DOI 10.37882/2223-2966.2024.7.32

РАЗРАБОТКА И ИССЛЕДОВАНИЕ СЕРВИСА ДЛЯ ДЕДУПЛИКАЦИИ ДАННЫХ В ХРАНИЛИЩЕ

DEVELOPMENT AND RESEARCH OF A DATA DEDUPLICATION SERVICE FOR STORAGE SYSTEMS

A. Pantykhin
V. Gladun
I. Malinin
S. Molodyakov

Summary. This paper examines a deduplication service based on hash functions to minimize the volume of stored data. The main algorithm involves segmenting data into fixed-size blocks, calculating unique hash values for each segment, and saving only unique data blocks while creating references for duplicates. The technological stack includes Python, MongoDB, and the MongoEngine library. The paper presents research results related to the use of different hashing algorithms and data segment sizes.

Keywords: data deduplication, hash functions, data storage systems, storage optimization, MongoDB, Python, MongoEngine.

Пантюхин Андрей Максимович
Санкт-Петербургский политехнический университет Петра Великого
panandafog@gmail.com
Гладун Владимир Вадимович
Санкт-Петербургский политехнический университет Петра Великого
vladimir.gldn@gmail.com
Малинин Илья Игоревич
Санкт-Петербургский политехнический университет Петра Великого
malinin.ilja@gmail.com
Молодяков Сергей Александрович
Д.т.н., профессор, Санкт-Петербургский политехнический университет Петра Великого
molodyakov_sa@spbstu.ru

Аннотация. Рассмотрен сервис дедупликации на основе хеш-функций для минимизации объема хранимых данных. Основной алгоритм работы включает разделение данных на сегменты фиксированного размера, вычисление уникальных хеш-значений для каждого сегмента, и сохранение только уникальных блоков данных с созданием ссылок на дубликаты. Технологический стек включает Python, MongoDB и библиотеку MongoEngine. Приводятся результаты исследования, связанные с использованием разных алгоритмов хеширования, размеров сегментов деления данных.

Ключевые слова: дедупликация данных, хеш-функции, система хранения данных, оптимизация хранения, MongoDB, Python, MongoEngine.

Введение

В условиях постоянно растущего объема генерируемых данных актуальность эффективного и экономичного способа их хранения становится очевидной. Увеличение объема данных обусловлено как расширением информационных потребностей бизнеса, так и прогрессом в области интернета вещей и цифровой аналитики. Существующие подходы к хранению данных, в том числе использование облачных технологий и распределенных систем, обеспечивают масштабируемость и доступность данных, однако вызывают рост затрат на обслуживание данных из-за их объема.

Текущие технологии решают проблему хранения за счет применения различных методов сжатия и оптимизации данных, в том числе дедупликацию [1]. Дедупликация позволяет исключить сохранение множественных копий одних и тех же данных, эффективно снижает физический объем информации и, соответственно, затраты на ее хранение. Несмотря на значительные достижения в этой области, большинство существующих систем ориентированы на использование в облачных или масштабных распределенных средах, что делает их не всегда приемлемыми для локальных или специализированных применений.

В связи с этим, целью данной работы является разработка прототипа локальной системы дедупликации, способной анализировать потоки данных, выделять уникальные блоки и минимизировать дублирование данных на физических носителях путем использования эффективной хеш-функции и механизмов управления ссылками на данные.

Анализ методов дедупликации данных

Дедупликация — это процесс идентификации и удаления дублирующихся копий данных. Основные методы дедупликации — это пост-обработка и встроенная дедупликация (в реальном времени). Дедупликация данных с использованием пост-обработки была выбрана как наиболее эффективный метод для оптимизации процесса хранения данных в рамках данного исследования.

Классифицировать методы дедупликации можно на основе различных критериев, таких как время выполнения операции дедупликации (в реальном времени или отложенное), уровень дедупликации (на уровне файлов или блоков), а также способ идентификации дубликатов (с использованием хеш-функций или других методов). Перечислим методы дедупликации.

Использование хеш-функций [2]. Хеш-функции преобразуют блоки данных в короткие уникальные хеш-коды, что позволяет быстро и эффективно идентифицировать дубликаты данных. Это обеспечивает высокую скорость и надежность при сравнении данных и широко используется для дедупликации из-за простоты интеграции и управления.

Битовое сравнение (Byte-level comparison) [3]. Этот метод включает полное посимвольное сравнение данных, что гарантирует высокую точность в определении дубликатов. Однако такое сравнение требует значительных вычислительных ресурсов и времени, особенно при работе с большими объемами данных.

