STUDIA INFORMATICA 2015
Volume 36 Number 1 (119)

Łukasz GRALA, Zbyszko KRÓLIKOWSKI
Politechnika Poznańska, Instytut Informatyki

# OCENA EFEKTYWNOŚCI ARCHITEKTURY *IN-MEMORY* W SQL SERVER 2014

**Streszczenie**. Rozwój sprzętu komputerowego w ostatnich latach zainicjował badania, mające na celu opracowanie nowych architektur baz danych. Jednym z takich nowych rozwiązań jest architektura nazywana „*In-Memory*" lub też „*Main-Memory*". Stanowi ona radykalną zmianę podejścia do sposobu przechowywania i przetwarzania danych w systemach baz danych, wykorzystujących model przetwarzania transakcyjnego OLTP (ang. *On-Line Transaction Processing*). Celem niniejszej pracy jest analiza wybranych aspektów architektury *In-Memory* oraz ocena jej efektywności w porównaniu do rozwiązań konwencjonalnych, wykorzystujących pamięci dyskowe.

**Słowa kluczowe**: kolumnowe bazy danych, *In-Memory*, OLTP, OLAP

## IN-MEMORY SOLUTIONS FOR OLTP DATABASES

**Summary**. The changing realities of hardware caused the search for new solutions in the field of architecture databases. The new architecture called "*In-Memory*" or "*Main-Memory*" radically changed many typical existing solutions in the field of data structures that store rows of data, as well as the concept of indices. The work aims to discuss the architecture and compare the solutions "*In-Memory*" for on-disk storage solutions.

**Keywords**: database, In-Memory database, OLTP, OLAP

## 1. Wprowadzenie

Rozwój sprzętu komputerowego, a w szczególności pojawienie się procesorów wielordzeniowych oraz istotny wzrost rozmiaru dostępnej pamięci operacyjnej, spowodował zmianę podejścia do założeń architektury konwencjonalnych baz danych, w których przyjmowano, że dane pamiętane są w pamięci dyskowej i w miarę potrzeb są sprowadzane do pamięci

operacyjnej. W tym nowym podejściu, nazywanym *In-Memory*, zakłada się, że całość danych lub przynajmniej ich zdecydowana większość rezyduje na stałe w pamięci operacyjnej, która, jak powszechnie wiadomo, jest o kilka rzędów wielkości szybsza od pamięci dyskowych.

W przypadku systemów wykorzystujących model przetwarzania transakcyjnego OLTP (ang. *On-Line Transaction Processing*) istotne są dwa parametry: czas trwania jednostkowej transakcji oraz przepustowość systemu, to jest liczba transakcji możliwych do wykonania w jednostce czasu. Minimalizacja operacji dyskowych, a najlepiej ich uniknięcie, ma istotny wpływ na wydajność jednostkowych transakcji i przepustowość systemu.

Część komercyjnych systemów zarządzania bazami danych (SZBD) oferuje rozwiązania, pozwalające efektywniej wykorzystywać pamięć. Są to tak zwane *bazy kolumnowe*, które w odróżnieniu od konwencjonalnych baz danych, w których dane są składowane poziomo (wiersze tabel), stosują składowanie kolumnowe (ang. *column store*). Takie rozwiązania wprowadziła firmy Oracle w [11] i SAP [8, 9, 10], Terradata [13]. Prekursorem takich rozwiązań był SZBD MonetDB [14, 15]. Warto również zwrócić uwagę na projekt badawczy HyPer [3, 4], jak również rozwiązanie komercyjne VoltDB [16]. W wersji SQL Server 2012, firma Microsoft wprowadziła kolumnowe składowanie danych [12], co dało duży wzrost wydajności dla zastosowań OLAP (ang. *On-Line Analytical Processing*). W OLAP pobiera się duże ilości danych, dotyczących wybranych kolumn i wykonuje się na nich różnego rodzaju obliczenia. Jak już wspomniano powyżej, w tym rozwiązaniu właśnie kolumny są oddzielnie przechowywane, bardzo efektywnie kompresowane i znajdują się w dużej mierze w pamięci operacyjnej.

