В. М. Ничипорчук, В. В. Лопаткин

РАЗРАБОТКА РЕШЕНИЯ ПО ОБЕСПЕЧЕНИЮ ВЫСОКОЙ ДОСТУПНОСТИ И БАЛАНСИРОВКИ НАГРУЗКИ В РАСПРЕДЕЛЕННЫХ И ВЫСОКОНАГРУЖЕННЫХ СИСТЕМАХ В СФЕРЕ ЭЛЕКТРОННЫХ ПЛАТЕЖЕЙ

На сегодняшний день все большее распространение получают сервисы электронной оплаты, электронных магазинов, электронных торгов и интернет-банкинга. Важнейшими факторами, обеспечивающими успешное выполнение бизнес-модели организации, являются гарантии доступности данных, их целостности, а также быстрота доступа к ним. Одним из подходов к решению этой проблемы является применение систем обработки транзакций — OLTP-систем.

В системе обработки транзакций единичное выполнение бизнес-приложения производит единственную транзакцию. Конечные пользователи имеют онлайн-доступ к системе и корпоративным данным, в соответствии с которыми и производят транзакции. В среде обработки транзакций множество пользователей неоднократно производят похожие транзакции и требуют быстрого отклика по каждой транзакции. Примеры подобных пользователей — это служащие по обработке заказов, служащие по бронированию авиабилетов, банковские служащие. Они занимают общую среду программ и данных. Типичная система обработки транзакций характеризуется следующими качествами:

- множество конечных пользователей выполняют похожие транзакции, обращаясь к общим базам данных и файлам;
- система может устанавливать расписание выполнения транзакций в соответствии с атрибутами приоритетов;
- транзакции вызываются в режиме он-лайн, получают на вход информацию и выдают результат в режиме реального времени;
- транзакции разработаны с учетом выполнения условий удобного пользовательского интерфейса и быстрого времени отклика [1].

Обработка транзакций — это эффективное решение в случае, когда конечным пользователям требуется совершать следующие действия:

- обрабатывать отдельные порции информации в любое время в непредсказуемых разделах и последовательностях;
- получать доступ к корпоративным данным, которые были обновлены с целью соответствия последним транзакциям;
- осуществлять мгновенные изменения корпоративных данных и применение этих изменений на сервере [1].

Пример на рис. 1 показывает систему обработки заказов.

Основные функции, которые должны поддерживаться в системе обработки транзакций:

- обработка в режиме реального времени;
- высокая доступность;
- обеспечение требуемого уровня отклика;
- низкая стоимость (с точки зрения вычислительных ресурсов) каждой транзакции;
- доступ и обновление общих ресурсов при сохранении целостности [1].

Базовая конфигурация системы пакетной обработки данных представляет собой:

- компонент, связывающий терминалы или рабочие станции;
- систему управления баз данных;
- программы и приложения, исполняющие бизнес-логику.

