Санкт-Петербургский государственный университет

Математическое обеспечение и администрирование информационных систем

Информационно-аналитические системы

Говоруха Оксана Сергеевна

Анализ СУБД в качестве основы для построения системы управления метаданными

Выпускная квалификационная работа

Научный руководитель:

д. ф.-м. н., профессор Нестеров В. М.

Санкт-Петербург

2018

SAINT-PETERSBURG STATE UNIVERSITY

Software and Administration of Information Systems

Analytical Information Systems

Govorukha Oxana

Performance Analysis of DBMS as foundation of metadata management system

Graduation Project

Scientific supervisor:

Ph.D., Professor Vyacheslav Nesterov

Saint Petersburg

2018

Содержание

Введение 2

1. Цель работы 4

2. Задачи 5

3. Обзор 6

   3.1 Обзор СУБД 6

   3.2 Наборы данных для тестирования 12

   3.3 Обзор инструментов и метрик тестирования 13

4. Основная часть 16

   4.1 Определение схемы хранения данных 16

   4.2 Описание данных и особенности СУБД 19

   4.3 Методика тестирования 21

   4.4 Тестирование операционной задержки 22

   4.5 Тестирование пропускной способности 24

   4.6 Измерение использования дискового пространства 26

   4.7 Анализ системной нагрузки 27

5. Заключение 28

   5.1. Результаты 28

   5.2. Итоги 28

Литература 31

Приложение 33

   1. Скрипты для загрузки и запросов 33

   2. Образцы наборов данных 35

Введение

При растущем объеме информации мы сталкиваемся с проблемой рационального управления данными. Оптимальная система поиска и хранения данных помогает не только быстрее находить объекты с нужными атрибутами, но и анализировать данные. Реализация такой системы невозможна без явно представленных описаний свойств данных - метаданных. Метаданные хранят основную информацию об объекте, обеспечивают сохранность данных и защищают от несанкционированного доступа. Они существенно повышают ценность данных и обеспечивают более широкие возможности их использования. Содержание метаданных, их структура, функции и средства их представления зависят от описываемых ресурсов, предметной области использующих их систем, контекста и характера их использования, а также от многих других факторов.

Метаданные могут храниться вместе с объектом, что гарантирует, что они не будут потеряны и при изменении будут обновлены вместе. Однако не всегда возможно встроить метаданные в некоторые типы объектов, кроме того хранение метаданных отдельно может улучшить и упростить поиск этих объектов. Поэтому метаданные часто хранятся в базе данных и «связывают» с описываемыми объектами[3]. Использование различных СУБД может требовать разного способа представления данных, а также влиять на легкость управления данными. Поэтому выбор системы для хранения метаданных является важным решением.

Для выбора такой системы важно рассмотреть СУБД с разными моделями хранения данных, так как структура метаданных может быть различна, необходимо оценить эффективность той или иной модели представления данных и сравнить возможные схемы хранения. Конечно, важное значение имеют стандарты для представления метаданных, которые определяют схему хранения и позволяют легче переиспользовать данные в дальнейшем и управлять ими. Однако, часто встречаются метаданные с

нефиксированным набором атрибутов или имеющие сильно иерархическую структуру, что затрудняет определение схемы. Это нужно учитывать при выборе хранилища данных. Поэтому требуется оценить сложность представления метаданных с различной структурой в разных базах данных.

С увеличением количества данных, возрастает сложность управления этими данными, растет время записи, чтения или просмотра метаданных. Поэтому необходимо протестировать производительности СУБД для определения этих показателей и проанализировать системы для выявления оптимального решения в качестве хранилища метаданных.

В данной работе будем тестировать на различных по структуре наборах метаданных. Для анализа выберем наиболее распространенные системы управления базами данных, имеющие разные модели хранения данных. Сравним из относительно ключевых характеристик в контексте цели данной работы и измерим их производительность при работе с метаданными на разных запросах и выполняя операции записи и чтения.

1. Цель работы

В данной работе была поставлена следующая цель:

- Сравнить СУБД с разными моделями хранения данных на примере разных наборов метаданных, оптимизируя хранение выбранных наборов в БД с учетом преимуществ и особенностей СУБД, определить наиболее оптимальное решение в качестве хранилища для системы управления метаданными

2. Задачи

В связи с поставленной целью были выделены следующие задачи:

- Рассмотреть основные характеристики систем управления базами данных
- Найти и подготовить наборы данных для тестирования
- Определить метрики тестирования, методологию и инструменты
- Определить схемы хранения наборов данных
- Подготовить скрипты для загрузки данных, выполнения запросов, чтения и записи данных
- Произвести замеры на различных наборах
- Сравнить результаты, полученные при тестировании разных систем на различных наборах

3. Обзор

3.1. Обзор СУБД

В данной работе будем исследовать следующие СУБД: ArangoDB, MongoDB, Apache Cassandra и PostgreSQL. Рассмотрим их основные характеристики и ключевые отличия относительно требуемой задачи.

ArangoDB - мульти-модельная СУБД, поддерживает различные модели хранения данных в виде графа, документов или как ключ/значение[11]. Документы передаются в формате JSON[1]. Все документы группируются в коллекции, внутри коллекции документы могут иметь различные атрибуты. Существуют два типа коллекций: коллекция документов (вершин), которая имеет обязательные атрибуты id и key, и коллекция ребер, имеющая атрибуты from и to, для создания связей между документами в базе. Возможно выполнение сложных запросов и управление данными с помощью AQL (Arango Query Language). Существуют специальные драйверы, позволяющие использовать базу данных через различные среды и языки программирования. Для ускорения поиска поддерживаются индексы. С v1.3 возможно определение транзакций пользователем. Для всех транзакций в ArangoDB выполняются ACID[3] свойства, кроме операций над несколькими документами или коллекциями при распределенном хранении данных на кластере.

Архитектура кластера ArangoDB представляет собой master/master без единой точки отказа. Для различных случаев выделяются следующие роли: агенты, координаторы, основной и вторичный сервера. Каждому элементу кроме агентов выделяется id, по которому будет происходить взаимодействие внутри кластера. Конструкция кластера Arango позволяет минимизировать

1. JSON (JavaScript Object Notation) – текстовый формат обмена данными, имеющий структуру *ключ: значение* или упорядоченный список *значений* [16].
2. ACID (Atomicity, Consistency, Isolation, Durability) – свойства атомарности, согласованности, независимости и устойчивости, обеспечивающие для СУБД наиболее надежную и предсказуемую работу.

администрирование и должна обеспечивать самостоятельное восстановление в случае временных сбоев.

MongoDB - документно-ориентированная NoSQL база данных[14]. Все записи хранятся как BSON-объекты[3]. Документы могут иметь различную структуру в пределах одной коллекции, с v3.2 можно определять правила валидации для коллекции во время обновления или вставки. В базе определены некоторые ограничения для хранимых документов: порядок полей, размер документа, наличие id. Существует множество вариантов внешнего REST[4] интерфейса для MongoDB и различные драйверы для простой работы с базой данных из разных сред и с разными языками программирования, поддерживающие все основные операции. В MongoDB операция записи атомарна только на уровне одного документа. Для того, чтобы изолировать операцию записи, которая затрагивает несколько документов, можно использовать дополнительный изоляционный оператор. База данных позволяет чтение незафиксированных записей. Для ускорения поиска также возможна индексация полей.

MongoDB поддерживает репликацию для увеличения доступности, и шардинг для распределенного хранения данных. Репликация может быть представлена набором реплик или master/slave. В наборе реплик поддерживается автоматическая отказоустойчивость в случае прекращения работы основного сервера. Кластер в MongoDB состоит из сервера маршрутизации, шардов и сервера конфигураций. Возможно одновременное хранение разделенных и неразделенных данных. Дополнительно в MongoDB предоставляются функции для аутентификации, управления доступом и

---

3. BSON (Binary JavaScript Object Notation) - формат электронного обмена цифровыми данными, бинарная форма представления простых структур данных и ассоциативных массивов (объектов), подмножество JSON, включающее дополнительно регулярные выражения, двоичные данные и даты [17].
4. REST (Representational State Transfer) - архитектурный стиль взаимодействия компонентов распределённого приложения в сети, представляет собой согласованный набор ограничений, учитываемых при проектировании распределённой системы.

