МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВАСТОПОЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Институт информационных технологий и управления в технических системах
(полное название института)

кафедра «Информационные системы»
(полное название кафедры)

Пояснительная записка

к выпускной квалификационной работе магистра

на тему «Исследование методов оптимизации распределенного хранилища данных для системы электронного документооборота»

Выполнил: студент III курса, группы: ИС/м-18-1-з

Направления подготовки (специальности) 09.04.02

Информационные системы и технологии
(код и наименование направления подготовки (специальности))

профиль (специализация) Интеллектуальные информационные системы

Мжачев Илья Александрович
(фамилия, имя, отчество студента)

Руководитель Пелипас В.О., доцент
(фамилия, инициалы, степень, звание, должность)

Дата допуска к защите «____» _____ 20 ___ г.

Зав. кафедрой _____ И.П. Шумейко
(подпись) (инициалы, фамилия)

2021 г.

АННОТАЦИЯ

Пояснительная записка содержит в себе 90 страниц основного текста, 38 иллюстраций, 5 таблиц, 1 приложение и 31 использованный источник.

Ключевые слова: электронные документы, электронный документооборот, оператор электронного документооборота, система хранения данных, распределенное хранилище данных, методы оптимизации хранилищ данных.

Рассмотрена задача оптимизации распределенного хранилища данных для системы электронного документооборота.

Проанализировать существующие подходы к организации систем хранения данных, методы оптимизации хранилищ данных, а также требования, выдвигаемые к системам операторам электронного документооборота, была предложена распределенная архитектура хранилища с механизмами перераспределения данных, которая позволит экономить денежные средства при увеличении объема хранилища, сохраняя высокие показатели производительности.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Актуальность темы. Первые системы электронного документооборота появились еще в 80-х годах прошлого века, но широкого распространения они не получили, из-за существующих в то время технических возможностей. Они были дороги в разработке, обслуживании и решали узкие задачи отдельно взятых компаний.

Сегодня же можно сказать, что данные системы захватывают все отрасли жизни. С их помощью повышают эффективность деятельности коммерческих компаний и промышленных предприятий, а в государственных учреждениях на базе технологий электронного документооборота решаются задачи внутреннего управления, межведомственного взаимодействия и взаимодействия с населением.

Система электронного документооборота (СЭД) – организационно-техническая система, обеспечивающая процесс создания, управления доступом и распространения электронных документов в компьютерных сетях, а также обеспечивающая контроль над потоками документов в организации.

Электронный документ – документ, созданный с помощью средств компьютерной обработки информации.

Неотъемлемой частью любой системы электронного документооборота является система хранения данных – это очевидно, так как электронные документы должны где-то храниться.

Система хранения данных (СХД) – комплекс аппаратных и программных средств, который предназначен для хранения и оперативной обработки информации.

С каждым годом, объем хранимой информации системами электронного документооборота неуклонно растет. Также в нынешнее время из-за пандемии COVID-19, системы ЭДО (Электронный документооборот) стали жизненно необходимы для дистанционной работы предприятий. Стоит отметить, что компании не торопятся увеличивать бюджет на увеличение объема хранилищ и их

поддержку, разрыв между ростом объема данных и необходимыми расходами на их сопровождение продолжает увеличиваться. Вследствие этого вопросы оптимизации хранилищ данных становятся все более и более остро.

Цель и задачи работы. Целью данной работы является программная реализация оптимизированного хранилища данных для системы оператора электронного документооборота, которое реализует хранение, обработку и организацию доступа к данным.

Для достижения поставленной цели необходимо решить следующие задачи:

– исследовать существующие методы и подходы к организации хранилищ данных;

– разработать оптимизированный метод организации хранилища данных для решения задач системы электронного документооборота;

– разработать программный модуль хранения данных.

Предмет и объект исследования. Объектом исследования настоящей работы является система-оператор электронного документооборота. Предметом исследования является методы организации хранения данных.

Практическое значение работы. Результаты данной работы могут представлять интерес для инженеров программного обеспечения, разрабатывающих системы электронного документооборота, а также организациям, использующим системы электронного документооборота.

Структура работы. Данная работа состоит из пояснительной записки, включающей в себя введение, три раздела, заключение, список использованных источников, перечень сокращений и условных обозначений и приложения.

В первом разделе рассматриваются существующие подходы к организации систем хранения данных, существующие облачные хранилища данных. Производиться описание предметной области систем операторов электронного документооборота, описываются методы оптимизации хранилища данных.

Во втором разделе предлагается архитектура распределенной системы хранения данных полностью отвечающей требованиям оператора электронного

документооборота, производиться проектирование системы, описание критериев оценки эффективности и принципов верификации и тестирования.

В третьем разделе обосновывается выбор технологических средств для реализации системы хранения данных, разрабатывается структура данных, разрабатываются программные модули и проводиться тестирование и верификация полученной системы.

1 АНАЛИЗ ЗАДАЧИ ОПТИМИЗАЦИИ ХРАНЕНИЯ ДАННЫХ ДЛЯ СИСТЕМ ЭЛЕКТРОННОГО ДОКУМЕНТООБОРОТА

1.1 Технологии хранения данных

В первую очередь, перед обзором существующих подходов к решению задачи хранения данных, необходимо дать определение систем хранения данных (СХД), их разновидности и их устройства.

