УДК 004.052

А.Л. БОЙКО

Таврический национальный университет им. В.И. Вернадского, Украина

АНАЛИЗ МНОГОВЕРСИОННЫХ АРХИТЕКТУР ДЛЯ ПОВЫШЕНИЯ ГАРАНТОСПОСОБНОСТИ СИСТЕМ ХРАНЕНИЯ ДАННЫХ

Анализируются проблемы повышения надежности и производительности систем хранения данных, основанных на многоверсионных технологиях. Предлагается архитектура отказоустойчивого FT-сервера, базирующаяся на идее создания промежуточного программного обеспечения, осуществляющего логику транслирования SQL-запросов в диалектные формы. Формулируются задачи исследований, связанных с оптимизацией режимов работы FT-сервера, сопоставлением показателей производительности диверсных SQL серверов в различных конфигурациях, созданием настраиваемого промежуточного программного обеспечения.

системы хранения данных, гарантоспособность, надёжность, производительность, многоверсионность, отказоустойчивость

Введение. Постановка задачи

Для улучшения характеристик систем хранения данных возможно применение достаточно широкого спектра различных технологических приемов, решающих частные задачи и, как правило, требующих определенных затрат. Наиболее важ-ными характеристиками SQL серверов являются показатели надежности и производительности. Выбор конкретной Database Management System (DBMS) определяет базовые характеристики системы, которые могут быть в значительной степени улучшены благодаря использованию специальных механизмов. Специфика разра-батываемой системы определяет расстановку приоритетов для каждой аппаратно-программной платформы: когда наиболее важно минимизировать время отклика, применяются схемы повышения производительности, возможно, за счет некоторого ухудшения показателей надежности; для гарантоспособных систем необходимо обеспечить требуемый уровень достоверности получаемых результатов, показателей безотказности, готовности, отказоустойчивости и др. Наиболее сложная ситуация возникает, когда оба критерия важны, но улучшение показателя надежности системы приводит к падению производительности.

Исследования, проведенные в [1], показали, что во многих из существующих DBMS могут возникать очевидные (self-evident) и неочевидные (non self-evident) сбои, связанные с неверным представлением результатов выполнения SQL-запроса. При очевидных сбоях возникает исключительная ситуация, которая может быть обнаружена и обработана одним из механизмов репликаций [2, 3].

В работе [1] был произведен анализ ошибок четырех распространенных DBMS, который показал, что процентное соотношение отказов к общему числу сбоев варьируется в пределах от 13% (для MSSQL) до 21% (для PostgreSQL). Наибольшее число сбоев носит неочевидный характер, приводящих к получению неверных данных и не выявляемых программным обеспечением клиента. Механизм репликации может быть полезен только в случае обнаружения очевидных аварийных отказов и не способен предотвратить возникновение неочевидных сбоев.

Для улучшения надежности и парирования физических дефектов сложных аппаратных платформ, а также для защиты от дефектов программного обеспечения, приводящих к возникновению неочевид-

ных сбоев, может применяться версионная избыточность или принцип многоверсионности [4, 5].

Результаты выполнения скриптов с использованием диверсных конфигураций SQL серверов показали, что такие системы позволяют определять около 94% всех возможных сбоев (в т. ч. и неочевидных), что является весьма высоким показателем.

Цель данной работы – анализ архитектурных методов повышения надежности и производительности систем хранения данных, основанных на многоверсионных технологиях.

Анализ многоверсионной архитектуры систем хранения данных

Ограничение, связанное с невозможностью обнаружения неочевидных сбоев при выполнении операций чтения/записи одним сервером, может быть преодолено путем реализации комплексных систем, включающих различные Off-the-shelf серверы (O-серверы), и представляющихся клиенту как единый SQL сервер [6] (рис. 1).

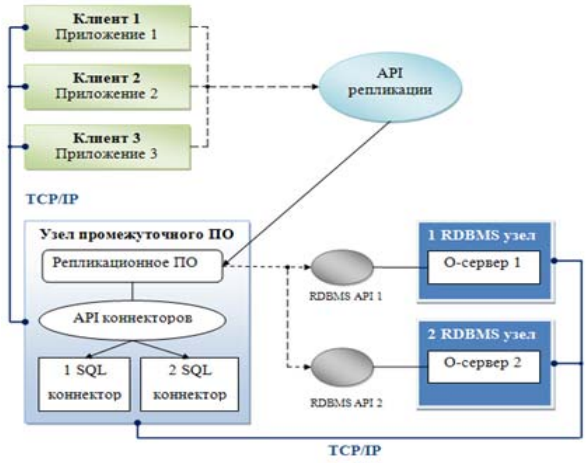

Рис. 1. Схема отказоустойчивого FT-сервера

Несколько клиентских узлов выполняют различные приложения, остальные узлы формируют ПО сервера (промежуточное ПО, RDBMS 1, RDBMS 2) или Fault Tolerant-узел (FT-узел). К разделяемым компонентам относится Репликаци-онное ПО, два вида коннекторов для сервера 1 и 2, содержащих правила перефразирования запросов. Каждый коннектор реализует собственный API интерфейс, используемый репликационным ПО, для поддержания API промежуточного программного обеспечения, через который клиент взаимодействует с FT-сервером.

