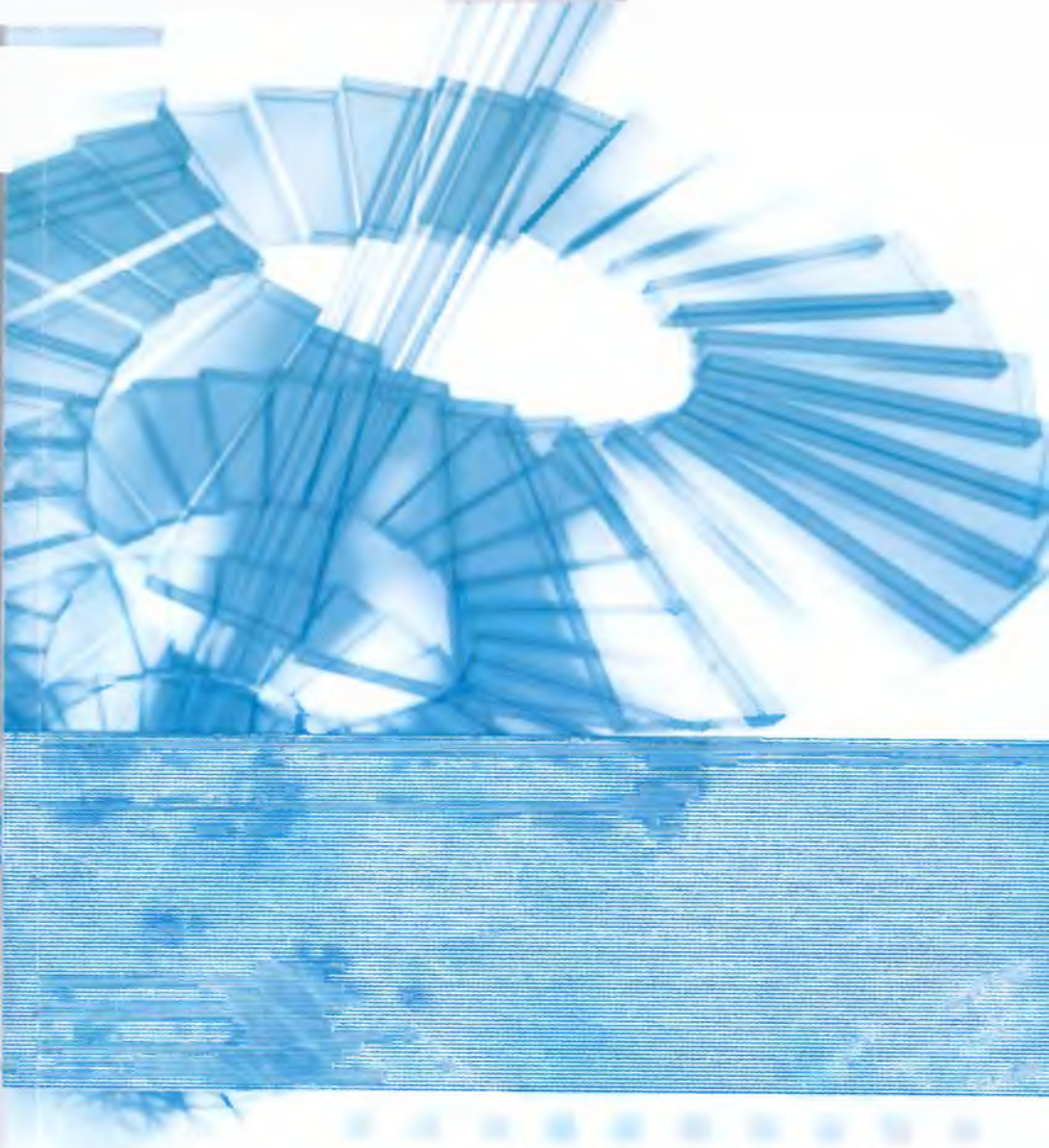

ИТММ-2009

«ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ И МАТЕМАТИЧЕСКОЕ МОДЕЛИРОВАНИЕ»

ЧАСТЬ 1

Томский государственный университет

Кемеровский государственный университет

Кемеровский научный центр СО РАН

Институт вычислительных технологий СО РАН

Филиал Кемеровского государственного университета
в г. Анжеро-Судженске

ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ И МАТЕМАТИЧЕСКОЕ МОДЕЛИРОВАНИЕ (ИТММ-2009)

Материалы VIII Всероссийской научно-практической конференции с международным участием

12–13 ноября 2009 г.