1.1.1 Понятие систем хранения данных

Система хранения данных – комплекс аппаратных и программных средств, который предназначен для хранения и оперативной обработки информации, как правило, большого объема. Информация – это файлы, в том числе медиа, структурированные и неструктурированные данные, резервные копии, архивы. В качестве носителей информации используются запоминающие устройства [1]. СХД различаются по уровням хранения:

– блочное хранилище: СХД используется как обычный диск, который можно форматировать, устанавливать на него ОС (Операционные системы), создавать логические диски. Данные хранятся не файлами, а блоками, что ускоряет операции ввода-вывода. Подходит для высокопроизводительных вычислений, СУБД (Система управления базой данных), хранения больших объемов данных;

– файловое хранилище: данные хранятся в виде файлов, которые размещаются в каталогах. Такая СХД используется для хранения «холодной» информации, которая не требуется для операционных вычислений;

– объектное хранилище: ориентировано на работу с большими неструктурированными данными объемом до петабайтов. Информация хранится не в виде файлов, а в виде «объектов» с уникальными идентификатором и метаданными. Используется в аналитике, машинном обучении, для хранения

«тяжелых» медиа-файлов и резервных копий, разработки и эксплуатации приложений в облаке, хостинга веб-сайтов.

Также системы хранения данных различаются по частоте использования хранимых данных:

– системы краткосрочного хранения (online storage). Такого рода системы обязаны быть иметь высокие показатели скорости доступа к данным, содержат небольшой объем информации и как правило данные «живут» в них от двух недель до одного месяца;

– системы средней продолжительности (near-line storage). Имеют средний показатели скорости доступа, средний объем информации, данные могут храниться на протяжении года;

– системы долговременного хранения (offline storage). У таких систем низкой уровень скорости доступа, большие объемы информации, данные хранятся от года и больше.

Помимо этого, их также различают по типам подключения [2, 3]:

– DAS (Direct-attached storage). система хранения данных с прямым непосредственным подключением к серверу или рабочей станции, без помощи сети;

– NAS (Network-attached storage). Файловый сервер, который включен в локальную сеть;

– SAN (Storage Area Network). Сеть, которая объединяет разнотипные хранилища (диски, оптические приводы, ленточные массивы), но которые воспринимаются операционной системой как единое логическое хранилище данных, или как сетевой логический диск.

DAS – подразумевает прямое (непосредственное) подключение носителей информации к серверу либо рабочей станции. При этом накопители могут быть внутренними (установлены непосредственно в корпусе сервера) либо внешними. Самый простой пример DAS-системы – это жесткий диск, который расположен,

внутри сервера или рабочей станции. Архитектура DAS системы представлена на рисунке 1.1.

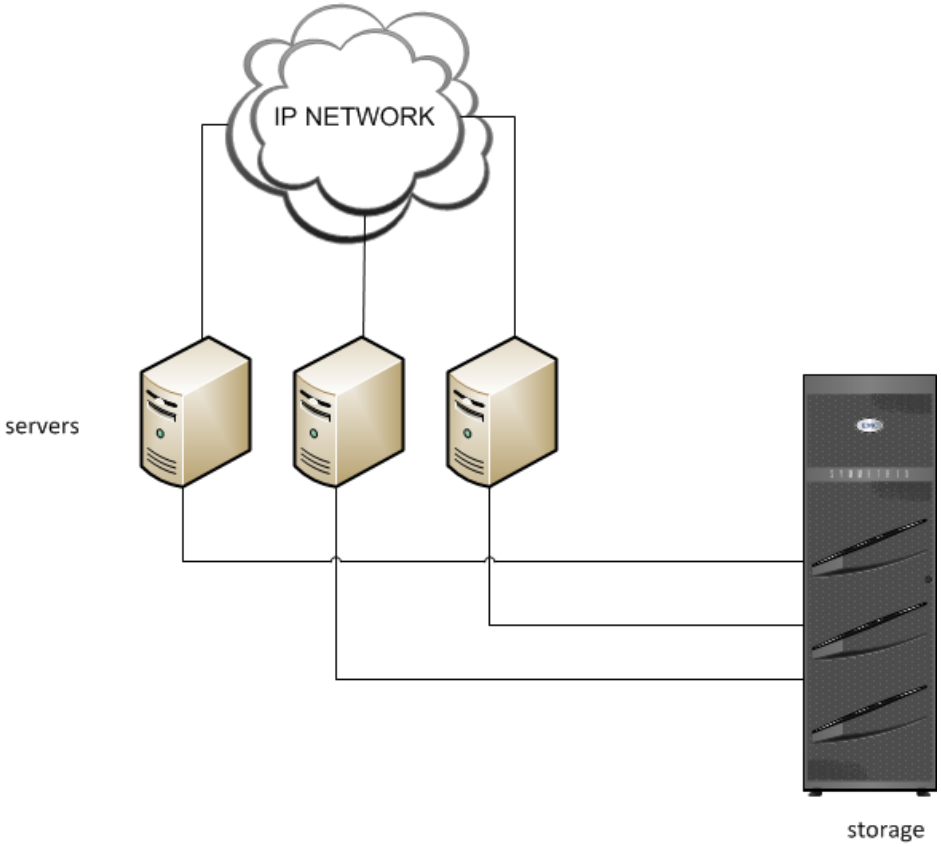


Рисунок 1.1 – Архитектура DAS системы [3]

Данная система имеет относительно низкую стоимость оборудования и очень хорошо подходит для хранения мультимедиа данных большого объема благодаря высокой скорости обмена данными. Можно выделить преимущества данной системы:

- низкая стоимость;
- простота эксплуатации;
- может быть быстро развернута;
- высокая скорость обмена данными.