Дальнейшим усовершенствованием данной архитектуры будет обеспечение возможности проведения диверсифицированных репликаций посредством middleware.

Основная задача middleware состоит в распространении запросов, генерируемых прило-жениями клиента, для всех SQL серверов, состав-ляющих отказоустойчивый диверсный SQL сервер. Результаты выполнения запросов обрабатываются и отправляются ПО клиента.

Для формализации языка структурных запросов (SQL) был разработан ряд стандартов (SQL-92, SQL-99), однако большинство современных DBMS обладают набором дополнительных нестандартизованных возможностей, реализация которых обеспечивается уникальным синтаксисом, не имеющим поддержки в DBMS других произво-дителей.

Для эффективного использования SQL сервера в диверсной конфигурации необходимо преодолеть ряд ограничений, накладываемых необходимостью применения различных SQL диалектов.

Серверы, обладающие значительным количеством пользователей, как правило, гарантируют совместимость со стандартом SQL-92, описывающим базовые типы запросов и позволяющий в большинстве случаев портировать исходный код запросов на другие программные платформы без внесения модификаций в исходные тексты.

В проектируемой системе приложение клиента может использовать один из видов SQL диалектов и изначально ориентироваться на работу с конкретной DBMS. Промежуточное ПО должно быть прозрачно для клиента: его основная функция заключается в перехвате запросов клиента, транслировании его в один из поддерживаемых видов SQL диалектов,

конкурентном выполнении запросов диверсными компонентами FT-сервера и возвращении результатов клиенту.

Промежуточное ПО должно реализовывать функциональность коннекторов, осуществляющих трансляцию между синтаксисом запросов клиента и сервера на основе предопределенных правил. Эти правила могут быть сформулированы для каждого типа SQL выражений и для каждого диалекта.

Анализ диалектных языков

В работе [6] был проведен анализ диалектных языков для DBMS Interbase 6.0, Firebird 1.0, PostgreSQL 7.0 и PostgreSQL 7.2 на основе известных сообщений об ошибках. Были сформулированы некоторые общие правила для перефразирования запросов (например, исполь-зование представлений (View) моет быть заменено исполнением размещенных процедур (Stored Procedure) или временных таблиц (Temporary Table)). Исследования ошибок названных DBMS показали, что 80% запросов чтения и 60% DDL выражений могут быть заменены корректными логическими эквивалентами.

Главными факторами деградации производительности при включении перефразирования являются: задержки middleware при сравнении результатов выполнения запросов, использование механизмов транзакции, выполнение SELECT запроса после модификации данных, пере-фразирование выражений и пр. [6].

В работе [2] рассмотрены другие факторы падения производительности FT-сервера (например, обеспечение консистентности данных, перефразирование с использованием более чем одного правила). Некоторым упрощением при исследовании влияния задержек, вносимыми механизмом реализации диверсности данных, является выполнение одного и того же запроса повторно вместо использования логического эквивалента, что обусловлено отсутствием ПО для автоматизации данной задачи.

В диверсной конфигурации FT-сервера применение перефразирования запросов необхо-димо только в случае обнаружения несовпадения результатов выполнения SQL выражения серверами (для SELECT запросов); в случае поступления выражений, модифицирующих содержимое БД, всегда осуществляется выполнение логически эквивалентного запроса. Очевидно существенное повышение производительности системы по сравнению с сервером в недиверсной конфигурации, поскольку перефразирование выполняется на меньшем подмножестве запросов.

Диверсификация данных должна обеспечить высокий уровень определения неочевидных сбоев, в некоторых случаях недоступный при использовании диверсности дизайна [6]. Основным преимуществом данного подхода является относительно низкая стоимость реализации по сравнению с решениями, основанными на диверсности дизайна. Диверсификация данных может быть использована как совместно с различными версиями DBMS, так и отдельно.

Режимы работы многоверсионной архитектуры

В [1, 6] был проведен анализ работы SQL сервера в диверсной конфигурации с использованием различных режимов ‘slowest response’, ‘fastest response’, ‘optimistic response’.

Режим ‘slowest response’ характеризуется временем обработки запросов наиболее медленного сервера, что вызвано необходимостью верификации полученных результатов.

Наиболее важным нефункциональным требованием к SQL серверам является требование к производительности, которая может быть значи-тельно увеличена при выборе режима ‘fastest response’ для работы SQL сервера в диверсной конфигурации.

Характерной особенностью данного режима является сокращенное время ожидания результата выполнения SQL запросов: первый полученный результат сразу передается ПО клиента. Верификация полученных результатов производится при накоплении некоторого количества ответов серверов до завершения выполнения транзакции.

Режим ‘optimistic response’ во многом аналогичен режиму ‘fastest response’, но имеет некоторые особенности: в нем не производится сравнение результатов выполнения запросов различными серверами, а также реализуется функция пропуска запросов выборки [1].