Сравнение по сигнатурам [4]. Данные обрабатываются для создания сокращенных представлений или сигнатур, которые затем используются для сравнения. Это похоже на хеш-функции, но сигнатуры могут быть адаптированы для отражения специфических аспектов данных, что позволяет гибко подходить к процессу дедупликации.

Семантический анализ [5]. Семантический анализ учитывает смысловое содержание данных, а не только их структуру или содержание на битовом уровне. Это особенно полезно для систем, в которых необходимо понимать контекст и значение информации, чтобы корректно определить дубликаты.

Фильтрация на основе правил (Rule-based filtering) [6]. Этот метод использует предопределенные правила для идентификации дубликатов, основываясь на атрибутах данных, таких как дата, размер или автор. Это позволяет настроить процесс дедупликации под конкретные нужды и требования.

Дедупликация на основе контекста [7]. Контекстнозависимая дедупликация учитывает окружение или ситуацию, в которой используются данные. Это помогает минимизировать ошибки при дедупликации данных, которые могут быть одинаковыми, но иметь разное значение в разных условиях.

Выбор хеш-функций для дедупликации в нашем исследовании обусловлен несколькими ключевыми факторами. Во-первых, хеш-функции обеспечивают высокую скорость обработки данных, что критически важно при работе с большими объемами информации. Во-вторых, они позволяют значительно сократить объем необходимого хранилища, сохраняя только уникальные блоки данных. В-третьих, хеш-функции предоставляют достаточную надежность и точность для большинства приложений, так как современные алгоритмы хеширования минимизируют риск коллизий. Наконец, этот метод легко интегрируется в существующие системы и не требует сложной настройки, что делает его идеальным выбором для эффективного и надежного решения задачи оптимизации хранения данных.

Анализ существующих решений

В процессе разработки прототипа системы дедупликации был проведен анализ существующих решений на рынке, чтобы определить текущие тенденции, возможности и ограничения. Сравнение с аналогичными решениями позволило выявить ключевые аспекты, которые необходимо учесть при создании системы. Далее приведены некоторые популярных систем дедупликации и их особенности.

1. Data Domain (Dell EMC) [8]. Преимущества: поддерживает различные уровни данных и типы нагрузок; имеет высокую производительность и надежность. Ограничения: высокая стоимость решений, которая может быть неоправданной для малых и средних предприятий.
2. Microsoft Windows Server Deduplication [9]. Преимущества: интегрированная с Windows Server функция дедупликации, обеспечивающая удобство использования и управления при низкой стоимости. Ограничения: менее эффективна при большом объеме маленьких файлов и в условиях высокой нагрузки.
3. Veeam Backup & Replication [10]. Преимущества: предлагает решения, ориентированные на виртуальные среды, с возможностями резервного копирования и дедупликации. Ограничения: Преимущественно ориентирован на виртуальные и облачные среды, что может ограничивать его применение в традиционных IT-инфраструктурах.
4. Veritas NetBackup [11]. Преимущества: обеспечивает комплексное решение для защиты данных с высоким уровнем масштабируемости и поддержкой разнообразных типов данных и нагрузок. Ограничения: требует значительных инвестиций в лицензирование и поддержку.
5. Experian Data Quality Software [12]. Преимущества: обеспечивает высокое качество данных с функциями дедупликации, помогая организациям поддерживать чистоту и точность баз данных. Ограничения: может быть сложно интегрировать и настроить в существующие базы данных.

Многие из существующих решений требуют значительных начальных вложений и имеют высокую стоимость обслуживания. На основе проведенного анализа было принято решение разработать систему дедупликации, которая будет экономически доступна для малых и средних предприятий, обладая при этом высокой производительностью и простотой интеграции и управления.

Архитектура системы

Архитектура системы включает следующие компоненты (рис. 1).

- File Storage. Директория, где хранятся файлы для обработки.
- File Reader. Чтение/Запись файлов: Считывает файлы из/в хранилища.
- File Splitter. Разделение файлов на сегменты: Для процесса дедупликации файлы разделяются на сегменты одинакового размера. Соединение сегментов в файлы: Объединяет сегменты обратно в исходные файлы в процессе восстановления.
- Hash Generator. Генерация хеша из данных. Создает уникальный идентификатор для каждого сегмента файла, используя выбранный алгоритм хеширования.
- Deduplicator / Restorer. Сохранение сегментов и их хешей: Отвечает за сохранение уникальных сегментов и их соответствующих хешей. Чтение сегментов: извлекает сегменты из хранилища для обработки. Восстановление файлов из сегментов: собирает файлы из сегментов, удаляя дубликаты.
- Database Connector. Чтение/Запись в базу данных.
- Hash Table. Хранит хеши сегментов файлов для быстрого доступа и сопоставления.
- Segments Storage. Хранит уникальные сегменты файлов для последующего восстановления исходных файлов.