К большому сожалению, DAS-системы плохо масштабируются. Устройства хранения имеют относительно небольшое количество портов, что ограничивает количество хостов, которые могут непосредственно подключаться к хранилищу.

Ресурсы, которые не используются, не могут быть перераспределены. Именно по этим причинам были разработаны системы NAS и SAN архитектуры, которые являются сетевыми.

NAS-системы – это сетевые системы хранения данных, непосредственно подключаемые к сети точно так же, как и сетевой принт-сервер или маршрутизатор. Данная система предоставляет доступ к файлам через IP сети.

Устройство NAS использует собственную операционную систему, которая оптимизирована для файловых операций ввода/вывода, а также освобождена от всех функций операционной системы, не связанных с обслуживанием файловой системы.

NAS используется для работы с данными файлового типа, к которым нужен коллективный одновременный доступ. NAS работает «поверх» существующей локальной сети, через общие коммутаторы/маршрутизаторы. Архитектура системы изображена на рисунке 1.2.

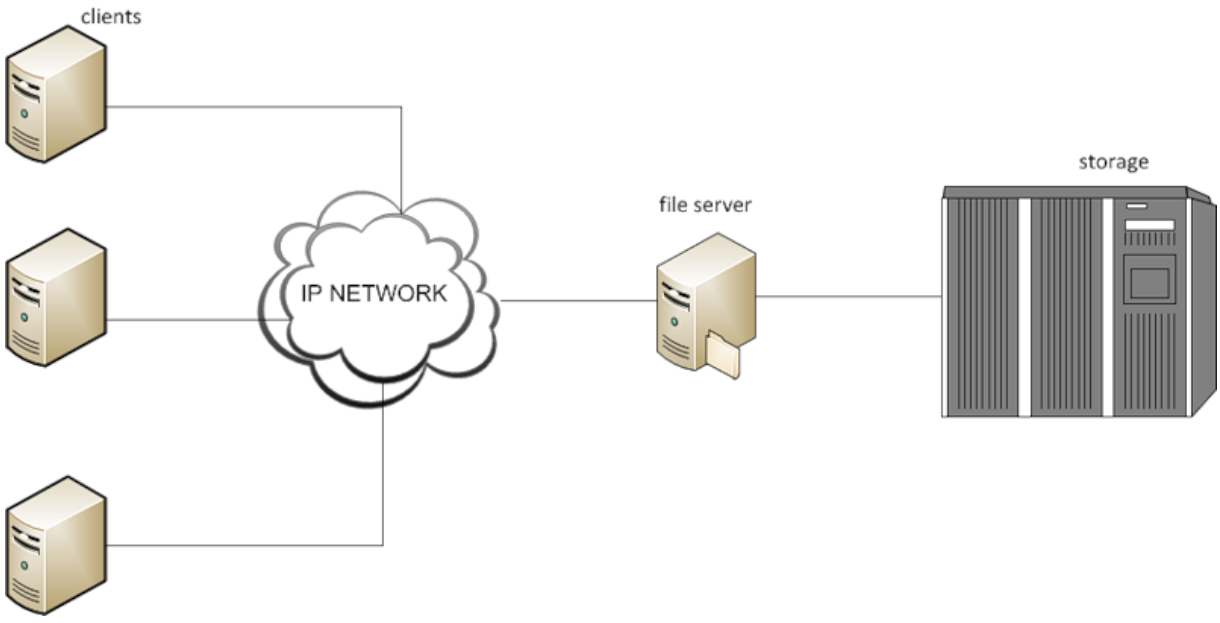


Рисунок 1.2 – Архитектура NAS системы [3]

Преимуществами данной системы являются:

– эффективное использования всех ресурсов памяти;

– высокая производительность серверов;

– уменьшение нагрузки на сервер приложений;

– прост в эксплуатации.

Из недостатков можно выделить сложность масштабируемость серверов, а также повышенную нагрузку на сеть.

SAN-системы – представляет собой специализированную сетевую инфраструктуру для хранения данных, которая связывает один или несколько разнотипных хранилищ в единую сеть. Данная сеть позволяет большому числу пользователей хранить свои данные в одном месте и совместно использовать их. В основном используется блочный тип хранения данных. Преимущества данной системы следующие:

– централизованное управление данными;

– высокий уровень быстродействия;

– высокий уровень отказоустойчивости;

– высокий уровень масштабируемости.

На рисунке 1.3 изображена архитектура SAN-системы.

