Департамент образования и науки города Москвы

Государственное автономное образовательное учреждение высшего образования города Москвы

«Московский городской педагогический университет»

Институт цифрового образования

Департамент информатики, управления и технологий

ДИСЦИПЛИНА:

Инструменты для хранения и обработки больших данных

**Лабораторная работа № 4.1**

**Тема:**

**«**Сравнение подходов хранения больших данных**»**

Выполнил(а):

Вереина М.С., АДЭУ-211

Преподаватель: Босенко Т.М.

Москва

2024

**Цель работы:** сравнить производительность и эффективность различных подходов к хранению и обработке больших данных на примере реляционной базы данных PostgreSQL и документоориентированной базы данных MongoDB.

Оборудование и программное обеспечение

- Компьютер с операционной системой Ubuntu.

- PostgreSQL.

- MongoDB.

- Python 3.x.

- Библиотеки: psycopg2, pymongo, pandas, matplotlib.

**Теоретическая часть**

В современном мире объемы данных растут экспоненциально, что приводит к необходимости использования эффективных методов их хранения и обработки. Существует два основных подхода к хранению больших данных:

1. Реляционные базы данных (например, PostgreSQL)

2. NoSQL базы данных (например, MongoDB)

Каждый из этих подходов имеет свои преимущества и недостатки, которые мы рассмотрим в ходе выполнения лабораторной работы.

**Практическая часть**

**Вариант 3**

Сформировать данные или использовать источники данных не менее 1000 записей для обычных данных и 100000 для больших данных.

**Задание:** сравнить потребление памяти при работе с большими объемами данных логистической компании (информация о доставках, маршрутах, складах).

**# 3. Данные логистической компании**

**Шаг 1. Генерация больших данных**

1. Устанавливаем библиотеку Faker

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

Рисунок 1 – Установка библиотеки для генерации

1. Создаем функцию для генерации таких данных, как идентификатор доставки, склад, вес груза, статус доставки, предполагаемая поставка, маршрут доставки.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 2 – Создание функции и определение данных для генерации

1. Генерируем 1000 и 100 000 данных в соответствии с условиями задачи. Далее сохраняем их в файлы формата JSON и csv.

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

Рисунок 3 – Генерация данных и сохранение их в файлы

1. Убеждаемся в корректности сгенерированных данных

Изображение выглядит как текст, снимок экрана, число, Шрифт

Автоматически созданное описание

Рисунок 4 – Просмотр сгенерированных данных

1. Подключаемся к mongodb и проверяем подключение, а также указываем базу данных и коллекции, в которые в дальнейшем будут храниться сгенерированные данные.

Изображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение

Автоматически созданное описание

Рисунок 5 – Проверка подключения, создание базы данных, коллекций

1. Далее загружаем файлы со сгенерированными данными в mongodb

Изображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение

Автоматически созданное описание

Рисунок 6 – Загрузка данных в mongodb

1. Загруженные данные успешно отображаются в MongoExpress

Изображение выглядит как текст, снимок экрана, программное обеспечение, число

Автоматически созданное описание

Рисунок 7 – Наличие документов в коллекциях

1. Далее собираем статистику по потреблению памяти. Для начала определим и проанализируем размеры коллекций и использование ими хранилища.

Изображение выглядит как текст, снимок экрана, дизайн

Автоматически созданное описание

Рисунок 8 – Сбор статистики по потреблению памяти

1. Далее получим информацию о состоянии сервера с помощью команды serverStatus для получения информации о текущем и максимальном использовании памяти в кэше WiredTiger (встраиваемая СУБД класса «ключ — значение», используемая в качестве подсистемы хранения по умолчанию в MongoDB).

Изображение выглядит как текст, снимок экрана, диаграмма, дизайн

Автоматически созданное описание

Рисунок 9 – Отношение занимаемой коллекцией памяти

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Количество данных | Время на загрузку данных в коллекцию | Размер коллекции | Использование хранилища | Используемая память, % |
| 1000 записей | 468 мс | 331,32 КБ | 168 КБ | 3,4% |
| 100 000 записей | 5.83 с | 32,59 МБ | 16,05 МБ |

**Интерпретация данных. Выводы.**

Коллекция с 100 000 документами значительно больше по размеру (32,59 МБ) по сравнению с коллекцией из 1000 документов (331,32 КБ). Это ожидаемо и указывает на линейное увеличение размера при добавлении больших объемов данных.

