizações com os dados lidos na varredura original da tabela antes de registrar o índice no catálogo do sistema. Segundo o estudo, há um importante compromisso entre o tempo necessário para construir o índice e a vazão que pode ser alcançada pelas transações de atualização durante o período de construção.

Alguns sistemas comerciais já possuem facilidades para criar índices sem impedir a realização de transações concorrentes. É o caso dos SGBDs Oracle [45] e IBM DB2 [34], por exemplo. O sistema PostgreSQL não oferece facilidades para a criação *on-line* de índices. Discutiremos as conseqüências desta limitação à medida que apresentarmos nossas propostas de arquiteturas de agentes para auto-sintonia de índices.

Repare que, mesmo quando um mecanismo de criação on-line de índices está disponível no SGBD, é interessante que as operações de reorganização ocorram em momentos de baixa atividade do sistema. Ainda que não haja um impacto de disponibilidade, a reorganização física pode trazer uma degradação de desempenho inaceitável nos picos de processamento. Esta noção está relacionada ao princípio de auto-sintonia de realização de ajustes complexos *off-line* [36] e é exemplificada em [58] no contexto de realocação de dados em *arrays* de disco. A reorganização física é intencionalmente suspendida quando o nível de atividade no sistema se encontra acima de um patamar previamente determinado. Discutiremos melhor como podemos aplicar este tipo de técnica no projeto de uma arquitetura de agentes para o problema de auto-sintonia de índices na seção 3.3.

3.2 Arquitetura com um Único Agente

Em um trabalho anterior de nosso grupo de pesquisa, foi proposta uma arquitetura com um único agente para a auto-sintonia de índices em SGBDs relacionais [8]. Nesta seção, descrevemos, detalhamos e estendemos

esta arquitetura originalmente proposta. O agente proposto será aqui denominado de *Agente de Benefícios*, uma vez que utiliza uma heurística que se baseia no benefício trazido por cada índice para determinar sua materialização ou destruição. Esta dissertação leva adiante a proposta original de Costa e Lifschitz [8] através da revisão e extensão da heurística utilizada pelo agente para criação e remoção de índices, da análise de vantagens e desvantagens da arquitetura, da apresentação do projeto detalhado do agente (veja a seção 4.5.1) e da obtenção de resultados experimentais com a sua implementação (presentes na seção 4.5.2).

Modelos Local e Global de Auto-sintonia

O primeiro passo para a descrição de como o *Agente de Benefícios* deve proceder para criar e destruir índices de forma autônoma é apresentar como este agente deve ser incluído no SGBD. Como pretendemos estender a funcionalidade de SGBDs relacionais já existentes a fim de reduzir suas necessidades de ajuste, escolhemos uma arquitetura embutida para o acoplamento do agente no sistema (veja a seção 2.2). Nesta arquitetura, o sistema de agentes possui acesso direto às interfaces dos componentes do SGBD e vice-versa.

Podemos classificar o foco das iniciativas de auto-sintonia de sistemas de bancos de dados em local ou global [36]. Quando tratamos de uma iniciativa específica de sintonia, como é o nosso caso com a sintonia de índices, estamos preocupados eminentemente com o modelo local de auto-sintonia a ser empregado. A Figura 3.1 mostra este modelo para o *Agente de Benefícios*.

No modelo local, o agente interage com os componentes do SGBD referentes à sua área de ajuste. Esta interação é regida por um processo de auto-sintonia com as seguintes etapas:

- Recuperação de Informações: o agente se utiliza de sensores para receber informações e métricas dos componentes do SGBD.
- Avaliação de Métricas: a partir das informações coletadas, o agente estabelece crenças que dão suporte aos algoritmos utilizados para decidir se alguma ação de sintonia deve ser realizada.
- Enumeração de Alterações Possíveis: o agente aplica algoritmos ou heurísticas para enumerar alternativas de ajustes que podem levar a um melhor desempenho do sistema. Durante esta enumeração, o agente pode se utilizar de efetuadores [66] para simular cenários

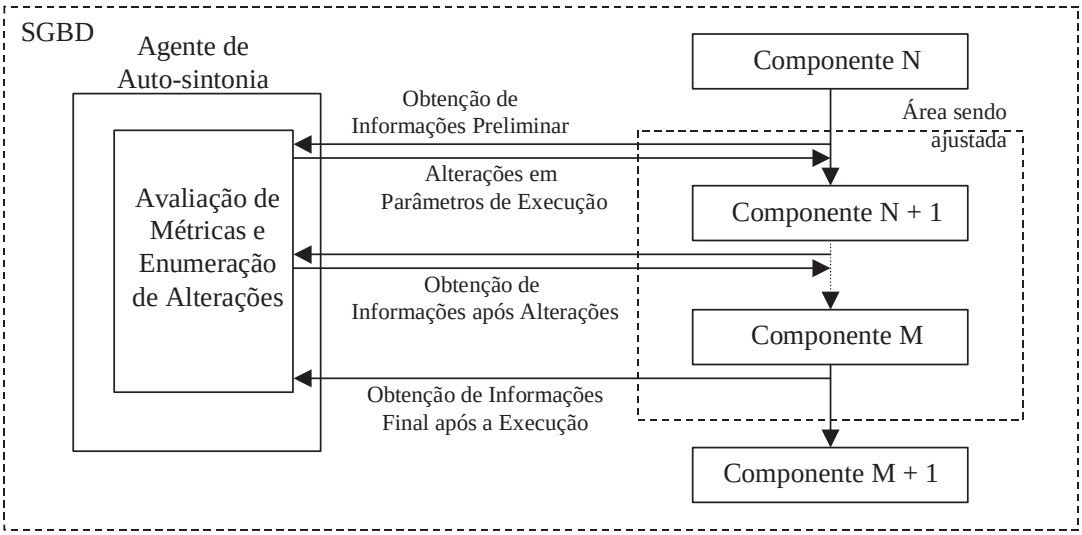

Figura 3.1: Modelo local para auto-sintonia usando agentes

utilizando mecanismos previamente implementados nos componentes do SGBD.

– Realização de Alterações: os ajustes escolhidos pelos algoritmos ou heurísticas empregados pelo agente são aplicados aos componentes do sistema através do uso de efetuadores.

No caso do *Agente de Benefícios*, a etapa de *Recuperação de Informações* é realizada através da coleta dos comandos SQL processados pelo otimizador. Em seguida, ocorre a *Avaliação de Métricas* com a atualização de crenças necessárias para a aplicação da heurística de escolha de candidatos e da heurística de benefícios (veja a seguir). O agente então passa pela *Enumeração de Alternativas Possíveis*. Nesta etapa, são criados índices hipotéticos no sistema e o agente utiliza o otimizador para obter planos e custos estimados para a execução de comandos com estas configurações físicas simuladas. Por fim, é conduzida a *Realização de Alterações*, que é implementada pela criação ou remoção de índices reais do sistema de bancos de dados.