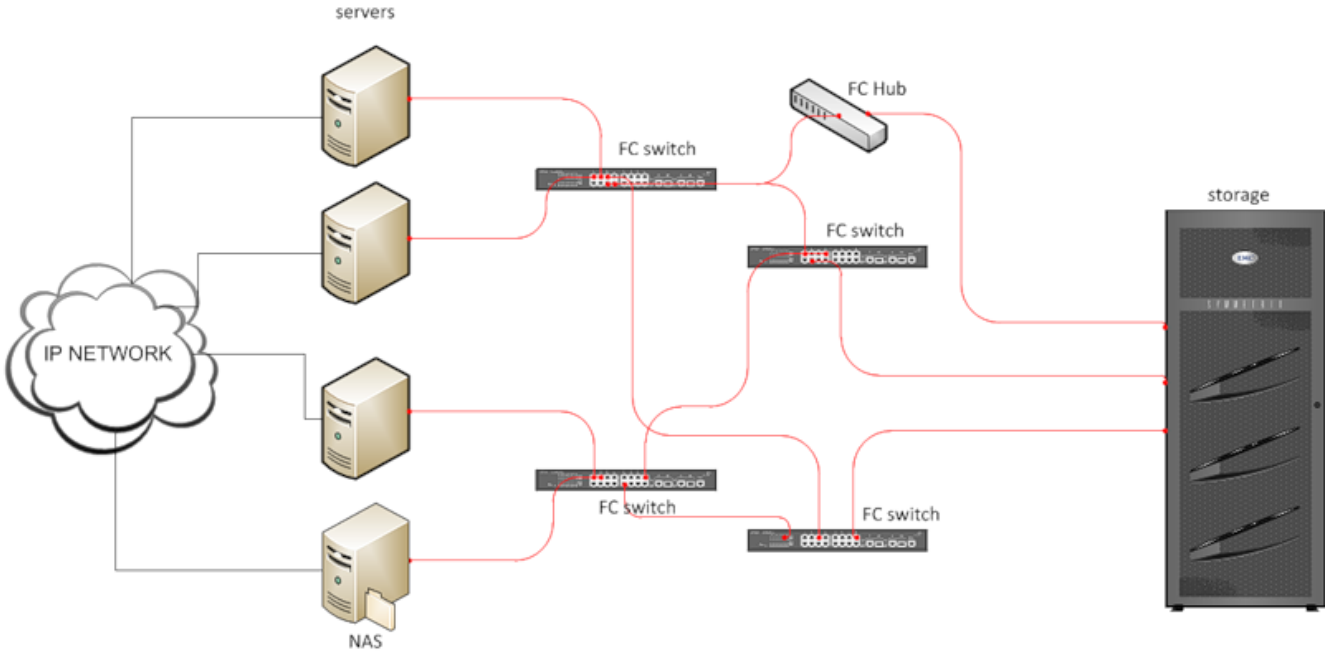


Рисунок 1.3 – Архитектура SAN-системы [3]

Но самым главным недостатком SAN архитектуры является его дороговизна, далеко не каждая фирма может позволить себе данную систему.

1.1.2 Виды и характеристики запоминающих устройств

Запоминающие устройство (ЗУ) – устройство, предназначенное для записи и хранения данных. В основе работы запоминающего устройства может лежать любой физический процесс, обеспечивающий приведение системы к двум или более устойчивым состояниям [4].

По типу доступа к данным ЗУ делятся на:

– устройства с последовательным доступом. Обращения к данным, происходят в заранее заданном порядке;

– устройства с произвольным доступом. Единовременно можно получить доступ к любым данным по адресу.

По зависимости от электропитания:

– энергозависимые. Данные после потери электропитания исчезают;

– энергонезависимые. Данные после потери электропитания никуда не исчезают.

По возможности записи:

– устройства с однократной записью, без возможности перезаписи;

– перепрограммируемые устройства. ЗУ с возможностью многократной перезаписи, затрудненной долгим временем записи или ограниченным числом циклов записи;

– устройства со свободной многократной записью.

По назначению:

– внутренние устройства: предназначены для хранения данных непосредственно необходимых во время выполнения программы. Обладают быстрым уровнем доступа к данным (так как подключены на прямую к процессору);

– внешние устройства: предназначены для хранения больших объёмов данных на сменных или фиксированных носителях. Обладают медленным уровнем доступа к данным.

Для решения задач систем электронного документооборота не подходят устройства с однократной записью, а также перепрограммируемые устройства, так как постоянно поступают новые данные, которые подлежат хранению. Также не подходят энергозависимые запоминающие устройства.

Рассмотрим самые распространенные запоминающие устройства на сегодня:

– оперативные запоминающие устройства (ОЗУ). Является энергозависимым ЗУ, с произвольным доступом к данным, возможностью многократной записи данных и относиться к внутренним устройствам;

– HDD-диски (Hard Disk Drive). Энергонезависимое ЗУ, с произвольным доступом к данным, возможностью многократной записи данных и относиться ко внешним устройствам. Основан на принципе магнитной записи;

– SSD-диски (Solid State Drive). Энергонезависимое запоминающие устройство, с произвольным уровнем доступа, возможностью многократной записи данных и относиться ко внешним устройствам. Использует технологию флеш-памяти;

– ленточные накопители (Стриммер). Энергонезависимое ЗУ, с последовательным уровнем доступа, возможностью многократной записи и относиться ко внешним устройствам. Основан на принципе магнитной записи.

Запоминающие устройства имеют следующие характеристики:

– емкость: максимально-возможный объем хранимых данных;

– среднее время доступа: высчитывается по простой формуле, среднее время поиска (mean seek time) суммируется с временем ожидания (mean wait time), т.е. временем извлечения информации с диска;

– скорость передачи данных;

– число операций ввода-вывода в секунду (Input/Output Per Second, IOPS);

– пропускная способность передачи данных (data throughput). Показывает, какой объем данных можно передать в единицу времени.

ОЗУ обладает самой маленькой емкостью среди описанных ЗУ, при этом обладает самым высоким средним временем доступа, скорости передачи данных, числом операций ввода-вывода, пропускной способностью. Все этим преимущества – благодаря прямому подключению к процессору. Из минусов стоит отметить, что данные ЗУ, помимо малой емкости, еще и самые дорогие в соотношении цены и объема хранимой информации. Также ОЗУ является энергозависимой, и нельзя гарантировать надежность хранения данных.

