Главный редактор - д.э.н., профессор К.А. Кирсанов
тел. для справок: +7 (925) 853-04-57 (с $11^{00}$ – до $18^{00}$)
Опубликовать статью в журнале - http://publ.naukovedenie.ru

**Игумнов Артём Олегович**
Igumnov Artyom Olegovich
Национальный исследовательский Томский политехнический университет
National Research Tomsk Polytechnic University
Аспирант/ Postgraduate
E-Mail: artishiro@gmail.com

**Сонькин Дмитрий Михайлович**
Sonkin Dmitry Mikhaylovich
Национальный исследовательский Томский политехнический университет
National Research Tomsk Polytechnic University
Ассистент/Assistant
Кандидат технических наук
E-Mail: sonkind@tpu.ru

05.13.01 – Системный анализ, управление и обработка информации

# Об одном из подходов к оптимизации высоконагруженных систем на примере системы диспетчерского управления таксомоторным парком

## One approaches to optimize highload systems on example of dispatching taxi system

**Аннотация:** Предметом этой статьи является проблема высокой нагрузки базы данных в условиях ограниченных аппаратных ресурсов. Приводится анализ структуры дата базы с точки зрения общих характеристик сущностей, и способ решения проблемы кэшированием определённых данных. Механизм кэширования позволяет существенно снизить нагрузку базы данных, и тем самым сделать систему более выгодной в сравнении с аппаратными решениями увеличения производительности.

**The Abstract:** This article describes the problem of accessing database in state of limited hardware resources. It provides the analytics of the structure of database and the way to reduce the workload by caching certain data. The cache mechanism reduces the workload that makes the whole system that uses it cheaper on hardware. An example of practical use in existing active system of dispatching taxi park is described.

**Ключевые слова:** Нагрузка базы данных, кэширование, высоконагруженные системы, ORM.

**Keywords.** Database load, cache, highload, ORM.

***

Последние 20 лет информационные системы активно наращивают свой функционал, удовлетворяя возрастающие потребности пользователей. По мере роста функциональности систем вырастает объём данных, оперируемых системой, возрастает требование к оперативности их работы. Для решения возникающих задач применяются новые технологии и подходы к построению распределённых систем, в том числе облачные вычисления. В общем виде под высоконагруженными системами подразумеваются распределённые системы с большим объёмом передаваемых данных [1]. Как правило, это крупные серверные программы использующие ресурсы нескольких компьютеров для обеспечения высокой производительности. Наиболее ярким примером являются интернет-приложения, в которых каждому пользователю выделяется часть аппаратных ресурсов для своевременного отклика и

Главный редактор - д.э.н., профессор К.А. Кирсанов
тел. для справок: +7 (925) 853-04-57 (с $11^{00}$ – до $18^{00}$)
Опубликовать статью в журнале - http://publ.naukovedenie.ru

хранения информации [2]. С другой стороны, современные системы масштаба предприятия так же предполагают обмен и хранение больших объёмов информации. В рамках данной статьи под высоконагруженными системами понимаются системы, осуществляющие активные изменения большого количества данных при ограниченных аппаратных ресурсах.

Любая система, оперирующая с большими объёмами данных, так или иначе, реализует механизмы кэширования различных данных [3]. Для интернет-приложений кэш разделяется на несколько уровней, и большая часть используемых данных кэшируются у самого пользователя на файловой системе, что снижает обращения к сети, и повышает отзывчивость системы. Для систем масштаба предприятия бессмысленно кэшировать что-либо в файловой системе одного клиентского места. Как правило, данные изменяются достаточно динамично, буквально каждое действие пользователя сохраняется. Поэтому все оперативные данные хранятся только в базе данных, для роста производительности в системе целесообразно выполнять централизованное кэширование для снижения нагрузки при оперативной работе, при этом объекты для кэширования следует подбирать тщательно, анализируя каждую отдельную сущность в базе данных.