O processo de auto-sintonia apresentado se baseia em um *ciclo de controle de realimentação* para refinar progressivamente as decisões de sintonia tomadas com o conhecimento das métricas locais obtidas a partir dos componentes do SGBD. O uso deste tipo de abordagem para processos de auto-sintonia é comum na literatura, conforme é detalhado em [36].

É sabido que os componentes de um SGBD possuem inter-relações que podem afetar o desempenho do sistema como um todo [59]. Por

exemplo, a criação de índices pode resolver problemas de bloqueios através da restrição do conjunto de linhas que precisam ser bloqueadas pelo SGBD em operações de atualização. Assim, apesar de o agente proposto possuir um foco local, é interessante que ele possua a capacidade de colaborar com outros agentes quando incluído em uma arquitetura global para a auto-sintonia do sistema. A Figura 3.2 mostra uma possibilidade para este modelo global de cooperação entre agentes.

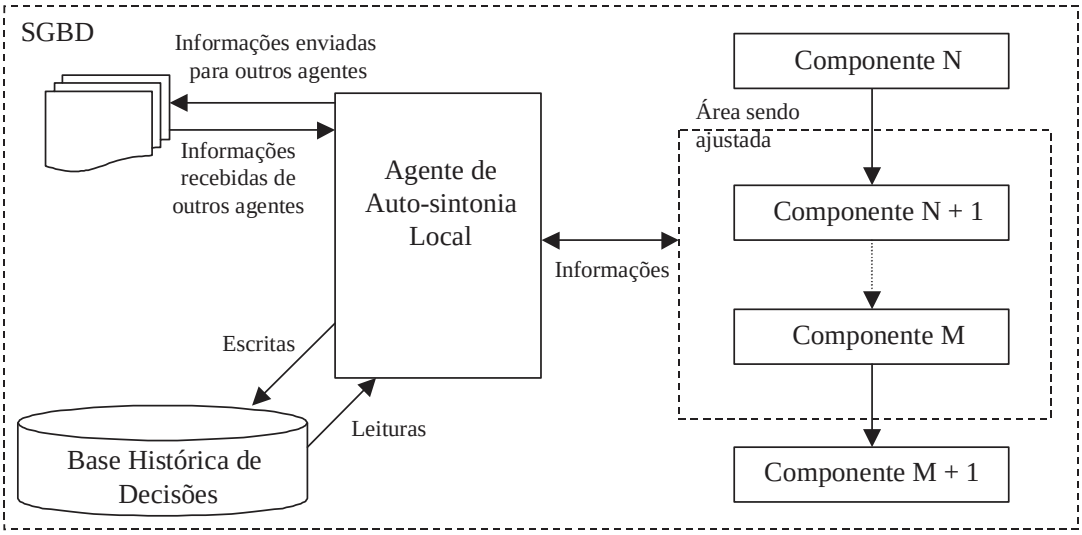

Figura 3.2: Modelo global para auto-sintonia usando agentes

No modelo global, cada agente inserido no sistema é responsável pela sintonia local de um determinado conjunto de componentes do SGBD. Antes da etapa de *Realização de Alterações*, cada agente faz um intercâmbio de informações com os demais agentes presentes no sistema. Esta colaboração visa determinar se as ações que um agente irá efetuar sobre o sistema podem ter algum impacto negativo sobre o seu desempenho global. Este impacto é avaliado tanto através do conhecimento de sintonia específico embutido em cada agente especializado quanto através da comparação dos ajustes sendo realizados com uma base histórica de decisões anteriormente realizadas. Este tipo de organização supõe que os agentes deverão chegar a um consenso sobre quais ajustes devem ser realizados e quais devem ser desprezados com o uso de alguma estratégia de negociação [31].

O detalhamento de uma arquitetura global para auto-sintonia de sistemas de bancos de dados utilizando agentes pode ser encontrado em [42]. Este trabalho de pesquisa foi desenvolvido em paralelo com esta dissertação. Aqui, iremos restringir nosso enfoque ao modelo local para a auto-sintonia de índices no sistema.

Heurística de Benefícios

As ferramentas de seleção de índices propostas na literatura [36] dão apoio a um profissional especializado através da sugestão de um conjunto de índices adequados a uma carga de trabalho dada como entrada. O uso de um agente para auto-sintonia de índices visa permitir a obtenção automática da carga de trabalho e possibitar que as decisões de criação ou remoção de índices sejam tomadas sem intervenção humana. Para alcançar este grau de autonomia, o agente deve possuir heurísticas adequadas de decisão.

Propomos que as heurísticas embutidas no agente sigam a metodologia apresentada na seção 2.1. Teremos uma heurística de escolha de índices candidatos, aplicada para cada comando SQL individualmente, e uma heurística de seleção final de índices que procura obter um boa relação custo-benefício para a carga de trabalho como um todo.

Devemos notar que, nas propostas existentes na literatura [36], as heurísticas para seleção de índices são tipicamente executadas *off-line*, podendo ser processadas até mesmo em um servidor distinto de onde o SGBD está instalado. Já no caso de um agente embutido no SGBD, as heurísticas de escolha de candidatos e de seleção final de índices devem ser executadas juntamente com outros serviços providos pelo SGBD. Para evitar uma penalidade de desempenho sobre estes outros serviços, temos duas opções. A primeira é agendar a execução das heurísticas para momentos "quietos" no sistema. Isto exigiria que o agente fosse dotado de algum mecanismo de manutenção do histórico de comandos SQL submetidos ao SGBD para processamento posterior por parte das heurísticas. A segunda opção, por nós escolhida nesta arquitetura, é limitar o espaço de busca percorrido pelas heurísticas de forma a manter sob controle o seu tempo de execução e uso de recursos.

Utilizamos no agente uma heurística para escolha de índices candidatos adaptada de Lohman et al. [37]. Para um detalhamento da heurística utilizada na implementação e considerações sobre a sua escolha, veja a seção 4.5.1.

Já para a seleção final de índices, apresentamos nesta seção a *heurística de benefícios*. Esta heurística procura decidir *on-line*, à medida que o sistema recebe novos comandos SQL, quais índices devem ser criados ou removidos para tornar o processamento da carga de trabalho como um todo mais eficiente. Esta característica é inspirada na heurística proposta no trabalho anterior de Costa e Lifschitz [8]. Nesta dissertação revisamos e estendemos a abordagem proposta neste trabalho. Em primeiro lugar, a heurística aqui

apresentada computa explicitamente o benefício trazido por índices reais, o que não era realizado em [8]. Isto permite que tomemos decisões não somente de criação de índices candidatos, mas também de remoção de índices reais previamente existentes. Além disto, apresentamos aqui equações revisadas para cálculo dos benefícios trazidos pelos índices candidatos enumerados.