Несмотря на то, что вторая коллекция значительно больше по размеру, использование хранилища не увеличивается пропорционально. Это может указывать на то, что MongoDB эффективно управляет памятью и хранит данные более компактно.

В обеих коллекциях наблюдается разница между размером и использованием хранилища, что может свидетельствовать о наличии свободного пространства или фрагментации. Важно следить за этими показателями для оптимизации производительности базы данных.

**Шаг 2. Сравнение потребления памяти при работе с большими объемами данных логистической компании (информация о доставках, маршрутах, складах).**

**Мониторинг использования памяти**

Для мониторинга использования памяти серверами во время выполнения операций используются утилиты командной строки, такие как htop или top на Linux. Эти инструменты позволят вам видеть использование ресурсов в реальном времени.

Основные параметры данной утилиты, которые следует учитывать:

1. Использование памяти

%MEM: процент использования оперативной памяти каждым процессом.

RES: количество резидентной памяти, используемой процессом.

VIRT: Общий объем виртуальной памяти, используемой процессом.

SHR: Объем разделяемой памяти, который может быть использован другими процессами.

2. Использование CPU

%CPU: Процент использования процессорного времени каждым процессом.

3. Время работы процессов

TIME+: Общее время работы процесса.

4. Состояние процессов

S (Status): Статус процесса (например, R - выполняется, S - спит).

5. Средняя нагрузка (Load Average)

Load Average: Отображает среднее количество процессов, ожидающих выполнения за последние 1, 5 и 15 минут. Высокие значения могут указывать на перегрузку системы.

**Проведение эксперимента**

Генерация 1000 строк:

Изображение выглядит как текст, снимок экрана, Шрифт, число

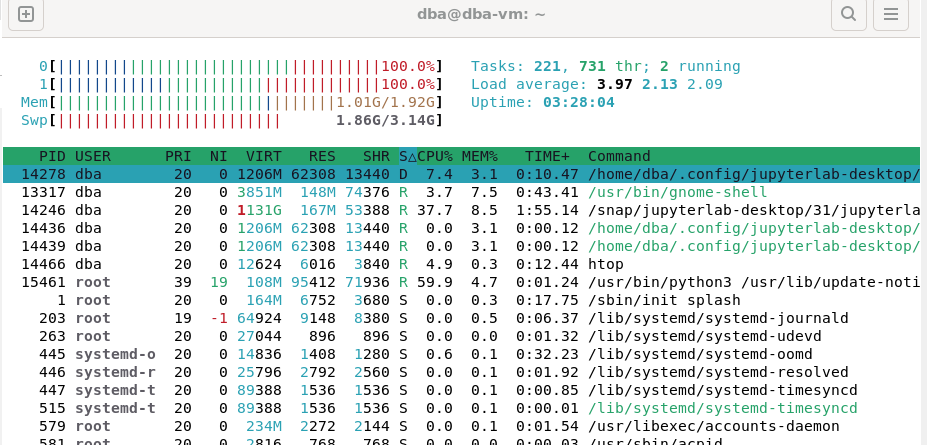
Автоматически созданное описание

Генерация 100 000 строк:

Изображение выглядит как текст, Шрифт, число, линия

Автоматически созданное описание

Загрузка 1000 записей в MongpDB из файла.



Загрузка 100 000 строк в MongoDB из файла.

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Получение 1000 документов из MongoDB

Изображение выглядит как текст, Шрифт, число, документ

Автоматически созданное описание

Получение 100 000 документов из MongoDB.

Изображение выглядит как текст, Шрифт, число, документ

Автоматически созданное описание

Обновление одного из 1000 документов

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Обновление одного из 100 000 документов.

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Нахождение документов из 1000 по статусу

Изображение выглядит как текст, снимок экрана, число, Шрифт

Автоматически созданное описание

Нахождение документов по статусу из 100 000.

