**Введение**

Современный мир становится все более информатизирован, появляется все больше веб-сервисов, которые имеют значительное количество клиентов (например, различные сервисы электронной коммерции и банковские сервисы). Для таких сервисов является очень важной возможность работать непрерывно под большими нагрузками, так как простой может повлечь большие финансовые потери, а также миграцию пользователей к конкурентам, которые в состоянии обеспечить высокую доступность своих сервисов. При проектировании сервиса необходимо также учитывать вопросы производительности работы приложения, потому как зачастую затрата большого количества ресурсов на простейшие действия пользователя может повлечь фатальные последствия (например, такое простое действие, как поиск какой-либо информации — если при каждом поисковом запросе происходит обращение к базе данных, то время отклика приложения будет достаточно большим, что влечет неудовлетворение пользователя работой системы; при этом каждый раз происходит нагрузка на базу данных, что также сказывается на общей работе системы).

Решением данных проблем является использование балансировщика нагрузки для кластера, на котором развернуто приложение, и эффективного алгоритма кэширования данных, позволяющего снизить затраты ресурсов при работе системы. Использование метода балансировки нагрузки позволяет многим компаниям практически не беспокоиться о возможности потерять клиентов из-за сбоя серверов, на которых находится веб-сервис, так как балансировщик нагрузки реализует грамотное управление поступающими запросами к серверам, что обеспечивает равномерную нагрузку на узлы кластера и снижает опасность потери работоспособности.

Однако, использование балансировщика нагрузки не гарантирует оптимальный по времени ответ от сервера на поступивший запрос. Именно для решения такой проблемы и используются различные алгоритмы кэширования данных, которые минимизируют затраты ресурсов при обработке очередного запроса к серверу за какими-либо данными.

Правильная реализация данных технологий гарантирует максимально эффективную и надежную работу приложения, позволяя не беспокоиться о потери пользователей системы.

Таким образом, данные проблемы являются очень важными для реализации применительно к крупным веб-сервисам, ориентированным на непрерывную высоконагруженную работу с клиентами.

Работа выполняется в рамках созданной ранее системы прогнозирования инкассаций Cash Management.

Предполагается следующая структура пояснительной записки:

В первом разделе проводится анализ методов балансировки нагрузки на кластер серверов приложений и алгоритмов кэширования данных, позволяющих наиболее эффективно организовать работу системы.

Во втором разделе выбирается метод для решения задачи балансировки нагрузки на кластер и эффективный алгоритм кэширования данных, а также анализируется возможность их модификации и адаптации применительно к работе системы прогнозирования инкассаций CashManagement.

В третьем разделе проектируется модуль, решающий задачу кэширования данных в условиях частых модификаций в БД, а также разрабатывается архитектура для кластера серверов с использованием метода балансировки нагрузки между узлами в кластере.

В четвертом разделе приводится программная реализация спроектированных модулей и результаты тестирования и интеграции с системой CashManagement.

**Раздел 1. Анализ алгоритмов кэширования данных и методов балансировки нагрузки на кластер серверов**

В данном разделе описаны результаты анализа, связанного с изучением целей и особенностей работы системы прогнозирования инкассаций CashManagement. Проводится анализ целей работы системы, а также рассматриваются различные недостатки работы системы с целью их устранения и увеличения надежности и производительности работы системы. На основе проведенного анализа предполагается рассмотреть наиболее эффективные алгоритмы кэширования данных для улучшения производительности работы системы, а также методы балансировки нагрузки на кластер серверов, которые позволяют обеспечить оптимальное управление запросами, поступившими к кластеру, на котором развернуто приложение. Проводится анализ применимости данных алгоритмов и методов для использования в системе CashManagement для улучшения таких характеристик системы, как надежность, отказоустойчивость и производительность.

**1.1. Цели работы системы прогнозирования инкассаций Cash Management**

SmartVista ATM Cash Management (сокращенно SVCM) является самостоятельным приложением, позволяющим снизить затраты на содержание сети банкоматов. Решение предназначено, чтобы помочь банкам:

1. Снизить операционные издержки на обслуживание сети банкоматов
2. Уменьшить объемы денежных средств в сети
3. Поддерживать необходимое количество денежных средств в каждом банкомате, проводя инкассации в нужное время и по самым низким ценам
4. Повысить отдачу денежных средств
5. Увеличить доступность сети за счет уменьшения количества экстренных инкассаций и улучшить обслуживание клиентов путем сокращения проблем с нехваткой денежных средств в банкоматах
6. Сократить время построения маршрутов инкассации и повысить качество их построения
7. В данный момент построение оптимальных маршрутов инкассаций основано на метаэвристическом генетическом алгоритме.

В ходе анализа работы системы были выявлены такие недостатки, как низкая производительность при обработке поисковых запросов за выборкой данных из БД, выражающаяся в неоптимальном обращении к базе данных при каждом запросе к серверу, а также отсутствие балансировки нагрузки, что подвергает большой угрозе отказоустойчивость и надежность работы системы при условиях работы большого количества пользователей в системе.

**1.2.**  **Изучение и сравнительный анализ методов и алгоритмов кэширования данных с целью улучшения производительности работы приложения**

Кэш — промежуточная структура с быстрым доступом, содержащий информацию, которая может быть запрошена с наибольшей вероятностью. Доступ к данным в кэше осуществляется быстрее, чем выборка исходных данных из БД, однако её объём существенно ограничен по сравнению с хранилищем исходных данных.[3]

Существует 3 основных типа кэша:

1. Lazy cache — кэш, сохраняющий данные и отдающий их, пока данные не устарели. 2. Synchronized cache — клиент вместе с данными получается метку последнего изменения и может спросить у поставщика не изменились ли данные, чтобы повторно из не запрашивать.