Para apresentarmos a heurística de benefícios, precisamos definir os seguintes fatores:

1. C_R: o custo, gerado pelo otimizador, do melhor plano de execução sobre a configuração de índices reais.

2. C_H: o custo, gerado pelo otimizador, do melhor plano de execução sobre a configuração de índices hipotéticos e reais. Usualmente, os índices hipotéticos escolhidos para uso neste plano são sugeridos pela heurística de escolha de candidatos como os índices candidatos para o comando[4].

3. C_N: o custo, gerado pelo otimizador, do melhor plano sobre uma configuração física que não contém qualquer índice (nem real, nem hipotético).

4. C_A: o custo estimado de atualização de um índice durante o processamento de um comando de atualização. Normalmente, este custo não é calculado pelo otimizador do sistema e deve ser estimado em sintonia com o modelo de custos empregado no SGBD.

5. B_K: o benefício do índice K para o comando SQL sendo atualmente analisado. O benefício será calculado pela heurística de forma distinta para índices reais e hipotéticos.

6. BAc_K: o benefício acumulado do índice K para todos os comandos já processados. Novamente, a heurística atualizará o benefício acumulado de acordo com o fato de o índice ser real ou hipotético.

7. CC_K: o custo estimado de criação do índice K. Normalmente, este custo não é calculado pelo otimizador do SGBD. Assim, foi também necessário estimá-lo em nossa implementação.

Quando uma nova operação é submetida ao SGBD, o otimizador gera o melhor plano de acesso dada a configuração real de índices existente. Neste

[4]Nesta dissertação, consideramos que um conjunto de índices hipotéticos são enumerados pela heurística de escolha de índices candidatos. Os índices candidatos são um subconjunto selecionado dos índices hipotéticos enumerados pela heurística.

momento, conforme descrito na próxima subseção, o agente é notificado sobre o novo comando submetido e aplicará a heurística de escolha de candidatos para obter, interagindo com o otimizador, o melhor plano de execução dada uma configuração de índices hipotéticos e reais. Uma vez determinados os candidatos para o comando, executamos um passo adicional de obtenção, novamente a partir de interação com o otimizador, do custo de processamento do comando sobre uma configuração em que não há índices definidos. Após estes procedimentos, temos os fatores C_R, C_H e C_N calculados para o comando. É, então, invocada a *heurística de benefícios* mostrada nas Figuras 3.3 e 3.4.

Se o comando submetido for uma consulta, aplicaremos o procedimento mostrado na Figura 3.3. Para cada índice candidato à criação, iremos calcular o seu benefício e atualizar o seu benefício acumulado. O benefício é a diferença entre o custo da consulta com a configuração de índices reais e o seu custo usando uma configuração de índices reais e hipotéticos. É importante notar que, como estes custos são obtidos sempre para o *melhor plano de execução* escolhido pelo otimizador, o benefício calculado para o índice candidato neste passo é positivo ou nulo.

Avaliação de Consultas

Para cada índice candidato IC da consulta faça
$B_{IC} = C_R - C_H$;
$BAc_{IC} = BAc_{IC} + B_{IC}$;
Se $BAc_{IC} > CC_{IC}$ *então*
Criar o índice
$BAc_{IC} = 0$;
Fim se ;
Fim para ;

Para cada índice real IR da consulta faça
$B_{IR} = C_N - C_R$;
$BAc_{IR} = BAc_{IR} + B_{IR}$;
Fim para ;

Figura 3.3: Estratégia de avaliação de consultas

Se vários índices candidatos são usados no plano de execução da

consulta, de certa forma estamos fazendo uma contagem duplicada de benefícios para índices distintos. Esta abordagem já foi empregada, por exemplo, em [37]. Aproximamos o benefício de cada índice como o benefício total trazido pelo uso da configuração hipotética. É difícil obter uma regra intuitiva para dividir o benefício entre os índices utilizados. Se o otimizador escolheu o uso simultâneo de vários índices, acreditamos que devemos recompensar estes índices igualmente em nossa heurística. Observe que tendemos a incorrer neste tipo de aproximação apenas em consultas mais complexas, em que o uso de diversos índices traz melhor desempenho.

Após o cálculo de benefício, iremos materializar o índice somente se o seu benefício acumulado tiver superado o seu custo estimado de criação. Nosso objetivo é criar índices cujo uso repetido se prove interessante o suficiente para incorrermos no seu custo de criação. Repare que uma suposição implícita da heurística é a de que índices necessários para consultas passadas continuarão a ser interessantes para as consultas a serem processadas pelo SGBD no futuro. Assim, o agente tende a encontrar bons projetos de índices para cargas de trabalho relativamente estáveis.

Para cada índice real, a heurística calcula o benefício como sendo a diferença entre o custo de processar o comando em uma configuração em que não há índices definidos e o custo na configuração de índices reais. Repare que novamente este benefício deve ser positivo ou nulo. Como o comando submetido é uma consulta, apenas atualizamos o benefício acumulado para o índice real.

Se o comando submetido ao SGBD for uma atualização, aplicaremos o procedimento mostrado na Figura 3.4. Este procedimento se inicia com uma avaliação idêntica à realizada para consultas. É interessante observar que comandos de atualização devem inicialmente trazer os dados para a memória, exatamente como consultas, para somente então processar modificações sobre os dados. Normalmente, os otimizadores de SGBDs relacionais oferecem estimativas de custos para recuperar os dados necessários à atualização, mas não para os custos envolvidos na modificação propriamente dita dos dados.

Após a aplicação do mesmo procedimento para avaliação de consultas, iremos contabilizar os custos de manutenção de índices em que incorreríamos para o processamento do comando. Estes custos de manutenção serão debitados do benefício acumulado tanto de índices candidatos quanto de índices reais. Para estimativa do custo de manutenção de índices, introduzimos o fator C_A. Em princípio, supomos que o custo de atualizar qualquer índice afetado pelo comando é idêntico e igual a C_A. Se em algum sistema específico houver uma grande variação nos custos de manutenção de diferentes tipos

Avaliação de Atualizações

Executar "Avaliação de Consultas"

Para cada índice I afetado pela atualização faça

$BAc_I = BAc_I - C_A$;

Se (*I é real*) *e* ($BAc_I < 0$) *e* ($|BAc_I| > CC_I$) *então*

$BAc_I = 0$;

Remover o índice

Fim se ;

Fim para ;

Figura 3.4: Estratégia de avaliação de atualizações

de índices, podemos tornar o fator C_A uma função do índice afetado pelo comando.