Изображение выглядит как текст, снимок экрана, число, Шрифт

Автоматически созданное описание

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Действия/Показатели | MEM% | RES | VIRT | SHR | CPU% | TIME+ |
| Генерация 1000 строк | 6.0 | 118M | 976M | 34048 | 108.0 | 0:11 |
| Генерация 100 000 строк | 7.5 | 147M | 1009M | 60404 | 129.0 | 1:34 |
| Загрузка 1000 строк в MongoDB из файла | 3.1 | 62308 | 1206M | 13440 | 7.4 | 0:10 |
| Загрузка 100 000 строк в MongoDB из файла | 12.2 | 239M | 1596M | 10112 | 67.1 | 3:55 |
| Получение 1000 документов из MongoDB | 9.7 | 190M | 12780 | 60772 | 12.2 | 0:13 |
| Получение 100 000 документов из MongoDB | 14.9 | 292M | 1654M | 9856 | 52.2 | 3:59 |
| Обновление одного из 1000 документов | 1.8 | 438M | 1554M | 33548 | 0.7 | 0:06 |
| Обновление одного из 100 000 документов | 8.0 | 156M | 1660M | 5376 | 2.8 | 1:01 |
| Нахождение документов из 1000 по статусу | 2.6 | 4912 | 307M | 2432 | 3.4 | 0:11 |
| Нахождение документов по статусу из 100 000 | 18.3 | 360M | 1131G | 53024 | 7.6 | 1:06 |

**Интерпретация таблиц. Выводы**

**Потребление памяти**

Увеличение объема данных приводит к росту потребления памяти, что ожидаемо, так как больше данных требует больше ресурсов для обработки.

Значительное увеличение потребления памяти при загрузке больших объемов данных указывает на необходимость оптимизации процесса загрузки.

Рост потребления памяти также наблюдается при увеличении числа запрашиваемых документов.

Обновление большого количества документов требует больше ресурсов.

Поиск по большому количеству документов значительно увеличивает нагрузку на систему.

**Использование процессора**

Использование CPU превышает 100%, что указывает на использование нескольких ядер процессора для обработки данных.

Значительное увеличение использования CPU при загрузке больших объемов данных.

Увеличение числа запрашиваемых документов также приводит к росту использования процессора.

Обновление и поиск также показывают рост использования CPU с увеличением объема обрабатываемых данных, что подтверждает зависимость между объемом данных и нагрузкой на процессор.

**Время выполнения**

Время выполнения операций значительно увеличивается с ростом объема данных, что отражает более длительные вычисления и операции ввода-вывода.

Масштабируемость и гибкость

MongoDB демонстрирует свою способность масштабироваться и обрабатывать большие объемы данных, что делает ее подходящей для приложений, работающих с полуструктурированными данными. Возможность хранения различных типов данных в одном документе и отсутствие строгой схемы позволяют легко адаптироваться к изменениям в структуре данных.

**Заключение**

MongoDB хорошо подходит для приложений, требующих высокой доступности и масштабируемости, особенно когда речь идет о больших объемах данных. Однако важно учитывать потребление ресурсов и время выполнения операций при проектировании системы. Для достижения оптимальной производительности необходимо правильно настраивать индексы и обеспечивать достаточное количество оперативной памяти для работы с рабочими наборами данных.

PostgreSQL

Загрузка данных из датафрейма на 1000 строк

Изображение выглядит как текст, число, снимок экрана, Шрифт

Автоматически созданное описаниеИзображение выглядит как текст, снимок экрана, число, Шрифт

Автоматически созданное описание

Выполнение запроса по таблице, которая содержит 1000 строк

Изображение выглядит как текст, число, программное обеспечение, снимок экрана

Автоматически созданное описание

Загрузка в Postgresql 100 000 строк из датасета.

Изображение выглядит как текст, число, программное обеспечение, Шрифт

Автоматически созданное описаниеИзображение выглядит как текст, снимок экрана, программное обеспечение, число

Автоматически созданное описание

Данная операция по времени заняла ~ 7 минут.

Выполнение запроса к таблице, которая содержит 100 000 строк.