На данный момент существует множество систем масштаба предприятия, предполагающих манипулирование большим количеством информации. Рассмотрим в качестве примера систему диспетчерского управления таксомоторным парком, которая предполагает использование единой базы данных заказов, сотрудников, машин и др. Такую систему следует проектировать как клиент-серверную систему, представляющую собой единый сервер реализующий бизнес логику и множество автоматизированных рабочих мест (клиентов) осуществляющих работу по различным направлениям. Если речь идёт о крупном таксопарке, то количество заказов, с которыми оперируют клиентские приложения, может быть очень велико. Крупный таксопарк может иметь порядка 3000-5000 заказов в сутки, соответственно за несколько лет, база данных заказов аккумулирует миллионы заказов. Для своевременной обработки (приём и распределение) заказов, таксопарк, как правило, имеет несколько диспетчеров. В диспетчерскую службу привлекается оптимальное количество диспетчеров. В среднем для обслуживания тысячи заказов в сутки нужно 3 диспетчерских места. Каждый заказ предполагает минимум 4 изменения в своей истории: ввод нового заказа, передача заказа водителю, отметка о начале выполнения заказ, закрытие заказа, а также может содержать ещё ряд изменений (например: отзвонка клиенту, прибытие водителя к клиенту и др.). Предположим, что в час пик каждый диспетчер безостановочно принимает и обрабатывает заказы. В среднем для принятия заказа диспетчер тратит 40 секунд (на разговор с клиентом и занесение заказа). Обработка занимает меньше времени, поскольку не требует переговоров с клиентом, поэтому будем считать, что каждую минуту диспетчер тратит 40 секунд на переговоры и занесение заказа и 20 секунд на обработку заказов. Значит, каждую минуту каждый диспетчер вносит 4 изменения. Три диспетчера это 12 изменений в минуту, а если говорить о крупном таксопарке с более чем 3000 тысячами заказов, это выходит что нужно использовать 9-15 диспетчеров, это 36-60 изменений в минуту и после каждого изменения снова обращаться к базе за списком актуальных заказов. Даже такая нагрузка, может замедлить систему, однако стоит учитывать, что это не единственные операции, которые производятся над базой данных. Одновременно с этим может вестись учёт путевых листов и финансов по ним, формирование отчётов и, практически в любом таксопарке, сбор навигационных данных.Таким образом, описанная система является высоконагруженной, и без оптимизации программных и технологических решений невозможно обеспечить удовлетворительную производительность работы системы.

В такой системе, камнем преткновения становится база данных, в которой невозможно, одновременно вносить и возвращать информацию в большом количестве за приемлемый промежуток времени. Очевидное решение заключается в расширении аппаратной части и

Главный редактор - д.э.н., профессор К.А. Кирсанов
тел. для справок: +7 (925) 853-04-57 (с $11^{00}$ – до $18^{00}$)
Опубликовать статью в журнале - http://publ.naukovedenie.ru

масштабирование системы для распределения нагрузки. Однако такое решение затратно по финансам. Альтернативным решением является внесение изменений в технологию работы программного обеспечения, которые позволят снизить нагрузку на базу данных.

На сегодняшний день практически любой язык программирования предоставляет набор инструментов для обращения к базе данных. В привычном виде эти инструменты позволяют соединяться с базой, используя логин и пароль, и выполнять текстовые запросы на языке SQL (Structured Query Language). Но, использование такого подхода, снижает гибкость программного обеспечения, усложняя его развитие, и обязывает каждого клиента проходить аутентификацию к базе данных, усложняя тем самым само использование базы данных, и требования к администрированию. Если система уже используется и активно развивается, то изменения в структуре базы данных неизбежны, и, следовательно, нужно будет каждый раз при таком изменении исправлять все обращения к базе данных. Для решения этой проблемы существуют **ORM** (англ. Object-relational mapping, рус. Объектно-реляционное отображение) — технология программирования, которая связывает базы данных с концепциями объектно-ориентированных языков программирования, создавая «виртуальную объектную базу данных»[4].ORM предоставляет набор объектных представлений сущностей базы данных для определённого языка программирования. Таким образом, программист работает только с объектной моделью сущностей, абстрагируясь от структуры базы данных, что в большинстве случаев позволяет не возвращаться к старому коду при изменении структуры базы данных. Посредством ORM можно, по крайней мере, частично абстрагироваться от самой базы данных и её структуры.

Однако, использование ORM не избавляет от необходимости каждому клиенту проходить аутентификацию к базе данных (Рис. 1а). Хранить логин и пароль соединения с базой данных в конфигурации клиента не безопасно, поскольку клиентов может быть неограниченное количество, и безопасность каждого рабочего места не гарантируется. Использовать фиксированный в коде пароль повышает зависимость клиентов от изменений безопасности базы данных и необходимости обновления клиентских приложений.