3. Write-through cache – любое изменение данных выполняется сразу в хранилище и в кэше.[2, 5]

Главная проблема, связанная с кэшом — устаревание данных, потеря их актуальности. Поэтому необходимо проверять, что данные еще не устарели и не были модифицированы.[6] Вследствие ограниченности размера кэша при достижении кэшом некоторого объема часть данных вытесняется. Основной параметр, который характеризует систему кэширования – это процент попаданий запросов в кэш. Этот параметр довольно легко измерить, чтобы понять насколько ваша система кэширования эффективна. Частые сбросы кэша, кэширование редко запрашиваемых данных, недостаточный объем кэша – все это ведет к пустой трате оперативной памяти, не повышая эффективность работы. Иногда данные меняются настолько часто и непредсказуемо, что кэширование не даст эффекта, процент попаданий будет близок к нулю. Но обычно данные считываются гораздо чаще, чем записываются, именно поэтому кэши эффективны.[4]

Различают следующие алгоритмы кэширования данных:

1. Алгоритм Белади: основан на принципе удаления из кэша информации, которая не будет требоваться в дальнейшем. Однако реализация данного алгоритма невозможна, так как не известно, когда данные понадобятся в следующий раз;

2. LRU (Вытеснение давно неиспользуемых данных): основан на вытеснении дольше всех неиспользуемого кэша. В реализации данного алгоритма необходимо учитывать, когда данные использовались, что является довольно-таки накладным при большом объеме данных;

3. MRU (Наиболее недавно использовавшийся элемент): основан на удалении из кэша элемента, к которому происходило последнее обращение;

4. Псевдо-LRU: используется для кэшей с большой ассоциативностью, так как для таких кэшей реализация LRU становится неоптимальной;

5. Сегментированный LRU: кэш делится на два сегмента. пробный сегмент и защищенный сегмент. Строки в каждом сегменте упорядочены от частоиспользуемых к наименее используемым. Данные при промахах добавляются в кэш, причем в область последних использованных элементов пробного сегмента. Данные при попаданиях убираются где бы они не располагались и добавляются в область частоиспользуемых элементов защищенного сегмента. К строкам защищенного сегмента обращения таким образом происходят по крайней мере дважды. Защищенный сегмент ограничен. Такой перенос строки из пробного сегмента в защищенный сегмент может вызвать перенос последней использованной (LRU) строки в защищенном сегменте в MRU-область пробного сегмента, давая этой линии второй шанс быть использованной перед вытеснением;

6. Кэш прямого отображения: очень быстрый кэш, используется в основном для процессоров;

7. Least-Frequently Used (Наименее часто используемый): подсчитывает как часто используется элемент. Те элементы, обращения к которым происходят реже всего, вытесняются в первую очередь;

8. Адаптивная замена (ARC): балансирует между LRU и LFU, что дает хорошие результаты.[5]

**1.3. Изучение и анализ методов балансировки нагрузки для повышения отказоустойчивости и надежности кластера серверов, на которых развернуто приложение**

Балансировка нагрузки — метод распределения запросов между несколькими [серверами](https://ru.wikipedia.org/wiki/Сервер_(аппаратное_обеспечение)) с целью оптимизации использования ресурсов, [горизонтальног](https://ru.wikipedia.org/wiki/Масштабируемость" \l ".D0.93.D0.BE.D1.80.D0.B8.D0.B7.D0.BE.D0.BD.D1.82.D0.B0.D0.BB.D1.8C.D0.BD.D0.BE.D0.B5_.D0.BC.D0.B0.D1.81.D1.88.D1.82.D0.B0.D0.B1.D0.B8.D1.80.D0.BE.D0.B2.D0.B0.D0.BD.D0.B8.D0.B5)о масштабирования кластера и реализации отказоустойчивости.

Балансировка нагрузки может быть использована для расширения возможностей кластера, состоящего из нескольких узлов. Она также позволяет продолжать работу даже если некоторые из узлов кластера вышли из строя. Вследствие этого растёт отказоустойчивость, а также возможность горизонтального масштабирования.[13, 15, 8]

Метод балансировки нагрузки должен обладать следующими свойствами:

1. Предсказуемость;

2. Равномерная загрузка ресурсов системы;

3. Масштабирумость.[14, 12]

Методы балансировки нагрузки:

1. Round Robin (алгоритм кругового обслуживания): представляет собой перебор по круговому циклу: первый запрос передаётся одному серверу, затем следующий запрос передаётся другому и так до достижения последнего сервера, а затем всё начинается сначала; преимущества данного метода заключаются в независимости от протокола высокого уровня (имеется возможность использовать любой протокол, в котором обращение к серверу идёт по имени), отсутствии необходимости в связи между серверами и низкой стоимости реализации подобного метода. Однако, у данного метода есть серьезные недостатки: у каждого сервера в кластере должен быть одинаковый набор ресурсов и не учитывается загруженность того или иного узла в кластере;

2. Weighted Round Robin: улучшенная версия метода Round Robin. Суть усовершенствований заключается в следующем: каждому серверу присваивается весовой коэффициент в соответствии с его производительностью и мощностью. Это помогает распределять нагрузку более гибко: серверы с большим весом обрабатывают больше запросов.  
3. Least connections: данный метод учитывает количество подключений, поддерживаемых серверами в текущий момент времени. Каждый следующий вопрос передаётся серверу с наименьшим количеством активных подключений.

4. Алгоритм Destination Hash Scheduling был создан для работы с кластером кэширующих прокси-серверов, но он часто используется и в других случаях. В этом алгоритме сервер, обрабатывающий запрос, выбирается из статической таблицы по IP-адресу получателя.[8, 15, 10]

**1.4. Анализ алгоритмов кэширования данных и методов балансировки нагрузки применительно к задачам повышения производительности работы и отказоустойчивости системы прогнозирования инкассаций CashManagement**

Требования, которые необходимо учитывать при выборе алгоритма кэширования данных и метода балансировки нагрузки для реализации в системе прогнозирования инкассаций CashManagement:

1. Выбранный алгоритм кэширования данных оптимально использовать оперативную память и не заполнять ее ненужными данными;

2. Обязательно должны учитываться вопросы актуальности данных, содержащихся в кэше;

3. Метод балансировки нагрузки должен обеспечивать максимальную надежность работы системы;

4. Должна быть реализована равномерная нагрузка на все сервера в кластере.

**1.5. Выводы**

* Система прогнозирования инкассаций Cash Management не обладает возможностью оптимально совершать запросы к БД, следовательно, необходимо реализовать алгоритм кэширования данных, который бы минимизировал количество обращений к базе данных.
* При реализации алгоритма кэширования данных необходимо учитывать вопрос актуальности данных в кэше, так как работа по модификации данных может быть очень активной.
* В работе системы не обеспечивается необходимая отказоустойчивость, для чего должен быть реализован балансировщик нагрузки, который бы эффективно управлял распределением запросов к узлам кластера.

**1.6.** **Цели и задачи УИР/НИР**

Целями данной работы являются:

1. Выбор алгоритма кэширования данных с учетом частых модификаций в базе данных и метода балансировки нагрузки на кластер серверов для обеспечения отказоустойчивости кластера серверов приложений.

2. Модификация и адаптация выбранного алгоритма и метода для системы CashManagement.

3. Программная реализация и тестирование выбранного метода и алгоритма.

Для достижения цели №1 необходимо проанализировать алгоритмы кэширования данных, а также методы балансировки и выбрать наиболее эффективные.

Для достижения цели №2 необходимо изучить выбранный алгоритм и метод, после чего адаптировать их под требования для системы CashManagement.

Для достижения цели №3 необходимо реализовать модифицированные алгоритмы и протестировать их работу, после чего проанализировать полученные результаты оптимизации.

