МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

МАТЕРИАЛЫ

Первой Всероссийской молодежной научной конференции

«МАТЕМАТИЧЕСКОЕ И ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ИНФОРМАЦИОННЫХ, ТЕХНИЧЕСКИХ И ЭКОНОМИЧЕСКИХ СИСТЕМ»

Томск, 17–18 мая 2013 г.

Под общей редакцией
кандидата технических наук И.С. Шмырина

Томск
Издательский Дом Томского государственного университета
2013

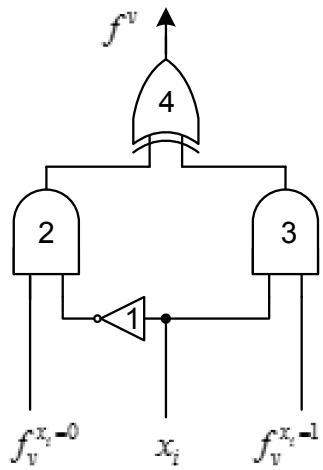

Рис. 4. Вентильная реализация формулы $f^v = \overline{x_i} f_v^{x_i=0} \oplus x_i f_v^{x_i=1}$

Возможен еще один подход к устранению лишних вершин в графах. Можно в процессе построения FreeBDD проверять дочерние функции на совпадения, построив для них ROBDD (ДНФ каждой из функций известны).

Предложены подходы к упрощению Free BDD-графов. В полученных по упрощенным графам комбинационным схемам неисправность задержки каждого пути обнаружима.

ЛИТЕРАТУРА

1. *Матросова А. Ю., Николаева Е. А., Румянцева Е. В.* Синтез логических схем, контролепригодных относительно неисправностей задержек путей // Известия высших учебных заведений. Физика. 2012. Т. 55, № 11. С. 114–116.

АНАЛИЗ ПОДХОДОВ К РАБОТЕ С ДАННЫМИ В СЕРВЕРНЫХ ПРИЛОЖЕНИЯХ

В. В. Томков

Томский государственный университет
E-mail: isirbis@gmail.com

Обработка и хранение данных – основная задача серверных приложений в рамках многозвенных (Multitier) и клиент-серверных (client-server) архитектур. Соответственно, от того, насколько эффективно серверы будут справляться с этой задачей, зависит эффективность всей Системы в целом. Так, например, если серверная части Системы имеет проблемы с производительностью на уровне обработки данных, то никакое, даже самое гениальное клиентское приложение, не сможет обеспечить приемлемый уровень производительности Системы.

Прежде чем начинать работу с данными, необходимо определиться с моделью ввода/вывода, которую будет использовать сервер для получения данных. Сейчас существует шесть основных моделей ввода/вывода, будут рассмотрены только че-

тыре из них. Оставшиеся две модели не будут рассмотрены в силу того, что в серверных приложениях их использование крайне нежелательно.

Одним из самых распространенных и простых подходов получения данных является модель мультиплексированного ввода/вывода (Multiplexing I/O). Она позволяет сократить время простоев до минимума за счёт блокирования не на самом вводе/выводе, а на системном вызове мультиплексора. На практике эта модель порождает требование вынесения обработки данных в отдельный поток либо создание алгоритмов, обеспечивающих быструю обработку данных в потоке, где происходит блокировка.

Существует ещё одна старая и проверенная модель ввода/вывода в потоках/дочерних процессах (One descr per thread or process). Она используется в некоторых работающих серверных приложениях уже несколько лет. Отличительной особенностью является простота в обработке данных, но за эту простоту приходиться платить увеличенными расходами на создание/удаление потоков/дочерних процессов. Стоит так же отметить тот факт, что мы можем лишиться простоты обработки данных, если потоки/процессы работают с разделяемыми ресурсами. В этом случае необходимо продумывать взаимодействия между потоками/процессами.