Após o cálculo do novo valor para o benefício acumulado de um índice real, verificamos se a sua manutenção na base permanece vantajosa. Caso o benefício trazido pelo índice seja negativo e superior, em módulo, ao custo estimado de criação do índice, o mesmo é removido. Novamente, supomos que índices prejudiciais para comandos processados no passado continuarão sendo prejudiciais ao processamento de comandos pelo SGBD no futuro.

Em relação a outras heurísticas presentes na literatura, a heurística de benefícios se destaca por ter um comportamento *on-line*, permitindo a criação e destruição de índices com maior reatividade e sem intervenção de um administrador. Destaca-se, ainda, pela consideração explícita dos custos associados à criação dos índices no procedimento de decisão.

Arquitetura para Construção do Agente

Para especificarmos a arquitetura do *Agente de Benefícios*, utilizaremos o *framework* para construção de agentes em camadas descrito na seção 2.2. Nesta seção, descreveremos como cada camada será estendida para construirmos um agente para a auto-sintonia de índices.

A Figura 3.5 mostra o mapeamento proposto entre as camadas do *framework* de construção de agentes de Kendall et al. [35] e nosso modelo local de auto-sintonia. Mostramos, ainda, as funcionalidades que devem ser

implementadas em cada camada para que instanciemos o *framework* para a construção do *Agente de Benefícios*.

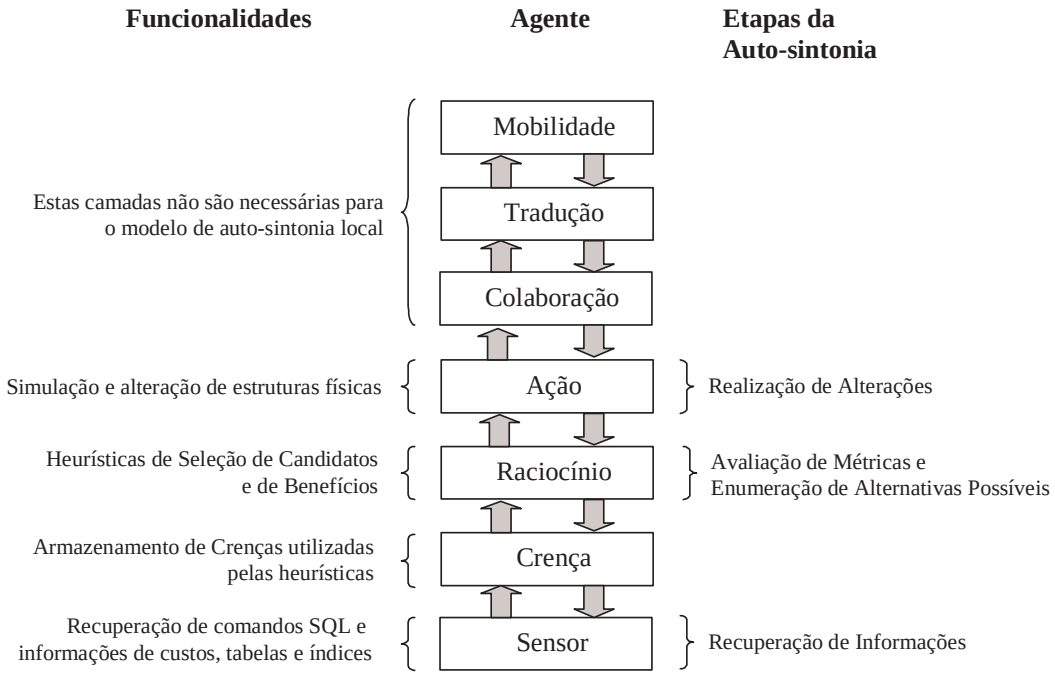

Figura 3.5: Instanciação do *framework* de construção de agentes para auto-sintonia de índices

As funcionalidades necessárias para cada camada são:

- *Sensor*: é responsável pela recuperação de informações provenientes dos módulos do SGBD. O agente deve receber cada comando SQL processado pelo otimizador bem como suas informações de custos, tabelas e índices reais. Para isto, o SGBD deve ser instrumentado para que estas informações sejam enviadas após o passo de otimização e antes do passo de execução do comando. A alteração do código do SGBD permite que o sensor siga um modelo de notificação, o que representa um menor *overhead* em relação a alternativa de coleta (*polling*). Também é interessante notar que uma padronização no formato de envio das informações pode facilitar embutir o agente em diferentes SGBDs.
- *Crença*: armazena informações sobre tabelas, índices reais e índices candidatos necessárias para o cálculo das heurísticas implementadas na camada de *Raciocínio*.
- *Raciocínio*: é responsável pela avaliação das métricas baseadas nas informações recuperadas pela camada *Sensor* e pela enumeração de

alternativas possíveis de ação do agente. O *Agente de Benefícios* aplica duas heurísticas nesta camada. A primeira objetiva escolher índices candidatos para o comando SQL vindo do sensor do agente. Durante a execução desta seleção, a camada de *Raciocínio* simula índices hipotéticos recorrendo à camada de *Ação*. Após a execução da heurística de escolha, os candidatos são armazenados (ou atualizados) na camada de *Crença*. Para cada candidato, é então necessário aplicar a *Heurística de Benefícios*, descrita anteriormente, que decide se o índice candidato deverá ou não ser criado neste momento. Também através da *Heurística de Benefícios* é avaliado se existe algum índice real que deve ser removido.

– *Ação*: é responsável por realizar as alterações escolhidas pela camada de *Raciocínio* do agente. Deverá oferecer facilidades para a simulação de índices hipotéticos e para a criação e remoção de índices reais. Como a implementação destas funcionalidades exige interação direta com o SGBD, os componentes desta camada deverão acionar efetuadores implementados especificamente para o SGBD utilizado.

As camadas *Colaboração*, *Tradução*, e *Mobilidade* não são necessárias para a construção de um agente que segue o modelo local de auto-sintonia desta dissertação. A utilidade e as funcionalidades destas camadas em um modelo global de auto-sintonia, em que existe comunicação e colaboração entre agentes, é debatida por Milanés em [42].

Repare que, para o *Agente de Benefícios*, as camadas *Crença*, *Raciocínio* e *Ação* utilizam os fluxos de informações tanto *top-down* quanto *bottom-up* na arquitetura do agente. Já a camada *Sensor* somente envia informações no sentido *bottom-up* na arquitetura, para a camada de *Crença*. Na seção 4.5.1, iremos mostrar o projeto detalhado de cada uma destas camadas, especificando as classes, interações e padrões de projeto empregados.

