МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н.Г.ЧЕРНЫШЕВСКОГО»

Кафедра информатики и программирования

ОПТИМИЗАЦИЯ ПРОИЗВОДИТЕЛЬНОСТИ ВЫСОКОНАГРУЖЕННЫХ ПРИЛОЖЕНИЙ С ИСПОЛЬЗОВАНИЕМ КЭШИРОВАНИЯ

АВТОРЕФЕРАТ БАКАЛАВРСКОЙ РАБОТЫ

студента 4 курса 441 группы
направления 02.03.03 Математическое обеспечение и администрирование информационных систем
факультета компьютерных наук и информационных технологий
Мартышина Ивана Алексеевича

Научный руководитель:
Доцент _____ Е.В. Кудрина
подпись, дата

Зав. кафедрой:
Ученая степень, звание _____ М.В. Огнева
подпись, дата

Саратов 2022

ВВЕДЕНИЕ

Актуальность темы. В настоящее время любое высоконагруженное приложение обрабатывает большие объемы данных, что ставит перед разработчиками задачу более тщательно подходить к вопросу оптимизации и находить такие решения, которые позволят приложению передавать большие объемы информации за минимальное время.

Сейчас кэш применяется во многих проектах и фреймворках, этот мощный паттерн значительно помогает ускорить работоспособность приложения и увеличить производительность. Так как быстродействие приложения сейчас является одним из залогов его успеха, внедрение кэша позволит достичь данной цели с минимальными затратами и рисками, а также повысит пропускную способность и стабильность системы при высоких нагрузках.

Цель бакалаврской работы – исследовать возможность оптимизации производительности высоконагруженных клиентоориентированных приложений за счет использования кэширования.

Поставленная цель определила **следующие задачи**:

1. рассмотреть классификацию методов оптимизации программ;
2. изучить особенности использования кэширования при оптимизации программ;
3. представить обзор инструментальных средств и библиотек для программной реализации кэширования;
4. описать структуру высоконагруженного клиентоориентированного приложения;
5. реализовать локальное и распределенное кэширование, а также кэш-менеджер для их синхронизации;
6. провести анализ производительности приложения после внедрения в него кэширования.

Методологические основы оптимизации программ, в том числе с помощью кэширования представлены в работах Прайса М. [1], Дж. Рихтера [2], С.С. Скиена Дж.[3], Скита Д. [4].

Практическая значимость бакалаврской работы. Практическая значимость данной работы заключается в изучении подхода для реализации распределенного кэширования с возможностью контролирования зависимостей сущностей для корректной синхронизации кэша между экземплярами приложений и поддержание его в актуальном состоянии.

Структура и объём работы. Бакалаврская работа состоит из введения, двух разделов, заключения, списка использованных источников и 5 приложений. Общий объем работы – 57 страниц, из них 44 страницы – основное содержание, включая 16 рисунков и 2 таблиц, цифровой носитель в качестве приложения, список использованных источников информации – 25 наименований.

КРАТКОЕ СОДЕРЖАНИЕ РАБОТЫ

Первый раздел «Теоретический часть» посвящен классификации методов кэширования, особенностям использования кэширования при оптимизации программ и обзору инструментальных средств и библиотек для программной реализации кэширования.

1.1 Классификация методов оптимизации программ

В данном разделе рассмотрены и изучены основные понятия оптимизации и самые распространённые классификации методов оптимизации программ такие как, обновление программного обеспечения (ПО), избавление от избытков кода, кэширование, распараллеливание и отложенные вычисления, использование более оптимальных структур данных и алгоритмов, обновление или добавление нового оборудования.

1.2 Особенности использования кэширования при оптимизации программ

Одним из наиболее часто используемых паттернов в разработке

программного обеспечения является **кэширование**. Это простая и эффективная система для повторного использования результатов уже выполненных операций.

Выполняя запрос за большими данными, они будут кэшироваться и повторное выполнение запроса будет использовать уже за кэшированные данные, вместо того что бы обращаться за ними в базу данных.