Следующие две модели обработки данных, ввод/вывод, управляемый сигналами (Signal-driven I/O), и асинхронный ввод/вывод (Asynchronous I/O), объединяет свойство – отсутствие блокировки. Каждая из них обладает определёнными ограничениями и проблемами. Первая модель ограничена языками программирования, где возможен доступ напрямую к сигналам ядра. Например, прямое использование данной модели в Java невозможно, и компонент серверного приложения, работающий с вводом/выводом становиться платформозависимый. Для другой модели должен существовать механизм, который бы уведомил процесс о том, что ввод/вывод завершён.

Одними из первых и простых способов обработки данных – это обработка в том же потоке, где происходит получение данных, и обработка в отдельном потоке. Данные модели не является предпочтительными в современных реалиях по ряду причин. Первая – доступные ресурсы используются не полностью. В многоядерных системах мы используем только одно ядро из нескольких. Вторая – увеличение накладных расходов на создание/удаление потоков. Стоит так же отметить, что при большом количестве одновременных соединений мы будем тратить практически все время только на переключение между потоками.

Другой подход обработки данных – это использование пула потоков. Он позволяет избавиться от недостатков предыдущих, но порождает новые проблемы, а именно проблемы синхронизации потоков.

Предыдущие подходы решали нужды на заре появления серверных приложений и, разумеется, не были рассчитаны на удовлетворение эволюционировавших нужд. Нужен был новый подход в обработке данных, и в такой ситуации, как нельзя кстати, подошла «асинхронная событийная модель». Идея модели заключается в том, что поведение сервера определяется событиями. Модель хорошо подходит для систем, где обработка события происходит очень быстро. Типичной ошибкой при использовании этого подхода является долгая обработка события. Дело в том, что при росте количества одновременных соединений число циклов обработки информации может заметно сократиться. На практике разработчик может столкнуться с несколькими проблемами. Первая – утечка памяти. Разработчику нужно очень внимательно следить за выделяемой памятью, иначе процесс будет пожирать все больше и больше памяти с каждым новым запросом. Вторая – обработка ошибок программы. Если

какой-либо обрабатывающий код выбросит исключение, то это приведёт к тому, что «положит» весь сервер целиком.

Другим подходом для обработки данных стала «модель акторов». Изначально она представляет собой математическую модель для параллельных вычислений [1]. Эта модель очень хорошо подходит для больших распределенных серверных приложений, где обработка информации может происходить разное количество времени. Несмотря на многие достоинства этой модели, разработчикам приходится самостоятельно искать решение проблем, возникающих на практике и связанных с ограничениями и допущениями модели акторов, её конкретной реализации и распределенных систем в целом [2].

Существует также целый класс подходов в обработке данных, который можно назвать «комбинированными модели». Такие модели создаются, когда разработчика не устраивают существующие модели по ряду причин. Они состоят из элементов уже рассмотренных систем и наследуют все их плюсы и минусы.

В некоторых системах можно обойтись без долговременного хранения данных, что позволяет хранить всю полученную информацию в памяти, но, в основном, местом хранения выступают базы данных, диски и различные структуры данных: хеш-таблицы, деревья и другие. Они являются очень узким местом в работе серверных приложений, т. к. дают дополнительные задержки.

Одним из первых способов хранения данных являются диски. Здесь мы храним всю информацию в том виде, в котором нам удобно, но платим за это тем, что все операции по работе лежат на нашей совести. При неправильной реализации мы можем получить систему, которая будет работать неимоверно долго из-за записи/чтения данных.

Этот подход очень удобен для хранения статической информации, которую не нужно изменять (изображения, страницы текста и другие).

Другим подходом для хранения данных являются реляционные базы данных. На текущий момент они наиболее распространены по причине хорошей производительности и отсутствия необходимости разработки своих методов хранения и работы с данными. Одной из проблем реляционных баз данных, является проблемы при работе с данными большого объёма и в системах с высокой нагрузкой (около 70 тысяч запросов в секунду).