шифрования.

Cassandra - распределённая система управления базами данных, относится к NoSQL и предназначена для обработки и хранения больших объемов данных, обеспечивая высокую доступность без единой точки отказа[12]. Представляет гибридную модель ключ/значение и колоночно-ориентированной модели. Каждому ключу в базе соответствует определенный объект - набор столбцов. Столбцы объединяются в группы - семейство столбцов (таблицы). Ключ может идентифицировать строку из переменного числа элементов. Колонка в базе данных состоит из трех частей: имени, метки времени и значения. Возможно выполнять запросы для определения и изменения данных при помощи CQL (Cassandra Query Language). Не поддерживается выполнение join или подзапросов. Конкретное значение, хранимое в базе, идентифицируется однозначно следующими атрибутам: пространством ключей (оно позволяет на одном кластере размещать данные разных приложений), колоночным семейством (осуществляет привязку к запросу), ключом (-привязка к узлу кластера), именем колонки (-привязка к атрибуту в записи, позволяет в одной записи хранить несколько значений). Запись в базу данных Cassandra осуществляется быстрее, чем чтение. Поддерживается транзакционность и выполнение свойств ACID на уровне одной записи - набора колонок с одним ключом.

Узлы кластера - равноценны. Для координации между узлами и отправления им запросов используется координатор. При распределенном хранении возможно задать стратегию выбора узла кластера в зависимости от значения ключа. Можно регулировать временные задержки операций чтения, записи и настраивать согласованность и доступность каждого вида операций.

PostgreSQL - реляционная система управления базами данных[13]. Любая таблица представляет собой именованный набор строк. Все строки таблицы имеют одинаковый набор именованных столбцов, при этом каждому столбцу назначается определённый тип данных. Таблицы объединяются в

базы данных, а набор баз данных, управляемый одним экземпляром сервера PostgreSQL, образует кластер баз данных. Для выполнения запросов используется язык SQL, позволяющий выполнять сложные вложенные запросы и использовать join. Для ускорения поиска возможна индексация по одному столбцу или набору. Поддерживаются транзакции на уровне базы данных. Существуют функции для хранения и работы с файлами и JSON данными. Для шардинга PosgreSQL существует несколько решений: Postgres-XC, PL/Proxy, HadoopDB, Greenplum Database.

Рассмотрев общие характеристики выбранных СУБД, выделим ключевые для системы хранения метаданных и управления ими и сравним СУБД относительно них.

Так как метаданные могут иметь сложную иерархическую структуру или быть неструктурированными, для СУБД важно иметь возможность хранения такого вида данных, например, с помощью поддержки хранения данных типа JSON или возможности изменения схемы в любой момент работы с базой данных (гибкость схемы). При этом, даже при сложной структуре метаданных необходимо исключать вероятность добавления некорректных данных, на уровне СУБД этого можно добиться с помощью поддержки валидации данных.

Для управления данными, ускорения поиска по определенным атрибутам, сортировки элементов и анализа полезна индексация, поддерживается ли она СУБД и то, каким способом она осуществляется, сказывается на эффективности управления данными в базе. Метаданные или некоторые атрибуты в них, могут быть совсем не типизированы и представлять собой текстовый тип, однако по таким данным может быть необходимо тоже выполнять поиск, например, для выделения данных содержащих определенные слова или атрибуты, в таких случаях будет полезна поддержка полнотекстового поиска. Также, немаловажен язык запросов, поддерживаемый СУБД, он должен быть достаточно гибкий для

выполнения сложных запросов на уровне базы данных, что может обеспечить более быстрый и эффективный поиск.

При увеличении количества данных и масштабируемости системы важную роль играют скорость чтения и записи данных и поддержка их согласованности. Обеспечение полной согласованности данных и их долговечности может влиять на производительность, однако это поддерживает корректность данных в любой момент времени и исключает их потерю в случаи сбоя системы.

Сравнение по ключевым характеристикам представлено в Табл.1

Таблица сравнения СУБД

	ArangoDB	**MongoDB**	**Cassandra**	**PostgreSQL**
модель хранения данных	Мульти - модельная СУБД	Документо - ориентированная СУБД	Модель ключ - значение и колоночно – ориентированная	Реляционная СУБД
хранение неструктурированных данных	Возможно различное количество атрибутов и разные типы значений в документах в одной коллекции.	Возможно различное количество атрибутов и разные типы значений в документах в одной коллекции.	Семейство столбцов определяется заранее. Для каждого ключа – разное число колонок.	Схемы таблиц определяются заранее. Поддержка JSON.
валидация атрибутов	Нет	Возможно типизировать значения атрибутов.	Типы атрибутов описываются при определении схемы и не изменяются.	Типы полей указываются при создании таблицы и не изменяются.
индексация	Индексация по одному или нескольким атрибутам. Автоматическая индекация id и key или from и to	Индексация по одному или нескольким атрибутам. Автоматическая индексаця Id.	Обязательно Primary Key. Возможен вторичный индекс.	Возможна индексация по одному или нескольким полям. При хранении данных в формате JSON также возможна индексация
операции чтение/запись	Отложенная запись только для документов, касающихся одной коллекции (увеличивает пропускную способность).	Асинхронная запись (увеличивает пропускную способность, но возможно незафиксированное чтение).	Операция записи работает быстрее, чем чтение. (На времени записи и чтения сказывается устанавливаемый уровень согласованности данных).	Время выполнения операций чтения и записи может увеличиваться в зависимости от типа индекса и количества данных

поиск	Возможны сложные вложенные запросы и аналог join с помощью AQL.	Возможен поиск по шаблону и фильтрация.	Только несложные запросы с помощью CQL.	Возможны сложные вложенные запросы и join с помощью SQL
обеспечение согласованности данных	Возможны пользовательские транзакции. Выполнение ACID. (кроме операций над распределенными данными на кластере).	Запись атомарна только на уровне документа. Можно изолировать несколько операций (но при ошибки не будет выполнен откат всех изменений). Возможно чтение незафиксированных записей.	Атомарность действует на уровне одной строки. Возможно определять уровень согласованности данных.	Возможно выполнение транзакций на уровне всей базы данных. Для всех транзакций выполняются ACID.

Табл.1

3.2. Наборы данных для тестирования

Для тестирования СУБД были подобраны два набора метаданных из разных источников. Датасеты представляют собой структурированные наборы данных, чтоб бы можно было протестировать СУБД не только на запись и чтение данных, но и более сложных запросах. Полная структура экземпляров двух наборов “amazon-meta” и “All_Viruses.gene_info” приводится в Приложении[2] (рис.1, рис.2).

Таблица описание наборов данных

наим.	All_Viruses.gene_info	amazon-meta
краткое описание	часть набора gene_info из базы данных геномов	часть набора данных amazon-meta.txt, содержит собранную информацию о продуктах с amazon
количество записей	322.547	300.000
общий размер файла	147 Мб	971 Мб
атрибуты	**1. tax_id** **2. GeneID** **3. Symbol** **4. Locus Tag** **5. Synonyms** **6. dbXrefs** **7. chromosome** **8. map_location** **9. description** **10. type_of_gene** **11. Symbol_from_nomenclature_authority** **12. Full_name_from_nomenclature_authority Nomenclature_status** **13. Other_designations** **14. Modification_date**	**1. Id** **2. ASIN** **3. title** **4. group** **5. salesrank** **6. similar** **6.1. total** **6.2. asin** **7. categories** **7.1. total** **7.2. categories** **8. reviews** **8.1. total** **8.2. downloaded** **8.3. avg_rating** **8.4. reviews** **8.4.1. data** **8.4.2. customer** **8.4.3. rating** **8.4.4. votes** **8.4.5. helpful**
источник	ftp://ftp.ncbi.nih.gov/gene/DATA/GENE_INFO/Viruses/All_Viruses.gene_info.gz	https://snap.stanford.edu/data/bigdata/amazon/amazon-meta.txt.gz