Необходимые задачи для достижения целей №1-3 не нуждаются в дальнейшей детализации.

**Раздел 2. Выбор алгоритма кэширования данных и метода балансировки нагрузки и адаптация их для использования в системе CashManagement**

В данном разделе детально рассмотрены основные алгоритмы кэширования данных, приведены их преимущества и недостатки, а также проведен анализ на предмет использования их задаче реализации алгоритма кэширования данных для системы CashManagement. Из всех рассмотренных алгоритмов выбирается один, который наиболее удовлетворяет требованиям, предъявляемым к модулю кэширования данных системой. Для выбранного алгоритма также приведена блок-схема и необходимые модификации для использования в системе CashManagement. Также в разделе рассматриваются наиболее хорошо себя зарекомендовавшие методы балансировки нагрузки на кластер серверов приложений. Из них выбирается наиболее соответствующий для использования в системе. Для выбранного метода приводится набор UML-диаграмм, построенных в результате проектирования.

**2.1. Выбор алгоритмов кэширования данных и балансировки нагрузки на кластер**

В данном подразделе сравниваются различные алгоритмы, реализующие кэширование данных и балансировку нагрузки. Для каждого из алгоритмов представлено подробное описание и их преимущества и недостатки. По результатам сравнительного анализа производится выбор одного алгоритма кэширования и балансировки нагрузки, которые будут в дальнейшем модифицированы и реализованы.Анализ алгоритмов кэширования данных представлен в следующей таблице:  
 Таблица 1. Алгоритмы кэширования

|  |  |  |  |
| --- | --- | --- | --- |
| Название алгоритма | Описание | Преимущества | Недостатки |
| Алгоритм Белади (алгоритм предвидения) | Добавление в кэш всех запрашиваемых данных с последующим удалением из него всей информации, когда не потребуется в будущем.  Используется для сравнения, теоретический алгоритм. | 1. Оптимальность. | 1. Невозможно реализовать на практике, т. к. неизвестно, когда данные понадобятся в следующий раз. |
| LRU (Least recently used) | Вытеснение дольше всех неиспользуемых элементов из кэша. | 1. Нет проблем с валидностью кэша. 2. Высокая эффективность на относительно небольшом количестве данных. | 1. Необходимость учитывать время последнего использования элементов. 2. Уменьшение эффективности в зависимости от количества данных. |
| MRU (Most Recently Used) | Удаление из кэша элемента, к которому происходило последнее обращение. Стремление к сохранению старых данных. | 1. Высокая эффективность при циклическом сканировании больших наборов данных. | 1. Малая эффективность при маленьком размере кэша. 2. Подходят только в случаях, когда чем старше элемент, тем больше обращений к нему происходит. |
| Псевдо-LRU (PLRU) | Используется для кэшей с большой ассоциативностью, когда почти всегда нужно отбрасывать наименее используемый элемент. Алгоритм кэширования для использования в процессорах. | 1. Высокая эффективность для кэшей с большой ассоциативностью, когда LRU становится неэффективен. 2. Требует для элемента кэша только один бит. | 1. В остальных случаях неэффективен. |
| Least-Frequently Used | Подсчет частоты использования элемента. Те элементы, обращения к которым происходят реже всего, вытесняются в первую очередь. | 1. Нет проблем с валидностью кэша. 2. Высокая эффективность на относительно небольшом количестве данных. | 1. Необходимость иметь счетчик обращений к элементу, что снижает эффективность при большом количестве данных. |
| Адаптивная замена (ARC) | Баланс между LRU и LFU. | 1. Соединяет в себе преимущества LRU и LFU. |  |

Для эффективной работы в системе CashManagement необходимо иметь всегда актуальные данные в кэше, помещая в него результаты выполнения различных поисковых запросов. Поэтому по результатам сравнительного анализа для реализации был выбран алгоритм LRU, который обеспечивает валидность кэша и минимальный риск потери актуальности данных. Данный алгоритм является самым подходящим из-за модификаций, которым должен быть подвергнут алгоритм для работы в системе CashManagement.  
 Сравнительный анализ методов балансировки нагрузки представлен в следующей таблице:

Таблица 2. Алгоритмы балансировки нагрузки

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Описание | Преимущества | Недостатки |
| Round Robin (алгоритм кругового обслуживания) | Перебор по круговому циклу: первый запрос передаётся одному серверу, затем следующий запрос передаётся другому и так до достижения последнего сервера, а затем всё начинается сначала | 1. Независимость от протокола высокого уровня. 2. Низкая стоимость реализации. 3. Отсутствие необходимости в связи между серверами в кластере. | 1. У каждого сервера в кластере должен быть одинаковый набор ресурсов. 2. Не учитывается загруженность того или иного узла в кластере. |
| Weighted Round Robin | Каждому серверу присваивается весовой коэффициент в соответствии с его производительностью и мощностью. Серверы с большим весом обрабатывают больше запросов. | 1. Гибкое распределение нагрузки. 2. Высокая эффективность при точно известном составе серверов в кластере. | 1. Необходимость знать производительность и мощность серверов в кластере. |
| Least connections | Учитывает количество подключений, поддерживаемых серверами в текущий момент времени. Каждый следующий вопрос передаётся серверу с наименьшим количеством активных подключений. | 1. Алгоритм предоставляет высокую надежность за счет передачи запроса менее загруженному серверу. 2. Низкая стоимость и простота реализации. 3. Отсутствует необходимость знать данные о составе кластера. | 1. Не учитывает разность нагруженности отдельных запросов. |
| Destination Hash Scheduling | Чаще всего используется в кластере кэширующих прокси-серверов. В этом алгоритме сервер, обрабатывающий запрос, выбирается из статической таблицы по IP-адресу получателя. | 1. Гибкость. 2. Большая эффективность, если используется класте из кэширующих прокси-серверов. | 1. Необходимость привязки к данным пользователя. |

Для реализации в системе CashManagement исходя из анализа был выбран алгоритм баласировки Least connections, дающий необходимую надежность в работе и низкую стоимость реализации.

**2.2. Модификация выбранных алгоритмов кэширования данных и балансировки нагрузки на кластер**

В данном подразделе представлены описываются требования, предъявляемые системой к алгоритмам кэширования данных и балансировки нагрузки. Производится модификация выбранных в предыдущем подразделе алгоритмов для использования в системе прогнозирования инкассаций CashManagement и производится выбор средств для проектирования модифицированных алгоритмов.

Система прогнозирования инкассаций CashManagement предъявляет следующие требования к реализации кэширования данных:

1. Выбранный алгоритм должен обладать высокой эффективностью;
2. Алгоритм должен обеспечивать актуальность данных;
3. Жизненный цикл кэша и время, которое элемент в кэше считается актуальным, должно быть настраиваемым.