Таким образом, помимо сервера базы данных, целесообразно использовать так же сервер приложений, содержащий в себе бизнес логику. Клиентские приложения предоставляют только пользовательский интерфейс. Сервер приложений предоставляет доступ клиентов, к объектным представлениям сущностей базы данных, созданных ORM добавляя прослойку между сервером базы данных и клиентами, позволяя использовать собственные механизмы аутентификации вместо аутентификации базы данных (Рис. 1б).

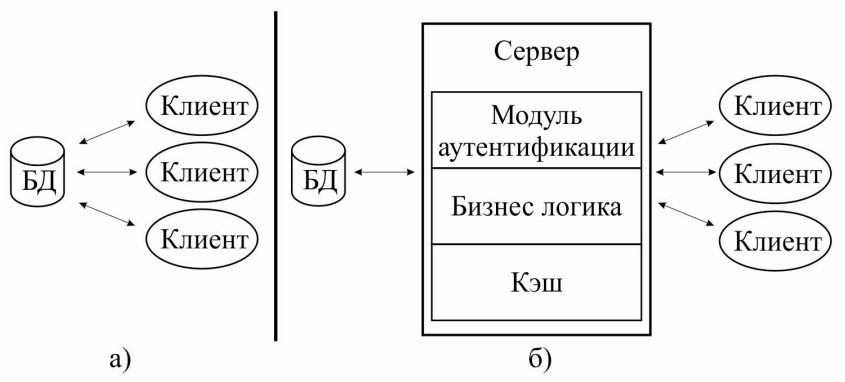


***Рис. 1.*** *Авторизация к базе данных*

Таким образом, логин и пароль могут храниться только в серверном приложении, а клиентские модули будут получать данные только от сервера приложений, не взаимодействуя с сервером базы данных напрямую. Очевидно, что применение предлагаемой программной

Главный редактор - д.э.н., профессор К.А. Кирсанов
тел. для справок: +7 (925) 853-04-57 (с $11^{00}$ – до $18^{00}$)
Опубликовать статью в журнале - http://publ.naukovedenie.ru

оптимизации так же позволяет повысить безопасность хранимых данных и наращиваемость функционала.

При реализации бизнес логики сервера приложений следует проанализировать структуру базы данных, с точки зрения частоты использования сущностей.

Большинство сущностей можно разделить на оперативные и справочные. Справочными называются сущности, которые редко подвергаются изменению, в основном только считываются. Оперативные сущности – это сущности, которые часто подвергаются изменениям. На примере работы таксопарка, можно разделить сущности следующим образом:

- Справочные: списки автомобилей и сотрудников. Данные в них редко изменяются, и почти всегда используются только единичные сущности, или небольшие группы (напр. один автомобиль, который будет выполнять заказ, или группа автомобилей удовлетворяющих условиям). Данные сущности не подвергаются большому количеству изменений, и не предполагают каких-либо колоссальных объёмов (даже если взять все таксопарки любого крупного города, едва ли количество автомобилей превысит несколько десятков тысяч)

- Оперативные: списки заказов, наоборот, имеют явный признак актуальности (выполнен или нет), обновляются очень часто, и хранятся в большом количестве (даже 1 средний таксопарк за год может накопить более миллиона заказов).

С точки зрения организации кэширования, справочные сущности можно кэшировать непосредственно у клиента, без введения явной синхронизации, достаточно лишь с некоторой периодичностью их синхронизировать.

Оперативные же данные, отличаются тем, что у всех пользователей должны быть актуальные данные, при этом они интенсивно изменяются. Целесообразно организовать кэширование на стороне сервера, чтобы поддерживать актуальное состояние. Для заказов это особенно актуально, в связи с тем, что есть явный признак актуальности и соответственно чёткий список заказов, которые и актуальны пользователям. Полное разделение сущностей представлено на Рис. 2.