Часть 1

Издательство Томского университета

2009

УДК 519

ББК 22.17

И74

Информационные технологии и математическое моделирование (ИТММ-2009): Материалы VIII Всероссийской научно-практической конференции с международным участием (13–14 ноября 2009 г.). – Томск: Изд-во Том. ун-та, 2009. – Ч. 1. – 332 с.

И74

ISBN 978-5-7511-1929-4

В часть 1 вошли материалы секций «Вероятностные методы и модели», «Информационные технологии» и «Экономико-математические модели».

Для специалистов в области информационных технологий и математического моделирования.

УДК 519

ББК 22.17

Конференция проводится при поддержке Российского фонда фундаментальных исследований (проект № 09-07-06061-г)

ISBN 978-5-7511-1929-4

12. Спецификация JSR 286: Portlet Specification 2.0 [Электронный ресурс] // URL: http://jcp.org/en/jsr/detail?id=286 (дата обращения: 01.10.2009).
13. Официальный сайт разработчика Jetspeed [Электронный ресурс] // URL: http://portals.apache.org/jetspeed-2/ (дата обращения: 01.10.2009).
14. Официальный сайт разработчика Google Web Toolkit [Электронный ресурс] // URL: http://code.google.com/intl/be-BY/webtoolkit/ (дата обращения: 01.10.2009).

ОБ ИСПОЛЬЗОВАНИИ КЕША ПРЕДВАРИТЕЛЬНОГО ЧТЕНИЯ В ВЫСОКОНАГРУЖЕННЫХ ВЕБ-ПРИЛОЖЕНИЯХ

С. П. Кондратьев
Томский государственный университет

За последние годы всё больше и больше программистов сталкиваются с необходимостью написания приложений для Интернета. Эта платформа диктует свои правила, отличные от тех, которые используются при разработке локальных приложений [1]. Одним из таких фундаментальных отличий локального (десктопного) приложения от сетевого (веб) приложения является необходимость поддержки работы множества клиентов одновременно. В результате возникают проблемы и явления, не свойственные локальным приложениям, такие как огромные количества одновременных запросов, требущих данные из постоянного хранилища (базы данных). В этом случае пропускная способность приложения ограничивается пропускной способностью сервера базы данных, что во многих случаях нежелательно.

Существует два способа увеличения производительности веб-приложения, не зависящих от базы данных.

В качестве кардинального решения в веб-приложениях перестают использовать реляционную базу данных вообще. Вместо неё возможно использование хеширующих структур (таких, как Berkeley DB) напрямую. Данное решение позволяет отказаться от использования структурированных запросов (на языке SQL или аналогичных ему) и получать данные непосредственно из хешей. Вместо Berkeley DB, которая использует комбинацию хешей в памяти и на диске, иногда используется memcached, который полностью хранит данные в памяти. Данное решение позволяет получить очень высокую пропускную способность за счет потенциального отсутствия блокировок при чтениях. Однако данное решение имеет очевидный минус: разработчику приходится самостоятельно реализовывать все необходимые манипуляции над структурой, хранящейся в хеше.

Другим решением служит использование кеширующих механизмов между базой данных и приложением.