В соответствии с требованиями выбранный алгоритм LRU был модифицирован следующим образом: при реализации была учтена необходимость иметь возможность регулировать время жизни кэша и время актуальность данных в кэше. Проверка актуальности данных для исключения лишних обращений к БД проводится с помощью извлечения данных о последней модификации данных в таблице.

Требования к реализации балансировки нагрузки для кластера, на котором развернута

система:

1. Реализация должна быть независимой от состава серверов приложений в кластере.
2. Должна обеспечиваться высокая надежность работы системы.

Реализация балансировки была основана на алгоритме Leasts Connections, в основе которого лежит механизм хранения информации о количестве текущих подключений к серверу в кластере. Данная реализация обеспечивает необходимую надежность в работе. Алгоритм не учитывает, какие виды серверов имеются в кластере, работая с самими http-запросами. В случае, если в кластере имеется сервер с меньшим количеством активных соединений чем тот, на который пришел запрос, запрос перенаправляется на этот сервер.

Для представления результатов проектирования был выбран стандарт графического описания абстрактной модели системы — UML.

В языке UML имеются следующие виды диаграмм:

* use case diagram (диаграммы прецедентов);
* activity diagram (диаграммы описаний технологий, процессов, функций);
* sequence diagram (диаграммы последовательностей действий);
* collaboration diagram (диаграммы взаимодействий);
* component diagram (диаграммы компонентов);
* deployment diagram (диаграммы развертывания);

**Раздел 3. Результаты проектирования модулей кэширования данных и метода балансировки нагрузки на кластер серверов**

В данном разделе описываются результаты проектирования модулей на основе выбранных и адаптированных алгоритмов кэширования данных и балансировки нагрузки для последующей их программной реализации. Приводится информация о выбранной модели жизненного цикла (итеративная модель), а также ее преимущества и недостатки. Обосновывается выбор именно такой методологии разработки в сравнении с другими. Строится архитектура разрабатываемых модулей кэширования и балансировки нагрузки, также приводится общая архитектура системы с учетом разрабатываемых модулей. Приводится список основных требований к разрабатываемым модулям применительно к использованию в системе прогнозирования инкассаций CashManagement (список нефункциональных требований).

**3.1. Итеративная модель жизненного цикла**

В данном подразделе представлено описание выбранной модели жизненного цикла разработки ПО, ее преимущества и недостатки. Также приводится список ситуация, когда использование выбранной модели оправданно. Приводится обосновывание выбора именно итеративной модели жизненного цикла для разработки функционала для системы прогнозирования инкассаций CashManagement.

Плюсы и минусы итеративной модели:

Таблица 3: Преимущества и недостатки итеративной модели

|  |  |
| --- | --- |
| Преимущества | Недостатки |
| Ранее создание работающего ПО | Каждая фаза – самостоятельна, отдельные итерации не накладываются |
| Гибкость – готовность к изменению требований на любом этапе разработки | могут возникнуть проблемы с реализацией общей архитектуры системы, поскольку не все требования известны к началу проектирования |
| Каждая итерация – маленький этап, для которого тестирование и анализ рисков обеспечить проще, чем для всего жизненного цикла продукта |

Когда можно использовать итеративную модель:

1. для крупных проектов;
2. когда известны, по крайней мере, ключевые требования;
3. когда требования к проекту могут меняться в процессе разработки.

Таким образом, выбранная модель ЖЦ является наиболее подходящей, так как позволяет реализовывать функционал постепенно, с возможностью изменить некоторые моменты в реализации.



Рисунок 1. Схема итеративной модели ЖЦ

**3.2. Проектирование модуля кэширования на основе выбранного алгоритма кэширования данных**

В данном подразделе представлены результаты проектирования модуля кэширования данных на основе выбранного и адаптированного к использованию в системе CashManagement алгоритма кэширования данных. Приводятся результаты проектирования модуля в виде скриншотов, разрабатываемого в соответствии с нефункциональными требованиями к работе системы.

Результаты проектирования модуля представлены на следующих UML-диаграммах:

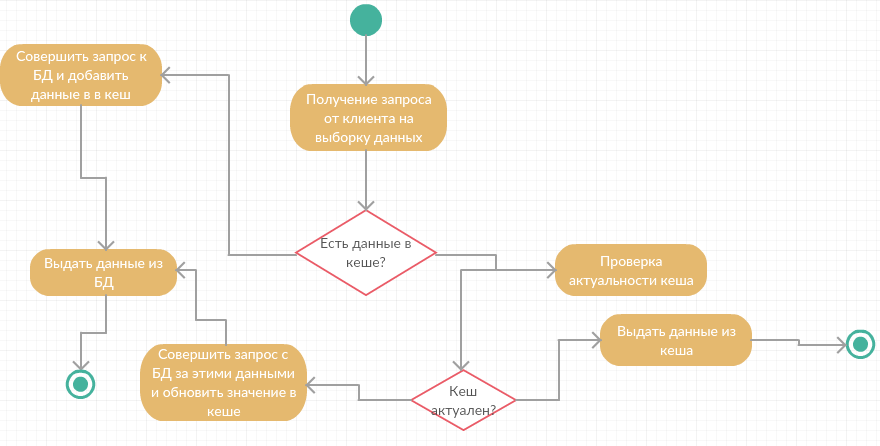
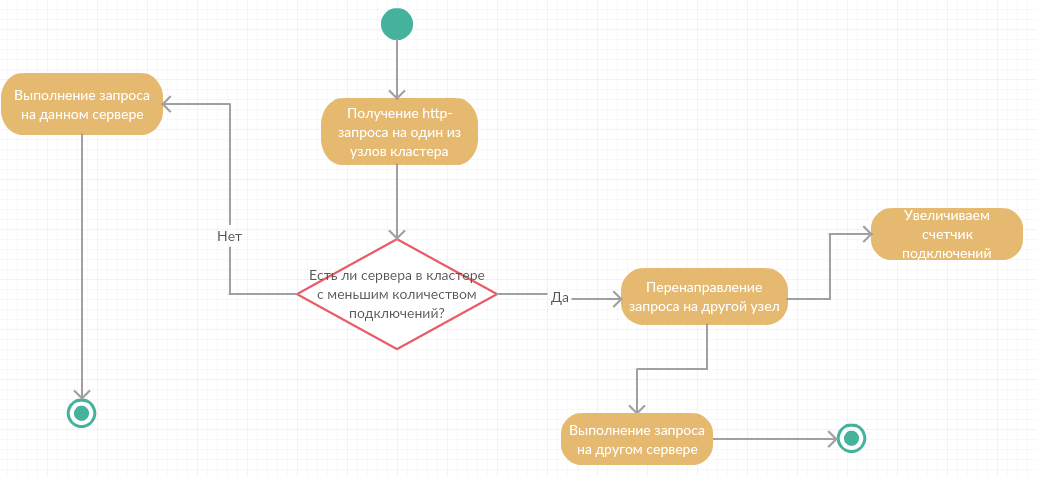


Рисунок 2. Диаграмма деятельности модуля кэширования данных

**3.3. Проектирование модуля, отвечающего за балансировку нагрузки на основе выбранного метода балансировки**

В данном подразделе представлены результаты проектирования модуля балансировки нагрузки на основе выбранного и адаптированного к использованию в системе CashManagement метода балансировки нагрузки на кластер серверов приложений. Приводятся результаты проектирования модуля в виде скриншотов, разрабатываемого в соответствии с нефункциональными требованиями к работе системы.