Табл.2

3.3. Обзор инструментов и метрик тестирования

Сравнение производительностей баз данных довольно распространенная задача. Реализовано большое количество тестов, многие из которых находятся в открытом доступе и при желании их можно воссоздать на собственной машине. Часто тесты, например, как в статьях [1,15], создаются под определенную нагрузку, и имеют задачу продемонстрировать лучшие характеристики определенной базы данных, на специально подобранных наборах данных и запросах, которые в большей мере отразят желаемые показатели. Такие тесты раскрывают возможности выбранной СУБД, а запросы к другим сравниваемым базам данных обычно редко оптимизируют исходя из их особенностей, поэтому возможен не совсем объективный результат. В некоторых работах [6,8] наоборот сравнивают более общие показатели СУБД, касающиеся модели хранения, поддержки согласованности, и других характеристик, не связанных с хранимыми данными. Выбор СУБД для сравнения определяется задачами, которые ставятся в статье, или темой, которую исследуют авторы. Например, как в статье [2] тестируются две мульти-модельные СУБД и две другие СУБД, имеющие разные модели хранения, которые объединяют для хранения одного специально синтезированного набора. Авторы сравнивают производительность выбранных средств хранения, когда требуется использование двух NoSQL-схем в одном приложении. В зависимости от сформулированных запросов СУБД с наилучшей производительностью меняется. В данной работе нас больше интересует сравнение производительности и анализ СУБД для организации хранения метаданных, мы рассматриваем базы данных с различными моделями хранения на двух наборах данных разных по структуре.

Для некоторых замеров, например, определения операционной задержки определенного запроса, можно воспользоваться возможностями самой СУБД, для определения пропускной способности и создания большей

нагрузки на систему можно разработать собственные тесты, однако существуют и готовые довольно гибкие решения для воссоздания разных типов нагрузки на базы данных, например, для тестирования NoSQL баз данных распространен инструмент YCSB (Yahoo! Cloud Serving Benchmark) (например, [4,7]). Он имеет несколько профилей нагрузки, различающихся соотношением количества выполняемых операций на запись и чтение или чтения и обновления, при которых и прогоняет базы данных на наборе тестов. Тестирование с помощью данной утилиты отражает общие характеристики: пропускную способность и время ожидания работы баз данных. Основной целью является разработка рабочих нагрузок для хранилищ «ключ-значение» или облачных хранилищ и сравнение производительности в созданных условиях.

Важное значение при тестировании имеют характеристики окружения, то, где будет располагаться сервер базы данных, храниться данные и запускаться тесты при желании масштабирования системы и создания кластера, но отсутствии требуемого оборудования, можно воспользоваться возможностями облачных сервисов. Так, довольно часто для тестирования больших нагрузок разворачивают кластер базы данных на серверах от Amazon (Amazon Web Services), которые предоставляют дополнительные вычислительные возможности для хранения и анализа производительности баз данных, такие как EC2 (the Elastic Compute Cloud) и EBS (Elastic Block Storage) (например, [9,10]). С помощью собственных скриптов и дополнительных инструментов сервис позволяет тестировать базы данных. Использовать для замеров производительности можно как собственную тестовую систему, так и, например, утилиту YCSB (как в статье [10]).

В данной работе на начальных этапах - определения схемы хранения и пробных замерах времени при исполнении запросов и операций загрузки данных и вставки, базы данных тестируются без использования внешних ресурсов. На последующих этапах при загрузке больших объемов данных и тестировании предполагается развернуть базы данных на кластере. Для

тестирования необходимо выбрать подходящий инструмент. Например, в статье [5] рассматривается разработка инструмента тестирования, а также возможные сценарии. На первом этапе замеры, осуществляются возможностями СУБД.

Для составления верной оценки и выбора СУБД, которая будет больше соответствовать нашей задаче, важно правильно определить метрики, которые будем измерять и их условия. В качестве ключевых метрик сравнения баз данных могут быть рассмотрены разные показатели, наиболее подробно возможные метрики описываются в статье [9].

Исходя из цели данной работы, для тестирования были выбраны следующие метрики:

Операционная задержка (operational latency) - данная метрика будет измерять задержку выполнения запроса. Параллельно будет запущенно несколько потоков, которые непрерывно будут отправлять системе запросы и будем замерять разницу в секундах между отправлением запроса и получением результата от системы.

Пропускная способность (throughput) - измерение количества выполненных операций(запросов) за единицу времени.

Использование дискового пространства (disk space usage) - измеряем размер, занятый данными, метаданными и другими вспомогательными структурами системы базы данных.

Системная нагрузка (system load) - анализируем использования CPU, памяти и диска на протяжении эксперимента. Собираются данные о нагрузке каждые 5 секунд во время тестирования системы.

4. Основная часть

4.1. Определение схемы хранения данных

Поскольку для системы хранения метаданных может быть необходима поддержка хранения нетипизированных данных или гибкость схемы, преимуществом является то, что в NoSQL базах данных, таких как ArangoDB и MongoDB, не требуется определять схему и можно добавлять документы в формате JSON. Датасеты можно сразу загрузить в базы данных в соответствующие коллекции.

Можно сравнить насколько эффективен такой способ хранения для управления. Для этого рассмотрим небольшую часть данных каждого набора, и загрузим их в базы данных таким образом, что каждый набор будет храниться в одной коллекции. От каждого набора взяли для эксперимента по 5.000 записей и загрузили их в отдельные коллекции в базы данных ArangoDB и MongoDB (как в Приложение[1]).

Первый набор имеет достаточно простую структуру и данный способ хранения не скажется на эффективности управления данными. Рассмотрим более подробно часть второго набора “amazon-meta”, на примере выполнения запросов проанализируем усложняет ли данная схема хранения поиск или анализ данных. Исходя из структуры документа (Приложение[2], рис.1) были выбраны следующие запросы:

1. Запрос, возвращающий продукты, для которых были написаны отзывы конкретного числа (Приложение[1.7.1]).

2. Запрос, возвращающий несколько первых продуктов с максимальным числом отзывов (Приложение [1.7.2]).

3. Запрос, возвращающий оценку нашего товара и схожих с ним (Приложение [1.7.3]).

4. Запрос, возвращающий группы товаров и количество товаров в каждой группе (Приложение [1.7.4]).

Измерили время выполнения данных запросов и добавление документа - для некоторых запросов в ArangoDB возможно сократить среднее время, изменив схему хранения. Так как при исходной схеме параметризованные атрибуты в запросах являются глубоко вложенными и на текущий момент их индексация в данной СУБД не поддерживается, можем воспользоваться следующей возможностью ArangoDB - хранить документы в виде графов. Представим нашу коллекцию в виде нескольких связанных коллекций вершин и ребер (связей). В наборе “amazon-meta” выделим отдельно коллекцию отзывов и коллекцию оставшихся частей документов, а для связей между ними и связей между схожими продуктами создадим коллекции-ребра.

Новая структура документа набора “amazon-meta” теперь состоит из нескольких документов и имеет вид, приведенный в табл.3. Будем загружать в таком виде коллекции в ArangoDB (Приложение [1.2]).

product_meta	**similars_edges**	**Reviews**	**rev_edges**

Id
ASIN
_key
title
group
salesrank
categories
reviews
  total
  downloaded
  avg_rating
	_from
	_to
		_key
		data
		customer
		rating
		votes
		helpful
			_from
			_to

Табл.3

С ростом желания поддержки гибких схем развивались и реляционные базы данных, так, в PostgreSQL появились типы столбцов json и jsonb. Благодаря чему можно загрузить документы набора “amazon-meta” целиком в базу данных в PostgreSQL, однако, при такой загрузке мы также не сможем оптимизировать запросы, используя индексы по атрибутам в глубоко вложенной структуре документа. Поэтому воспользуемся представлением из табл.3, создав таблицы со столбцами типа jsonb. Для загрузки первого набора “all_viruses” также создадим таблицу со столбцом типа jsonb (Приложение[1.5]).

В MongoDB при загрузке набора “amazon-meta” в виде нескольких коллекций некоторые сложные запросы невозможно будет реализовать, используя на уровне СУБД, так как в ней не поддерживаются join и придется использовать дополнительный модуль для реализации операций map-reduce. Поэтому при тестировании будем рассматривать первую схему хранения - в одной коллекции (Приложение[1.3-1.4]).