HDD-диски – можно сказать, что они самые распространенные на сегодняшний день. Имеют следующие преимущества:

– низкая стоимость в перерасчете на объем хранимой информации;

– неограниченное количество циклов записи;

– возможность восстановления информации. Имеется в виду то, что при выходе из строя диска, данные находящиеся на нём можно восстановить.

Из недостатков можно отметить низкое среднее время доступа к информации, а также небольшое количество операций ввода-вывода в секунду.

SSD-диски по сравнению с HDD-дисками, имеют одно очень важное преимущество – это высокое среднее время доступа к данным, на порядок выше IOPS [5]. Для сравнения у HDD-дисков IOPS равняется 80-100, в то время как у SSD-дисков он более 8000 [5]. Но за такими важными преимуществами скрываются свои недостатки:

– высокая стоимость в перерасчете на объем хранимой информации;

– ограниченное количество циклов записи. По сути, после каждой записи информации на диск, физическая ячейка памяти «сжигает» ее.

Ленточные накопители являются самыми дешевыми (в соотношении цены и объема хранимой информации), надежными и самыми медленными.

Таким образом ОЗУ хоть и самые быстрые, но не подходят для надежного хранения данных. HDD-диски – это медленные и большие носители информации,

которые могут успешно решать задачи хранения большого объема данных, не требующих быстрой скорости доступа к данным. В свою очередь SSD-диски – это быстрые хранилища меньшего объема (по сравнению с HDD), которые могут решать задачи быстрого доступа к данным. Также стоит отметить, что данные диски имеют ограниченный ресурс циклов перезаписи, что в свою очередь не позволяет использовать их в системах с частой перезаписью данных. Ленточные носители – большие хранилища данных, но очень медленные, такого типа ЗУ подойдут для архивного хранения данных.

В данной научной работе намеренно отсутствует описание аппаратной реализации данных носителей информации, а также описание некоторых характеристик, потому что это не является целью данной работы.

На сегодняшний день, в свободном доступе можно приобрести HDD-диски максимального объема в 16 терабайт (ТБ), в свою очередь SSD-диски максимального объема в 8 ТБ. Очевидно, что данных объемов будет недостаточно, для средних и больших систем электронного документооборота. Также HDD и SSD диски не обладают требуемыми уровнями надежности, прогноз выхода из строя диска является сложной. Согласно статистике компании «Backblaze», из 13 тысяч жестких дисков за 3 года отказывают примерно от 3.1% до 26.5% [5]. С целью увеличения общего объема памяти и для повышения уровня надежности систем хранения данных – была разработана технология RAID (Redundant Array of Independent Disks).

1.1.3 Технология RAID

Технология RAID – массив независимых дисков с избыточностью хранения данных [6]. Избыточность означает то, что все байты данных при записи на один диск дублируются на другом диске, и могут быть использованы в том случае, если первый диск откажет. Кроме того, эта технология помогает увеличить IOPS. Существует множество различных уровней (конфигураций) RAID-массивов, которые каждый по-своему решает поставленные задачи надежности и производительности [6, 7].

RAID 0. Предполагает одновременное использование нескольких жестких дисков с целью существенного увеличения производительности рабочей станции. Информация разбивается на блоки данных фиксированной длины и записывается на несколько дисков поочередно. При отказе одного из дисков неработоспособной оказывается вся система, так как данные до этого были равномерно записаны по всем хранилищам из массива. Схема данного уровня представлена на рисунке 1.4.

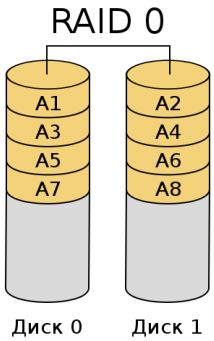


Рисунок 1.4 – RAID 0 [6]

RAID 1. Полностью решает проблему надежности, массив состоит из двух (или более) дисков, являющихся полными копиями друг друга. Только половина ёмкости массива отводиться под данные. Скорость считывания данных выше чем скорость записи данных. Схема уровня изображена на рисунке 1.5.

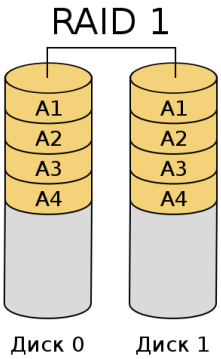


Рисунок 1.5 – RAID 1 [6]

RAID 2. Представляет собой улучшенную версию RAID 0. Данные все также записываются на несколько дисков, логически представляющих единое целое

дисковое пространство, но было введено использования кода Хэмминга при записи данных. Данный код способен исправлять возникающие ошибки (в случае выхода из строя одного из дисков в массиве), используя проверочные последовательности. Коды коррекции ошибок требуют достаточно много дискового пространства, что повышает избыточность (для этих целей выделяются отдельные диски из массива). Схема представлена на рисунке 1.6.