В системе объектного брокера Hibernate предполагается использовать кеш второго уровня для достижения необходимой производительности. Достоинство этого метода, такое как прозрачность для разработчика, является недостатком для сильнонагруженных систем – кеш автоматически сбрасывается при любых массовых обновлениях. Разработчик не может манипулировать механизмами устаревания данных в кеше. Поэтому

для реализации системы предоставления реклам в проекте Mo'Jiva было решено реализовать отдельный кеширующий механизм, основанный на принципе «предчтения».

Принцип предварительного чтения (read-ahead cache) используется во многих системах для получения данных большими частями. Это позволяет уменьшить время общения с базой либо локализовать его в одном временном промежутке. Обычно такое кеширование предполагает только доступ для чтения. Кроме того, необходим какой-то способ для обновления в кеше данных, которые были измены в базе данных.

Кеш основан на перехвате двух событий внутри механизма брокера Hibernate для языка Java и веб-серверов соответствующей спецификации J2EE [2].

Брокер позволяет расширять свой функционал путем перехвата событий. За загрузку объектов из базы данных отвечает событие LoadEvent, а за загрузку связанных коллекций – событие InitializeCollectionEvent. В данные события передаются типы объектов, их идентификаторы и требуемый уровень изоляции (минимальный: LockMode.NONE означает, что объект можно взять из любого источника без доп. блокировок, LockMode.READ означает, что объект должен быть взят из базы без блокировок, LockMode.WRITE требует взятие объекта из базы с блокировкой на запись [3]). Для работы кеша обработчики этих событий перегружены.

В обработке события LoadEvent [3] первоначально необходимо проверить тип доступа (кеш должен работать только уровня изоляции LockMode.NONE), а затем извлечь объект из уже заполненного кеша по типу объекта и его идентификатору.

Обработка события InitializeCollectionEvent [3] сложнее. Оно возникает, когда необходимо загрузить данные для связи «один-ко-многим» или «многие-ко-многим». В событие, кроме объекта-сущности, передается еще имя связи, которую необходимо инициализировать. При обработке события необходимо загрузить список объектов по требуемой связи. Для этого используется сложный идентификатор объекта-коллекции, состоящий из имени и идентификатора объекта, имени поля-связи и имени фильтра.

При обработке события LoadEvent может возникнуть ситуация, когда объект не найден в кеше. В этом случае мы предполагаем, что объекта не существует и в базе данных (таким образом, мы никогда не обращаемся к базе данных напрямую).

Кеш поддерживает как частичную инвалидацию объектов, так и полную.

Для частичной инвалидации используется принцип временных меток. Каждый объект содержит в себе время своей последней модификации. Это время обновляется за счет использования триггеров в базе данных. Кеш раз в определенный период времени запрашивает из базы все новые измененные объекты на основе этих меток. Обновление связей также приводит к обновлению временной метки у собственно объекта. Тем самым обеспечивается когерентность кеша с базой данных. Частичная инвалида-

ция также поддерживает удаление объектов. Для этого используется специальное значение временной метки.

При полной инвалидации соответствующие сущности полностью повторно вычитываются. Это имеет смысл, если количество соответствующих объектов в базе невелико, а манипуляции с ними сложно отследить.

При загрузке объекта в кеш сразу обрабатываются все его прямые связи («многие-ко-одному» и «многие-ко-многим» в прямую сторону). В результате этой обработки в кеш попадают обратные связи («один-ко-многим» и «многие-ко-многим» в обратную сторону). По этим связям все объекты загружаются «лениво» и хранятся в виде прокси-ссылок, что позволяет уменьшить объем кеша и дублирование данных в нем. Эти прокси-ссылки будут инициализированы при первом же обращении через событие LoadEvent.

В рамках такой системы кеширования с нижележащим хранилищем всегда общается только небольшое количество потоков (для реализации механизма синхронизации кеша с базой данных). При операциях чтения общение с базой не происходит. Это позволяет сильно уменьшить нагрузку на базу данных и избавиться от проблемы блокировок во время множества параллельных чтений/записи.