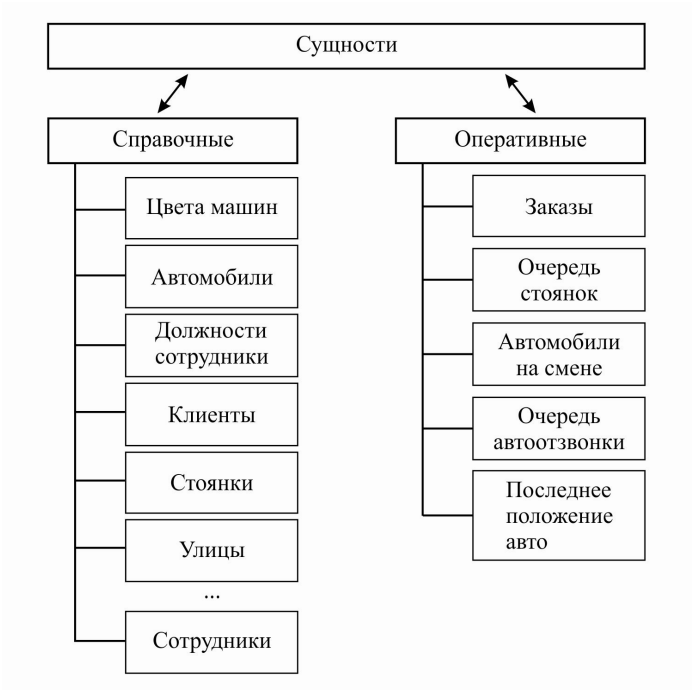


***Рис 2.*** *Списки сущностей таксомоторного парка*

Главный редактор - д.э.н., профессор К.А. Кирсанов
тел. для справок: +7 (925) 853-04-57 (с $11^{00}$ – до $18^{00}$)
Опубликовать статью в журнале - http://publ.naukovedenie.ru

Общий механизм кэширования можно представить так: сервер может хранить в памяти некий список актуальных сущностей и принимать изменения \ добавления. При получении очередного изменения, сервер сохраняет его в базе, модифицирует коллекцию, хранимую в памяти, и отсылает уведомления об изменении клиентам. Соответственно можно определить правила, по которым определённые сущности считать активными и хранить кэш в памяти сервера. В момент соединения с сервером клиенты запрашивают коллекции различных сущностей с сервера и, в дальнейшем, лишь изменяют отображение, в зависимости от приходящих с сервера уведомлений об изменении. Сервер периодически синхронизует свою коллекцию с базой данных, а клиенты периодически запрашивают полностью коллекцию с сервера. Использование такого механизма позволяет минимизировать обращения к базе данных, снижая нагрузку до одного запроса раз в заданный период. Для отправки уведомлений об изменениях клиентам необходимо реализовать механизм подписки: клиенты подписываются на получение изменений некоторой сущности и, как только такие изменения происходят, получают соответствующие данные (Рис. 3).

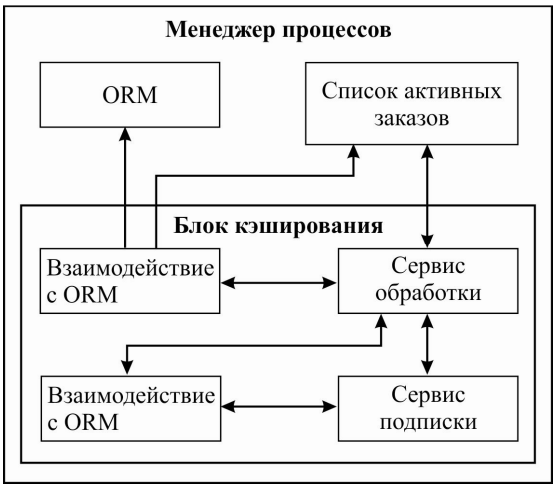


***Рис 3.*** *Структура сервера приложений*

Сервер, при старте запрашивает все невыполненные заказы, и хранит эту совокупность в памяти. Клиенты, соединяясь с сервером, проходят авторизацию на сервере, после чего получают соответствующую совокупность заказов. После чего клиенты подписываются на изменения сущностей заказов. Таким образом, если один из клиентов внёс изменения в заказе (напр. назначил водителя для выполнения заказа), сервер немедленно сохраняет изменение и отправляет остальным клиентам по подписке это изменение (Рис. 4). Соответственно у всех клиентов всегда будет актуальная информация о заказах, без необходимости часто запрашивать список заказов из базы данных. Для сравнения, при n количестве клиентов и p частоты изменений в базе данных, без кэширования, будет осуществляться p+p*n количество запросов к базе данных, а с кэшированием только p, то есть в n+1 раз меньше.

