УДК 004.652.5

Бузовский О.В., д.т.н.,
Подрубайло А.А.

АКТУАЛЬНЫЕ ПРОБЛЕМЫ РАСПРЕДЕЛЕННЫХ ХРАНИЛИЩ ДАННЫХ В ОПЕРАТИВНОЙ ПАМЯТИ

Национальный технический университет Украины «КПИ»

sxl_sas@ukr.net

Требования к быстродействию систем хранения данных постоянно повышаются. Для достижения высокой скорости обработки запросов созданы системы хранения данных в оперативной памяти. В данной статье приведен обзор технологии распределенных хранилищ данных в оперативной памяти, а также проведен анализ недостатков существующих реализаций таких систем

Ключевые слова: распределенное хранилище данных в оперативной памяти, распределенное объединение таблиц, консистентность данных

Введение

Объем генерируемых и хранимых человечеством данных постоянно растет. По оценкам [1], каждые два года суммарный объем накопленных данных удваивается, и в 2015 году перешагнет порог в $10x2^{70}$ байт. В то же время, требования к быстродействию поиска данных непрерывно повышаются. В этих условиях системы управления базами данных, да и сами концепции таких хранилищ, должны непрерывно эволюционировать.

Требования к уменьшению времени обработки запроса к базе данных растут быстрее, чем скорость чтения данных из долговременных запоминающих устройств (ДЗУ). Это привело к появлению хранилищ данных, целиком размещенных в оперативной памяти [2] (*In-memory database, IMDB*), скорость чтения из которой более чем на порядок превосходит соответствующие показатели наиболее быстродействующих ДЗУ. Вследствие ограниченности объема оперативной памяти одного компьютера, были созданы распределенные хранилища данных в оперативной памяти (*In-memory data grid, IMDG*).

В данной статье выполнен обзор актуальных проблем, связанных с использованием распределенных систем хранения данных в оперативной памяти, и предложены подходы к их решению.

In-memory database

База данных в оперативной памяти в концепции *in-memory database* по своим функциональным особенностям подобна реляционным базам данных на долговременных носителях. Схема работы с такими хранилищами практически не отличается от работы с традиционной базой данных [3,4]: *IMDB* предоставляют полную поддержку языка *SQL* и поставляются с *JDBC/ODBC* драйверами, что минимизирует сложность перехода на эту технологию с традиционных хранилищ на магнитных носителях. Такие базы данных обязаны удовлетворять требования *ACID* [5] (*atomicity, consistency, isolation, durability*), т.е. ключевым понятием в таких системах является транзакция, как группа объединенных последовательных операций с базой данных. Транзакция не может быть выполнена частично, не нарушает согласованность системы, не влияет на одновременную работу других транзакций [6]. Кроме того, выполненные успешной транзакцией изменения не должны быть потеряны вследствие каких-либо сбоев системы (к примеру, обесточивания либо сбоя в оборудовании). Эти требования хорошо соответствуют традиционным бизнес-моделям, однако в полной мере реализовать их в распределенных системах, с учетом необходимого быстродействия, крайне сложно. Таким образом, хранилища, организованные по кон-

цепции in-memory database, в подавляющем большинстве случаев, реализованы в рамках одного сервера. Однако максимальный объем оперативной памяти одного компьютера ограничен и не может удовлетворить ряд задач, требующих хранения большого объема данных.

In-memory data grid

Идеология распределенных хранилищ данных в оперативной памяти, в свою очередь, резко отличается от этой концепции. Основной парадигмой для таких система стала *BASE – basically available, soft state, eventual consistency*. Эта парадигма была специально разработана для горизонтально масштабируемых (распределенных) систем [7]. Взамен пессимистичного подхода *ACID*, ставящего во главу угла консистентность (согласованность) хранимых в системе данных и проверяющей консистентность в конце каждой операции, *BASE* предполагает, что что консистентность данных в распределенной системе находится в состоянии потока, т.е. постоянно изменяется. Хранилище обязано обеспечить только конечную консистентность данных, т.е. хранимые в системе данные будут, в конечном счете, согласованы, при условии отсутствия изменений и через некоторое время после последнего изменения. Следствием этого является то, что данные в такой системе доступны лишь в большинстве случаев. Это означает, что в то время, пока данные в различных частях хранилища не пришли в консистентное состояние, запросы к ним могут не возвращать результатов.

