データベース管理システムにおける性能劣化を抑制するソフトウェア若化

十文字　優斗[†]　山田浩史[†]

ソフトウェア若化とは稼動しているソフトウェアを不安定な状態から安定した状態に戻す実践的な手法である[4]．ソフトウェア若化の典型的な例は対象ソフトウェアの再起動であり，これを用いることでソフトウェアの可用性が向上することが知られている．ソフトウェア若化の運用方式にはプロアクティブとリアクティブとがある．プロアクティブなソフトウェア若化では，ソフトウェアが稼動している最中に定期的に再起動を行なうことで，クラッシュなどのサービス停止を予防する．特にソフトウェア老化[4]と呼ばれる，メモリリークなど長期に渡り稼動しているソフトウェアにおいて生じる問題に対して効果的である．また，リアクティブなソフトウェア若化では，実際にソフトウェアが停止した際にソフトウェアを再起動する．これにより，サービスの再開を実現する．

ソフトウェア若化は強力な手法であるが，データベース管理システム (DBMS) に対してはコストの大きい操作となる．多くの DBMS は，低速なディスク I/O を回避するためにバッファプールをメモリに展開して，その上でデータをキャッシュしながら操作を行なう．また，memcached など永続化の機能を有さない DBMS は，データをすべてメモリ上のテーブルに展開してそこでの操作のみで処理を終える．こうした DBMS に対してソフトウェア若化を実行すると，性能が著しく低下してしまう．再起動によってメモリの内容が破棄されてしまうため，バッファプールやテーブルを再構築する必要があるためである．再起動のコストは甚大であり，ある文献[2]では，Facebook において，再起動後の memcached がウォームアップするのに 100 分程度要することが報告されている．大規模なソフトウェア若化方式は多く提案されており，オペレーティングシステム (OS)[5),9)]や仮想マシンモニタ[6),7)]といったソフトウェアレイヤを対象とした手法は提案されているが，著者らが知る限り近年のデータベース管理システムを対象とした方式は存在しない．

本研究では，DBMS のソフトウェア若化による性能劣化を軽減する手法を提案する．提案方式では，DBMS のソフトウェアの構造を，バッファプールやテーブルといったデータベース領域と実際に実行するコードや参照するデータ構造などの領域とを分離し，再起動実行時に前者の領域を解放せずに維持する．これにより，ソフトウェア若化を実行する際は，コードなどの領域のみが再起動され，再起動後にデータベース領域を再接続することによって，ソフトウェア若化の効果を得つつ，それによる性能劣化を抑制することを狙う．

再起動後に DBMS のメモリ領域を維持するために，提案方式では OS の共有メモリ機構を利用する．データベース領域を共有メモリとして割り当てることによって，DBMS プロセスが消失しても，該当領域は OS によって保持される．再起動後は，共有メモリを読み込むことで，データベース領域を再度参照するようメモリオブジェクトを用意する．また，Slab Calcification[3]と呼ばれる，長期稼動によるスラブアロケータの不具合を解消するために，再起動時にデータベース領域内のオブジェクトの取捨選択をできるようにする．現在はランダムにスラブオブジェクトを破棄して Slab Calcification が再起動後に起きないようにしている．

クラッシュなどのリアクティブなソフトウェア若化にも対応するため，バリューのチェックサムを取得して同じく共有メモリに保存する方式，それを複数持つことによってデータの安全性をより堅牢にする方式，またバッファキャッシュを OS のクラッシュから保護する手法[1),8)]を応用して，データベース領域を操作する処理を実行しているとき以外はデータベース領域を Read-only にする方式などを組み込み，クラッシュ時のデータの堅牢性と性能のトレードオフを明らかにする予定である．

現在，memcached を対象に提案方式を組み込んでおり，初期プロトタイピングがほぼ完了している．今後は，予めセットしておいたキーバリューを取得するような単純なワークロードや Twitter のアクセスを模したベンチマークを稼動させながら提案方式を用いてソフトウェア若化を実施するというプロアクティブなソフトウェア若化の効果を確認する．また，ソフトウェアフォールトインジェクションを行い，リアクティブなソフトウェア若化の有効性も検証する予定である．

参考文献

1) P. M. Chen, W. T. Ng, S. Chandra, C. Aycock, G. Rajamani, and D. Lowell. The Rio File

† 東京農工大学
Tokyo University of Agriculture and Technology

Cache: Surviving Operating System Crashes. In *Proc. of the 7th ACM Int'l Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS '96)*, pages 74–83, 1996.

2) A. Goel, B. Chopra, C. Gerea, D. Mátáni, J. Metzler, F. U. Haq, and J. Wiener. Fast Database Restarts at Facebook. In *Proc. of the 2014 ACM SIGMOD Int'l Conf. on Management of Data (SIGMOD '14)*, pages 541–549, 2014.
3) X. Hu, X. Wang, Y. Li, L. Zhou, Y. Luo, C. Ding, S. Jiang, and Z. Wang. LAMA: Optimized Locality-aware Memory Allocation for Key-value Cache. In *Proc. of 2015 USENIX Annual Technical Conference (USENIX ATC '15)*, pages 57–69, 2015.
4) Y. Huang, C. Kintala, N. Kolettis, and N. D. Fulton. Software Rejuvenation: Analysis, Module and Applications. In *Proc. of the IEEE 25th Int'l Symp. on Fault-Tolerant Computing (FTCS '95)*, pages 381–390, 1995.
5) K. Kourai. Fast and Correct Performance Recovery of Operating Systems Using a Virtual Machine Monitor. In *Proc. of the 7th ACM International Conference on Virtual Execution Environments (VEE '11)*, pages 99–109, 2011.
6) K. Kourai and S. Chiba. A Fast Rejuvenation Technique for Server Consolidation with Virtual Machines. In *Proc. of the 37th IEEE/IFIP Int'l Conf. on Dependable Systems and Networks (DSN '07)*, pages 245–254, 2007.
7) K. Kourai and S. Chiba. Fast Software Rejuvenation of Virtual Machine Monitors. *IEEE Transactions on Dependable and Secure Computing (TDSC)*, 8(6):839–851, 2011.
8) W. T. Ng and P. M. Chen. The Systematic Improvement of Fault Tolerance in the Rio File Cache. In *Proceedings of the 29th Symposium on Fault-Tolerant Computing (FTCS '99)*, pages 76–83, Jun. 1999.
9) K. Yamakita, H. Yamada, and K. Kono. Phase-based Reboot: Reusing Operating System Execution Phases for Cheap Reboot-based Recovery. In *Proc. of the 41st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN '11)*, pages 169–180, 2011.