W SQL Server 2014 poza kolumnowym składowaniem danych, wprowadzono nowe rozwiązanie *In-Memory* dla zastosowań OLTP o nazwie Hekaton [2]. Rozwiązanie to bazuje na nowej idei przechowywania wiersza danych. Nowa architektura jest w pełni zintegrowana z dotychczasowym systemem zarządzania bazą danych, jest więc łatwa w użyciu wraz z dotychczasowymi rozwiązaniami oraz nie komplikuje procesu administracji i utrzymania systemu.

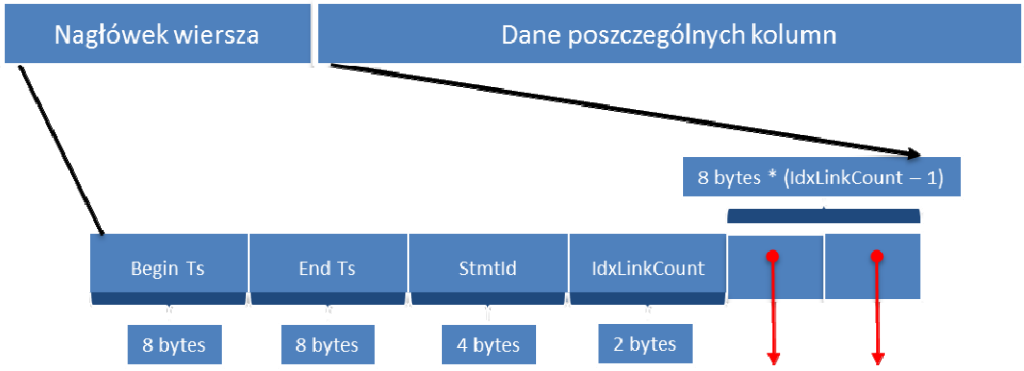
W kolejnych rozdziałach niniejszego artykułu zostanie przedstawiona analiza architektury

*In-Memory* w SZBD SQL Server 2014. Jest to niewątpliwie jedno z najciekawszych i najbardziej zaawansowanych rozwiązań dostępnych na rynku. Przedstawione zostaną wybrane aspekty architektury *In-Memory* SQL Server 2014, to jest struktury danych oraz mechanizm współbieżności transakcji zaimplementowany w tym rozwiązaniu oraz ocena jej efektywności w porównaniu do rozwiązań klasycznych wykorzystujących pamięci dyskowe.

## 2. Architektura *In-Memory* w SQL Server 2014

### 2.1. Struktury danych

Opracowując koncepcję architektury *In-Memory* w SQL Server 2014 [12], brano pod uwagę wąskie gardła rozwiązań klasycznych, wykorzystujących pamięci dyskowe, to jest operacje odczytu i zapisu (*I/O*), mechanizm blokad, a także mechanizm wersjonowania, mające duży wpływ na stopień współbieżności realizacji transakcji. Nowe struktury danych musiały być tak zaprojektowane, by maksymalnie wykorzystać dostępną pamięć, przy wysokim stopniu współbieżności transakcji. Zaimplementowano struktury danych, w których najmniejszą jednostką jest wiersz, a nie jak dotychczas strona zawierająca zbiór wierszy. Inspiracją dla tej architektury była koncepcja rozwiązania wolnego od blokad (ang. *Lock-Free*), bazującego na tabelach haszujących (ang. *Hash Tables*) oraz zbiorach definiowanych za pomocą list (ang. *List-Base Sets*), opisanych między innymi w [1]. Wiersz, jako najmniejsza struktura danych, jest przechowywany w pamięci i jego wielkość maksymalna nie może przekraczać 8 KB. Budowę wiersza w architekturze *In-Memory* [5, 17] przedstawiono na rysunku 1.




Rys. 1. Struktura wiersza w architekturze *In-Memory* [17]
Fig. 1. Record format in the architecture In-Memory [17]