Решением проблемы большой нагрузки в реляционных базах данных являются распределенные базы данных, ориентированные на работу в реальном времени. Как правило, такие БД поддерживают принцип NoSQL, в основе которого лежит теорема Брюера [3]. Они отличаются очень высокой производительностью на запись, гибкой схемой данных, осуществлением большинства операций в оперативной памяти. Однако некоторые из них имеют высокую задержку при случайном доступе.

Исходя из выше сказанного, предлагается использовать модель для работы с данными, представленную на рис. 1.

Модель состоит из следующих компонент: DataPool – отвечает за получение данных (компонент IOOperations) и добавление их в очередь (компонент QueueData). Компонент SlightProcessing работает в отдельном потоке, где забирает данные из очереди и осуществляет предварительную обработку данных, например, добавление в пул потоков, выбора сервера для обработки данных и т. д. Набор компонент, названных DataProcessing, отвечают за основную обработку данных. Они могут располагаться физически как на той же машине, где и происходит получение данных, так и на других. Для получения данных возможно использование как мультиплексированного, так и асинхронного ввода/вывода, а выбор модели хранения

данных зависит от задач, которые будет решать серверное приложение, использующее это решение.

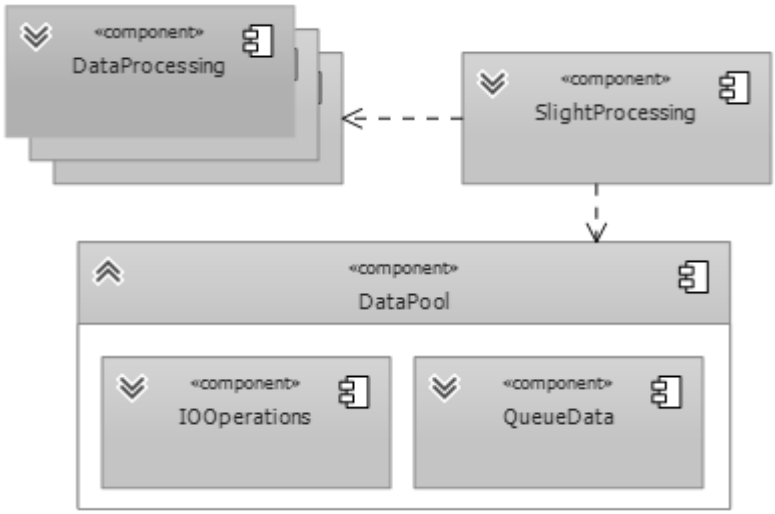

Рис. 1. UML диаграмма компонент для обработки данных в серверных приложениях

Предложенная модель сочетает в себе две модели обработки данных, а именно «модель акторов» и «асинхронную событийную модель». Данный подход позволяет частично решить проблемы утечки памяти и обработки ошибок, которые может допустить разработчик при использовании «асинхронной событийной модели», т. к. наиболее опасные места вынесены либо на отдельные машины, либо в отдельные потоки.

Необходимо заметить, что все основные данные для запросов должны находиться в памяти, а диски/БД используются только для записи.

ЛИТЕРАТУРА

1. *Хьюитт К., Бишоп П., Штайгер Р.* Универсальный модульный формализм акторов для искусственного интеллекта // International Joint Conferences on Artificial Intelligence, 1973.

2. *Цесько В. А.* Потоковая обработка данных с помощью модели акторов // Application Developer Days, Москва, Российская Федерация, 2012, 11–12 мая.

3. *Brewer E. A.* Towards robust distributed systems (abstract) // Proc. of the 19th Annual ACM Symposium on Principles of Distributed Computing, Portland, Oregon, United States, 2000, July 16–19. P. 7.

СИМУЛЯЦИЯ ТОРГОВЛИ ЦЕННЫМИ БУМАГАМИ НА МОСКОВСКОЙ БИРЖЕ С ПОМОЩЬЮ МОДЕЛЬНОГО ПОРТФЕЛЯ

С. Н. Якименко

Томский государственный университет
E-mail: truvor187@gmail.com

Введение

Последние десять лет российский фондовый рынок активно развивается. Ранее преобладали комнатная торговля и торговля, базируемая на телефонной линии.