Стек технологий

Выбранный технологический стек для создания сервиса дедупликации данных включает использование Python, MongoDB и библиотеки MongoEngine [13].

MongoDB — это NoSQL база данных, которая предлагает ряд преимуществ для систем дедупликации. MongoDB не требует фиксированной схемы данных, что позволяет легко адаптировать базу данных под изменяющиеся требования к данным. MongoDB эффективно масштабируется, позволяя обрабатывать большие объемы данных и высокую нагрузку. MongoEngine — это библиотека-оболочка для MongoDB, написанная на Python. Она упрощает работу с MongoDB, предоставляя классы и функции, похожие на ORM, что облегчает манипуляцию данными. MongoEngine специально разработан для Python, он обеспечивает лучшую интеграцию и использование питонических идиом.

Выбор данного стека технологий обоснован необходимостью обработки больших объемов данных с гибкостью и эффективностью. Python обеспечивает удобную разработку и поддержку, MongoDB предлагает нужную производительность и масштабируемость для работы с большими объемами данных, а MongoEngine упрощает взаимодействие между Python и MongoDB, делая разработку более интуитивной и эффективной.

Алгоритм работы системы

Алгоритм дедупликации неструктурированных данных, применяемый в разрабатываемой системе, основывается на следующих последовательных шагах, которые обеспечивают оптимизацию процесса хранения данных:

1. Разделение входного потока данных. Исходные данные первоначально разделяются на сегменты фиксированного размера. Этот размер выбирается таким образом, чтобы обеспечить оптимальное

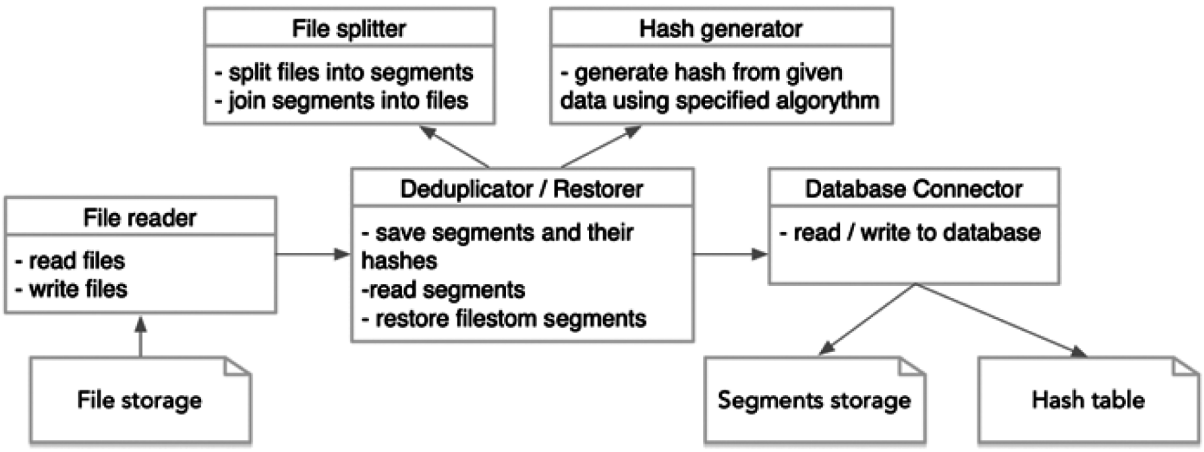

Рис. 1. Архитектура системы

сочетание производительности и эффективности хранения, минимизируя при этом дублирование содержимого.

2. Вычисление хеш-значений. Для каждого отдельного блока данных вычисляется уникальное хеш-значение. Этот процесс включает применение алгоритма хеширования, такого как SHA-256 или MD5, который преобразует блок данных в короткий числовой код, характеризующий данный блок.
3. Проверка наличия хеш-значения в таблице. Полученное хеш-значение каждого блока сравнивается с уже существующими хеш-значениями в таблице, которая содержит записи о всех ранее сохраненных блоках данных. Эта таблица служит своего рода индексом, позволяющим быстро определять, сохранялся ли ранее такой же блок данных.
4. Сохранение данных или создание ссылки. Если блок с таким же хеш-значением уже существует на носителе, то вместо повторного сохранения блока в хранилище создается только ссылка на уже существующий блок. Это позволяет избежать дублирования и значительно сократить объем используемого дискового пространства. Если же такого хеш-значения в таблице нет, новый блок данных сохраняется на диске, а его хеш-значение вместе с указанием на его расположение добавляется в таблицу.