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Получение 1000 документов в PostgreSQL

Изображение выглядит как текст, снимок экрана, программное обеспечение, число

Автоматически созданное описание

Получение 100 000 документов в PostgreSQL

Изображение выглядит как текст, снимок экрана, программное обеспечение, число

Автоматически созданное описание

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Действия/Показатели | MEM% | RES | VIRT | SHR | CPU% | TIME+ |
| Загрузка 1000 строк в PostgreSQL из файла | 8.9 | 175M | 1155G | 71288 | 13.1 | 0:32 |
| Загрузка 100 000 строк в PostgreSQL из файла | 8.7 | 170M | 1155G | 71800 | 36.6 | 7:37 |
| Получение 1000 документов из PostgreSQL | 10.3 | 202M | 1155G | 53580 | 19.5 | 1:08 |
| Получение 100 000 документов из PostgreSQL | 4.0 | 254M | 81500 | 4352 | 21.2 | 2:28 |
| Вычисление среднего по статусу из 1000 строк | 8.8 | 174M | 1155G | 71800 | 20.9 | 1:47 |
| Вычисление среднего по статусу из 100 000 строк | 8.5 | 166M | 1155G | 66808 | 32.1 | 2:32 |

**Интерпретация таблиц. Выводы**

**Загрузка данных в PostgreSQL**

Потребление памяти (MEM%) остается на схожем уровне, несмотря на увеличение объема загружаемых данных.

Использование CPU значительно увеличивается с увеличением объема данных, что указывает на более высокую нагрузку на процессор при загрузке больших объемов.

Время выполнения (TIME+) значительно увеличивается с 32 секунд до 7 минут и 37 секунд, что подтверждает, что загрузка большего объема данных требует значительно больше времени.

**Получение документов из PostgreSQL**

Потребление памяти для получения документов из базы данных показывает различия, где для получения меньшего объема (1000) оно выше (10.3%), чем для большего (4.0%). Это может указывать на то, что система использует кэширование для более крупных запросов.

Использование CPU увеличивается, но не так значительно, как при загрузке данных.

Время выполнения также увеличивается, но не так сильно, как при загрузке.

**Вычисление среднего по статусу**

Потребление памяти остается стабильным между двумя объемами данных.

Использование CPU увеличивается при вычислении среднего по большему количеству строк, что ожидаемо.

Время выполнения также увеличивается, но не так значительно.

**Общие выводы по сравнению показателей**

Увеличение объема обрабатываемых данных (сравнение между записями в количестве от 1000 до 100000) приводит к значительному росту времени выполнения операций, особенно при загрузке данных в базу.

Использование CPU значительно возрастает при работе с большими объемами данных, что указывает на необходимость оптимизации запросов и настройки системы для обработки больших наборов данных.

Потребление памяти остается относительно стабильным между двумя объемами данных, что может свидетельствовать о том, что PostgreSQL эффективно использует кэширование для обработки запросов.

Для оптимизации работы с большими объемами данных рекомендуется рассмотреть возможность использования индексов и настройки параметров конфигурации PostgreSQL для улучшения производительности.

**Общие выводы по сравнению PostgreSQL и MongoDB**

Скорость загрузки и получения данных:

MongoDB превосходит PostgreSQL в скорости загрузки и получения небольших объемов данных, что делает его подходящим для приложений с высокой частотой вставок и чтений небольших наборов данных.

PostgreSQL показывает более стабильную производительность при работе с большими объемами данных, особенно в сценариях, требующих сложных вычислений или транзакций.

Использование ресурсов:

PostgreSQL имеет более низкое использование CPU при выполнении сложных операций и вычислений по сравнению с MongoDB, что может быть важным фактором для приложений, требующих высокой производительности и низкой латентности.

Аналитические возможности:

PostgreSQL лучше подходит для аналитических задач благодаря своей поддержке сложных SQL-запросов и ACID-транзакций, что делает его предпочтительным выбором для приложений, требующих строгой целостности данных.

Гибкость схемы данных:

MongoDB предлагает большую гибкость в работе с неструктурированными данными благодаря своей документной модели хранения, что может быть полезно для динамически изменяющихся приложений.

Выбор СУБД в зависимости от задачи:

Выбор между PostgreSQL и MongoDB должен основываться на конкретных требованиях приложения: если требуется высокая скорость обработки небольших объемов данных и гибкость схемы — стоит выбрать MongoDB; если необходима высокая производительность при сложных запросах и транзакциях — стоит выбрать PostgreSQL.