Существует 3 типа кэша:

- **Кэш в памяти** (In-Memory Cache) Используется в рамках одного экземпляра приложения и храниться в оперативной памяти, что делает доступ к нему очень быстрым.
- **Постоянный локальный кэш (Persistent in-process Cache)** — идея заключается в том, чтобы хранить копию кэша вне текущего процесса, например в файле, или в той же базе данных.
- **Распределенный кэш (Distributed Cache)** — позволяет иметь общий кэш для нескольких экземпляров приложений или серверов. Распределенный кэш выноситься в отдельный модуль, чтобы он был слабо связан со всей структурой приложения и мог быть отключен по необходимости, само же хранилище – отдельная служба, к которой у всех серверов есть доступ, например база данных.

1.3 Обзор инструментальных средств и библиотек для программной реализации кэширования

В данном разделе были рассмотрены основные инструментальные средства и библиотеки для программной реализации и проведения сравнительного анализа, такие как, язык C#, платформа .NET, SQL Server, Entity Framework, Apache JMetr, NCache, FusionCache, EasyCache.

Второй раздел «Практическая часть» посвящен реализации библиотеки распределенного кэширования и сравнительному анализу производительности приложения с применением кэша и без него.

2.1 Постановка задачи

Разработать библиотеку распределенного кэширования и использовать ее для оптимизации производительности высоконагруженного клиентоориентированного приложения.

2.2 Структура высоконагруженного клиентоориентированного приложения

В данном разделе описана структура данных приложения, к которому по итогу будет применено распределенное кэширование, приложение является коммерческим продуктом, и его структура может быть описана только абстракциями.

Приложение представляет собой клиентоориентированную систему, хранящую большие наборы данных как о самом клиенте, так и о связанных с этим клиентом сущностях.

Структура приложения представлена на рисунке 1.

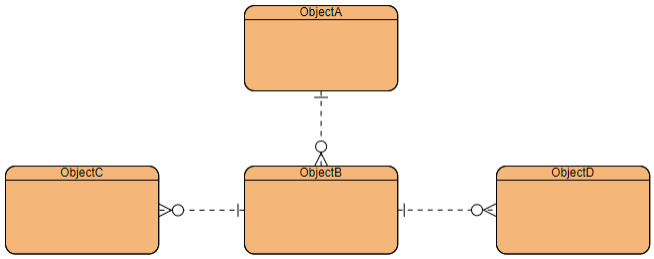

Рисунок 1 – Схема данных приложения

Объект А имеет связь 1 ко многим с Объект Б, это означает что одному Объекту А соответствует множество Объектов Б, аналогично с самим Объектом Б, ему в соответствие ставятся множества Объектов С и Объектов Д, так как они соединены с ним связью 1 ко многим.

Следовательно, при запросе объекта А, необходимо собрать информацию о всех связанных сущностях Объектов Б, потенциально их может быть несколько тысяч, затем для каждого объекта Б, собрать объекты С, количество таких объектов для каждого объекта Б может доходить до 1 миллиона записей, связных объектов Д для каждой записи может быть от 1 до 10.

2.3 Особенности реализации локального кэша

Кэш может находиться как на стороне клиента, так и на стороне сервера, но стоит учесть тот факт, что такой кэш будет локальным, он будет существовать в рамках одного клиента или сервера. Имея множество серверов, локальный кэш будет актуальным лишь для каждой машины в отдельности, что является его недостатком при распределенной системе.

Если рассматривать ситуацию, когда приложение работает в рамках одной машины, локальный кэш будет хорошим решением для оптимизации приложения.

Рассмотрим этапы реализации локального кэша. Первое что необходимо это продумать - как хранить кэш в памяти, есть множество вариантов от массивов, списков, самописных коллекций до специальных классов представляющимся платформой.

В .NET Framework 4.7.2 присутствует класс MemoryCache, он представляет сам по себе простой локальный кэш, который включает в себя хранилище, с базовым набором методов для работы и специальной структурой для хранения кэша и будет представлен в виде поля в данной реализации кэша. Далее необходимо реализовать основные методы для работы с кэшем.

Метод GetByKeyAsync, он представляет возможность извлекать запись из кэша, по переданному ему stringKey.

Далее идет второй основной метод в реализации кэша AddOrReplaceAsync, его основная задача - добавить кэш в память или заменить существующее значение новым.

Метод GetByKeyOrAddAsync фактически является аналогом метода GetByKeyAsync, за одним исключением, рассмотрим фрагмент метода.