Apache Cassandra не позволяет нам хранить документы произвольной структуры без потери функциональности. Если документ имеет сложную структуру и мы не можем определить его схему, приходится хранить его как строку и становится невозможным ни выполнять поиск по атрибутам, ни управлять данными в коллекции. Такой способ хранения не является оптимальным. Так как хотим сравнить время выполнения запросов в данной СУБД, то должны определить его схему в базе данных. Исходный набор “amazon-meta” содержит вложенные структуры, которыми сложно управлять в Apache Cassandra, поэтому для более эффективного управления загрузим этот набор, разделенный по отдельным коллекциям (как в табл.3) и создадим для него схему, где столбцы – атрибуты документов из табл.3. Также для загрузки первого набора “all_viruses” в Apache Cassandra определим схему, соответствующую представлению документа в табл.1.

4.2. Описание данных и особенности СУБД

Рассмотрим подробнее два выбранных набора данных и посмотрим их наиболее вероятные сценарии использования. Исходя из особенностей структур наборов, определим на каких запросах к этим наборам будем тестировать СУБД.

Первый набор не имеет вложенных структур и все типы атрибутов являются простыми, в данном случае атрибут - либо число, либо строка. Один из наиболее вероятных сценариев использования набора “all_viruses” – анализ (группировки, выборки, другие операции реляционной алгебры), т.е. потенциально более ориентирован на выполнение с ним OLAP-операций.

Второй набор является иерархическим с множеством вложенных параметров, которые могут быть различны. Набор данных “amazon-meta” меньше подходит для аналитики, с таким видом представленных метаданных обычно выполняют мелкие транзакции, такие как операции вставки, обновления, т.е. данная коллекция больше ориентирована на выполнение OLTP-операций.

При разработке тестов учтем возможные сценарии использования наборов. Рассмотрим, во время тестирования, разные запросы к этим коллекциям исходя из целей их наиболее вероятного использования. Так при тестировании на первом наборе данных, мы измеряем операционную задержку и пропускную способность, не только на запись и чтение строк, но и рассматриваем три запроса: выборка документов по значению атрибута, лежащего в заданном диапазоне, группировка и сортировка по выбранному атрибуту. Запросы к базам данных приводятся в Приложении[1]. При тестировании на втором наборе большее внимание стоит обратить на операции записи, чтения, но так как в работе рассматриваются и возможности управления данными на уровне СУБД, то протестируем также на некоторых запросах со сложно структурированным набором данных, такие как выборка с вложенным подзапросом, сортировка по вложенному

атрибуту и группировка с большим числом возможных групп. Запросы приводятся в Приложении[1]. Некоторые СУБД плохо подходят для выполнения сложных аналитических операций без использования дополнительных ресурсов и менее гибкие в этом отношении, так, например в Apache Cassandra нельзя выполнить сортировку по любому атрибуту документа заранее при создании таблицы не указав CLUSTERING ORDER по этому атрибуту, также язык CQL не обладает поддержкой вложенных запросов, поэтому не удалось реализовать и протестировать данную СУБД на некоторых запросах. Исходя из особенностей разных типов СУБД и их ориентированности на разные операции[18], показатели могут сильно разнится на запросах, будем искать наиболее оптимальное решение на основе проводимых замеров.

4.3. Методика тестирования

Для тестирования, определим, каким образом будем работать с базами данных, отправлять запросы и фиксировать метрики. Рассматриваем СУБД со следующими характеристиками ArangoDB v.3.1.17, MongoDB v.3.2, Postgresql 9.6, Apache Cassandra v3.0.9. Для замеров посылаются семантически эквивалентные запросы к разным СУБД – выполняются транзакции, операции записи, чтения и выбранные запросы для первого набора данных (Приложение[1.8-1.11]) и второго набора данных (Приложение[1.12-1.15]). Работа с базой данных организуется таким образом: в программе запускаются несколько потоков, параллельно отправляющие запросы к базе данных, общее число выполняемых операций в потоке 1000, во время работы фиксируются показатели, которые нужно измерить. Перед запуском основного тестирования, базы данных разогреваются, выполняя несколько раз те же запросы, что и в самом тесте. Для всех коллекций во время выполнения любых транзакций выполняется синхронизация данных на диск.

4.4. Тестирование операционной задержки

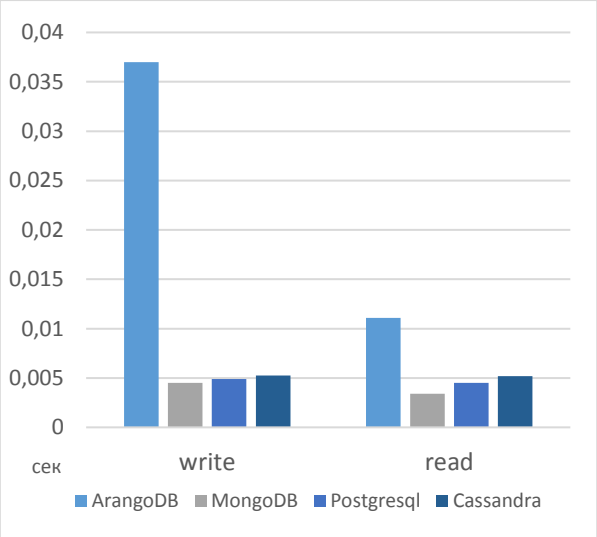
Одним из важных показателей производительности системы является время между отправкой запроса и получением ответа от системы – операционная задержка (operational latency). Данный показатель может быть определен как сумма времени операций, когда выполняется в линейных рабочих процессах или как время самой медленной операцией, выполняемой одним рабочим потоком, когда выполняется в параллельных рабочих процессах.

При увеличении данных и росте системы возрастает стоимость времени отклика. Однако низкая задержка может достигаться в ущерб сохранности данных (durability and consistency). Сохранение данных на диске и синхронизация гарантируют их прочность (durability), но увеличивает задержку, несинхронизация записи на диск влечет уменьшение задержки и увеличение пропускной способности, но и снижает гарантии долговечности данных. Обеспечение сохранности и согласованности данных гарантирует нам корректный результат выполнения запроса в любой момент и то, что данные не будут утеряны при сбое системы, поэтому при тестировании будем обеспечивать в системах синхронизацию данных на диск.

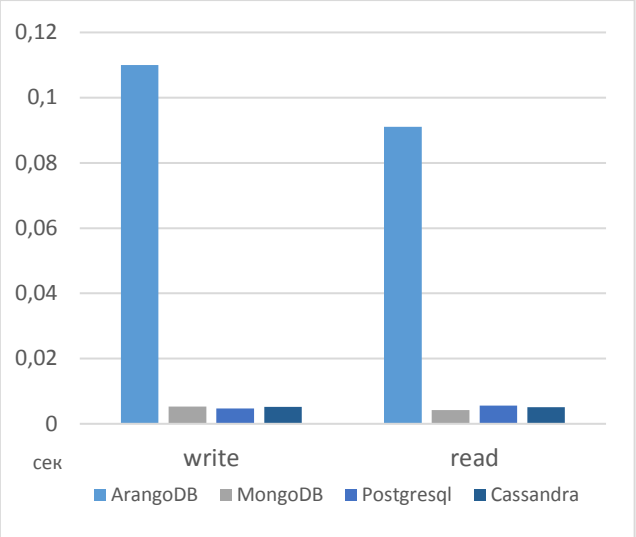
В данной работе для измерения операционной задержки параллельно запускались несколько потоков, которые непрерывно отправляли системе запросы, замеряя разницу в секундах между отправлением запроса и получением результата от системы, затем считалась средняя задержка для потока и брался самый медленный результат среди одновременно запущенных потоков.