Vantagens e Desvantagens da Arquitetura com um Único Agente

A arquitetura com um único agente proposta traz as seguintes vantagens:

– em relação a trabalhos anteriores da literatura, são eliminados todos os passos que exigiam intervenção humana para a seleção de índices. O agente se ocupa de monitorar o SGBD para derivar a carga de trabalho e, no momento escolhido pela heurística de benefícios, executa a

criação ou remoção de índices no sistema. Assim, podemos tornar o SGBD autônomo em relação a esta atividade.

– a escolha de índices candidatos para cada comando SQL pode ser implementada utilizando simplificações de heurísticas já existentes na literatura. Isto permite que reaproveitemos o trabalho já realizado sobre heurísticas de seleção de índices.

– o comportamento *on-line* da heurística de benefícios proposta para seleção final dos índices permite que façamos a escolha de quais índices devem ser criados ou removidos sem a necessidade de manter um histórico dos comandos executados no SGBD. Isto diminui significativamente os custos de observação do sistema em relação a métodos mais tradicionais de geração de arquivos de *trace*.

Por outro lado, temos algumas desvantagens no uso desta arquitetura:

– em um sistema que não oferece a possibilidade de criar índices sem interromper o acesso à tabela, é necessário tomar cuidados extras com a realização desta operação. Em [57], várias considerações de administração de banco de dados são levantadas. Entre elas, se destacam a definição de quando a operação deve ser realizada e de qual é o custo percebido em termos de desempenho e de disponibilidade. Os custos de desempenho podem existir mesmo quando o sistema oferece mecanismos para criação *on-line* de índices. No caso do *Agente de Benefícios*, não existe um mecanismo que permita que a criação ou remoção de índices seja agendada para execução em um momento mais conveniente.

– não conseguimos reaproveitar heurísticas desenvolvidas na literatura para a seleção final de índices, que leva a carga de trabalho como um todo em consideração. Estas heurísticas esperam como entrada todo o conjunto de comandos SQL que estão sendo submetidos ao SGBD e não têm um comportamento *on-line* como o da heurística de benefícios.

– a heurística de benefícios traz uma suposição implícita de que as cargas de trabalho que o SGBD processa são relativamente estáveis. Se tivermos cargas de trabalho com perfis distintos submetidas sazonalmente ao SGBD, o agente precisará passar constantemente por períodos de aprendizagem e adaptação dos índices para os perfis da carga. Por exemplo, tomemos um sistema em que há apenas consultas durante o dia e uma atividade forte de atualização durante a noite. Neste tipo

de sistema, o agente tenderia a remover índices após algum tempo do processamento noturno para então voltar a criá-los após algum tempo do processamento diurno. Isto levaria a um desempenho inferior em relação a uma abordagem em que fosse possível antever estes ciclos repetitivos na carga de trabalho e adaptar os índices preventivamente antes do início de cada ciclo.

Com base nas desvantagens identificadas acima, propomos nesta dissertação, na próxima seção, uma arquitetura para auto-sintonia de índices em SGBDs relacionais que se utiliza de múltiplos agentes. Apesar de endereçar as desvantagens da arquitetura com um único agente, esta nova arquitetura apresenta maior consumo de recursos e dificuldade de implementação. O uso de agentes, com suas propriedades de autonomia e reatividade, se mostra importante para a separação de responsabilidades entre os componentes utilizados.

3.3 Arquitetura com Múltiplos Agentes

Nesta seção, apresentamos uma arquitetura para auto-sintonia de índices em SGBDs relacionais que se utiliza de múltiplos agentes. Estes agentes assumem responsabilidades específicas e cooperam com o propósito de melhorar o desempenho do processamento da carga de trabalho, modificando a organização física do sistema. Procuramos endereçar os pontos fracos levantados sobre a arquitetura com um único agente.

Podemos ver a arquitetura proposta na Figura 3.6. Seguimos o modelo local de auto-sintonia apresentado na seção 3.2, desta vez adaptado para o uso de múltiplos agentes. Conforme já discutido, para a auto-sintonia de índices a etapa de *Recuperação de Informações* envolve a obtenção, a partir do SGBD, dos comandos SQL processados pelo otimizador juntamente com seus custos estimados de execução.

Estes comandos serão enviados para um *Agente de Coleta de Comandos*, responsável por montar uma imagem da carga de trabalho que é submetida ao sistema. Periodicamente, esse agente colaborará com o *Agente de Seleção de Índices*. Este é responsável por aplicar heurísticas de escolha de candidatos e de seleção final de índices sobre a carga de trabalho dada. Assim, está associado fortemente às etapas de auto-sintonia de *Avaliação de Métricas* e de *Enumeração de Alterações Possíveis*.

Por fim, o *Agente de Execução de Sintonia* será responsável por de fato realizar a criação ou remoção de índices no sistema. Estas alterações