И самый простой в реализации метод удаления записей из кэша DeleteByKey.

2.4 Особенности реализации распределенного кэша

Как решить проблему и синхронизировать кэш между серверами? Тут на помощь приходит распределенный кэш, он дает единое хранилище, в котором будет храниться кэш.

Для реализации распределенного кэша в данной работе будет использован MS Sql Server 2016. В случае локального кэша, хранилищем выступал MemoryCache класс, имеющий функционал, представляющий как минимум методы добавления, получения кэша и контроля его актуальности.

С реализацией распределенного кэша, всю эту функциональность необходимо прописывать руками. Первое, что необходимо, это определить схему того, как кэш будет представлен в базе данных, ее можно увидеть на рисунке 2.

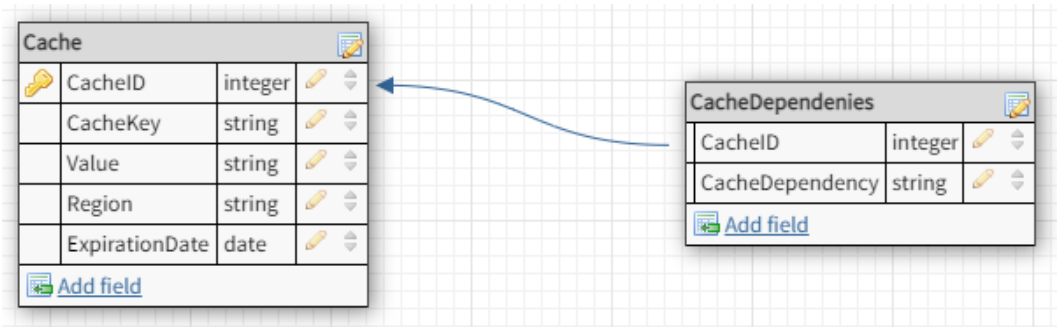

Рисунок 2 – Схема распределенного кэша в базе данных

Таблица Cache представляет из себя то самое хранилище, «ключ-значение», где CacheKey это ключ, а Value соответственно значение, хранящее данные в формате JSON строки, поле Region представляет дополнительную функциональность, которая позволяет разделять кэш на группы и производить более быструю фильтрацию и удалять кэш, относящийся к одной группе. Также есть поле ExpirationDate, которое содержит в себе информацию о сроке действия кэша. И так же продумать то как будет происходит управление зависимостями, что бы в случае удаления одной записи, очищался кэш записей зависящий от удаляемой данная функциональность будет достигаться по средствам реализации триггера INSTEAD OF.

Далее необходимо реализовать основные методы для работы с кэшем.

Метод GetByKeyAsync предоставляет возможность извлекать кэш из базы данных по переданному ему ключу кэша.

Метод AddOrReplaceAsync предоставляет возможность, как и в случае локального кэша, добавлять или заменять существующие записи в кэше, но за тем отличием, что он так же принимает в свои параметры список зависимостей кэшируемой сущности.

Метод GetByKeyOrAddAsync, он так же представляет из себя метод похожий по смыслу, с таким же методом в локальном кэше, но его сигнатура расширена одним дополнительным параметром для работы с зависимостями.

Метод GetExpirationDate, данный метод позволяет свести количество дорогостоящих обращений к базе данных к минимуму, он будет возвращать в ответ лишь дату окончания действия кэша для запрошенной записи.

2.5 Реализация кэш менеджера для синхронизации локального и распределенного кэша

Реализация кэш менеджера представляет из себя компоновку уже ранее реализованной логики распределенного и локального кэша, что и позволяет сделать единый механизм управления кэшем как в базе данных, так и на каждой машине в отдельности.

Далее приведен алгоритм работы кэш менеджера:

Шаг 1: Запрос приходит на сервер

Шаг 2:

- Если запись существует локально, идет запрос в базу и определяется, совпадают ли даты локальной записи и записи в базе, если кэш является актуальным, он извлекается;
- Если запись существует локально, но при этом кэш в базе имеет другой срок актуальности, кэш перезаписывается на основе кэша из базы;
- Если кэш не существует локально, идет запрос в базу и в случае, если кэш там существует, идет синхронизация с локальной машиной;
- Если кэш не существует локально и отсутствует в базе, создается новая запись кэша.