Распределенные хранилища в оперативной памяти, как правило, не поддерживают *SQL*, предоставляя взамен возможности массово-параллельной обработки данных [8, 9]. В рамках этого подхода данные распределяются по кластеру, состоящему из стандартных серверов, и обрабатываются параллельно. Основным способом доступа к данным в *IMDG* является доступ по ключу, *MapReduce*-алгоритмы, а также ограниченный *SQL*-подобный язык запросов.

Основной структурной единицей *IMDG* является кэш (в различных реализациях таких хранилищ также имеющий название региона). Кэш распределенного хранилища данных в оперативной памяти – это распределенный ассоциативный массив. Этот массив, в отличие от строго типизированных реляционных баз данных, хранит сериализованные объекты, что позволяет исключить затраты на десериализацию на стороне клиента при чтении данных из хранилища. Такая организация позволяет обеспечить высокую степень горизонтальной масштабируемости, что особенно важно для крупных вебсервисов. Крупнейшие интернет-компании *EBay* и *Amazon* несколько лет успешно эксплуатируют собственные хранилища данных, построенные на этой архитектуре. [7, 10]

Однако технология распределенных хранилищ данных в оперативной памяти имеет целый ряд недостатков, преодоление которых является сегодня актуальной задачей.

Энергозависимость

Наиболее очевидным недостатком *IMDG* является их энергозависимость. Так как основным носителем информации в таких хранилищах выступает оперативная память, не являющаяся долговременным запоминающим устройством, даже кратковременная потеря электропитания приведет к потере хранимой информации.

Для решения этой проблемы существует несколько подходов. Простейший состоит в создании резервной копии данных на долговременном запоминающем устройстве (ДЗУ). Существует несколько стратегий резервного копирования. При синхронном копировании (рис. 1) операция по вставке/изменению/удалению данных считается завершенной только после получения подтверждения от ДЗУ об успешном выполнении этой операции на долговременном устройстве.