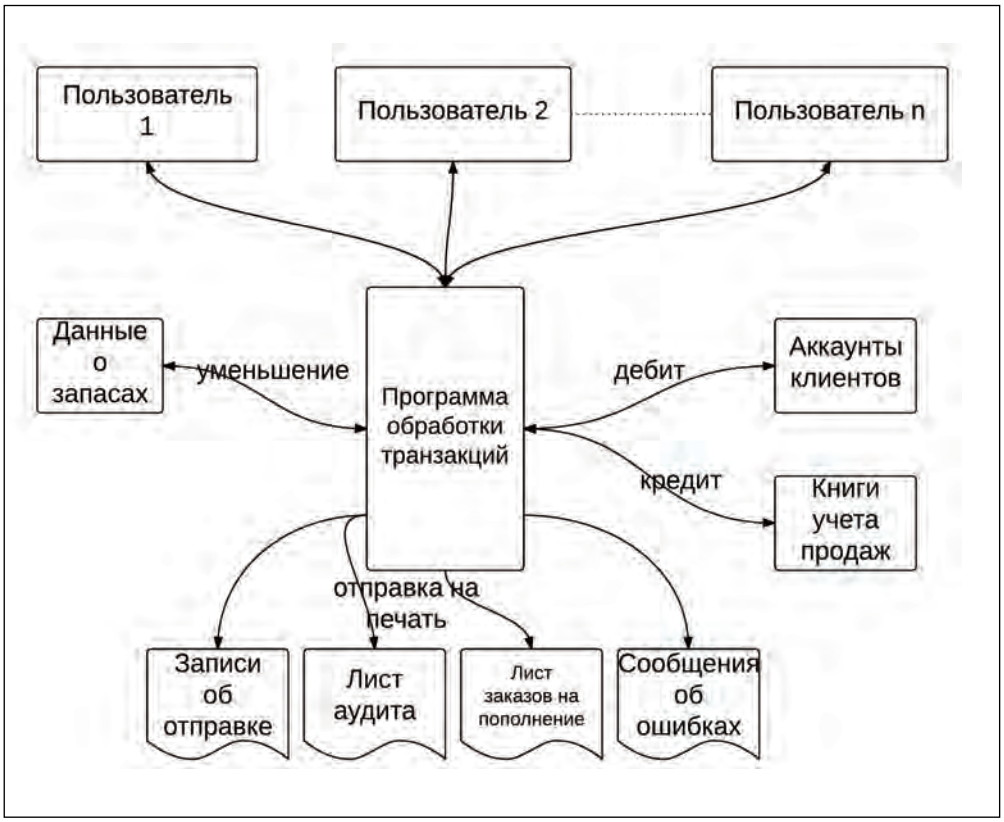

Рис. 1. Система обработки заказов

На рис. 2 изображена базовая схема конфигурации системы обработки транзакций.

С точки зрения хранилища – СУБД, требования выполняются применением схем по обеспечению балансировки нагрузки, репликации, отказоустойчивости и восстания после сбоя. Серверы баз данных могут работать совместно, чтобы второй сервер мог в кратчайшие сроки взять на себя управление в случае, если главный сервер выйдет из строя (высокая доступность), или чтобы позволить нескольким компьютерам обслуживать выдачу одной и той же информации (балансировка нагрузки). В идеале, серверы баз данных могут работать вместе незаметно для конечного пользователя. Веб-серверы, раздающие статические веб-страницы, могут быть объединены без особых усилий, исключительно с помощью балансировки веб-запросов между несколькими физическими машинами. В частности, могут быть применены базы данных, доступные только для чтения. К сожалению, большинство серверов баз данных работает в режиме как чтения, так и записи, комбинировать подобные серверы значительно труднее. Это происходит оттого, что, в то время как данные, размещаемые только для чтения, следует помещать на каждый сервер лишь один раз, данные, доступные для записи, должны быть распределены на все серверы, чтобы последующие запросы чтения к этим серверам возвращали последние изменения.

Эта проблема синхронизации является фундаментальной трудностью для любого объединения серверов. Из-за того, что не существует одного-единственного решения, которое исключает влияние проблемы синхронизации для всех случаев использования, существует множество решений данной проблемы. Каждое решает проблему по-своему и минимизирует нежелательные сценарии для некоторой определенной задачи нагрузки.

Для решения поставленной задачи предлагается схема на основе открытого ПО PostgreSQL и Pgpool-II с использованием комбинации нескольких подходов для достижения максимальной эффективности и функциональности.

Первый подход — серверы «теплой» и «горячей» замены, использующие восстановление по временным меткам Point-In-Time Recovery (PITR).

Серверы должны поддерживать текущее состояние базы данных, считывая поток от лога транзакций, который ведет главный сервер. Если главный сервер выйдет из строя, сервер замены уже будет содержать почти все данные, что и главный сервер, и сможет быстро взять на себя функции главного сервера. Этот процесс асинхронный и может быть осуществлен лишь для всего сервера.