Операционную задержу к рассматриваемым СУБД измеряем для операций записи и чтения документа, и на выбранных запросах: для первого набора это операции выборки, сортировки и группировки, скрипты запросов в Приложении [1.8-1.11] и для второго набора были написаны запросы с вложенными атрибутами и вложенными подзапросами, скрипты в

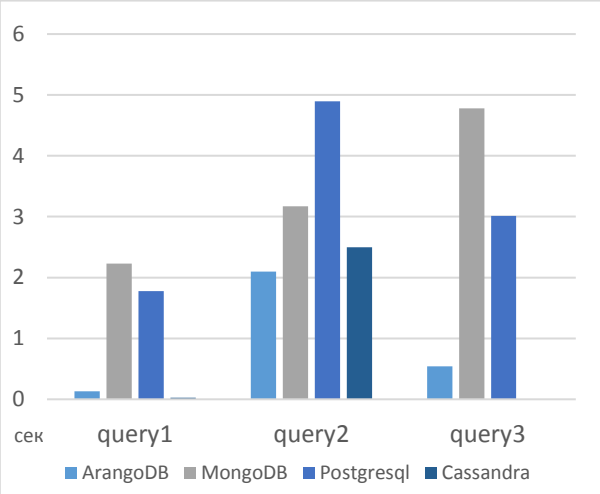
Приложении[1.12-1.15]. Результаты представлены на диаграммах (рис.1-рис.4)



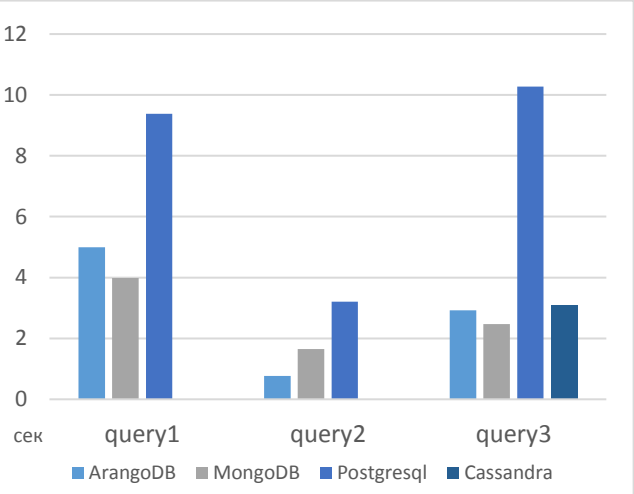
Первый набор “Viruses” Рис.1



Второй набор “Amazon-meta” Рис.2



Первый набор “Viruses” Рис.3



Второй набор “Amazon-meta” Рис.4

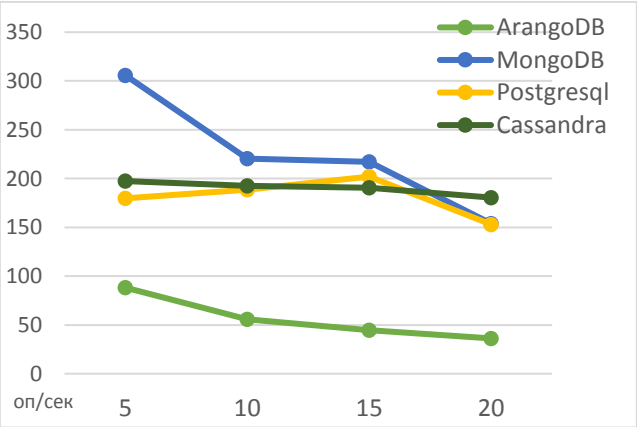
Запрос1: выборка по атрибуту – ‘дата изменения записи’, за определенный период. Дата изменения генерируется случайным образом в программе при тестировании.
Запрос2: группировка по типам генов.
Запрос3: сортировка записей по атрибуту – ‘дата изменения’.

Запрос1: выборка с вложенным подзапросом, всех записей с отзывами определенного числа. Дата генерируется случайным образом в программе при тестировании.
Запрос2: сортировка по вложенному атрибуту.
Запрос3: группировка по типам товаров

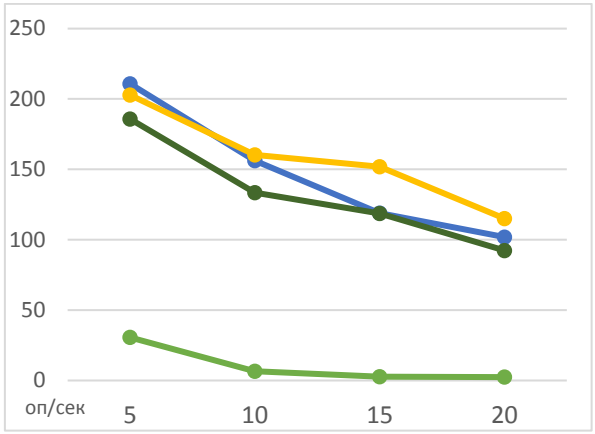
4.5. Тестирование пропускной способности

Необходимым показателем для оценки производительности СУБД является пропускная способность системы (throughput). Данная метрика измеряется количество выполненных транзакций за фиксированную единицу времени. Пропускная способность связана с операционной задержкой и также зависит от синхронизации данных СУБД.

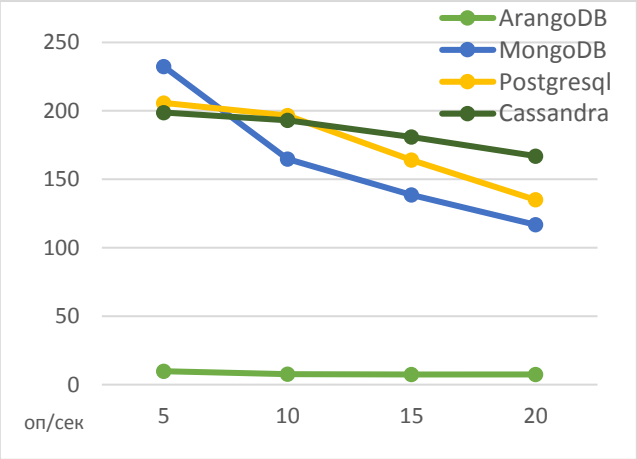
В тестах замерили количество операций чтения, записи и выбранных запросов (Приложение [1.8-1.15]) в секунду, запущенных в несколько параллельно выполняемых потоках для двух датасетов, результаты для ArangoDB, PostgreSQL, MongoDB и Cassandra представлены в виде графиков (рис.5 - рис.14), где нижняя ось – количество параллельно запущенных потоков, что позволяет отследить изменения пропускной способности при увеличении нагрузки на СУБД.



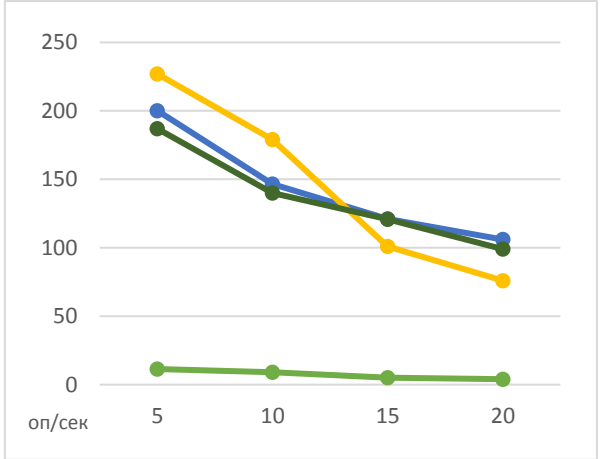
Первый набор “Viruses” – **“Чтение”** Рис.5



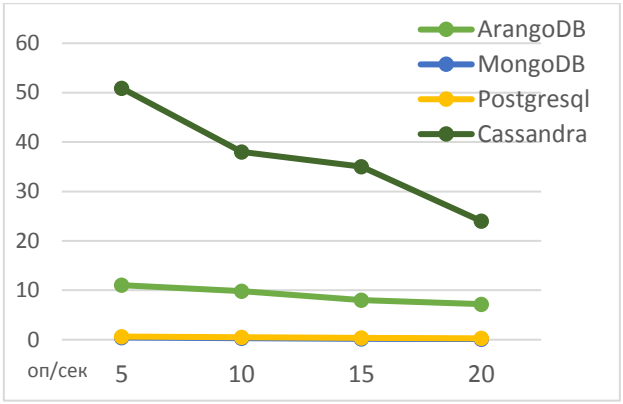
Первый набор “Viruses” – **“Запись”** Рис.6