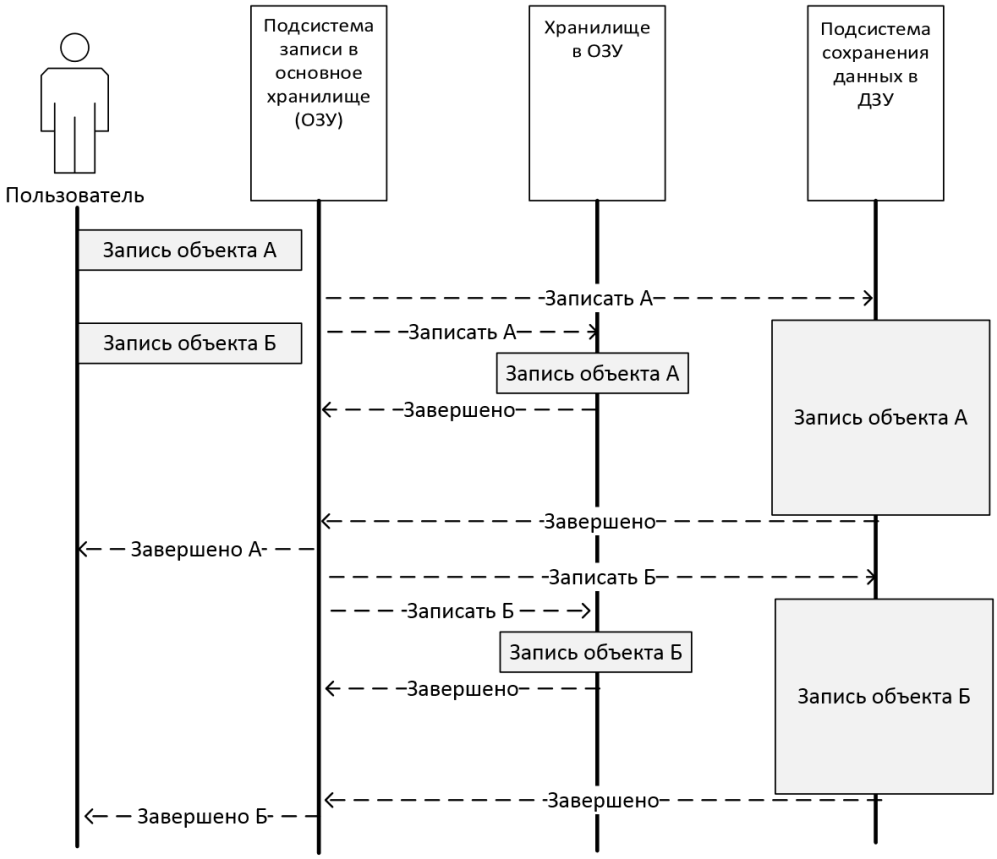

Рис. 1. Схема записи данных в *IMDG* с синхронным созданием резервной копии

Очевидно, что при синхронном резервном копировании производительность операций модификации данных деградирует до скорости выполнения операций записи\чтения на долговременном устройстве, которое на несколько порядков медленнее оперативной памяти. В то же время такой механизм позволяет гарантировать консистентность хранилища и его резервной копии, т.е. при любом сбое информация не будет потеряна. Поэтому синхронную запись используют только в хранилищах, где модификация данных – относительно редкое событие, а консистентность и надежность хранения очень важны.

Резервную копию можно также обновлять и в асинхронном режиме (рис. 2). При этом процесс записи на долговременное устройство выполняется в фоновом режиме, а квитанция об успешном выполнении операции возвращается пользователю сразу после модификации основного хранилища в ОЗУ. Очевидно, что в этом случае процесс резервного копирования практически не влияет на время выполнения операций по модификации данных, поэтому такой подход приемлем для систем с большим количеством операций записи. Однако, в случае отключения электропитания и отказа, в восстановленном из асинхронной резервной копии хранилище могут быть не отражены операции, об успешном выполнении которых пользователь уже получил квитанцию. К примеру, если на схеме, показанной на рис. 2, отказ электропитания произойдет сразу после получения пользователем квитанции о завершении записи объекта А, резервная копия на ДЗУ не успеет обновиться, соответственно, информация об этой операции будет потеряна, в то время как пользователь будет уверен в ее успехе. При одновременной модификации данных многими пользователями и последующем восстановлении из резервной копии определение того, какие операции не отражены в восстановленной системе – актуальная на сегодняшний день проблема.