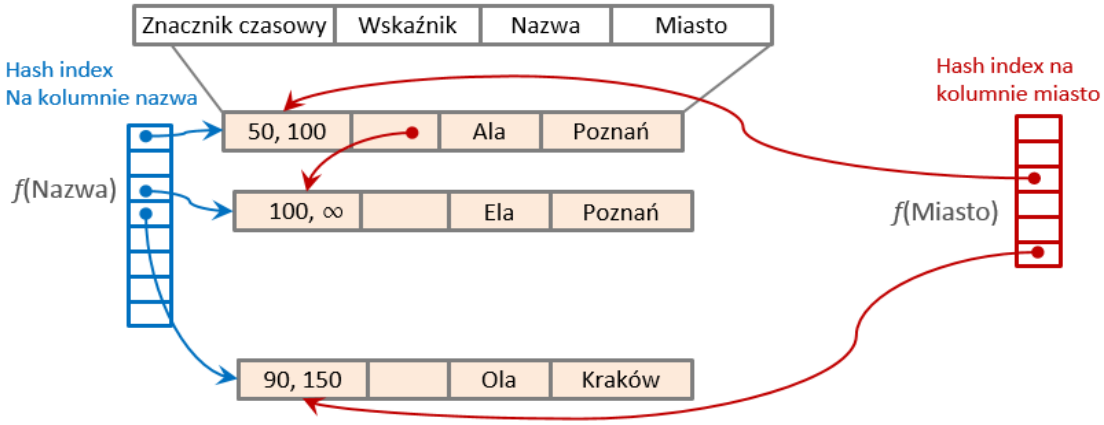
W nagłówku wiersza znajdują się informacje, dotyczące wersji danych, co powoduje, iż ta struktura danych sama w sobie jest mechanizmem wersjonowania. Ponadto występuje wskaźnik do pozostałych wersji wiersza, a także wskaźniki do indeksów.

Oprócz podstawowej struktury danych, jaką jest wiersz, w omawianym rozwiązaniu zaimplementowano dwa rodzaje indeksów. Struktury indeksowe nie są przechowywane w pamięci dyskowej, a ich reprezentacja w pamięci wykorzystuje funkcje haszujące i wskaźniki do wierszy. Konsekwencją nieprzechowywania indeksów w pamięci dyskowej jest fakt, iż mechanizm dziennika transakcji nie odnotowuje operacji związancych z indeksami, co w znaczym stopniu zmniejsza ilość operacji wejścia/wyjścia. Dziennik transakcji jest wykorzystywany do zagwarantowania własności atomowości i trwałości zmian

wykonanych przez transakcje zatwierdzone. W przypadku wystąpienia awarii, SZBD odbudowuje w pamięci struktury indeksowe, ale nie na podstawie zapisów w dzienniku transakcji i danych w pliku na dysku, tylko na podstawie danych pamiętanych w wierszu.

Jedną z zaimplementowanych struktur indeksowych jest indeks haszujący, bazujący na wspomnianej powyżej strukturze tabeli haszującej wolnej od blokad – *lock-free hash table* [1, 2]. Drugą strukturą jest indeks zakresowy zbudowany w formie BW-drzewa (ang. *BW-Tree*), będącego nową wersją bez blokadowej wersji B-drzewa (ang. *B-Tree*), opisanej szczegółowo w [6].

Na rysunku 2 przedstawiono przykładowy schemat procesu wstawiania i aktualizacji danych w omawianej strukturze *In-Memory*.




Rys. 2. Przykładowy proces wstawiania i aktualizacji danych [17]
Fig. 2. An example process of inserting and updating data [17]

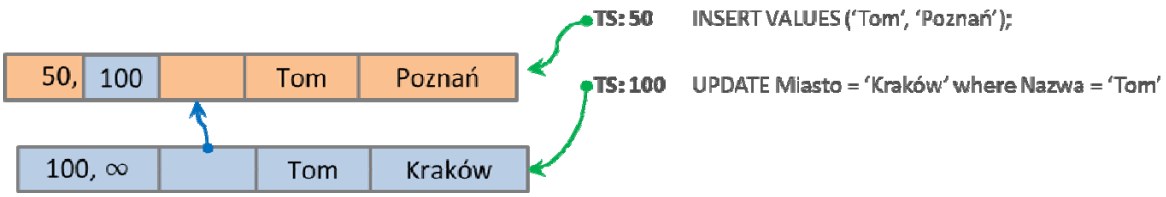
W przykładowym procesie wstawiania i aktualizacji z rysunku 2 wykorzystywane są dwa indeksy haszujące zbudowane na kolumnach *Nazwa* i *Miasto* oraz trzy przykładowe wiersze danych. Pierwszy wiersz wstawiono o czasie 50 z wartościami: (*Ala, Poznań*), drugi wiersz został wstawiony o czasie 90 z wartościami: (*Ola, Kraków*), a o czasie 100 nastąpiła aktualizacja wiersza z wartościami (*Ala, Poznań*) na (*Ela, Poznań*). Powstał nowy wiersz z początkowym znacznikiem czasowym 100. W wierszu pierwszym wstawiona została wartość końcowego znacznika czasowego 100. Pomiędzy tymi dwoma wierszami (*50, 100, Ala, Poznań*) oraz (*100, -, Ela, Poznań*) wstawiany jest wskaźnik tworzący łańcuch tych wierszy.