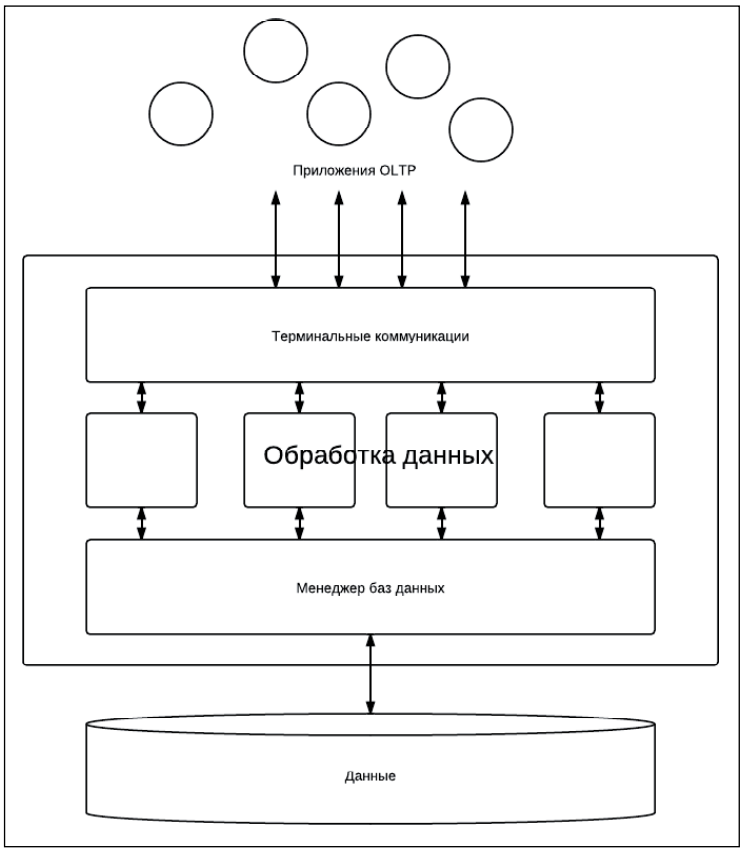

Рис. 2. Схема конфигурации системы обработки транзакций

Схема восстановления по PITR может быть применена с использованием отправки логов на уровне файлов или потоковой репликации либо комбинацией обоих методов [2].

Второй подход — репликация с использованием промежуточного программного обеспечения на основе менеджмента SQL-запросов. При использовании данной схемы программа перехватывает каждый SQL-запрос и отправляет его на один или на все серверы. Каждый сервер обновляется независимо. Запросы на запись должны быть отправлены на каждый сервер таким образом, чтобы каждый сервер получил изменения. Однако запросы только на чтение могут быть отправлены лишь на один сервер, позволяя распределить нагрузку на чтение между всеми серверами.

Если отправлять запросы без изменений, то такие функции, как random(), CURRENT_TIMESTAMP и последовательности, будут иметь разные значения на разных серверах [3]. Это происходит оттого, что каждый сервер работает независимо, и из-за того, что транслируются на все серверы SQL-запросы, а не конечные значения срок и столбцов.

Если это неприемлемо, то либо менеджер запросов, либо само приложение должны отправлять эти запросы на один сервер, а затем копировать полученные значения на другие серверы. Другой способ — это использование данного метода репликации с традиционной схемой «главный—ведомый», чтобы запросы модификации отправлялись только на главный сервер, а на другие серверы распро-

странялись посредством традиционной репликации «главный—ведомый», а не с помощью менеджера запросов. Pgpool-II и Continuent Tungsten являются примерами подобного типа репликации [4].

Схема построения предложенной модели кластера изображена на рис. 3.