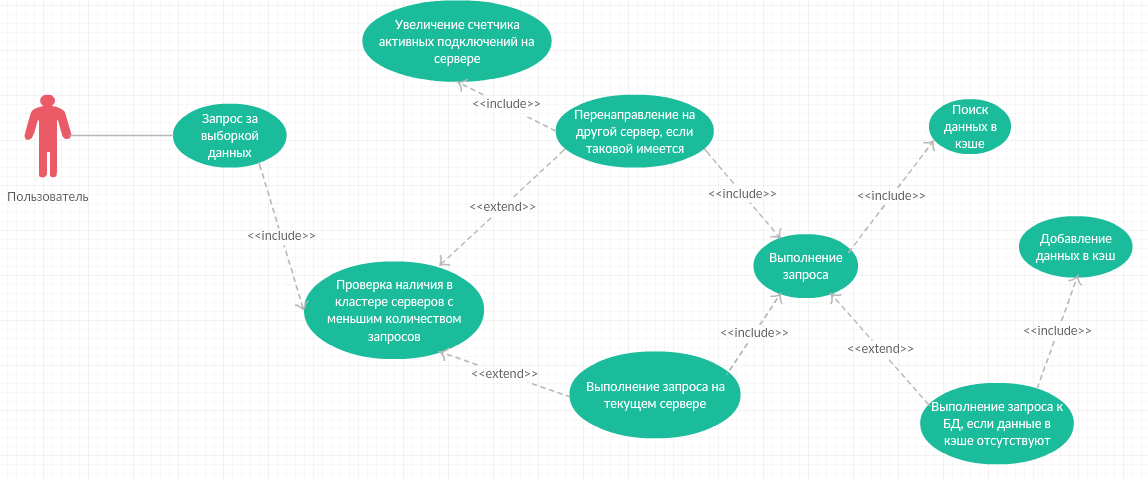
Результаты проектирования модуля представлены на следующих UML-диаграммах:

 Рисунок 3. Диаграмма деятельности для модуля балансировки нагрузки

**3.4. Результаты общего проектирования модулей**

В данном подразделе представлены результаты проектирования модулей балансировки нагрузки и кэширования данных на основе выбранных и адаптированных к использованию в системе CashManagement алгоритмов. Приводятся результаты проектирования модулей в виде скриншотов, описывается архитектура модулей, разрабатываемых в соответствии с нефункциональными требованиями к работе системы.

Результаты проектирования обоих модулей представлены на следующих UML-диаграммах:

 Рисунок 4. Диаграмма использования модулей

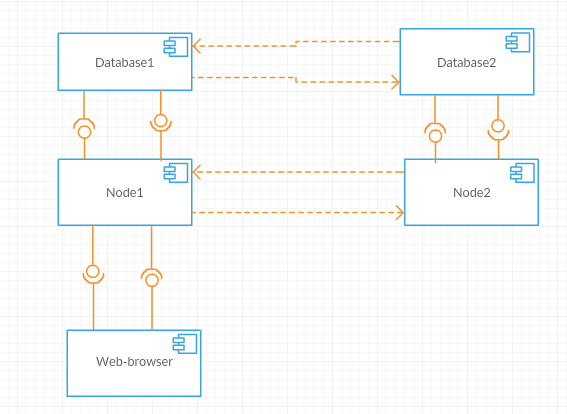


Рисунок 5. Диаграмма компонентов системы

**3.5. Нефункциональные требования, предъявляемые к работе модулей системой**

Нефункциональные требования к модулю балансировки нагрузки:

1. Работа модуля должна обеспечивать эффективную балансировку и равномерную нагрузку на все узлы кластера;
2. Улучшение производительности работы приложения;
3. Незаметность для пользователя;
4. Избегание возможности конфликтов данных на разных серверах при обращении к источнику данных.

Нефункциональные требования к модулю кэширования данных:

1. Работа модуля должна уменьшить количество обращений к БД за данными;
2. Работа модуля должна уменьшить время отклика системы путем уменьшения времени за счет выборки данных из кэша;
3. Кэширование должно снизить нагрузку на источник данных.

**Раздел 4. Результаты программной реализации, тестирования и экспериментальной проверки алгоритма кэширования данных и метода балансировки нагрузки**

В данном разделе приводится описание программной реализации спроектированных алгоритмов и методов, описываются используемые в процессе реализации инструментальные средства и платформы. Дается краткое описание всех инструментальных средств с причинами их использования в данной реализации. Приведены результаты тестирования, а также тестовые сценарии, по которым производится тестирование программной реализации. В сводную таблицу заносятся ожидаемые и фактические результаты от программной реализации. Описываются новые характеристики, которые система приобрела по итогам интеграции программной реализации в систему CashManagement. Проводится общий анализ эффективности примененных алгоритмов, сравнение с характеристиками системы до внедрения разработанных алгоритмов кэширования данных и метода балансировки нагрузки.

**4.1. Программная реализация алгоритмов кэширования данных и балансировки нагрузки на кластер**

В данном подразделе описываются инструментальные средства и платформы, используемые в процессе реализации алгоритмов. Дается краткое описание всех инструментальных средств с причинами их использования в данной реализации. В сводную таблицу заносятся ожидаемые и фактические результаты от программной реализации. Описываются новые характеристики, которые система приобрела по итогам интеграции программной реализации в систему CashManagement.

**4.1.1. Используемые технологии и инструментальные средства**

**4.1.1.1. Java Enterprise Edition**

Java Platform, Enterprise Edition — набор спецификаций и документации для языка Java, который описывает архитектуру серверной платформы для задач средних и крупных предприятий. Java EE разработана Sun Microsystems Inc и является кроссплатформенным программным обеспечением. Основная цель, с которой создавались спецификации — это обеспечить масштабируемость и кроссплатформенность приложений, а также целостность данных по время работы системы.