### 2.2. Współbieżność transakcji

W systemach baz danych wykorzystujących model przetwarzania OLTP transakcje wykonywane są współbieżnie. Jedną z podstawowych metod zarządzania współbieżną realizacją transakcji jest wykorzystywanie blokad danych, a także wersjonowanie wspierane mechanizmem blokad. W architekturze *In-Memory* w SQL Server 2014 odrzucono te rozwiązania –

nie wykorzystuje się blokad do synchronizacji transakcji wykonywanych współbieżnie, a sama struktura składowania wiersza zawiera w sobie mechanizm wersjonowania danych. Po każdej operacji aktualizacji tworzony jest nowy wiersz albo nowa wersja wiersza. W nagłówku każdego wiersza znajduje się znacznik czasowy powstania wiersza (na rysunku 1 – *Begin Ts*) oraz informacja dotycząca opcjonalnego końca „życia" takiego wiersza (na rysunku 1 – *End Ts*) i wskaźnik do kolejnej wersji wiersza. Szczegółowo mechanizm ten został przedstawiony w [2, 5, 17].

Na rysunku 3 przedstawiono prosty przykład wersjonowania wierszy w omawianej architekturze *In-Memory*.




Rys. 3. Wersjonowanie wierszy w architekturze In-Memory [17]
Fig. 3. Versioning records in architecture In-Memory [17]

## 3. Ocena efektywności transakcji w architekturze In-Memory i rozwiązaniach konwencjonalnych

### 3.1. Założenia i środowisko realizacji testów efektywnościowych

Opracowano dwa rodzaje testów, których celem była ocena efektywności wykonywania transakcji w architekturze *In-Memory* oraz w rozwiązaniach konwencjonalnych. W ramach pierwszej grupy badano efektywność wykonywania krótkich transakcji, realizujących wstawianie i usuwanie dużych wolumenów danych różnego typu. W ramach drugiej grupy badano efektywność współbieżnej realizacji wstawiania pojedynczych danych, tzw. wsadów przez wiele wątków.

W pierwszej grupie testów efektywnościowych tabela pamiętana na dysku miała indeks grupujący na wartości inkrementowanej, czyli wyeliminowano problem przebudowy struktury indeksu grupującego, oraz jeden indeks niegrupujący na wartości numerycznej. Ta sama tabela w architekturze *In-Memory* miała dwa indeksy haszujące oraz indeks zakresowy. Testy wykonano wielokrotnie, a prezentowane w rozdziale 3.2 wyniki mają charakter uśredniony. Przed każdą iteracją struktury danych były usuwane i zakładane ponownie. W drugiej części tego testu tabele były zdefiniowane jako tymczasowe.

W drugiej grupie testów efektywnościowych wykonano symulację jednoczesnego wstawiania nowych zleceń do bazy danych w wielu równoczesnych wątkach. W bazie danych istniały dwie tabele nagłówków i pozycji dokumentu. Równocześnie uruchomionych było

80 wątków, które w pętli wykonywały wsad danych. W innym ujęciu można powiedzieć, że wstawiały one nowy dokument – w każdym wsadzie wstawiano 1 rekord nagłówka i 100 rekordów wierszy. Jako wynik wykonania testu porównano ilość wstawionych wsadów w jednostce czasu, w architekturze *In-Memory* oraz w rozwiązaniach konwencjonalnych, wykorzystujących pliki dyskowe. Test składał się z trzech następujących etapów:

- uruchomienie testu na strukturach dyskowych,
- migracja struktur dyskowych do *In-Memory*,
- wykorzystanie natywnej kompilacji procedur specjalnie utworzonej dla architektury *In-Memory*.

Wszystkie testy wykonano na komputerze posiadającym 32 GB pamięci RAM, dyski SSD oraz procesor Intel Core i7 posiadający 4 rdzenie, pracujący w trybie *hypertreading*, czyli widocznych w systemie jest 8 jednostek obliczeniowych (8CPU), system operacyjny Windows Server 2012 oraz system zarządzania bazą danych SQL Server 2014 z Service Pack 1.