Плюсы такого подхода (в отличие от обычного кеша второго уровня) в отсутствии необходимости полной инвалидации кеша при получении обновленных данных из базы. При любых операциях, которые изменят содержимое в базе данных, также будут обновлены и временные метки, что позволит перечитать только измененные данные. Кроме того, никакие запросы клиентов к кешу не приведут к чтениям из нижележащей базы данных. Это позволяет использовать огромное количество чтений: в проекте Mo'Jiva такой кеш выдерживает порядка 10 000 транзакций-чтений в секунду при том, что используемый сервер баз данных PostgreSQL не способен выдать на имеющейся схеме более 500 транзакций в секунду. Еще одним плюсом такого кеша является тот факт, что он прозрачно кеширует все обращения по связям «многие-к-одному».

Таким образом, данный кеш позволяет, используя стандартные методы брокера, отдавать объекты из кеша при прямых запросах по идентификатору, при обращении к связанным объектам и при обращении к связанным коллекциям.

В качестве минусов использования такого кеша можно отметить тот факт, что в определенные моменты времени кеш не полностью отражает данные из базы (а имеет их предыдущие образы). Если критично использование именно последней версии объекта из базы, то возможен запрос объекта с уровнем изоляции LockMode.READ или выше. В этом случае кеш будет проигнорирован, а объект вычитан (возможно, повторно) из базы. В минусы этого подхода можно отнести и его непрозрачность для разработчика: необходимо полностью реализовать всю подсистему предваритель-

ного чтения объектов в память. Однако само использование кеша является прозрачным.

В результате использования такой реализации кеша удалось увеличить производительность подсистемы отдачи реклам в проекте Mo'Jiva в несколько раз: с 200 запросов в среднем в секунду при отсутствии такого кеша до 1000 запросов в среднем с ним (в пике система способна успешно обработать порядка 8000 запросов).

Таким образом, использование кеша предварительного чтения позволяет сильно уменьшить зависимость производительности и пропускной способности веб-приложения от сервера баз данных.

Литература

1. Ceri S., Fraternali P., Bongio A. et al. Designing Data-Intensive Web Applications. – USA: Elsevier Science, 2003.

2. Java Persistence API [Электронный ресурс] // URL: http://java.sun.com/javaee/technologies/persistence.jsp (дата обращения 01.10.2009).

3. HIBERNATE – Relational Persistence for Idiomatic Java // URL: http://docs.jboss.org/hibernate/stable/core/reference/en/html_single/ (дата обращения 01.10.2009).

ПЕРСПЕКТИВА РАЗВИТИЯ МЕТОДОЛОГИИ СОЗДАНИЯ ИНФОРМАЦИОННО-ИССЛЕДОВАТЕЛЬСКИХ ЭЛЕКТРОННЫХ ОЦИФРОВАНО-ФОРМАЛИЗОВАННЫХ БАЗ ДАННЫХ БОЛЬШИХ СИСТЕМ СФЕР ПОЗНАНИЯ И НАУКИ ПЛАНЕТЫ ЗЕМЛЯ НА УРОВНЕ КИБЕРГ-ИНТЕРНЕТ

С. В. Кондратьев

Филиал Кемеровского государственного университета в г. Анжеро-Судженске

За два последних десятилетия информационно-коммуникационные технологии стали неотъемлемой составляющей современного человека, общества и всей мировой цивилизации. Электронная информационная среда и её возможности предопределяют необходимость государственных структур и институтов гражданского общества, экономической и социальной сфер, науки и образования, культуры и образа жизни людей к широкому использованию в науке, производстве и быту. Ни у кого не вызывает сомнения тот факт, что движение к информационному обществу – это путь в будущее человеческой цивилизации. Именно это фиксирует Окинавская хартия глобального информационного общества, которая подписана руководителями семи ведущих стран и Президентом России в июле 2000 года [1]. Однако, в последние годы наметилась тенденция снижения возможностей ИТ-технологий по причине того, что Интернет уже достаточно перегружен, и в недалеком будущем его пользователи ощутят отсутствие рабочих ресурсов задействованных серверов.