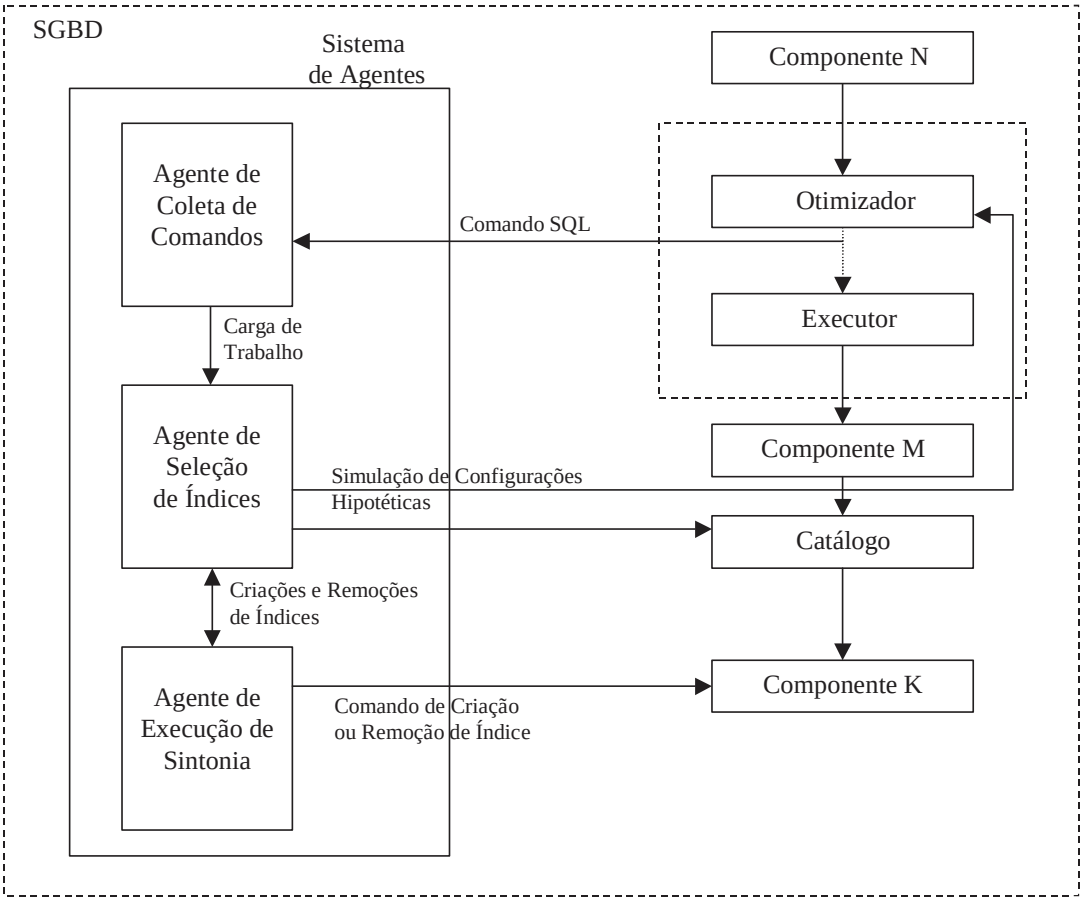

Figura 3.6: Arquitetura com múltiplos agentes

somente serão executadas pelo agente quando este julgar que o sistema se encontra em um estado conveniente para processá-las. Este agente está associado à etapa de *Realização de Alterações* do processo de auto-sintonia.

Nas próximas subseções iremos apresentar em maior detalhe cada um dos agentes que compõem a arquitetura. Após debatermos as questões envolvidas na construção destes agentes, discutiremos as vantagens e desvantagens do uso da arquitetura com múltiplos agentes.

Arquitetura para Construção do Agente de Coleta de Comandos

Para especificarmos a arquitetura do *Agente de Coleta de Comandos*, utilizaremos o *framework* para construção de agentes em camadas descrito na seção 2.2. Nesta seção, descreveremos como cada camada será estendida para construirmos um agente para a obtenção da carga de trabalho.

A Figura 3.7 mostra as funcionalidades que devem ser implementadas em cada camada para que instanciemos o *framework* de Kendall et al. [35]

para a construção do *Agente de Coleta de Comandos*.

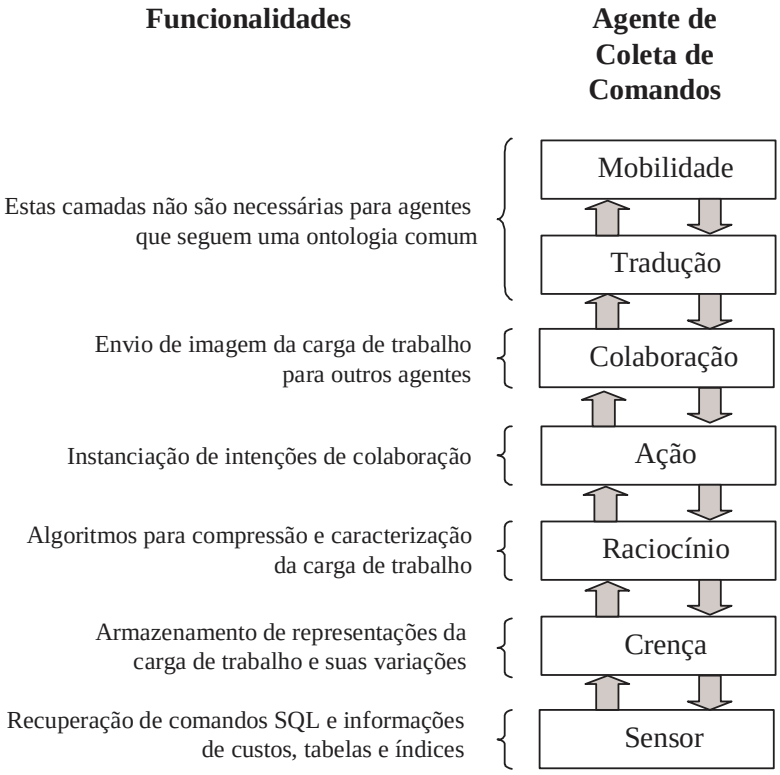

Figura 3.7: Instanciação do *framework* para construção do *Agente de Coleta de Comandos*

As funcionalidades necessárias para cada camada são:

- *Sensor*: é responsável por recuperar os comandos SQL e suas estatísticas de otimização. Assim como no *Agente de Benefícios*, podemos instrumentar o SGBD para que o agente seja notificado após o passo de otimização e antes do passo de execução.
- *Crença*: armazena representações da carga de trabalho e de seu comportamento cíclico. Devemos ser capazes de diferenciar os perfis da carga de trabalho submetida (transacional, suporte a decisão) e as periodicidades da submissão destes perfis (horário do dia, dia da semana, dia do mês).
- *Raciocínio*: implementa algoritmos para a caracterização de perfis da carga de trabalho e para a determinação de seu comportamento sazonal. Já existem trabalhos na literatura que estudam estes temas, como [22] e [32]. Entretanto, cabe observar que a caracterização automática da carga de trabalho é uma tarefa de alta complexidade e a intervenção de um administrador nesta tarefa pode ser necessária.

Esta intervenção poderia ser implementada através da criação de um arquivo de configuração para o agente que informasse características dos perfis da carga de trabalho processada pelo SGBD.

– *Ação*: permite que intenções de colaboração com outros agentes sejam instanciadas. Outros agentes recorrerão ao *Agente de Coleta de Comandos* para obter informações sobre a carga de trabalho que está sendo submetida ao sistema.

– *Colaboração*: envia mensagens para outros agentes contendo uma imagem da carga de trabalho que está sendo processada pelo sistema. Esta imagem é derivada de acordo com a sazonalidade e os perfis de carga de trabalho que são tratados pelo agente.

As camadas *Tradução* e *Mobilidade* não são necessárias para o agente, uma vez que a sua comunicação deve ocorrer com outros agentes possuidores de uma mesma ontologia e dentro de uma mesma sociedade [35].

Como a responsabilidade do *Agente de Coleta de Comandos* é a de obter uma imagem da carga de trabalho processada pelo sistema, sua aplicação não se restringe à auto-sintonia de índices. Este componente poderia ser reaproveitado por outros componentes de auto-sintonia local inseridos em uma arquitetura global como a de Milanés [42].