### 3.2. Wyniki wykonanych testów efektywnościowych

W wykonanym teście badane transakcje wykonywały operację wstawiania odpowiednio sto tysięcy, milion i dwa miliony wierszy – każda operacja była oddzielną transakcją. Badano czasy realizacji tych transakcji. Na poziomie logicznym tabele danych miały identyczną strukturę, a indeksy zostały zoptymalizowane pod kątem realizacji jednakowych zapytań. Operacje wstawiania miały identyczny kod TSQL.

Uzyskane średnie wyniki testów efektywnościowych dotyczących wstawiania danych przedstawiono w tabeli 1 oraz na rysunku 4.

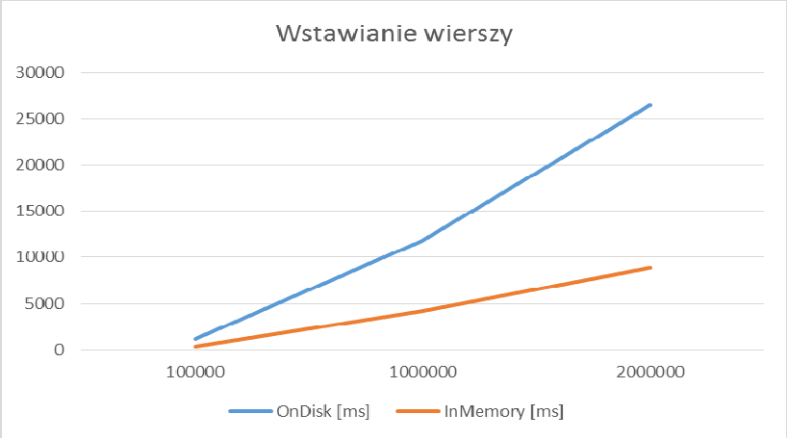
Tabela 1

Wynik testów wstawiania wierszy do obu struktur danych

| Lp. | Liczba wierszy | Tabela dyskowa [ms] | Tabela *In-Memory* [ms] |
|---|---|---|---|
| 1 | 100000 | 1189 | 330 |
| 2 | 1000000 | 11861 | 4189 |
| 3 | 2000000 | 26460 | 8901 |

Transakcje wykonujące operacje wstawiania różnych wolumenów danych do konwencjonalnych struktur dyskowych, zgodnie z przewidywaniami okazały się dużo wolniejsze od transakcji wykonujących te same operacje w architekturze *In-Memory*.

Jeszcze większa różnica pomiędzy tymi rozwiązaniami uwidoczniła się w przypadku usuwania wierszy. Uzyskane średnie wyniki wykonanych testów efektywnościowych dotyczących usuwania wierszy przedstawiono w tabeli 2 oraz na rysunku 5.

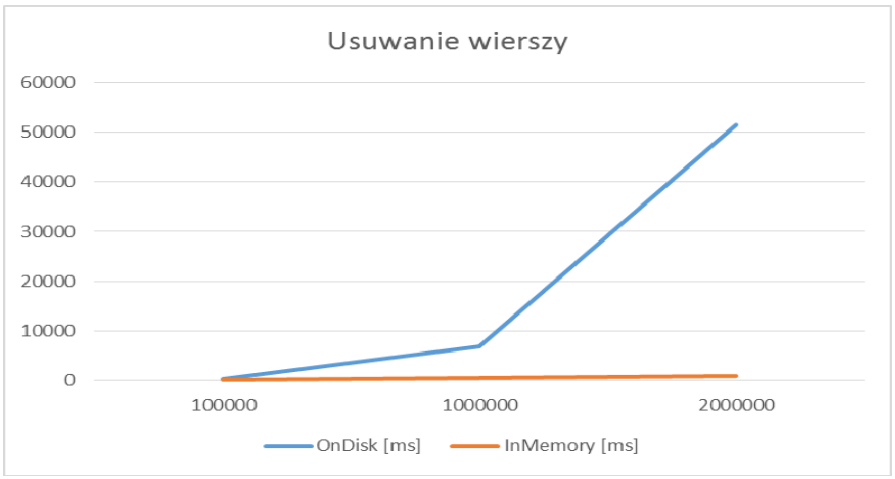



Rys. 4. Ocena efektywności transakcji wstawiającej wiersze
Fig. 4. Results of row insert