Программная реализация

Структура проекта выглядит следующим образом (см. рис. 2).

Python-скрипты:

- comparator.py: Используется для сравнения данных до и после дедупликации.
- config.py и config.ini: Файлы конфигурации, где config.py может использоваться для чтения и обработки данных из config.ini.
- deduplicator.py: Основной скрипт для процесса дедупликации данных.
- restorer.py: Используется для восстановления исходных файлов из данных, полученных при дедупликации.
- hasher.py: Используется для создания хешей.
- performance_measurer.py: Используется для измерения производительности сервиса дедупликации.
- reader_writer.py: Скрипт для чтения и записи данных в файлы.
- splitter.py: Скрипт для разделения данных на сегменты и создания файлов из набора сегментов.

Директории:

- database: Содержит скрипты для работы с базой данных, включая инициализацию (db.py) и модели данных (models.py).

```
.
├── README.md
├── comparator.py
├── config.ini
├── config.py
├── database
│   ├── __init__.py
│   ├── db.py
│   └── models.py
├── deduplicator.py
├── docker-compose.yml
├── restorer.py
├── file_storage
│   └── input
│       └── input.txt
├── hasher.py
├── mongo-init.js
├── mongodb-data
├── mongodb-log
├── performance_measurer.py
├── reader_writer.py
├── requirements.txt
├── splitter.py
├── tests
│   ├── files
│   │   ├── results.csv
│   │   ├── sample_100kb.txt
│   │   ├── sample_10kb.txt
│   │   ├── sample_2kb.txt
│   │   ├── sample_2mb.txt
│   │   ├── sample_500kb.txt
│   ├── test_database.py
│   ├── test_deduplicate_and_restore.py
│   ├── test_performance.py
│   └── test_splitter.py
└── venv
```

Рис. 2. Структура проекта

- file_storage: Используется для хранения входных файлов.
- mongodb-data и mongodb-log: Директории для хранения данных MongoDB и логов соответственно.
- tests: Содержит тесты для различных компонентов системы, включая базу данных, процесс дедупликации, производительность и сплиттер. Директория files содержит образцы файлов для тестирования.

Дополнительные файлы:

- docker-compose.yml: Используется для настройки и запуска Docker-контейнеров.
- mongo-init.js: Скрипт для инициализации MongoDB с предварительными настройками.
- requirements.txt: Содержит список зависимостей Python, необходимых для проекта.
- venv: Каталог для виртуального окружения Python, предназначенного для изоляции зависимостей проекта.

Результаты тестирования

В ходе тестирования выполнялась дедупликация и восстановление одного и того же файла с использованием разных значений входных параметров: изменялся алгоритм хеширования и варьировался размер сегмента.

Рис. 3. Время выполнения с использованием различных алгоритмов хеширования

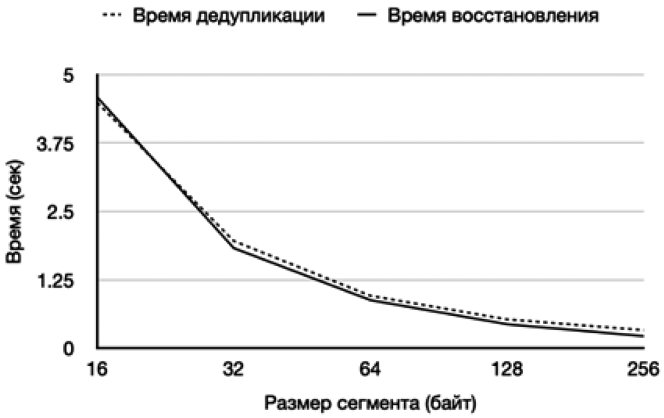

Рис. 4. Зависимость времени выполнения от размера сегмента

В процессе выполнения программы собирались следующие метрики:

- Затраченное время на дедупликацию (в секундах)
- Затраченное время на восстановление файла
- Соответствие восстановленного файла исходному (в процентах)
- Количество повторений сегментов
- Количество уникальных сегментов

По полученным результатам были построены зависимости, представим две из них. Зависимость времени выполнения программы от использованного алгоритма хеширования при размере сегмента 16 байт показывает эффективность алгоритма хеширования MD5 (рис. 3). Выбор MD5 как эффективного алгоритма хеширования обусловлен его скоростью, преимущественно в процессе восстановления файлов. Зависимость времени выполнения программы от размера сегмента при использовании алгоритма MD5 позволяет выбрать размер сегмента (рис. 4). Размер сегмента в 32 байта был определен как наилучший с точки зрения баланса между количеством дубликатов и временем выполнения программы.

