Atualização Dinâmica de *Software* em SGBDs com Suporte do Modelo de Componentes de *Software*

Cleandro Flores De Gasperi, Marcia Pasin

PPGI – Universidade Federal de Santa Maria (UFSM)
Av. Roraima 1000 – 97.105-900 – Santa Maria – RS – Brasil

cleandro@cpd.ufsm.br ,marcia@inf.ufsm.br

1. Introdução

Atualização Dinâmica de Software (ADS) [Fabry 1976] permite atualizar programas em execução sem interrupção de serviço. ADS não é novidade. A pesquisa mais remota data da década de 70. No contexto atual, destacam-se soluções para sistemas operacionais (Windows Update, atualizações para distribuições do Linux, etc) e trabalhos recentes enfatizando o modelo de componentes de *software* [Oreizy 1998, Wang 2004]. Soluções no nível de sistemas operacionais são tipicamente associadas a detalhes de implementação (mais restrições são reportadas em [Bellissimo 2006]), e por isso são difíceis de serem portadas para outras arquiteturas e sistemas. Soluções baseadas em componentes de *software* são focadas em servidores de aplicação e persistência (informação armazenada em memória estável), desconsiderando a camada transacional presente nos SGBDs.

Para promover a atualização do SGBD, as instituições dispõem, basicamente, de duas alternativas, a saber: **Solução baseada em *cluster*** atualiza gradualmente nodos de um *cluster* usando *hardware* redundante. Enquanto um nodo é atualizado, os demais assumem todas as requisições. Ocorre indisponibilidade do nodo, mas não do serviço. Notadamente esta é uma solução cara, baseada em configurações manuais e de complexa execução. **Indisponibilização do serviço** é adotada por grande parte das organizações, sendo trivial. O serviço é suspenso e, então, o servidor tem a versão do SGBD atualizada, sendo restabelecido o serviço ao final da atualização. Esta solução é popular em bancos de dados centralizados (tipicamente com um único nodo).

Na literatura, não constam trabalhos que apliquem técnicas de ADS automáticas em SGBD. Esta idéia é nova. Técnicas automáticas e semi-automáticas são típicas de sistemas operacionais. A necessidade de técnicas específicas de ADS para SGBDs decorre que esta categoria adota conceitos particulares como gerenciamento transacional, propriedades ACID, restrições de integridade e multiprocessamento.

2. Modelo de componentes de *software* para ADS em SGBDs

Arquitetura baseada em componentes de *software* consiste em uma solução eficiente para modelar sistemas computacionais e oferece uma visão geral dos componentes de um sistema enquanto detalhes de implementação são ocultados. Sob esta ótica, sistemas e programas são vistos como um conjunto de componentes, ou unidades binárias, conectados por interfaces bem definidas.

Em contraste, apesar de já existir implementações construídas no paradigma de orientação a objetos (OO), implementações de SGBDs tipicamente tem como base o modelo orientado a processos. Implementações OO, se conjugadas com *frameworks* para construção de sistemas com o conceito de componentes de *software* podem chegar a uma solução compatível ao que se propõe neste trabalho. Para tanto, possivelmente, se passará pela reescrita do SGBD, de tal forma que os objetos que implementam seus

módulos serão encapsulados em componentes. Assim, uma vez que o módulo está encapsulado em um componente, abstrações do modelo de componentes podem ser usadas para a substituição dos componentes de *software*.

3. Implementação

A implementação da solução proposta neste trabalho combina o modelo de componentes Fractal [Bruneton 2006] e conceitos de auto-gerenciamento [Kephart 2003]. Fractal permite definir o SGBD como um componente composto por subcomponentes. Em um sistema centralizado (um nodo), subcomponentes constituem-se dos elementos do SGBD como *parser*, analisador sintático, otimizador, etc. Uma vez definidos componentes, subcomponentes e dependências, controladores Fractal permitem a substituição dinâmica de componentes. Auto-gerenciamento, implementado por uma máquina que relaciona regras para satisfazer políticas pré-definidas especifica quando ADS deverá ser aplicada no SGBD (restrições temporais) bem como trata dependências entre componentes de *software* (restrições causais).

Inicialmente o SGBD está em operação normal, com dependências entre componentes. Cada componente a ser atualizado tornar-se-á indisponível, é atualizado e é reincorporado ao sistema. Finalmente, um procedimento de transferência de estado restaura atualizações processadas durante o período de indisponibilidade no componente recentemente atualizado. Esta solução promove continua disponibilidade e mantém o desempenho do sistema em níveis adequados, uma vez que o sistema é atualizado de forma gradativa.

4. Conclusões

Desde que a demanda de aplicações críticas tem aumentado, é interessante que sistemas computacionais sejam providos de técnicas eficientes, robustas e transparentes para ADS. Este trabalho apresentou uma arquitetura baseada em componentes de *software* para ADS em SGDB automática e sem uso de *hardware* redundante. A solução é viável respeitando restrições impostas pelo modelo de componentes e pela natureza do SGBD.

Referências

Bellissimo, A. *et al.* (2006), "Secure software updates: disappointments and new challenges", in Proc. HotSec '06: 1st USENIX Workshop on Hot Topics in Security.

Bruneton, E. *et al.* (2006), "The FRACTAL component model and its support in Java: experiences with auto-adaptive and reconfigurable systems", Software, Practice & Experience, v.36 n.11-12, pp. 1257-1284.

Fabry, R. S. (1976), "How to design a system in which modules can be changed on the fly". In: Proc. of the 2nd Int. Conf. on Software Engineering. IEEE Computer Society Press, Los Alamitos, CA. pp. 470-476.

Kephart, J. O.; Chess, D. M. (2003) "The vision of autonomic computing", Computer, vol. 36, pp. 41-50.

Oreizy, P.; Medvidovic, N.; Taylor, R. N. (1998), "Architecture-base runtime software evolution". In: 20th Int. Conf. on Software Engineering. IEEE Computer Society, Washington, DC, pp. 177-186.

Wang, Q. *et al.* (2004), "A component-based approach to online software evolution". Journal of Software Maintenance and Evolution: Research and Practice, pp. 1-25.