Шаг 3: Приложение продолжает работать с кэшированными данными.

Данный алгоритм позволяет реализовать кэш менеджер, который можно внедрить кэш в любую часть программы, где необходима оптимизация.

Основные методы для работы с кэшем:

- **AddOrReplaceAsync** – добавляет или заменяет уже существующую запись в кэше;
- **GetByKeyOrAddAsync** – получает или добавляет запись, если она отсутствует;
- **GetByKeyAsync** – возвращает кэш или в случае его отсутствия null;
- **GetCacheByKey** – возвращает сущность, включающую в себя все метаданные о записи в кэше, включая время действия кэша, регион и сущность.

2.6 Анализ производительности приложения после внедрения в него кэширования

Данный раздел посвящен сравнительному анализу того, как внедренный кэш влияет на производительность приложения. Данные тесты были произведены на коммерческом продукте, что позволяет произвести тестирование с большим набором данных. Далее приведены таблицы с результатами выше проведенного анализа, таблица 1 представляет данные запросов без кэша, таблица 2 с кэшем.

Таблица 1 – Средние показатели времени запросов без кэша

	Среднее (мс)	Макс. среднее (мс)	Мин. среднее (мс)	Медиана (мс)
Объект А	28	38	25	28
Объект Б	18	21	15	18
Объект С	26	27	26	27
Объект Д	17	19	16	17

Таблица 2 – Средние показатели времени запросов с кэшем

	Среднее (мс)	Макс. среднее (мс)	Мин. среднее (мс)	Медиана (мс)
Объект А	26	27	24	27
Объект Б	18	29	15	18
Объект С	26	27	26	27
Объект Д	17	19	16	17

Внедрение распределенного кэша в описанное выше приложение внесло значительный вклад в его оптимизацию и стабильную работу, находясь под высокими нагрузками.

ЗАКЛЮЧЕНИЕ

В ходе данной выпускной работы были изучены теоретические основы кэширования, его типы и место в разработке и оптимизации программного обеспечения. Также на практике были решены все поставленные задачи и как результат, была разработана библиотека распределённого кэширования, позволяющая внедрить кэш в проблемное место приложения, требующего оптимизации. Применение разработанной библиотеки позволило провести нагрузочное тестирование и проверить эффективность данной библиотеки. Результаты нагрузочного тестирования показали, что применение распределённого кэширования значительно увеличило производительность приложения и повысило его стабильность за счет повышения пропускной способности, что в итоге позволит выдержать высокие нагрузки.

Как показала практика, разработанная библиотека и сам кэш имеет актуальное практическое применение, библиотека может быть использована и в других приложениях, только с той поправкой, что необходимо учитывать детали реализации данной библиотеки, т. к. она была разработана с учетом особенностей приложения, в которое была внедрен. Но, несмотря на это, библиотека имеет открытый исходный код, что позволит изучить подход, примененный в данной библиотеке, использовав его для реализации новой или же внести изменения и расширить данную библиотеку под особенности нового проекта.

В будущем для данной библиотеки может быть изменен подход к реализации распределённого кэша, а конкретно, работа с зависимостями кэша может быть усовершенствована до универсального решения, что позволит внедрять ее в любой проект, не зависимо от особенностей его структуры данных. Но такая разработка потребует тщательного анализа проблемы и продумывания алгоритмов для ее решения.

Основные источники информации:

1. Прайс Марк П68 C# 9 и .NET 5. Разработка и оптимизация. — СПб.: Питер, 2022. — 832 с.: ил. — (Серия «Для профессионалов»).
2. Рихтер Дж. Р55 CLR via C#. Программирование на платформе Microsoft .NET Framework 4.5 на языке C#. 4-е изд. – СПб.: Питер. 2018. – 896 с.: ил. – (Серия «Мастер-класс»).
3. Скиена С. С42 Алгоритмы. Руководство по разработке. — 2-е изд.: Пер. с англ. — СПб.: БХВ-Петербург, 2018. — 720 с.: ил.
4. Скит, Джон. С42 C# для профессионалов: тонкости программирования, 3-е изд. : Пер. с англ. — М. : ООО “И.Д. Вильямс”, 2019. – 608 с. : ил. — Парал. тит. англ.