Заключение

В ходе работы была успешно спроектирован и разработан модульный сервис, который не только обладает гибкостью в плане модификаций, но и демонстрирует высокую эффективность благодаря использованию Python, MongoDB и MongoEngine. Эти технологии были выбраны с учетом их сильных сторон, таких как гибкость схем данных и простота масштабирования, что обеспечило надежную основу для проекта.

Проведено тестирование сервиса и был определен наиболее эффективный алгоритм хеширования и размера сегмента данных. Были выбраны алгоритм хеширования MD5 и размер сегмента в 32 байта. В ходе тестирования не было выявлено ошибок после дедупликации и процесса обратного восстановлении исходного файла.

Эти результаты подтверждают, что предложенный подход к дедупликации обеспечивает не только общую производительность системы хранения данных, но и высокий уровень надежности и целостности данных. На основе полученных данных можно сделать вывод, что разработанный сервис подходит для предприятий, стремящихся минимизировать расходы на хранение больших объемов информации.

ЛИТЕРАТУРА

1. Shynu P.G., Nadesh R.K., Menon V.G., Venu P., Abbasi M., Khosravi M.R. A secure data deduplication system for integrated cloud-edge networks // Journal of Cloud Computing. — 2020. — vol. 9. — no. 1.
2. Saeed A.S. M., George L.E., Fingerprint-based data deduplication using a mathematical bounded linear hash function // Symmetry. — 2021. — vol. 13. — no. 11.
3. Sujatha G., Raj J.R., A Comprehensive Study of Different Types of Deduplication Technique in Various Dimension // International Journal of Advanced Computer Science and Applications. — 2022. — vol. 13. — no. 3.
4. Gadre A., Pund P., Ajmire G., Kale S., Signature Recognition Models: Performance Comparison // 2021 International Conference on Advancements in Electrical, Electronics, Communication, Computing and Automation. — 2021.
5. Devarajan V., Subramanian R., Analyzing semantic similarity amongst textual documents to suggest near duplicates // Indonesian Journal of Electrical Engineering and Computer Science. — 2022. — vol. 25. — no. 3.
6. Wangikar V., Deshmukh S., Bhirud S., Exploring Research Pathways in Record Deduplication and Record Linkage // Communications in Computer and Information Science. — 2021.
7. Maze E., Dirhoussi A., Vernaz Y., Farahbakhsh R., Duplicate Detection of Technical Documents in Petroleum Context Based on Near-Duplicate Image Search, in Proceedings // SPE Annual Technical Conference and Exhibition. — 2023.

8. Faibish S., Armangau P., Gonczi I., Bassov I., Improving Data Reduction for SDNAS for Trident Backends. — 2019.
9. Saab A., Nakhle V., G. al Hajj Chehade, Moussawi H., Testing and Comparing the Performances of Windows Server 2022 // Ubuntu 20.04 and CentOS 8 under DDoS Attacks. — 2021.
10. Arora R. and Vetrithangam D., Advancements in deduplication techniques for efficient data storage // Journal of Theoretical and Applied Information Technology. — 2024. — vol. 102. — no. 5.
11. Pawar K., Phatale V., Kumari R., Kanade A., Ujgare S., Data De-Duplication Engine for Efficient Storage Management. — 2022.
12. Naghmouchi M., Laurent M., Levallois-Barth C., and Kaaniche N., Comparative Analysis of Technical and Legal Frameworks of Various National Digial Identity Solutions. — 2023.
13. Somasundar A., Chilakarao M., B.R.K. Raju, Behera S.K., Ramana C.V., Sethy P.K., MongoDB integration with Python and Node. js, Express. Js // 2024 Fourth International Conference on Advances in Electrical, Computing, Communication and Sustainable Technologies. — 2024.

© Пантюхин Андрей Максимович (panandafog@gmail.com); Гладун Владимир Вадимович (vladimir.gldn@gmail.com);
Малинин Илья Игоревич (malinin.ilja@gmail.com); Молодяков Сергей Александрович (molodyakov_sa@spbstu.ru)
Журнал «Современная наука: актуальные проблемы теории и практики»