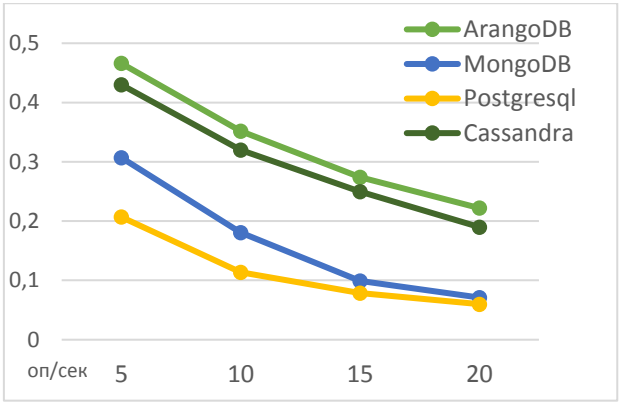
Второй набор “Amazon-meta” – **“Чтение”** Рис.7



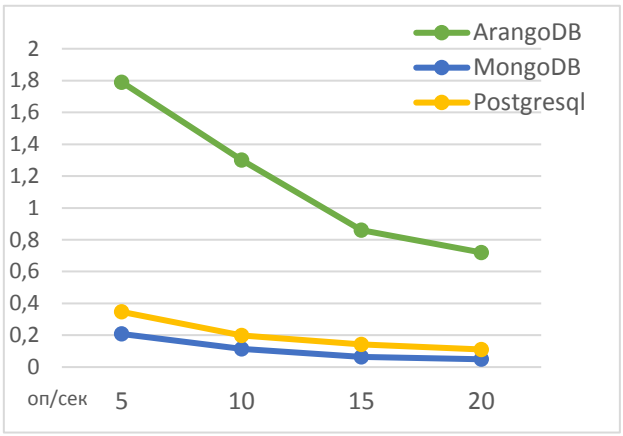
Второй набор “Amazon-meta” – **“Запись”** Рис.8



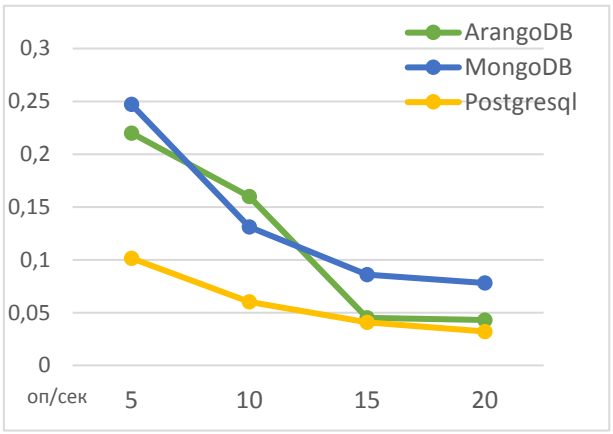
Первый набор “Viruses” – **Запрос1 (выборка)**
Рис.9



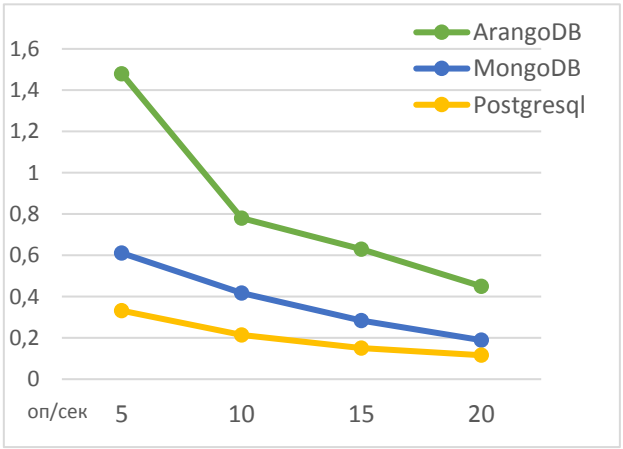
Первый набор “Viruses” – **Запрос2 (групп.)**
Рис.10



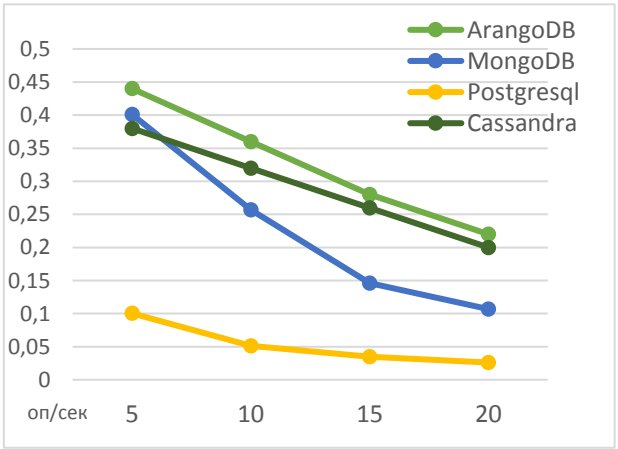
Первый набор “Viruses” – **Запрос3 (сортировка)**
Рис.11



Вт. набор “Amazon-meta” – **Запрос1 (выборка)**
Рис.12



Вт.набор “Amazon-meta”–**Запрос2(сортировка)**
Рис.13



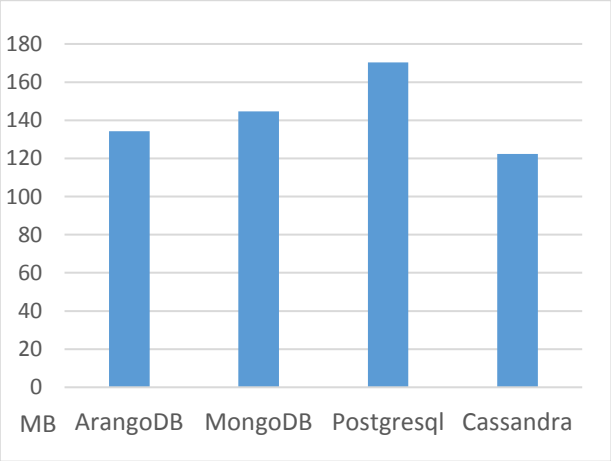
Вт. набор “Amazon-meta” – **Запрос3 (групп.)**
Рис.14

4.6. Измерение использования дискового пространства

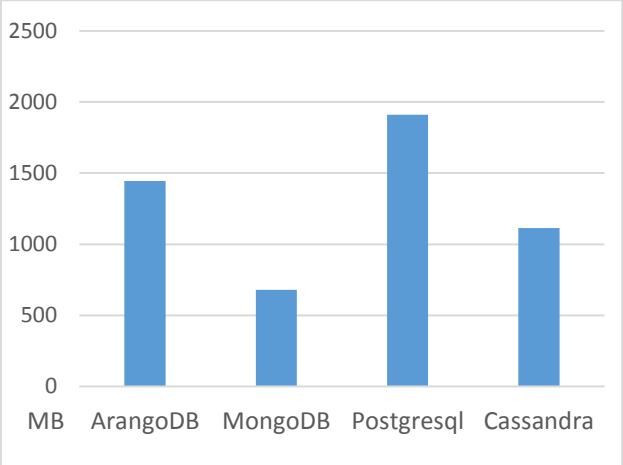
Память на диске (disk storage) – это всегда критический ресурс для любых систем, необходимо мониторить количество свободного пространства на диске и также предусмотрительно рассчитывать занимаемые размеры базы данных, когда речь идет об использовании облачных хранилищ.

При вычислении дискового пространства (disk space usage), занимаемого БД, следует учитывать размеры данных, метаданных и многих других вспомогательных структур системы базы данных. Из-за индексов и дополнительных структур для улучшения производительности, БД может занимать намного больше дискового пространства, чем может быть предусмотрено заранее с учетом действительных данных.

Расчеты по занимаемому количеству дискового пространства рассматриваемых БД в ArangoDB, PostgreSQL, MongoDB и Cassandra, включая дополнительные построенные структуры в данных СУБД, представлены ниже (рис.15 – рис.16).