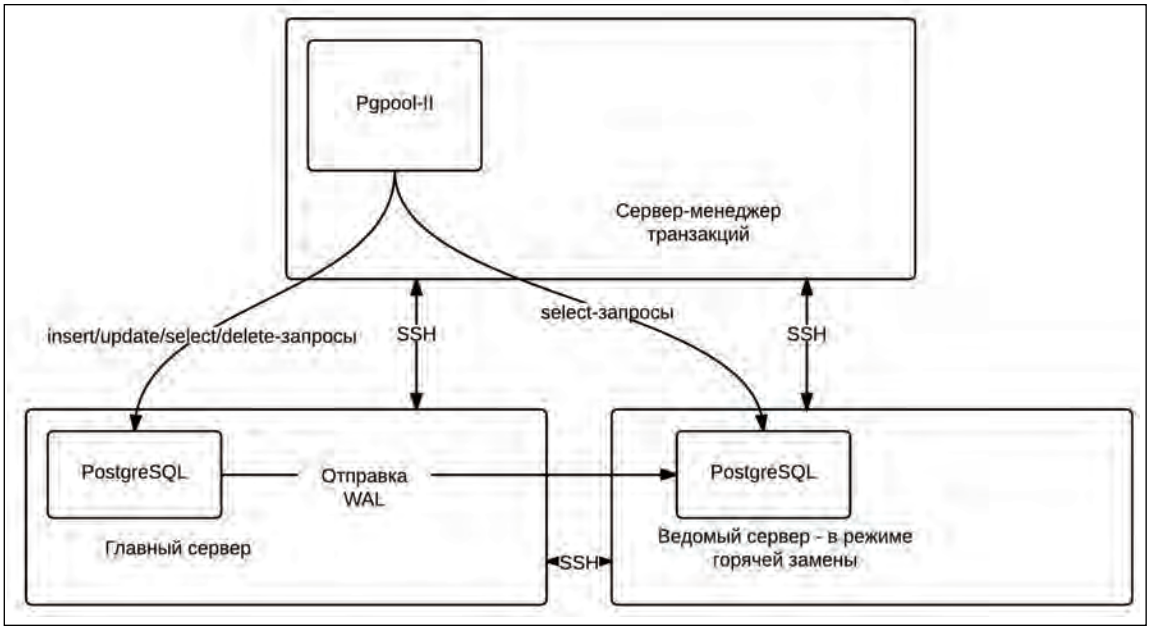

Рис. 3. Схема модели кластера БД

В данной схеме репликация между главным (master) и зависимым (slave) сервером осуществляется с помощью потоковой передачи лога транзакций, сами транзакции рассылаются на узлы с помощью менеджера транзакций Pgpool-II. Главный сервер действует в режиме как записи, так и чтения, в то время как ведомый сервер действует только в режиме чтения — с учетом этого транзакции и отправляются на узлы. Кроме того, зависимый сервер функционирует в режиме горячей замены таким образом, что в случае потери связи с главным сервером менеджер транзакций посылает соответствующую команду на зависимый сервер, по которой он (зависимый сервер) осуществляет процедуру восстановления, после чего функционирует в режиме как записи, так и чтения, тем самым обеспечивая доступ к данным как у главного сервера. В такой схеме зависимых серверов может быть несколько для достижения более высоких показателей надежности, а также для балансировки нагрузки (при чтении данных).

СПИСОК ЛИТЕРАТУРЫ:

1. Transaction Processing: Concepts and Products, GC33-0754-00. 1st ed. (Dec. 1990). International Business Machines Corporation, 1990.
2. *Smith G.* PostgreSQL 9.0 High Performance. Packt Publishing, 2010.
3. PostgreSQL Documentation. URL: http://www.postgresql.org/docs/9.1/static/high-availability.html (дата обращения 20.05.2012).
4. *Васильев А. Ю.* Работа с Postgresql: настройка, масштабирование. Creative Commons Attribution - noncommercial 2.5, 2010.