Arquitetura para Construção do Agente de Seleção de Índices

A Figura 3.8 mostra as funcionalidades que devem ser implementadas em cada camada para que instanciemos o *framework* de Kendall et al. [35] para a construção do *Agente de Seleção de Índices*.

As funcionalidades necessárias para cada camada são:

– *Crença*: armazena informações sobre os comandos SQL da carga de trabalho, os índices candidatos escolhidos para cada comando e seus benefícios estimados. Também pode armazenar outras informações que sejam relevantes para as heurísticas implementadas na camada de *Raciocínio*.

– *Raciocínio*: implementa uma heurística para escolha de índices candidatos e uma heurística para seleção final de índices. Uma apresentação de como trabalhos da literatura utilizam estes tipos de heurísticas pode ser encontrada na seção 2.1. Como este agente já recebe uma imagem integral da carga de trabalho, heurísticas previamente propostas na literatura podem ser facilmente aplicadas.

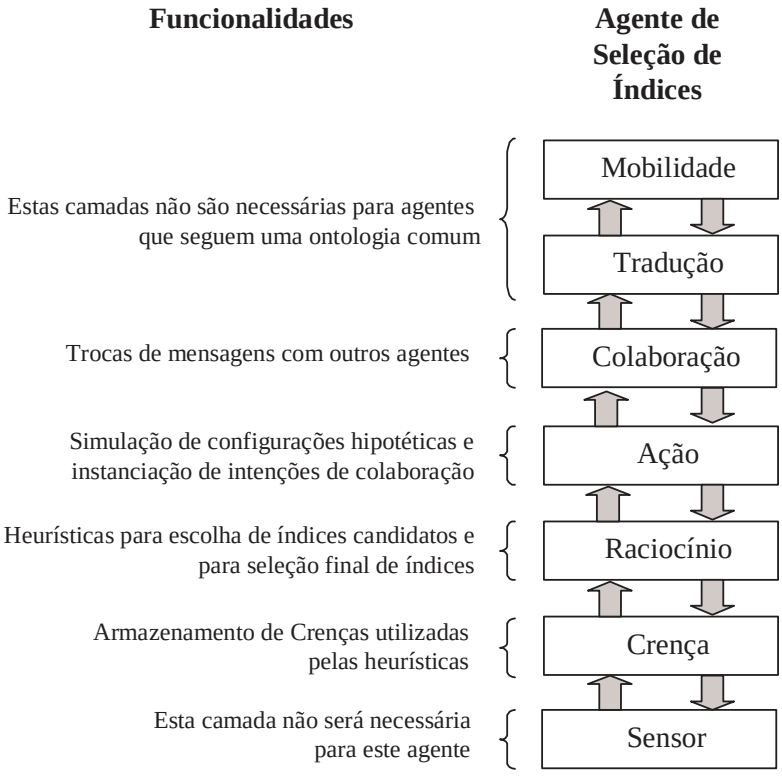

Figura 3.8: Instanciação do *framework* para construção do *Agente de Seleção de Índices*

- *Ação*: realiza a simulação de configurações hipotéticas de índices utilizando facilidades previamente incorporadas ao SGBD (veja a seção 3.1). Estas simulações são requisitadas pelas heurísticas da camada de *Raciocínio* do agente. Também devemos tratar aqui a instanciação de intenções de colaboração com outros agentes.
- *Colaboração*: permite que o *Agente de Seleção de Índices* receba a carga de trabalho sendo processada pelo SGBD do *Agente de Coleta de Comandos* e encaminhe sugestões de criações e remoções de índices para o *Agente de Execução de Sintonia*.

Novamente, as camadas *Tradução* e *Mobilidade* não são necessárias para o agente.

Neste agente, em especial, não é implementada a camada *Sensor*, uma vez que a interação do agente com o SGBD se restringe à simulação de configurações de índices hipotéticos. O agente não precisa interagir com o ambiente do SGBD para recuperar informações sobre o andamento de operações reais do sistema.

Arquitetura para Construção do Agente de Execução de Sintonia

Seguindo a abordagem das seções anteriores, mostramos, na Figura 3.9 as funcionalidades que devem ser implementadas em cada camada para que instanciemos o *framework* de Kendall et al. [35] para a construção do *Agente de Execução de Sintonia*.

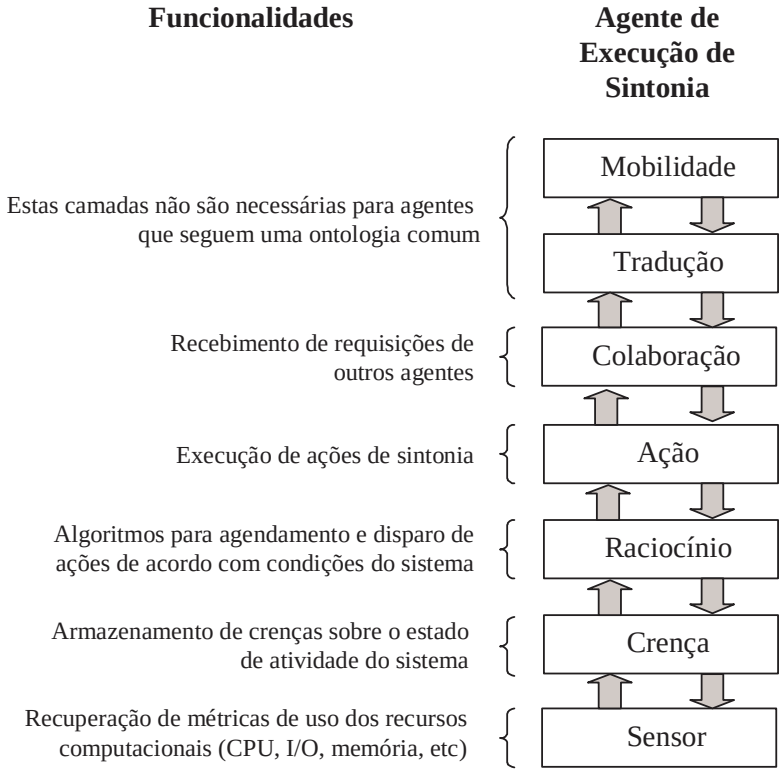

Figura 3.9: Instanciação do *framework* para construção do *Agente de Execução de Sintonia*

As funcionalidades necessárias para cada camada são:

- *Sensor*: recupera informações sobre o nível de atividade do sistema. Estas informações podem ser provenientes do SGBD (por exemplo, taxa de geração de *logs* por segundo) ou do sistema operacional (por exemplo, percentual de CPU ocupado ou I/O por dispositivo). A implementação da coleta de informações tanto do SGBD quanto do seu ambiente externo é um dos diferenciais do uso de agentes em SGBDs [40].
- *Crença*: armazena informações sobre o estado atual de atividade do sistema. Estas informações são usadas pelo agente para decidir quando levar a cabo ações de sintonia. Por exemplo, podemos adiar uma ação

de criação de um índice caso seja detectado que há uma sobrecarga de I/O nos dispositivos em que o índice seria materializado.