‘Fastest response’ обеспечивает трехкратное преимущество относительно наиболее быстрого Interbase сервера при выполнении 10000 транзакций. Падение производительности, обусловленное при-менением ‘slowest response’, составило 40% отно-сительно работы самого медленного PostgreSQL сервера, однако данный режим обеспечил возможность верификации обрабатываемых данных [1].

В среднем наибольшей уровень производительности обеспечивает сервер в диверсной конфигурации (PostgreSQL и Interbase). В работе сервера была применена опция пропуска запросов (skip), при этом 70% запросов выборки данных была отработана DBMS Interbase, в то время как PostgreSQL выполнил 51% запросов [6] .

Диверсификация позволяет улучшать показатели надежности и производительности отказоустойчивых серверов, что невозможно в гомогенных конфигурациях. Значительный прирост в произво-дительности объясняется оптимизацией различных DBMS на выполнение специфических типов запросов. Работа каждого сервера комплиментарна по отношению к другому серверу, что позволяет улучшить показатели производительности и надежности за счет диверсификации.

Разработка экспериментальной многоверсионной системы

В настоящее время производится разработка экспериментальной системы, создается имитационная модель FT-сервера, которая позволит проверить полученные в [3, 4] результаты. Планируется адаптировать модель к работе FT-сервера, состоящего более чем из двух диверсных DBMS, будут проведены измерения с учетом новых возможностей конфигурирования системы.

Модель позволит определить оптимальную конфигурацию FT-сервера в зависимости от предъявляемых требований: наиболее производительную пару DBMS, а также конфигурация, обеспечивающая наилучший уровень надежности. Входными данными для модели будут значения времени выполнения SQL-запросов для различных DBMS, указанные в [5].

Также планируется получить исходные данные для распространенной DBMS MySQL, провести серию экспериментов по выполнению указанных в [4] SQL запросов с целью выявления неочевидных сбоев, а также определить основные характеристики FT-сервера, состоящего из DBMS MySQL и других видов СУБД.

Промежуточное ПО FT-сервера включает функциональность для обнаружения, экранирования и устранения сбоев. Механизм определения очевидных сбоев аналогичен методу, применяемому в недиверсных серверах (с помощью сообщений об ошибках, тайм-аутов на выполнение операций). Диверсификация предоставляет возможность к обработке неочевидных сбоев в работе О-серверов.

Механизм экранирования ошибок может быть осуществлен мажоритарно-резервированной системой; правильным считается результат, полученный большинством серверов. Также необходимо определить давший сбой сервер и провести действия по его восстановлению.

Важной частью при создании промежуточного ПО является разработка формального описания системы на языке B нотаций, что позволит проводить строгий математический анализ разрабатываемой системы [1].

Выводы. Направления дальнейших исследований

В данной статье проведен анализ многоверсионной архитектуры системы хранения данных, предложена концепция создания промежуточного программного обеспечения, являющегося составной частью FT-сервера, а также определены свойства и исходные данные имитационной модели FT-сервера.

При разработке промежуточного ПО актуализируется ряд вопросов, связанных с особенностями реализации middleware. Важными направлениями в исследованиях являются:

1) демонстрация возможности практической реализации автоматического транслирования запро-сов из диалектных форм в стандартизованные представления (ANSI/ISO SQL);

2) дальнейшее исследование и оптимизации режима ‘fastest response’;

3) расширение набора исследуемых видов DBMS на наличие ошибок;

4) сопоставление показателей производительности диверсных SQL серверов в различных конфигурациях;

5) создание настраиваемого промежуточного программного обеспечения, предоставляющего клиенту право выбора приоритета функционирования SQL сервера. Возможно наделение middleware способностью интеллектуального принятия решения о необходимости использования того или иного режима работы сервера в зависимости от сложности передаваемого запроса и предыстории обработки подобных типов запросов.

Литература

1. Dondolossa G. Formal Methods in the development of safety critical knowledge - based components // ACM. – 2005. – P. 23-35.

2. Patino-Martinez M., Jimenez-Peris R., Kemme B., Alonso G. Consistent database replication at the middleware level, ACM Transactions on Computing Systems, 2005. – P. 375-423.

3. Bernstein P.A., Hadzilacos V., Goodman N. Concurrency Control and Recovery in Database Systems, 1987. – 370 p.

4. Gashi I., Popov P., Stankovic V. On designing dependable service with off-the-shelf SQL servers. – London, 2004. – 24 p.

5. Stankovic V., Popov P. Improving DBMS Performance through Diverse Redundancy. – London : Centre for Software Reliability, 2006. – 11 p.

6. Gashi I., Popov P. Rephrasing Rules for Off-The-Shelf SQL Database Servers. - London : Centre for Software Reliability, City University, 2006. – 10 p.

Поступила в редакцию 28.01.2008

Рецензент: д-р техн. наук, проф. В.С. Харченко, Национальный аэрокосмический университет им. Н.Е. Жуковского "ХАИ", Харьков.