Tabela 2
Wynik testów usuwania wierszy w obu strukturach danych

| Lp. | Liczba wierszy | Tabela dyskowa [ms] | Tabela *In-Memory* [ms] |
|---|---|---|---|
| 1 | 100000 | 426 | 99 |
| 2 | 1000000 | 6981 | 496 |
| 3 | 2000000 | 51583 | 1004 |

Druga część tej grupy testów efektywnościowych polegała na wykonaniu tych samych transakcji, ale tabele były strukturami tymczasowymi, czyli system zarządzania bazą danych nie zapewniał trwałości danych w przypadku wystąpienia awarii. W przypadku konwencjonalnych struktur dyskowych wymaga to innego zdefiniowania nazwy obiektu w operacji *CREATE*. Natomiast w architekturze *In-Memory* wymagana jest specjalna klauzula *DURABILITY = SCHEMA_ONLY*, wówczas SZBD nie gwarantuje własności trwałości danych podczas wykonywania transakcji – są one przechowywane jedynie w pamięci operacyjnej.




Rys. 5. Ocena efektywności transakcji usuwającej wiersze
Fig. 5. Results delete rows

W tabeli 3 przedstawiono średnie dane z wykonanych w ten sposób testów dotyczących wstawiania danych.

Tabela 3

Wynik testu wstawiania wierszy do obu struktur danych – tabel tymczasowych

| Lp. | Liczba wierszy | Tabela dyskowa [ms] | Tabela *In-Memory* [ms] |
|---|---|---|---|
| 1 | 100000 | 1156 | 88 |
| 2 | 1000000 | 11832 | 834 |
| 3 | 2000000 | 23989 | 1684 |

Podobnie jak w pierwszej części wykonano również test usuwania danych z obu typów struktur. Uśrednione wyniki przedstawia tabela 4.

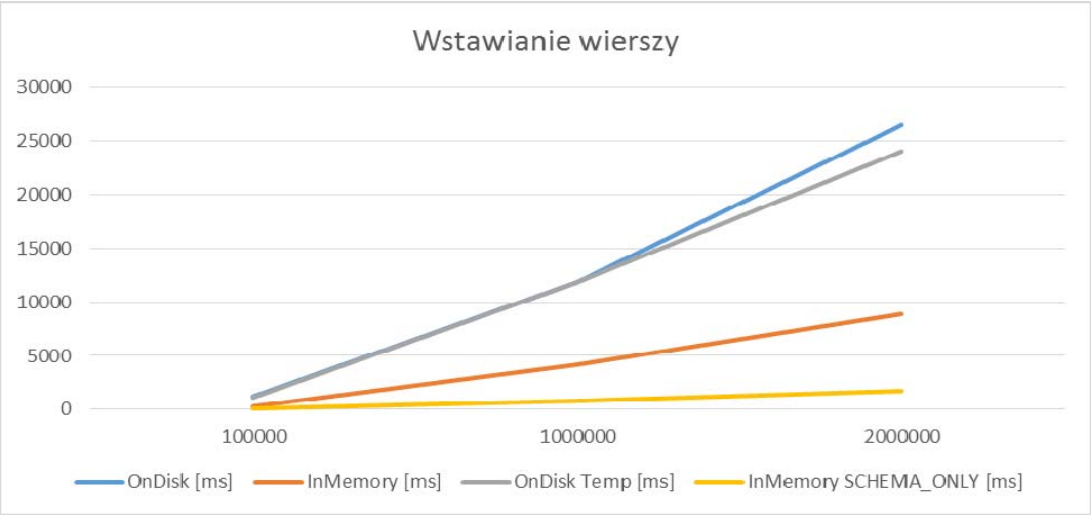
Tabela 4

Wynik testu usuwania wierszy do obu struktur danych – tabel tymczasowych

| Lp. | Liczba wierszy | Tabela dyskowa [ms] | Tabela *In-Memory* [ms] |
|---|---|---|---|
| 1 | 100000 | 740 | 109 |
| 2 | 1000000 | 11749 | 404 |
| 3 | 2000000 | 23048 | 738 |

Na rysunku 6 przedstawiono wyniki oceny efektywności transakcji wstawiania wierszy do tabel dyskowych i tymczasowych oraz *In-Memory* łącznie.