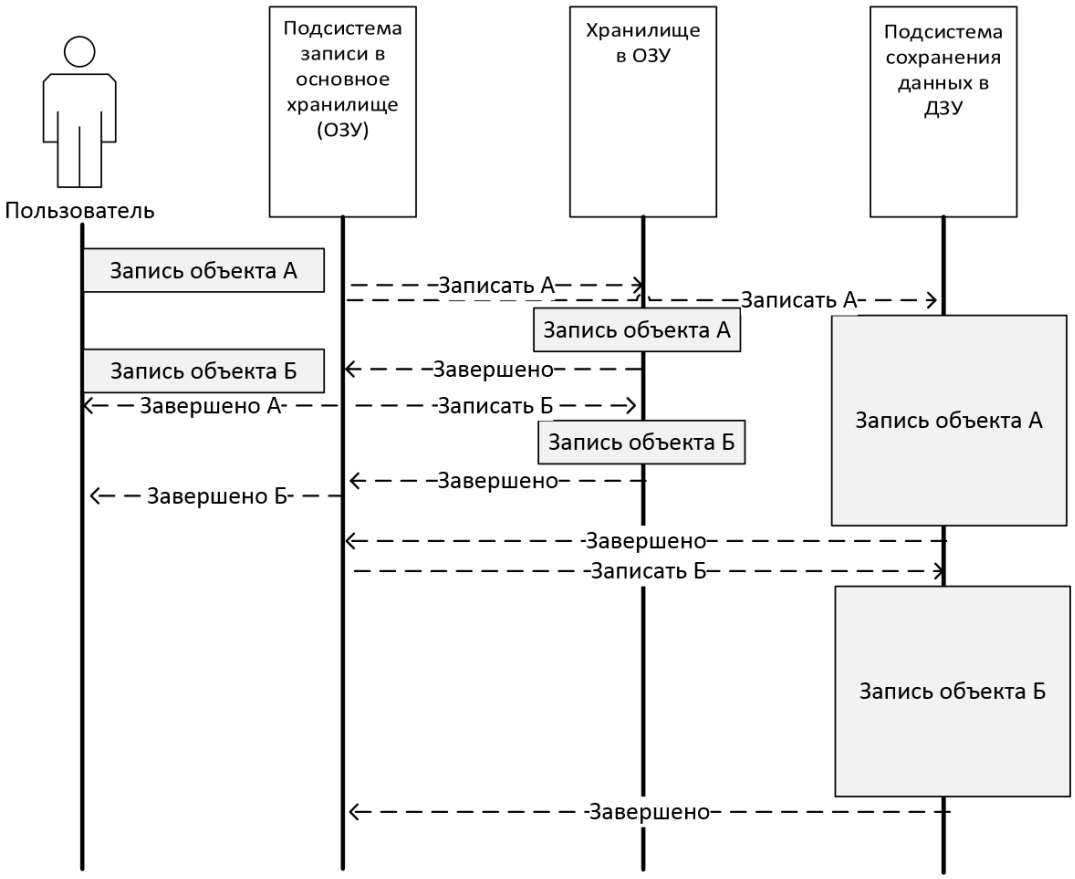

Рис. 2. Схема записи данных в *IMDG* с асинхронным резервным копированием

Резервное копирование позволяет избежать потери данных при отказе любого количества узлов хранилища, но для восстановления из резервной копии после отказа или потери электропитания потребуется перезапуск всего хранилища.

Другой подход к повышению энергонезависимости заключается в создании избыточных копий хранимых данных на различных узлах распределенной системы. При этом любая модификация данных выполняется на нескольких узлах одновременно. Таким подходом подразумевается независимое электропитание узлов системы хранения данных, содержащих идентичную информацию (реплики). В этом случае отказ одного или нескольких (в зависимости от степени избыточности) узлов распределенного хранилища не приведет к потере данных, а пользовательские запросы будут перенаправлены на другие узлы системы. Перезапуск системы в этом случае не потребуется, но количество отказов, которое система может перенести без потери данных, в этом подходе ограничено и задается заранее, перед запуском системы. «Бутылочным горлышком» такого подхода выступает скорость передачи данных по сети.

Оба подхода можно использовать одновременно. На рис. 3 приведен пример хранилища, способного перенести отказ любого из своих узлов, и способного к восстановлению при отказе более одного узла.

Переполнение оперативной памяти

Как уже упоминалось, вследствие высокой стоимости и конструктивных ограничений, объем оперативной памяти существенно меньше аналогичного показателя долговременных запоминающих

устройств. Поэтому, при неконтролируемом росте хранилища возможно полное заполнение оперативной памяти, что приведет, как минимум, к подкачке страниц ОЗУ на жесткий диск [11], и, соответственно, к существенному падению производительности, а, возможно, и к отказу хранилища. Механизм подкачки страниц заключается в переносе содержимого ОЗУ на долговременное запоминающее устройство, фактически низводя производительность хранилища в оперативной памяти до соответствующих значений в традиционных базах данных на ДЗУ, и, следовательно, неприемлем для быстродействующих хранилищ. Это означает, что переполнения оперативной памяти в распределенном хранилище следует избегать.