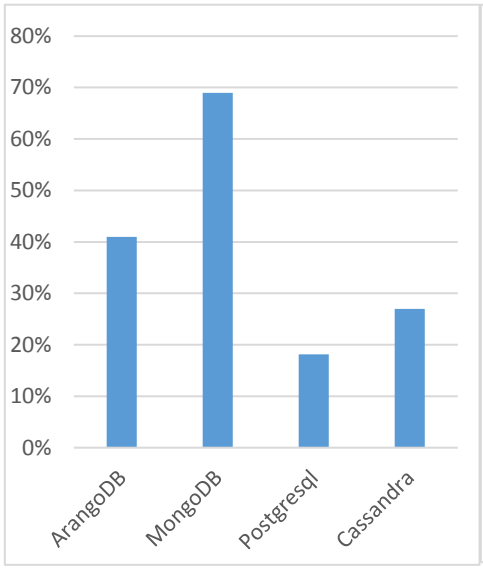
Первый набор “Viruses” **(disk space usage)**
Рис.15



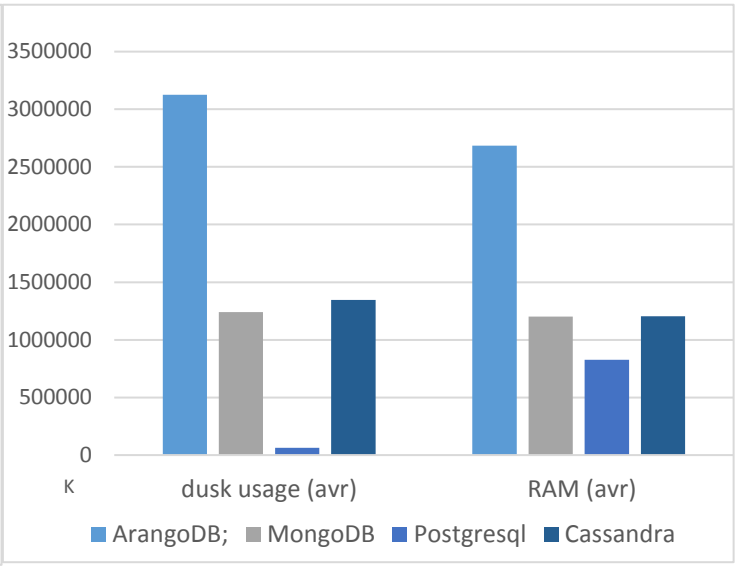
Второй набор “Amazon-meta” **(disk space usage)**
Рис.16

4.7. Анализ системной нагрузки

Метрики использования памяти, диска и CPU при выполнении транзакций, наряду с другими метриками измерения производительности, так же являются важными показателями эффективности доступа к данным. Поэтому, мы проанализировали системную нагрузку (system load), т.е. измерили использование CPU, памяти и диска на протяжении тестирования. Данные о нагрузке собирались каждые 5 секунд во время тестирования системы и вычислялась средняя нагрузка. Результаты представлены в диаграммах (рис. 17 – рис. 18).



CPU Рис.17



disk usage & RAM Рис.18

5. Заключение

5.1. Результаты

При тестировании операционной задержки и пропускной способности на запись/чтение выявили, что в сравнении с другими СУБД MongoDB в среднем быстрее работает на выполнение этих операции с выбранными датасетами, чем другие рассматриваемые СУБД. Однако, при увеличении нагрузки она начинает уступать по этим показателям, сохраняя преимущество только при записи сложно структурированного документа. При большей нагрузке лучшие показатели на чтение и стабильность пропускной способности демонстрирует Cassandra.

В большинстве рассмотренных аналитических запросов лидирует ArangoDB, но в запросе - выборке по первому набору, уступает по показателям Apache Cassandra. Минусом использования СУБД Cassandra может являться меньшая гибкость языка этой СУБД, в сравнении с SQL и возможностями, предоставляемыми ArangoDB и Mongo, и, как следствие - невозможность реализовать сложные запросы. При выполнении аналитических запросов Mongo и Postgresql по-разному обрабатывают запросы на уровне СУБД и их результаты разняться при тестировании на датасете с простой схемой и большим числом атрибутом, при работе именно с вложенными документами и выполнении с ними запросов Mongo лидирует, сразу после Arango, а при выборке с подзапросом даже превосходит. ArangoDB в свою очередь очень сильно уступает по записи и чтению документов из выбранных датасетов. Так, если рассмотреть в совокупности показатели замеров при выполнении операций на запись и чтение и запросах MongoDB демонстрирует в среднем более стабильные результаты, также данная СУБД предоставляет удобные средства для загрузки данных в базу.

При тестировании было показано, что Mongo создает большую системную нагрузку на cpu, но использует во время работы намного меньше

памяти и диска, чем ArangoDB. Postgresql создает меньшую системную нагрузку, чем другие сравниваемые СУБД, но при этом хранение коллекций в данной СУБД занимает большее количество пространства. При оценке памяти на диске, занимаемого наборами в базах данных, Mongo лидирует при хранении сложно структурированного набора, так как в других СУБД для эффективного выполнения аналитических запросов пришлось его разбить на несколько коллекций/таблиц, что привело к увеличению занимаемого пространства на диске.

Таким образом, основываясь на результатах и показателях выбранных метрик, можно выделить MongoDB как более оптимальное решение для хранения и работы с метаданными и слабоструктурированными или иерархическими данными. Минусом для определенных систем может являться отсутствие на данный момент в MongoDB обеспечения ACID свойств в полной мере, как в реляционных базах данных, однако с версии 4.0 заявляется их поддержка при транзакции с множеством документов, что будет являться ее преимуществом относительно других NoSQL-решений.

5.2. Итоги

Таким образом, в данной работе, в пункте 3.1, были рассмотрены основные характеристики следующих СУБД ArangoDB, PostgreSQL, MongoDB и Cassandra, важные для обеспечения управления метаданными в базах данных. Для сравнения СУБД были выделены основные метрики, пункт 3.3, чтобы определить оптимальное решение в качестве хранилища для системы управления метаданными. В работе для тестирования СУБД были подобраны и подготовлены два разных по структуре набора метаданных (пункт 3.2), схемы хранения которых были оптимизированы с учетом особенностей СУБД, подробно рассмотрено в пунктах 4.1-4.2. Для определения производительности была определена методология тестирования (пункт 4.3), были разработаны скрипты для загрузки данных,

выполнения запросов, чтения и записи данных (скрипты запросов в Приложении[1.5-1.12]), были получены и наглядно представлены для сравнения результаты тестирования и замеров, выбранных метрик, приведено в пунктах 4.3-4.7. Анализ результатов и определение более оптимальный СУБД как основы для построения системы управления метаданными описаны в пункте 5.1.

Литература

1. Claudius Weinberger. Benchmark: PostgreSQL, MongoDB, Neo4j, OrientDB and ArangoDB [https://www.arangodb.com/2015/10/benchmark-postgresql-mongodb-arangodb/].October 13, 2015
2. Fábio Roberto Oliveira, Luis del Val CuraPerformance. Evaluation of NoSQL Multi-Model Data Stores in Polyglot Persistence Applications. July 11 - 13, 2016
3. National Information Standards Organization; Rebecca Guenther; Jaqueline Radebaugh. Understanding Metadata. 2004
4. Jorge Bernardino, Pedro Furtado, Veronika Abramova. Which NoSQL Database?, 2014
5. Ion LUNGU, Bogdan George TUDORICA. The Development of a Benchmark Tool for NoSQL Databases. 2013
6. Rick Cattell, R. Scalable SQL and NoSQL data stores, 2011.
7. Bogdan George Tudorica, Cristian Bucur. A comparison between several NoSQL databases with comments and notes. 2011
8. Haleemunnisa Fatima, Kumud Wasni. Comparison of sql, nosql and new sql databases in light of internet of things – a survey. 2016
9. Alexandre Fonseca, Anh Thu Vu, Peter Grman. Evaluation of NoSQL databases for large-scale decentralized microblogging. 2013
10. John Klein, Ian Gorton, Neil Ernst, Patrick Donohoe, Kim Pham, Chrisjan Matser. Performance Evaluation of NoSQL Databases: A Case Study. 2015
11. https://docs.arangodb.com/3.3/Manual/index.html
12. http://cassandra.apache.org/doc/latest/
13. https://www.postgresql.org/docs/10/static/index.html
14. https://docs.mongodb.com/manual/
15. Claudius Weinberger. NoSQL Performance Benchmark 2018 – MongoDB, PostgreSQL, OrientDB, Neo4j and ArangoDB. February 14, 2018
16. http://json.org/json-ru.html

17. http://bsonspec.org/

18. Jignesh M. Patel, Operational NoSQL Systems: What's New and What's Next? IEEE Computer, April 2016, IEEE Computer Society.