Wykonane testy efektywnościowe pokazały, jak znaczna jest różnica w wydajności transakcji realizujących operacje wstawiania i usuwania dużych wolumenów danych, wykonywanych w konwencjonalnych bazach danych wykorzystujących pliki przechowywane na dysku, a wykonywanych w architekturze *In-Memory*.




Rys. 6. Ocena efektywności transakcji wstawiania wierszy do tabel dyskowych i tymczasowych oraz *In-Memory*

Fig. 6. Results insert rows

Kolejna grupa wykonanych testów efektywnościowych miała charakter obciążeniowy. Testy miały symulować działanie typowych systemów transakcyjnych OLTP, takich jak: system rezerwacji lotów, sklep internetowy itp.

W trakcie wykonywania testów można zaobserwować znaczącą różnicę w zakresie wykorzystanych zasobów. W przypadku konwencjonalnych systemów baz danych występował problem opóźnień, dotyczący blokowania zasobów (*ang. Latch Contention*). Pomimo iż procesory były wykorzystywane na poziomie około 50%, to liczba blokad zasobów (*ang. Latch*) wynosiło około 100 000 na sekundę. W przypadku rozwiązania *In-Memory* przy wzroście wydajności przedstawionym w tabeli 5, liczba blokad zasobów (*ang. Latch*) spadła do zera, natomiast użycie procesorów przekraczało 80%. Testy efektywnościowe wykonano pięciokrotnie w okresach około 120 sekund i wyniki uśredniono dla czasu jednej sekundy.

Tabela 5

Wyniki testów symulacji wielowątkowej rejestracji zamówień

| Jednostka | Tabela dyskowa | Tabela In-Memory | Natywna procedura | Przyrost |
|---|---|---|---|---|
| Liczba wsadów na sekundę | 1600 | 5000 | 40000 | 25x |

Podsumowując wyniki wykonanych dwóch grup testów efektywnościowych, należy stwierdzić, że:

- Transakcje realizujące operacje wstawiania i usuwania dużych wolumenów danych, wykonywane w konwencjonalnych bazach danych wykorzystujących pliki przechowywane na dysku, charakteryzują się znacznie niższą efektywnością w porównaniu do takich samych transakcji wykonywanych w architekturze *In-Memory*.
- W architekturze *In-Memory* wyeliminowano problem blokad, co zwiększyło w znacznym stopniu stopień współbieżnej realizacji transakcji,
- Liczba wstawianych danych (wsadów) w jednostce czasu, w architekturze *In-Memory* jest 3-krotnie wyższa niż w rozwiązaniach konwencjonalnych wykorzystujących pliki dyskowe.

## 4. Podsumowanie

Wykonane testy efektywnościowe jednoznacznie pokazują, iż obrany przez firmę Microsoft kierunek zmian w architekturze systemów zarządzania bazą danych dla zastosowań OLTP jest właściwy i przynosi oczekiwane efekty. Na tę chwilę architektura *In-Memory* wiążę się jeszcze z wieloma ograniczeniami. Można na ich podstawie zidentyfikować pewne konieczne kierunki rozwoju i dalszych prac nad tymi rozwiązaniami.

Optymalizacja struktur indeksowych, a także mechanizmów, które służą do przebudowy czy też usuwania w tle wierszy nieaktualnych, jest niewątpliwie jednym z takich kierunków. Dotyczy to kwestii doboru indeksów do zapytań, a także algorytmów realizujących instrukcje porównania i wymiany CAS (*ang. Compare and Swap*), przedstawione w [6]. Widoczny jest również element, mający wpływ zarówno na wydajność, jak i na ilość zajmowanego miejsca, czyli specyficzny profil systemu OLTP, gdzie wykonuje się bardzo dużą ilość aktualizacji, co może powodować bardzo długi łańcuch wersji danych wiersza, a zarazem większy obszar zajmowanej pamięci.

Biorąc pod uwagę fakt, iż jest duża różnorodność proponowanych rozwiązań wykorzystujących w inny sposób współczesną architekturę sprzętową systemów, a w szczególności jej pamięć, w tym zmodyfikowane czy też całkowicie nowe struktury dla danych, korzystne byłoby wnikliwe porównanie takich rozwiązań, jak to na przykład wykonano dla części z nich w publikacji [19].