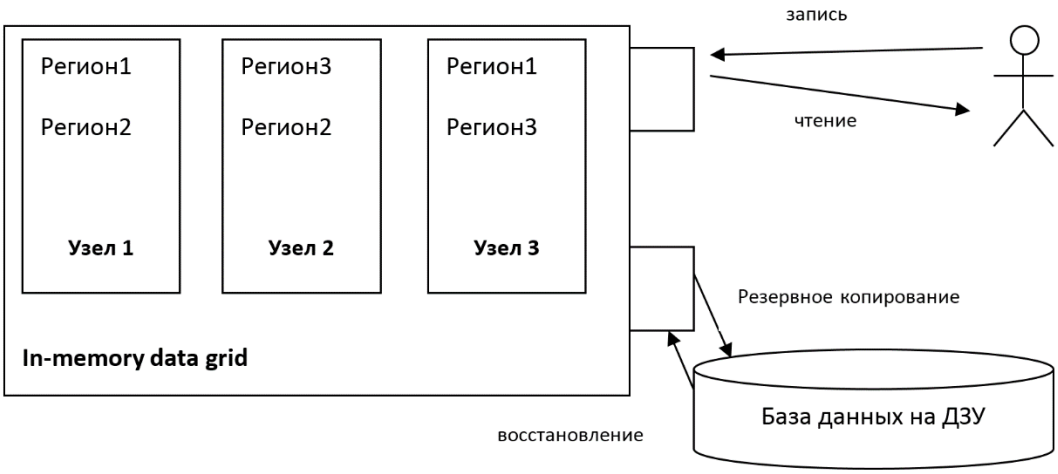

Рис. 3. Пример отказоустойчивого хранилища данных в оперативной памяти с резервным копированием на долговременное запоминающее устройство

Для предотвращения такой ситуации предложено несколько методов очистки хранимых данных:

а) Устаревание данных (*data expiration*) [12]: каждому объекту в кэше назначается заранее заданное время жизни, по истечению которого объект удаляется. Такая политика позволяет держать в оперативной памяти только актуальные данные. Время, в течение которого та или иная запись хранится в оперативной памяти, может быть увеличено путем перезаписи (обновления) этого объекта. Добавление в систему реплики на постоянных носителях позволит в этом случае не удалять данные из хранилища, оставляя архивные копии устаревших данных на долговременных запоминающих устройствах.

б) Изгнание данных (*data eviction*) является аварийным механизмом предотвращения переполнения оперативной памяти. Оно состоит в том, что для каждого узла хранилища определяется критический порог использования памяти, по достижению которого часть хранимых данных удаляется из основной памяти. При этом, в соответствии с заранее выбранной стратегией выбора, удаляются наиболее старые (давно обновленные), либо наиболее редко используемые данные. Отдельные стратегии предусматривают также удаление наиболее объемных данных.

Оба вышеуказанных подхода могут применяться совместно.

Отсутствие поддержки значительной части функциональности SQL

SQL – общепризнанный стандарт языка запросов к базам данных. Подавляющее большинство существующих приложений, работающих с традиционными базами данных, используют именно его. Однако ни в одной из существующих

реализаций распределенных хранилищ в оперативной памяти нет полноценной поддержки *SQL* запросов, что приводит к увеличению стоимости перехода существующих систем к использованию *IMDG*. В частности, сложность выполнения многих задач поиска и анализа данных увеличивается при отсутствии поддержки распределенных объединений (*distributed joins*) в указанном классе хранилищ. Это приводит к необходимости выполнять такие объединения на стороне клиента путем запросов к разным регионам по одному ключу, и, как следствие, к потере производительности и перегрузке сети передачи данных. Существующей альтернативой является использование хранимых на узлах кластера функций, в которых при помощи клиентского кода можно описать логику объединения, однако это требует наличия собственной хранимой процедуры для каждого вида объединения и также ведет к высокой нагрузке внутренней сети передачи данных кластера.

Не стандартизированный язык запросов

Часть *IMDG* предоставляют поддержку объектного языка запросов (*Object query language, OQL*), однако понимание функционального наполнения и синтаксиса этого языка различается от одной коммерческой реализации к другой. Таким образом, попытка перехода к иной реализации распределенного хранилища ведет к необходимости существенного изменения используемых запросов, что повышает стоимость такого перехода и порождает дополнительные риски. Кроме того, отсутствие стандартизированного языка запросов не позволяет напрямую сравнить производительность различных хранилищ применительно к конкретному коммерческому приложению без существенных затрат, связанных с изучением особенностей нового языка запросов и переносом необходимого функционала на новый язык.