Платформа Java EE основана на платформе Java SE и предоставляет набор интерфейсов API для разработки и запуска кроссплатформенных, надежных, масштабируемых серверных приложений.

Java EE содержит следующие компоненты, которые активно использовались в работе:

1. Технология Enterprise JavaBeans (EJB) — позволяет осуществлять быструю и упрощенную разработку распределенных, транзакционных и безопасных приложений, созданных на Java;
2. JPA (Java Persistence API) — технология, дающая возможность управлять данными с помощью объектно-реляционного сопоставления (ORM);
3. JDBC API, который входит в Java SE — инфраструктура, с помощью которой происходит взаимодействие с базой данных через специальный драйвер по URL, который специфичен для каждой СУБД. Преимуществами использования JDBC является его легковесность, отсутствие необходимости устанавливать громоздкую клиентскую программу, а также неизменность кода при переходе на другую базу данных[17].

**4.1.1.2. Oracle Database 11g.**

Oracle Database 11g — объектно-реляционная система управления базами данных, созданная компанией Oracle.

Архитектура СУБД Oracle рассчитана на работу с огромными объемами данных и большим (десятки и сотни тысяч) числом пользователей; она демонстрирует широкие возможности обеспечения высокой готовности, производительности, масштабируемости, информационной безопасности и самоуправляемости.

СУБД Oracle может быть развернута на любой платформе, начиная от небольших серверов и заканчивая симметричными многопроцессорными компьютерами и мейнфреймами. Уникальная способность СУБД Oracle работать со всеми типами данных, от традиционных таблиц до XML-документов и картографических данных, позволяет рассматривать ее в качестве оптимального выбора для работы с приложениями оперативной обработки транзакций, поддержки принятия решений и управления коллективной работой с информацией.

В возможности Oracle Database входит:

* Real Application Cluster: обеспечивает работу одного экземпляра базы данных на нескольких узлах и позволяет масштабировать систему в случае необходимости и управлять нагрузкой;
* Automatic Storage Management: позволяет автоматически распределять данные между имеющимися ресурсами систем хранения данных, повышая таким образом отказоустойчивость системы;
* Простые средства разработки;
* Самоуправление: специальные механизмы Oracle Database позволяют самостоятельно перераспределять нагрузку на систему, выявлять и прогнозировать ошибки;
* Большие БД: максимальный размер БД может достигать 8 экзабайт;
* Высокая производительность;
* Автономные транзакции;
* Oracle Enterprise Manager: набор инструментов для управления и мониторинга СУБД Oracle и серверов, на которых они установлены;
* Total Recall: позволяет создать резервный сервер, который может работать в паре с основным, снижая нагрузку на него, и который может автоматически заменить основной сервер в случае его отказа или планового отключения[16].

**4.1.1.3. Oracle WebLogic 12c.**

Oracle WebLogic 12c — сервер приложений компании Oracle, работающий на практически всех популярных операционных системах и поддерживающий следующие стандарты и технологии:

* + Http
  + Java2 EE 6.0
  + JMS
  + Microsoft .NET
  + JDBC
  + EJB
  + J2EE Connectors
  + Сервлеты

Oracle WebLogic 12c является одним из наиболее популярных серверов приложений для реализации Java-приложений с трехуровневой архитектурой (СУБД – сервер приложений – тонкий клиент), поддержания работы web-сайтов, корпоративных порталов и транзакционных приложений. Сервер приложений Oracle Web Logic представляет собой интегрированную платформу ПО промежуточного слоя (Middleware), созданную на основе сервис-ориентированной архитектуры (Service-Oriented Architecture, SOA) и технологии сетей распределенной обработки данных (Grid Computing).

Преимущества Oracle WebLogic 12c[16, 17]:

* Безусловная доступность и высокое время безотказной работы для приложений и сервисов
* Продвинутый механизм предвосхищения проблем вследствие более надежного мониторинга и управления промышленными приложениями
* Повышенная целостность данных благодаря очень надежному и быстрому обмену сообщениями
* Упрощенная разработка новых приложений и сервисов
* Поддержка новых и усовершенствованных Web-сервисов
* Управляемая платформа времени выполнения
* Лучшая в отрасли производительность
* Повышенная эффективность эксплуатации
* Возможность апгрейда из предшествующих выпусков

**4.1.1.4. IDE Eclipse Mars**

Eclipse Mars — свободная интегрированная среда разработки кроссплатформенных приложений на Java, поддерживающая JDT — модуль, нацеленный на групповую разработку, то есть среда интегрирована с системами управления версиями (Git, SVN).

В основе Eclipse лежит платформа расширенного клиента, в которую входят следующие компоненты:

* OSGI (стандартная среда поставки комплектов);
* Ядро платформы;
* SWT (портируемый инструментарий виджетов);
* Jface (файловые буферы, текстовые редакторы);
* Рабочая среда Eclipse (панели, редакторы).

Среда поддерживает стандарт Java EE и имеет плагины для Maven.

**4.1.2. Результаты программной реализации**

Таблица 4. Результаты реализации

|  |  |  |
| --- | --- | --- |
| Модуль | Ожидаемые результаты | Фактические результаты |
| Кэширование данных | Увеличение производительности | + |
| Снижение времени ожидания пользователем результатов поиска | + |
| Уменьшение количества обращений к БД | + |
| Балансировка нагрузки | Увеличение отказоустойчивости кластера | + |
| Повышение надежности работы | + |
| Уменьшение нагрузки на отдельный узел в кластере | + |

По результатам программной реализации система приобрела следующие характеристики:

* Повышенная отказоустойчивость и надежность;
* Улучшение масштабирования;
* Высокая производительность работы.

Программная реализация описана в приложении 1А.

**4.2. Тестирование и заключительный анализ результатов программной реализации**

В данном подразделе описываются результаты тестирования реализации, прикладываются тестовые сценарии и скриншоты процесса тестирования. Проводится общий анализ эффективности примененных алгоритмов, сравнение с характеристиками системы до внедрения разработанных алгоритмов кэширования данных и метода балансировки нагрузки.

**4.2.1 Тестирование**

Тестовые сценарии и скриншоты процесса тестирования предстлавлены в приложении 1B.

Также проводилось UNIT-тестирование и интеграционное тестирование с помощью средств фреймворка JUnit.

**4.2.1 Анализ результатов**

По результатам тестов можно сказать об успешной интеграции реализованных алгоритмов с системой прогнозирования инкассаций CashManagement, а также об увеличении производительности работы приложения. За счет реализации алгоритма балансировки нагрузки повысилась отказоустойчивость системы. Алгоритм кэширования данных обеспечил уменьшение времени отклика интерфейса при совершении запросов за выборкой данных, а также количество обращений к БД. Время отклика в среднем уменьшилось с 300-350 мс до 10 мс при совершении запросов за теми данными, которые имеются в кэше.