Kolejnym aspektem jest identyfikacja danych, czy też prognoza danych, które można przenieść z pamięci operacyjnej na większą, a zarazem wolniejszą pamięć dyskową. Powstało kilka propozycji realizacji takich działań, nazywanych często identyfikacją ciepłych i zimnych danych (*ang. Hot and Cold Data*) [7,18]. Implementacje rozwiązań *In-Memory* obecnie są również częstym kierunkiem badań w zastosowaniach *Big Data*, *NoSQL* czy też rozwiązań dla urządzeń mobilnych.

## BIBLIOGRAFIA

1. Michael M. M.: High performance dynamic lock-free hash tables and list-based sets. SPAA, 2002.
2. Diaconu C., Freedman C., Ismert E., Larson P. A., Mittal P., Stonecipher R., Verma N., Zwilling M.: Hekaton: SQL Server's Memory-Optimized OLTP Engine. SIGMOD, 2013.
3. Funke F., Kemper A., Neumann T.: HyPer-sonic Combined Transaction AND Query Processing. PVLDB, Vol. 4(12), 2011, s. 1367÷1370.
4. Kemper A., Neumann T.: HyPer: A hybrid OLTP&OLAP main memory database system based on virtual memory snapshots. ICDE, 2011, s. 195÷206.
5. Larson P. A., Blanas S., Diaconu C., Freedman C., Patel J. M., Zwilling M.: High-Performance Concurrency Control Mechanisms for Main-Memory Databases. PVLDB, Vol. 5(4), 2011, s. 298÷309.
6. Levandoski J. J., Lomet D. B., Sengupta S.: The Bw-Tree: A B-tree for New Hardware Platforms. ICDE, 2013.

7. Levandoski J., Larson P. A., Stoica R.: Classifying Hot and Cold Data in a Main Memory OLTP Engine. ICDE, 2013.
8. Lee J., Muehle M., May N., Faerber F., Sikka V., Plattner H., Krueger J., Grund M.: High-Performance Transaction Processing in SAP HANA. IEEE Data Engineering Bulletin, Vol. 36(2), 2013, s. 28÷33.
9. Lee J., Kwon J. S., Färber F., Muehle M., Lee C., Bensberg C., Lee J. Y., Lee A. H., Lehner W.: SAP HANA Distributed In-Memory Database System: Transaction, Session, and Metadata Management. ICDE, 2013.
10. SAP In-Memory Computing, http://www.sap.com/pc/tech/in-memory-computing-hana.html.
11. Oracle Whitepaper: Oracle Database In-Memory, October 2014 http://www.oracle.com/ technetwork/database/in-memory/overview/twp-oracle-database-in-memory-2245633. html.
12. Larson P. A., Clinciu C., Fraser C., Hanson E. N., Mokhtar M., Nowakiewicz M., Papadimos V., Price S. L., Rangarajan S., Rusanu R., Saubhasik M.: Enhancements to SQL Server Column Stores. SIGMOD, 2013.
13. Teradata Columnar, http://www.teradata.com/products-and-services/database/teradata-14.
14. MonetDB, http://monetdb.cwi.nl.
15. Boncz P. A., Zukowski M., Nes N.: MonetDB/X100: Hyper-pipelining query execution. CIDR, 2005, s. 225÷237.
16. VoltDB, http://voltdb.com.
17. Whitepaper Microsoft: SQL Server In-Memory OLTP internals Overview. 2014.
18. Eldawy A., Levandoski J., Larson P. A.: Trekking Through Siberia: Managing Cold Data in a Memory-Optimized Database. VLDB, 2014.
19. Bach M., Duszeńko A., Werner A.: Koncepcja pamięciowych baz danych oraz weryfikacja podstawowych założeń tych struktur. Studia Informatica, Vol. 31, No. 2B(90), Gliwice 2010, s. 63÷76.

**Abstract**

This article presents a new architecture databases - In-Memory for OLTP applications, implemented in SQL Server 2014, are presented data structures and indexes, versioning mechanism, as well as comparative tests performed on-disk architecture and In-Memory architecture. The tests covered both individual transactions, but also simulation parallel work.

**Adresy**

Łukasz GRALA: Politechnika Poznańska, Instytut Informatyki, ul. Piotrowo 2, 60-965 Poznań, Polska, lukasz.grala@cs.put.poznan.pl.

Zbyszko KRÓLIKOWSKI: Politechnika Poznańska, Instytut Informatyki, ul. Piotrowo 2, 60-965 Poznań, Polska, zbyszko.krolikowski@cs.put.poznan.pl.