Приложение
1. Скрипты для загрузки и запросов

1. Скрипт первого набора в Arangodb:

arangoimp --server.database viruses --file all_viruses.json --type json --collection all_viruses --batch-size 154980352

2. Скрипт загрузки второго набора в Arangodb:

arangoimp –server.database amazon --file am_meta_vert.json --type json --collection am_meta_vert –batch-size 267726848

arangoimp –server.database amazon --file am_reviews_vert.json --collection am_reviews_vert –batch-size 744325120

arangoimp –server.database amazon --file am_rev_edges.json --collection am_rev_edges –batch-size 371912704

arangoimp –server.database amazon --file am_similar_edges.json --collection am_similar_edges –batch-size 88395776

3. Скрипт загрузки первого набора в MongoDB:

mongoimport --db viruses --collection all_viruses --file all_viruses.json –jsonArray

4. Скрипт загрузки второго набора в MongoDB:

mongoimport --db amazon --collection am_meta --file am_meta.json –jsonArray

5. Скрипт для Postgresql:

Create database viruses;

Create table all_viruses (id serial primary key, doc jsonb);

Create database amazon;

Create table am_meta_vert (id serial primary key, doc jsonb);

Create table am_reviews_vert (id serial primary key, doc jsonb);

Create table am_rev_edges (id serial primary key, doc jsonb);

Create table am_similar_edges (id serial primary key,doc jsonb);

(загрузчик данных в таблицы реализуется с помощью программы на C#)

6. Скрипт для Cassandra:

Create keyspace viruses with replication={'class' : 'SimpleStrategy', 'replication_factor':1 };

Create keyspace amazon with replication={'class' : 'SimpleStrategy', 'replication_factor':1 };

(загрузчик данных в таблицы в пространстве ключей реализован с помощью программы на C#)

7. Запросы в ArangoDB при хранении набора метаданных Amazon в одной коллекции:

1. *‘RETURN LENGTH (FOR doc IN part_amazon FILTER (FOR rev IN doc.reviews.reviews FILTER rev.data == @data RETURN rev) != [ ] RETURN doc’;*

2. ‘*FOR doc IN part_amazon SORT doc.reviews.total DESC LIMIT 100 RETURN {‘title’: doc.title, ‘total_reviews’: doc.reviews.total}`;*

3. *‘FOR doc1 IN part_amazon LET sim_rat = (FOR doc2 IN part_amazon FILTER doc2.ASIN IN doc1.similar.asin RETURN doc2.reviews.avg_rating) RETURN {‘title’: doc.title , ‘avg_rating’: doc1.reviews.avg_rating, ‘similar_ratings’: sim_rat}’;*

4. *‘FOR doc IN part_amazon COLLECT group = doc.group WITH COUNT INTO g RETURN {‘group’: group, ‘InGroup’: g)’;*

8. Запросы в ArangoDB к БД “Viruses”:

1. *‘FOR doc in all_viruses FILTER doc.Modification_date=@date RETURN doc’;*

2.*’FOR doc in all_viruses COLLECT type_of_gene = doc.type_of_gene RETURN {type: type_of_gene}’;*

3. *‘FOR doc in all_viruses SORT doc.Modification_date RETURN doc’;*

9. Запросы в MongoDB к БД “Viruses”:

1.’db.all_viruses.find({“Modification_date”: @data})’

2.’ db.all_viruses.aggregate([{$group : {“type_of_gene:”$item”}}])’

3. ‘db.all_viruses.find().sort({“Modification_date”: -1})’

10. Запросы в Postgresql к БД “Viruses”:

1’select doc from all_viruses where doc @> '{"Modification_date":"@dat"}';’

2 ‘select doc->'type_of_gene' from all_viruses group by (doc->'type_of_gene');’

3 ‘select doc from all_viruses order by (doc->'Modification_date');’

11. Запросы в Cassandra к БД “Viruses”:

*1. ‘select * from Viruses.all_viruses where modification_date =@date’*

2. ‘select Viruses.group_and_count(type_of_gene) from Viruses.am_meta_vert;’ (aggregate - Viruses.group_and_count() – определяется заранее);

12. Запросы в ArangoDB к БД “Amazon”:

1.‘let k = (for i in am_rev_vert filter i.data == @date return distinct i.ASIN) for j in am_meta_vert filter j._key in k return j’

2. ‘for j in am_meta_vert sort j.reviews.total desc return j’

3. ‘for j in am_meta_vert collect group = j.group return {gr: group, 'count': count(group)}’

13. Запросы в MongoDB к БД “Amazon”:

1.’ db.am_meta.find({“reviews.reviews”: {$elemMatch: {“data”: @data}}”})’

2. ’ db.am_meta.find().sort({“reviews.total”: 1})’

3 ‘db.am_meta.aggregate([{$group : {“group:”$item”}}])’

14. Запросы в Postgresql к БД “Amazon”:

1. ‘select doc from am_meta_vert where doc->'ASIN' in (select doc->'ASIN' from am_rev_vert where doc @> '{"data": "@dat "}');’

2. ‘select doc from am_meta_vert order by doc->'reviews'->'total' desc;’

3. ‘select doc->'group' from am_meta_vert group by doc->'group';’

15. Запросы в Cassandra к БД “Amazon”:

3. ‘select Amazon.group_and_count(group) from Amazon.am_meta_vert;’ (aggregate - Amazon.group_and_count() – определяется заранее);

2. Образцы наборов данных

Экземпляр документа из набора ‘amazon-meta’


```
{
    "Id": 1,
    "ASIN": "0827229534",
    "title": "Patterns of Preaching: A Sermon Sampler",
    "group": "Book",
    "salesrank": 396585,
    "similar": {
      "total": 5,
      "asin": [
        "0804215715",
        "156101074X",
        "0687023955",
        "0687074231",
        "082721619X"
      ]
    },
    "categories": {
      "total": 2,
      "categories": [
        "   |Books[283155]|Subjects[1000]|Religion &
Spirituality[22]|Christianity[12290]|Clergy[12360]|Preaching[12368]",
        "   |Books[283155]|Subjects[1000]|Religion &
Spirituality[22]|Christianity[12290]|Clergy[12360]|Sermons[12370]"
      ]
    },
    "reviews": {
      "total": 2,
      "downloaded": 2,
      "avg_rating": 5.0,
      "reviews": [
        {
          "data": "2000-7-28",
          "customer": "A2JW67OY8U6HHK",
          "rating": 5.0,
          "votes": 10,
          "helpful": 9.0
        },
        {
          "data": "2003-12-14",
          "customer": "A2VE83MZF98ITY",
          "rating": 5.0,
          "votes": 6,
          "helpful": 5.0
        }
      ]
    }
  }
```


Рис. 1

Экземпляр документа из набора ‘All_Viruses.gene_info’


```
{
    "#tax_id": 10390,
    "GeneID": 4811502,
    "Symbol": "MDV041",
    "LocusTag": "MD5V_041",
    "Synonyms": "GaHV2Md5_gp041",
    "dbXrefs": "-",
    "chromosome": "-",
    "map_location": "-",
    "description": "similar to HSV1 UL28 DNA packaging protein",
    "type_of_gene": "protein-coding",
    "Symbol_from_nomenclature_authority": "-",
    "Full_name_from_nomenclature_authority": "-",
    "Nomenclature_status": "-",
    "Other_designations": "similar to HSV1 UL28 DNA packaging
protein|DNA packaging terminase subunit 2",
    "Modification_date": 20160806,
    "Feature_type": "-"
}
```


Рис. 2