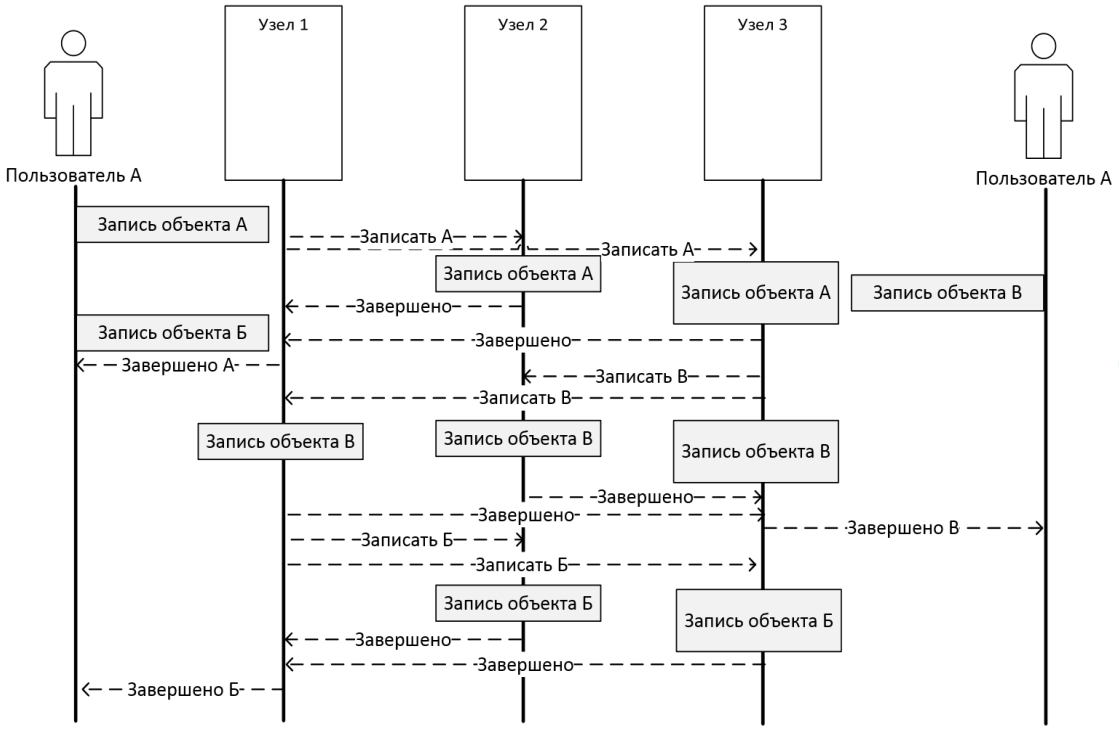

Рис. 4. Сценарий одновременной модификации данных в распределенном хранилище данных с блокировками

Блокировки

Для сохранения консистентности модифицируемых данных в современных распределенных хранилищах используется механизм блокировок. Он предусматривает запрет (блокировку) модификации данных до получения подтверждения о завершении предыдущей операции от всех узлов хранилища. Пример работы механизма блокировок показан на рис. 4. На рисунке показана ситуация, когда несколько пользователей выполняют запросы на модификацию хранимых данных в распределенном хранилище с полной репликацией, т.е. копии данных в котором хранятся на каждом узле распределенной системы, обеспечивая высокую доступность и отказоустойчивость. В описанном сценарии этот механизм существенно снижает производительность хранилища при частых операциях записи, т.к. несколько различных запросов, связанных с модификацией данных, в такой модели не могут выполняться одновременно, образуя очередь блокировки. Возможным подходом к решению этой проблемы могло бы быть использование программной транзакционной памяти (*software transactional memory, STM*), что позволило бы синхронизировать хранимые данные без блокировок.

Выводы

Несмотря на отмеченные особенности, рассмотренный тип хранилищ имеет существенное преимущество в виде малого времени доступа к данным. Недостатки *IMDG* могут быть частично устранены путем создания распределенных хранилищ данных в оперативной памяти, поддерживающего эффективные операции по распределенным слияниям таблиц, с полнофункциональным стандартизированным языком запросов и возможностью динамической загрузки классов. При решении этой задачи необходимо учитывать, что современные технологии передачи данных могут быть медленными по сравнению с обработкой данных в оперативной памяти. К примеру, сеть, построенная по технологии *InfiniBand* с номинальной пропускной способностью 40 Гбит/сек, показала 3 Гбит/сек на узле во время выполнения секционирования данных при помощи хэш-функции. При выполнении в оперативной памяти, секционирование на несколько тысяч частей выполняется со скоростью, приблизительно равной пропускной способности оперативной памяти [13, 14]. Следовательно, алгоритм объединения таблиц для хранилищ в оперативной памяти должен минимизировать пересылки данных по сети.