**Заключение**

В ходе выполнения учебно-исследовательской работы было проведено исследование различных алгоритмов кэширования данных и методов балансировки нагрузки на кластер серверов для применения в системе прогнозирования CashManagement.

После проведения анализа был выбран алгоритм кэширования LRU, основанный на удалении из кэша последнего неиспользуемого элемента, а также алгоритм балансировки нагрузки Leasts connections, идея которого состоит в переадресации запроса серверу с наименьшим количеством активных подключений.

Для реализации балансировки нагрузки был выбран кластер из нескольких серверов приложений Weblogic 12c. Для данного кластера был реализован выбранный механизм кэширования данных, обеспечивающий высокую производительность и надежность.

Таким образом, модули кэширования данных и балансировки нагрузки была успешно интегрированы с системой CashManagement, развернутой на кластере из двух серверов приложений Oracle WebLogic 12c.

В дальнейшем планируется перевести систему на использование ORM для обеспечения безопасной работы с БД с нескольких серверов, так как данная реализация имеет недостатки.

**Список литературы**

1. K. Brown. PRPL: A databaseworkload specification language, v1.3. M.S. thesis, Univ. of WI, Madison, 1992.

2. S. Chaudhuri, R. Krishnamurthy, S. Potamianos, K. Shim. Optimizing queries with materialized views. Proc. of IEEE Conf. on Data Engineering, 1995.

3. M. Franklin, Client data caching: A foundation for high performance object database systems, Kluwer, 1996.

4 A. Keller, J. Basu. A predicate-basedcaching scheme for client-server database architectures. VLDB J, 5(1), 1996

5. H.V. Jagadish. Linear clustering of objects with multiple attributes. Proc. ACM SIGMOD Conf., 1990.

6. J. O’Toole, L. Shrira. Hybrid caching for large scale object systems. Proc. 6th Wkshp on Pers. Object Sys., 1994.

7. Zheng, Chunmei. Design and Implementation of a LoadBalancing Model for Web-Sever Cluster Systems. Engineering and Technology (S-CET), 2012 Spring Congress on. IEEE, 2012

8. Deris, M., M. Rabiei, A. Noraziah and H. Suzuri.High service reliability for cluster server systems. IEEE Intl. Conf. Cluster Computing, pp: 280-287, 2003.

9. C. Raja Kumar1; M. Rajinikannan.A Literature Survey on various Java Application Servers». International Journal of Software & Hardware Research in Engineereing. ISSN No: 2347-4890 Volume 2 Issue 2, 2014.

10. T. Abdellatif, E. Cecchet, R. Lachaize. Evaluation of a Group Communication Middleware for Clustered J2EE Application Servers. Appears in Proceedings of CoopIS/DOA/ ODBASE, Cyprus, October 2004.

11. S. Elnikety, S. Dropsho, W. Zwaenepoel. Memory-Aware Load Balancing and Update Filtering in Replicated Databases. ACM Sigops/Eurosys European Conference on Computer Systems 2007 (EuroSys '07), Lisbon, Portugal, March 2007.

12. Cardellini, Valeria, Michele Colajanni, and Philip S. Yu. Dynamic load balancing on web-server systems. Internet Computing, IEEE 3.3, pages: 28-39, 1999.

13. Kansal, N. J., & Chana, I. (2012). Cloud load balancing techniques: A step towards green computing. IJCSI International Journal of Computer Science Issues, 9(1), 238-246.

14. Nishant, K., Sharma, P., Krishna, V., Gupta, C., Singh, K. P., Nitin, N., & Rastogi, R. (2012, March). Load balancing of nodes in cloud using ant colony optimization. In Computer Modelling and Simulation (UKSim), 2012 UKSim 14th International Conference on (pp. 3-8). IEEE.

15. Klaithem Al Nuaimi, Nader Mohamed, Mariam Al Nuaimi and Jameela Al-Jaroodi. A Survey of Load Balancing in Cloud Computing: Challenges and Algorithms. 2012 IEEE Second Symposium on Network Cloud Computing and Applications, 978-0-7695-4943-9/12,pp:137-142.

16. T. Abdellatif, E. Cecchet, R. Lachaize, «Evaluation of a Group Communication Middleware for Clustered J2EE Application Servers», Appears in Proceedings of CoopIS/DOA/ ODBASE, Cyprus, October 2004.

17. C. Raja Kumar1; M. Rajinikannan, «A Literature Survey on various Java Application Servers». International Journal of Software & Hardware Research in Engineereing, ISSN No: 2347-4890 Volume 2 Issue 2, 2014.

**Приложение 1A**

**Класс MonitoringCacheItem**

Данный класс представляет собой описание одного элемента кэша.

Таблица А1. Class MonitoringCacheItem

|  |  |  |
| --- | --- | --- |
| public class MonitoringCacheItem | | |
| Элементы класса | | Описание |
| Поля | private List<IFilterItem<Integer>> filterList | Список банкоматов или групп автомат, поиск по которым осуществляется. |
| private MonitoringFilter filter | Объект фильтра для поиска. |
| private List<AtmActualStateItem> resultList | Результат поиска, который заносится в кэш. |
| private Timestamp timeStamp | Время последнего обращения к этому элементу кэша. |
| Конструкторы | **public** MonitoringCacheItem() | Конструктор по-умолчанию. |
| **public** MonitoringCacheItem(List<IFilterItem<Integer>> filterList, MonitoringFilter filter, Timestamp timeStamp, List<AtmActualStateItem> resultList) | Конструктор, инициализирующий поля параметрами, переданными в параметрах. |
| Методы | **public** List<IFilterItem<Integer>> getFilterList() | Метод, возвращающий значение поля filterList. |
| **public** MonitoringFilter getFilter() | Метод, возвращающий значение поля filter. |
| **public** Timestamp getTimeStamp() | Метод, возвращающий значение поля timeStamp. |
| **public** List<AtmActualStateItem> getResultList() | Метод, возвращающий значение поля resultList. |
| **public** **void** setTimeStamp(Timestamp timeStamp) | Метод, устанавливающий значение поля timeStamp. |
| **public** **void** setFilterList(List<IFilterItem<Integer>> filterList) | Метод, устанавливающий значение поля filterList. |
| **public** **void** setFilter(MonitoringFilter filter) | Метод, устанавливающий значение поля filter. |
| **public** **void** setResultList(List<AtmActualStateItem> resultList) | Метод, устанавливающий значение поля resultList. |
| **public** List<AtmActualStateItem> getResultListPart(List<IFilterItem<Integer>> atms, MonitoringFilter filter) | Метод, возвращающий часть имеющегося значения из поля resultList. |

**Класс CacheUtils**

Данный класс реализует методы для работы с кэшом.