Главный редактор - д.э.н., профессор К.А. Кирсанов
тел. для справок: +7 (925) 853-04-57 (с $11^{00}$ – до $18^{00}$)
Опубликовать статью в журнале - http://publ.naukovedenie.ru


***Рис. 4.*** *Диаграмма последовательностей кэширования заказов*

Если анализировать количество обращений к базе данных, по количеству изменений/внесений заказов, а не по количеству клиентов вносящих изменения, то применение различного рода кэширования на каждом этапе сокращает количество запросов в определённое количество раз. Например, если предполагать вышеописанную структуру базы данных таксопарка, то к заказам, при каждом изменении, добавляются дополнительно два справочника: автомобили и сотрудники. А само по себе изменение заказа ведёт к трём обращениям к базе данных: запрос данных по заказу, сохранение изменения и запрос обновлённого списка заказов. Соответственно применение различного рода кэширования приведёт к следующим результатам (Рис. 5).

Главный редактор - д.э.н., профессор К.А. Кирсанов
тел. для справок: +7 (925) 853-04-57 (с $11^{00}$ – до $18^{00}$)
Опубликовать статью в журнале - http://publ.naukovedenie.ru

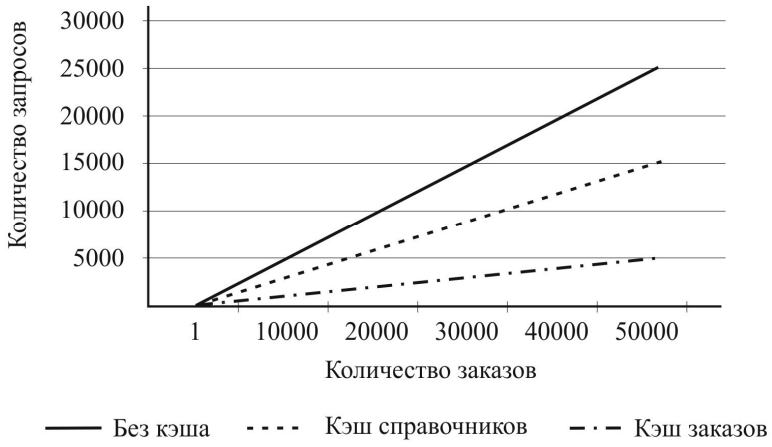


***Рис. 5.*** *Зависимость количества обращений от количества запросов*

Реализуя механизм кэширования на сервере приложения, количество запросов к базе данных снижается достаточно, чтобы используя один сервер базы данных обслуживать даже очень большой таксопарк или несколько таксопарков. Помимо кэширования реализация механизма обмена данными именно между серверным и клиентскими приложениями, без прямого взаимодействия с базой данных, позволяет использовать любые другие механизмы контроля получения/сохранения данных, например собственная реализация учёта прав доступа.

**Заключение**

В статье рассмотрен механизм кэширования сущностей базы данных. Отличительной особенностью предложенного механизма является то, что пользователи всегда имеют актуальное состояние данных, без лишних действий для их обновлений. Предложенный механизм позволяет снизить нагрузку на базу данных, при этом наибольший эффект достигается при больших количествах запросов к базе данных относительно кэшируемой сущности. Применение предложенного механизма кэширования сокращает количество запросов к базе данных в 5-7 раз для среднего таксопарка, а также убирает прямую зависимость нагрузки на базу данных от количества диспетчеров. Предложенные подходы к технологии построения программного обеспечения практически апробированы на системе диспетчерского управления таксомоторным парком ИТС «АГАТ» и показали свою эффективность

## ЛИТЕРАТУРА

1) О высокой нагрузке. [2013—2013]. Дата обновления: 08.01.2013. URLhttp://bazhenov.me/blog/2012/02/26/highload.html (дата обращения: 08.01.2013).

2) Архитектура высоконагруженных систем // InsightIT. [2013—2013]. Дата обновления: 08.01.2013. URL: http://www.insight-it.ru/highload/ (дата обращения: 08.01.2013).

3) Кэш // Википедия. [2013—2013]. Дата обновления: 12.03.2013. URL: http://ru.wikipedia.org/?oldid=53223165 (дата обращения: 12.03.2013).

4) ORM // Википедия. [2013—2013]. Дата обновления: 08.01.2013. URL: http://ru.wikipedia.org/?oldid=51444189 (дата обращения: 08.01.2013).

**Рецензент:** Ямпольский В.З., Профессор-консультант, доктор технических наук, Национальный исследовательский Томский политехнический университет