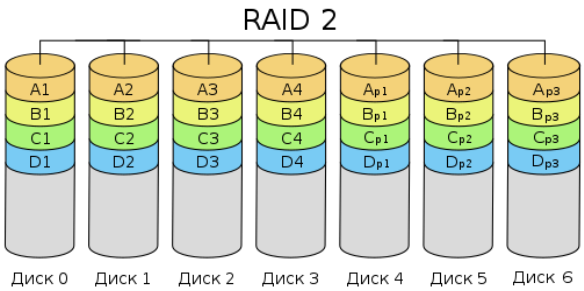


Рисунок 1.6 – RAID 2 [6]

RAID 3. Схожа с RAID 2, с одной разницей, что вместо кода Хэмминга используется обычная для побитовых проверок контрольная сумма, построенная по принципу «исключающего ИЛИ». Для хранения контрольных сумм выделяется диск из массива, который отвечает высоким требования отказоустойчивости, так как обращения к этому диску происходят каждый раз, когда необходимо записать данные. За счет этого уменьшается избыточность данных, но одновременная обработка нескольких обращений к данным невозможна. Схема конфигурации представлена на рисунке 1.7.

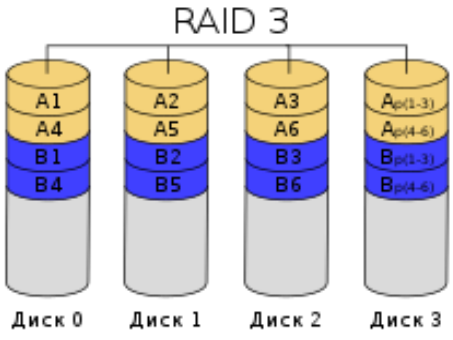


Рисунок 1.7 – RAID 3 [6]

RAID 4. Практическая полная копия RAID 3, отличие заключается лишь в увеличенном блоке записываемых данных. Появляется возможность параллельного доступа к данным для считывания, для записи по-прежнему данная возможность отсутствует. Схема изображена на рисунке 1.8.

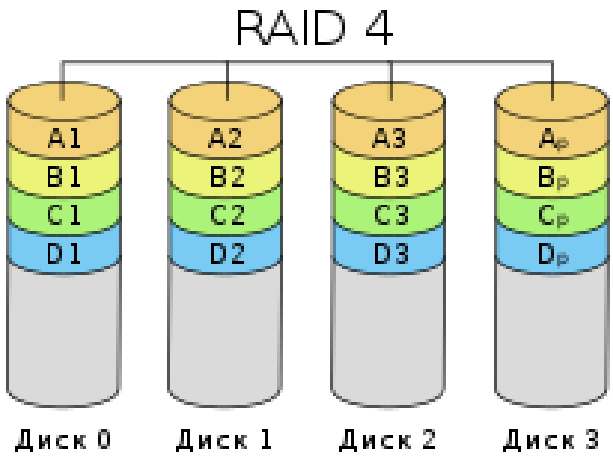


Рисунок 1.8 – RAID 4 [6]

RAID 5. Похож на RAID 3 и RAID 4, но контрольная сумма записывается не на отдельный диск, а по всему массиву, занимает примерно четвертую часть дискового пространства. Данное решение ускоряет операции считывания (так как не требуется обращаться к одному диску) и добавляет возможность параллельной записи данных. Схема данной конфигурации представлена на рисунке 1.9.

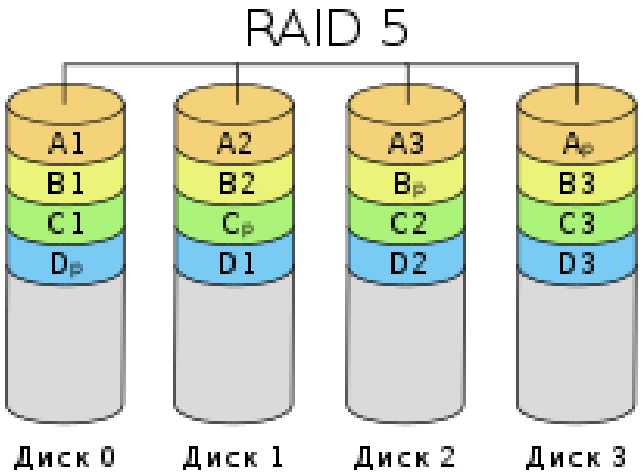


Рисунок 1.9 – RAID 5 [6]

Были рассмотрены основные базовые уровни RAID-массивов, также у этой технологии есть возможность комбинирования уровней. Ниже представлен один из популярных, который представляет интерес для данной работы.

RAID 10 – зеркалированный массив, данные в котором записывают последовательно на несколько дисков, также как в RAID 0. Но у этих дисков существует полная копия как в RAID 1. Таким образом, данный массив объединяет в себе высокую отказоустойчивость и производительность. На рисунке 1.10, представлена схема уровня. Сравнительные характеристики уровней RAID-массивов представлена в таблице 1.1 [7].

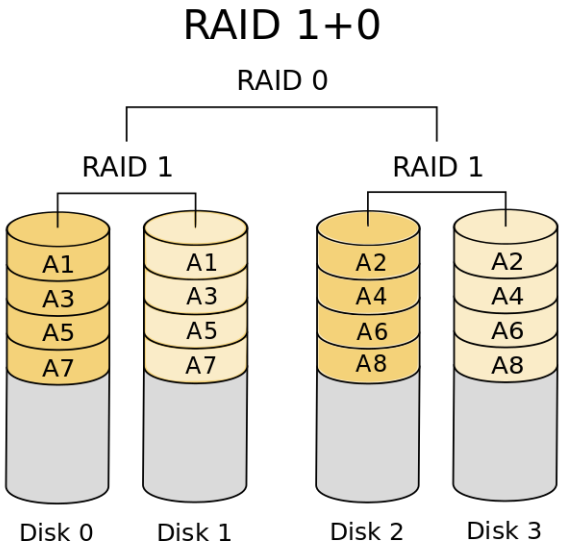


Рисунок 1.10 – RAID 10 [6]

Таблица 1.1 – Сравнение характеристик наиболее популярных уровней RAID.