Таблица А2. Class CacheUtils

|  |  |  |
| --- | --- | --- |
| public class CacheUtils | | |
| Элементы класса | | Описание |
| Методы | **private** **static** **boolean** isUpdated(MonitoringFilter filter, Object hash, CmMonitoring cmMonitoringEJB, CmCache cmCache) | Метод для проверки актуальности кэша и последующего его обновления в случае потери им актуальности. |
| **public** **static** List<AtmActualStateItem> fromCache(MonitoringFilter filter, CmMonitoring cmMonitoringEJB, CmCache cmCache) | Метод для получения данных из кэша. Если данных в кэше нет, они будут добавлены в кэш. |
| Поля | **private** **static** ConcurrentMap<Object, MonitoringCacheItem> *cacheMap* | Кэш в виде потокобезопасной карты (ключ → значение). |

**Интерфейс CmCache**

Данный интерфейс содержит методы для работы операций кэша, требующие обращений к БД.

Таблица А3. Interface CmCache

|  |  |  |
| --- | --- | --- |
| public interface CmCache | | |
| Элементы интерфейса | | Описание |
| Методы | **public** **boolean** isNeedUpdateCache(String schemaName, Timestamp timestamp) | Метод для получения данных сессии по идентификатору пользователя проверки необходимости обновления данных в кэше. |

**Класс CmCacheBean**

Данный класс реализует методы для работы с кэшом на уровне запросов к БД посредством использования методов класса CacheController.

Таблица А4. Class CmCacheBean

|  |  |  |
| --- | --- | --- |
| public class CmCacheBean implements CmCache | | |
| Элементы класса | | Описание |
| Методы | **public** **boolean** isNeedUpdateCache(String schemaName, Timestamp timestamp) | Метод, возвращающий true, если для данного пользователя есть запись в таблице, false - иначе |
| private Connection getConnection() | Метод для установления соединения с БД |
| Конструкторы | public CmCacheBean() | Конструктор по умолчанию |
| Поля | private static final Logger \_logger | Обеспечивает логирование действий |
| private DataSource dataSource | Data source |

**Класс CacheController**

Данный класс реализует методы для работы с базой данных на уровне бизнес-логики посредством использования JDBC API.

Таблица А5. Class CacheController

|  |  |  |
| --- | --- | --- |
| public class SessionReplicationController | | |
| Элементы класса | | Описание |
| Методы | **public** **static** **boolean** isNeedUpdateCache(Connection conn, String schemaName, Timestamp timestamp) | Метод для проверки необходимости обновления данных в кэше. |
| Поля | private static final Logger \_logger | Обеспечивает логирование действий |

**Интерфейс IClusterInfo**

Данный интерфейс содержит в себе методы для работы с узлами в кластере.

Таблица А6. Interface IClusterInfo

|  |  |  |
| --- | --- | --- |
| **public** **interface** IClusterInfo | | |
| Элементы интерфейса | | Описание |
| Методы | Integer getNumberNodes() | Метод, возвращающий количество узлов в кластере. |
| **boolean** needRedirect(ClusterNode node) | Метод, определяющий, надо ли перенаправить запрос на другой узел кластера. |
| ClusterNode initialize(ServletRequest request, ServletResponse response) | Метод, инициализирующий узел кластера и возвращающий его. |
| ConcurrentMap<Integer, ClusterNode> nodes() | Возвращает список узлов в виде карты. |

**Класс ClusterInfo**

Данный класс реализует методы для работы с узлами в кластере.

Таблица А7. Class ClusterInfo

|  |  |  |
| --- | --- | --- |
| **public** **class** ClusterInfo **implements** IClusterInfo | | |
| Элементы класса | | Описание |
| Методы | public static ClusterInfo getClusteInfo() | Создает новый инстанс класса ClusterInfo. |
| public boolean needRedirect(ClusterNode node) | Метод, определяющий, надо ли перенаправить запрос на другой узел кластера. |
| public ClusterNode initialize(ServletRequest request, ServletResponse response) | Метод, инициализирующий узел кластера и возвращающий его. |
| public ConcurrentMap<Integer, ClusterNode> nodes() | Возвращает список узлов в виде карты. |
| public static String computeURL(ClusterNode node, ServletRequest request) | Метод, возвращающий новый URL переадресованного запроса. |
| public Integer getNumberNodes() | Метод, возвращающий количество узлов в кластере. |
| Поля | **private** **static** ConcurrentMap<Integer, ClusterNode> nodes | Потокобезопасная карта узлов кластера. |

**Класс ClusterNode**

Данный класс представляет собой описание одного узла кластера.

Таблица А8. Class ClusterNode

|  |  |  |
| --- | --- | --- |
| public class ClusterNode | | |
| Элементы класса | | Описание |
| Поля | **private** **int** port | Номер порта узла. |
| **private** String hostName | Имя хоста узла. |
| **private** String address | Адрес узла. |
| **private** **long** requested | Счетчик активных подключений к узлу. |
| Конструкторы | ClusterNode() | Конструктор по-умолчанию. |
| **public** ClusterNode(String hostName, String address, **int** port) | Конструктор, инициализирующий поля параметрами, переданными в параметрах. |
| Методы | **public** **void** setPort(**int** port) | Метод, устанавливающий значение поля port. |
| **public** **void** setHostName(String hostName) | Метод, устанавливающий значение поля hostName. |
| **public** **void** setAddress(String address) | Метод, устанавливающий значение поля address. |
| **public** **int** getPort() | Метод, возвращающий значение поля port. |
| **public** String getHostName() | Метод, возвращающий значение поля hostName. |
| **public** String getAddress() | Метод, возвращающий значение поля address. |
| **public** **void** plus() | Метод, увеличивающий значение поля requested на 1. |
| **public** **static** ClusterNode getNode() | Метод, возвращающий инстанс данного класса. |
| **public** **void** minus() | Метод, уменьшающий значение поля requested на 1. |
| **public** **long** getRequestCounter() | Метод, возвращающий значение счетчика активных подключений. |
| **public** **void** doRedirect(ServletRequest request, ServletResponse response, FilterChain chain) | Метод, выполняющий перенаправление по новому URL. |

**Приложение 1B**

Тестовый сценарий:  
1. Авторизоваться в системе CashManagement (см. рисунок «Вход»).

2. Выполнить запрос за выборкой данных — засечь время выполнения операции.

3. Выполнить повторно поисковый запрос — засечь время выполнения.  
4. Выполнить подобный запрос, но с меньшим количеством поисковых параметров — засечь время выполнения операции.  
Результаты теста: после выполнения первого запроса время выполнения операции уменьшилось с 300-350 мс до 5-10 мс.

Рисунок 6. Вход