- *Raciocínio*: implementa algoritmos para agendamento e disparo de execução de tarefas. O agendamento é necessário pois outros agentes podem requisitar que a execução de ações de sintonia seja diferida. Já o mecanismo de disparo deve levar em conta se o nível de atividade atual do sistema permite a execução da ação de sintonia. Em situações de carga intensa, executar ações de sintonia pode agravar ainda mais o problema de desempenho por que passa o SGBD. Assim, o agente poderia apenas registrar a necessidade de execução da ação e disparála no momento em que a condição adversa de desempenho não mais exista. Este tipo de suspensão na execução de ações pode ser observado em outros trabalhos de auto-sintonia, como por exemplo [58].
- *Ação*: executa as ações de sintonia determinadas por outros agentes. As ações somente serão disparadas no momento determinado pela camada de *Raciocínio*.
- *Colaboração*: recebe requisições de execução de ações de sintonia de outros agentes e envia mensagens indicando se as ações foram realizadas com sucesso.

Mais uma vez e pelos mesmos motivos expostos anteriormente, as camadas *Tradução* e *Mobilidade* não são necessárias para o agente.

A exemplo do *Agente de Coleta de Comandos*, a utilidade do agente apresentado nesta seção tampouco se restringe apenas à auto-sintonia de índices. Este componente poderia ser reaproveitado em uma arquitetura de auto-sintonia como a proposta em [42].

Vantagens e Desvantagens da Arquitetura com Múltiplos Agentes

A arquitetura com múltiplos agentes possui as seguintes vantagens:

- em relação a trabalhos anteriores da literatura, são eliminados todos os passos que exigiam intervenção humana para a seleção de índices. Esta vantagem também já era verificada na arquitetura com um único agente.
- a arquitetura permite que a criação ou remoção de índices seja agendada para momentos de menor atividade no sistema. Isto facilita o seu uso em SGBDs que não oferecem facilidades para criação de índices *on-line*, uma vez que podemos realizar a materialização de um

índice apenas quando o impacto de disponibilidade e de desempenho sobre o sistema forem aceitáveis. Repare que isto resolve uma das desvantagens apresentadas para a arquitetura com um único agente.

- podemos reaproveitar o trabalho existente na literatura tanto sobre heurísticas para escolha de candidatos como para escolha final de índices. Isto é possível pois conseguimos fornecer como entrada para estas heurísticas imagens da carga de trabalho processada pelo sistema. Em relação a arquitetura com um único agente, temos a possibilidade de reaproveitar mais trabalho anteriormente desenvolvido na literatura.
- a arquitetura consegue se adaptar melhor a cargas de trabalho com perfis sazonais. Como o *Agente de Coleta de Comandos* consegue identificar os perfis e periodicidades existentes na carga de trabalho, este pode colaborar preventivamente com o *Agente de Seleção de Índices* e com o *Agente de Execução de Sintonia* para que a criação ou remoção de índices ocorra sempre antes da mudança de perfil da carga de trabalho. Este tipo de comportamento endereçaria melhor o exemplo dado anteriormente de um sistema em que há apenas consultas durante o dia e uma atividade forte de atualização no período noturno. Isto resolve outra das desvantagens identificada para a arquitetura com um único agente.

Temos, entretanto, algumas desvantagens na aplicação desta arquitetura:

- em relação a arquitetura com um único agente, o uso de três agentes que colaboram através da trocas de mensagens implica um maior consumo de recursos do sistema para conseguir realizar as atividades de auto-sintonia de índices.
- como o *Agente de Coleta de Comandos* precisa criar uma imagem da carga de trabalho, sua implementação importará em maior custo para monitoramento do sistema e manutenção do histórico, ainda que parcial, dos comandos processados. Isto não era necessário na arquitetura com um único agente.
- se não conseguirmos uma completa automatização da identificação de perfis e sazonalidades da carga de trabalho no *Agente de Coleta de Comandos*, pode ser necessário criar uma configuração do agente acessível pelo administrador do sistema. Este tipo de configuração é indesejável pois exige intervenção humana, ainda que em um nível de

abstração maior do que a realizada atualmente sobre os sistemas para a sua sintonia.

Também é importante notar que a arquitetura é de maior complexidade de implementação do que a arquitetura com um único agente proposta na seção 3.2. Não consideramos isto propriamente uma desvantagem, uma vez que esta dificuldade pode ser completamente contornada com a implementação da arquitetura.

3.4 Comentários Finais

Neste capítulo, inicialmente discutimos quais funcionalidades deveriam ser implementadas em SGBDs para a realização de auto-sintonia de índices. Mostramos a importância de existir um mecanismo para simulação de configurações hipotéticas de índices e recomendamos a existência da capacidade de criação de índices *on-line*.

Apresentamos, então, duas arquiteturas que se utilizam de agentes para realizar a auto-sintonia de índices em um SGBD relacional. Ambas permitem a total automatização dos passos envolvidos na escolha de índices para o sistema. Outros trabalhos presentes na literatura buscam algoritmos para a seleção de índices e propõem que estes algoritmos sejam utilizados em ferramentas de apoio ao DBA.

A primeira arquitetura se baseia em um único agente. Mostramos como este agente poderia ser encaixado no SGBD a partir da discussão de um modelo local de auto-sintonia. Discutimos, então, uma heurística que permite que as decisões de criação ou remoção de índices do sistema sejam tomadas de forma totalmente automática. Esta heurística é uma revisão e extensão da proposta de Costa e Lifschitz em [8]. Revisamos, nesta dissertação, o cálculo de benefícios realizado para índices candidatos e consideramos explicitamente o benefício trazido por índices reais. Por fim, debatemos sobre como construir o agente a partir da instanciação de um *framework* abstrato para construção de agentes.

A segunda arquitetura proposta se baseia em múltiplos agentes. Ela é interessante para lidar com SGBDs submetidos a cargas de trabalho com diferentes sazonalidades e perfis. Apresentamos como os agentes desta arquitetura interagem e discutimos a forma de construção de cada agente a partir da instanciação de um *framework* abstrato para a construção de agentes.

No próximo capítulo, iremos mostrar o projeto detalhado da arquitetura com um único agente e as questões envolvidas na sua integração com o SGBD de código aberto PostgreSQL. Apresentaremos, ainda, resultados experimentais obtidos com a implementação dessa arquitetura neste SGBD.