Тип	Использование емкости, %	Производительность		Надежность	Мин. количество дисков	Макс. количество дисков
		чтения	записи			
RAID 0	100	Высокая	Высокая	Низкая	2	16
RAID 1	50	Высокая	Средняя	Высокая	2	2
RAID 10	50	Высокая	Высокая	Высокая	4	16
RAID 5	67-94	Высокая	Средняя	Средняя	3	16

Исходя из таблицы 1.1 следует, что схема RAID 10 превосходит другие схемы, но использует объем памяти лишь на половину.

1.1.4 Распределенные хранилища данных

Распределенное хранилище данных (Distributed Data Storage) – это инфраструктура, которая может быть разделена между множественными физическими серверами (узлами хранения данных), которые чаще всего, распределены между разными центрами обработки данных (ЦОД) [8].

Обычно представлены в виде программно-определяемой сети хранения (SDS) данных, с механизмами синхронизации и координирования данных между узлами, находящимся в кластере.

Software Defined Storage (SDS) – это архитектура объединения систем хранения данных, которое отделяет программного обеспечение от оборудования, на котором храниться информация. В отличии от NAS и SAN систем, организация сети данных происходит на программном уровне, для функционирования нет необходимости использования одного и того-же аппаратного оборудования. В одной сети могут находиться сервера с различной операционной системой, характеристиками [9].

Распределенные хранилища данных имеют такие преимущества [8, 10, 11]:

– простота масштабируемости. Осуществляется очень просто, для этого необходимо добавить новый узел в сеть;

– высокая надежность. Распределенные хранилища как правило хранят более одной копии данных на разных узлах кластера;

– низкая стоимость. На сегодня представлено большое количество бесплатного программного обеспечения для организации распределенных хранилищ, можно использовать практические любые сервера;

– высокая производительность. Ввиду того, что данные размещены на разных узлах одновременно, возможно получение данных с «ближайшего» узла к пользователю, параллельный доступ к данным.

Из минусов можно выделить сложность ввода в работу распределенного хранилища, также из-за возможности использования различных серверов, возможны трудности в обслуживании кластера.

Распределенные системы хранения данных как правило реализуют не все функции, которые перечислены выше. Для конкретного хранилища выбираются и реализуются только необходимые для решения поставленных задач, именно поэтому существует большое количество разнообразных хранилищ. В последнее большое развитие получили облачные хранилища данных.

Облачное хранилище данных (Cloud Storage) – это хранилище, данные в котором хранятся на многочисленных распределенных в сети серверах (узлах данных), предоставляемых в пользование клиентам, в основном, третьей стороной. Внутренняя структура серверов клиенту, в общем случае, не видна. Данные хранятся и обрабатываются в так называемом «облаке», которое представляет собой, с точки зрения клиента, один большой виртуальный сервер. Физически же такие серверы могут располагаться удалённо друг от друга географически [12].

Преимущества пользования облачными хранилищами следующие [13]:

– оплачивается только тот объем хранилища, который фактически используется;

– доступ к данным возможен с любого сервера, рабочей станции, которая подключена к интернету;

– отсутствует необходимость заниматься приобретением, поддержкой и обслуживанием собственной инфраструктуры хранилища;

– высокий уровень надежности хранимых данных;

– обеспечение целостности данных предоставляются провайдером «облачного» центра. Все возможные аппаратные сбои под ответственностью центра.

Ниже представлены недостатки облачных СХД. В самую первую очередь это цена использования данных хранилищ, они очень высоки. Второй момент – это обязательное подключение к интернету, если из-за сбоев он недоступен,

недоступно и облачное хранилище. Самый большой минус – это возможная утечка данных. Известно множество случаев, к примеру, из-за некорректных прав доступа к облачным хранилищам Amazon S3, «утекло» 6 миллионов записей о клиентах сотовой компании связи «Verizon» [12].

Существует различные типы облачных хранилищ [13]: публичное, приватное, гибридное.

Публичное облачное хранилище – это инфраструктура, предоставляемая в пользование большому количеству пользователей. Данная инфраструктура располагается и находится в собственности компании предоставляющей услуги.

Приватное (корпоративное) облачное хранилище – это хранилище, развернутое на собственной закрытой инфраструктуре. Значительно повышается уровень безопасности, но сильно увеличиваются затраты на поддержание инфраструктуры.

Гибридное облачное хранилище – это объединение преимуществ публичного и приватного облачного хранилища. Также как в случае приватного облачного хранилища существует собственная закрытая инфраструктура, но значительно меньшего размера. В тоже время существует доступ к общедоступному хранилищу. При таком подходе можно гарантировать высокий уровень безопасности для «важных» данных и хранить большие объемы данных в публичном облаке.

1.2 Обзор существующих облачных хранилищ данных

1.2.1 Amazon S3

Amazon Simple Storage Service (S3) – это облачное хранилище данных, разработанная компанией «Amazon», впервые появился в марте 2006 года. Является системой хранения данных объектного типа.