Метод путевого объединения (*track join*), представленный в [15], минимизирует количество пересылок кортежей по сети. Основная идея этого метода состоит в определении цели отправки каждого конкретного кортежа. Использование путевого объединения существенно снижает использование сетевых ресурсов по сравнению с другими известными методами выполнения объединений распределенных таблиц. Однако он ориентирован на использование в реляционных базах данных, где хранимые кортежи строго типизированы и фактический объем хранимых данных в разных строках одной таблицы отличается несущественно. В отличие от них, *IMDG* хранят объекты, способные, в свою очередь, содержать другие объекты. Таким образом, объем хранимых данных в разных кортежах одной таблицы может отличаться на несколько порядков (а, теоретически, эта разница может быть и больше). Поэтому простой подсчет количества пересылаемых кортежей, используемый методом путевого объединения, не дает объективной количественной оценки объема пересылаемых данных. Эту проблему может решить модификация метода путевого объединения путем замены единицы измерения пересылаемых данных на объем памяти, занимаемой каждым пересылаемым объектом, представленная в [16].

Список литературы

1. V. Turner. The digital universe of opportunities. Research & analysis by IDC // Infobrief of EMC corporation – 04.2014 – 16 p.

2. P. Boncz, S. Manegold, M. Kersten. Database architecture optimized for the new bottleneck: Memory access // VLDB journal – 12.2000 – P. 231-246.

3. H. Plattner. A common database approach for OLTP and OLAP using an in-memory column database // Proceedings of the 2009 ACM SIGMOD International Conference on Management of data – 2009 – P. 1-2.

4. Gupta M. K., Verma V., Verma M. S. In-Memory Database Systems - A Paradigm Shift. // International Journal of Engineering Trends and Technology (IJETT) – 12. 2013– P. 333-336.

5. R. Cattell. Scalable SQL and NoSQL Data Stores // SIGMOD Record – 12.2010 (Vol. 39, No. 4) – P. 12-27.

6. J. Gray. The Transaction Concept: Virtues and Limitations.// Proceedings of the 7th International Conference on Very Large Databases – 1981 – P. 144—154.

7. D. Pritchett. BASE: an ACID alternative // Queue - Object-Relational Mapping Volume 6 Issue 3 – 06.2008 – P.48-55.

8. Williams, J.W., Aggour, K.S., Interrante, J., McHugh, J., Pool, E. Bridging high velocity and high volume industrial big data through distributed in-memory storage & analytics. // Big Data, 2014 IEEE International Conference on – 10. 2014 – P. 932 - 941.

9. N. Ivanov. In-Memory Database vs. In-Memory Data Grid: Revisited // GridGain Blog. [Электронный ресурс] – 06.2014 – Режим доступа: http://gridgain.com/in-memory-database-vs-in-memory-data-grid-revisited.

10. K. Birman, D.Freedman, Q. Huang, P. Dowell. Overcoming CAP with consistent soft-state replication // IEEE Computer – 2012 – P. 50-58

11. P. Denning. Thrashing: Its causes and prevention // Proceedings AFIPS, Fall Joint Computer Conference – 1968 – P. 915–922

12. B. Babcock, S. Babu, M. Datar, R. Motwani, J. Widom. Models and issues in data stream systems // Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems – 2002 – P. 1-16.

13. M.-C. Albutiu, A. Kemper, and T. Neumann. Massively parallel sort-merge joins in main memory multi-core database systems. // PVLDB, 5(10) –2012 – P. 1064-1075.

14. C. Balkesen et al. Multicore, main-memory joins: Sort vs hash revisited. // PVLDB, 7(1) – Sept. 2013 – P. 85-96.

15. O. Polychroniou, R. Sen and K. Ross. Track join: distributed joins with minimal network traffic. // SIGMOD Conference – 2014 – P. 1483-1494.

16. О. Бузовский, А. Подрубайло. Методы и алгоритмы объединения таблиц в распределенных хранилищах данных в оперативной памяти // Вестник КПИ. Информатика, управление и вычислительная техника. Выпуск 60. – 01.2015 – С.73-83.

Статью представлено к публикации 2.06.2015