Общий объем хранимых данных и количество объектов не ограничены. Размер отдельных объектов Amazon S3 может составлять от 0 байт до 5 ТБ. Можно

хранить практически любые типы данных в любом формате. Amazon S3 предоставляет простой интерфейс веб-сервиса, который можно использовать для хранения и извлечения любых объемов данных в любое время из любого места в Интернете. Amazon S3 – это простое хранилище объектов на основе ключа. При хранении данных объектам назначается уникальный ключ, который может использоваться впоследствии для доступа к данным. Ключи могут иметь любые строковые значения; их можно создавать так, чтобы имитировать иерархические атрибуты.

Ниже перечислены основные концепции для общего понимания работы [14]:

– корзина (Bucket). Именованный контейнер для хранения объектов. Каждый объект обязательно должен находиться в корзине. Такое решение позволяет: логически разделять файлы по назначению, защищать файлы по уровням доступа, агрегировать информацию для отчетов;

– объект (Object). Неделимая единица хранения, состоит из данных и метаданных (данных о данных). Метаданные представлены в виде пар ключ-значения, которые описывают объект, к примеру – дата последнего обращению к объекту. У каждого объекта обязательно должен быть определен ключ;

– ключ (Key). Уникальный идентификатор объекта в корзине. Сочетание наименования корзины, ключа объекта и его версии – однозначно идентифицирует объект;

– регион (Region). Центр обработки данных (ЦОД), в котором буду храниться объекты. При создании корзины, можно выбрать конкретный регион. Объекты никогда не покидают своего региона, только если это не было сделано намерено.

Версионность объектов – механизм, который позволяет сохранять состояние объектов до внесения изменений. В качестве примера рассмотрен обычный случай редактирования файла, после обновления которого старое содержимое никуда не пропадает, доступ к нему доступен, обновления же получат просто новый идентификатор версии.

По всему миру создано большое количество центров обработки данных компании Amazon. На рисунке 1.11 изображена географическая карта мира с отмеченными ЦОД компании Amazon. Как видно из рисунка 1.11 – на территории Российской Федерации нет ни одного центра обработки данных. Ближайшие доступные, это европейские центры, расположенные во Франкфурте, Ирландии, Лондоне, Париже, Стокгольме и Милане.

Рисунок 1.11 – Центры обработки данных компании Amazon

Amazon S3 предоставляет множество классов хранилищ, которые решают разные поставленные задачи и конечно же, имеют различную стоимость обслуживания. Существуют следующие классы хранилищ [14]:

– Amazon S3 Standard. Предлагает высокую надежность, доступность и производительность объектного хранилища для хранения часто используемых данных. Обеспечивая низкую задержку и высокую пропускную способность;

– Amazon S3 Intelligent-Tiering. Создан для оптимизации расходов путем автоматического перемещения данных на наиболее экономичный уровень доступа без ущерба для производительности и роста операционных издержек;

– Amazon S3 Standard-Infrequent Access. Является идеальным выбором для хранения данных, доступ к которым осуществляется относительно редко, но при этом должен обеспечиваться быстро;

– Amazon S3 One Zone-Infrequent Access. Подходит для хранения данных, с редким доступом. В отличии от других классов, которые хранят данные как минимум в трех регионах, он хранит только в одной зоне доступности.

– Amazon S3 Glacier. Это безопасный, надежный и экономичный класс для архивации данных. Время извлечения данных может составлять от нескольких минут до нескольких часов;

– Amazon S3 Glacier Deep Archive. Это самый экономичный класс хранилища, с поддержкой долгосрочного хранения и цифровой архивации данных, доступ к которым запрашивается один-два раза в год. Он создан для клиентов, которые хранят наборы данных 7–10 лет или дольше для выполнения нормативных требований.

Каждый из описанных классов гарантирует 99.99% надежность хранения данных, а также 99.99% доступности в течении года [14].

Особое внимание стоит уделить хранилищу Amazon S3 Intelligent-Tiering, который имеет возможность адаптивного управлению временем доступа к объектам. На рисунке 1.12 изображена схема принципа работы.

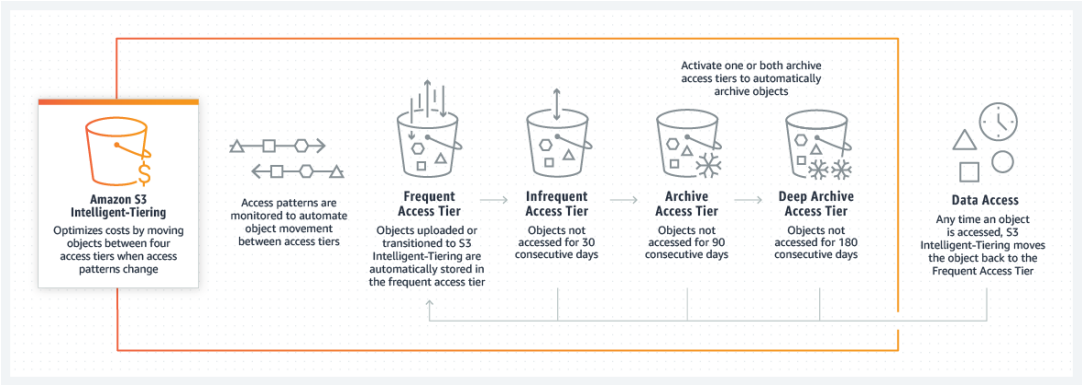


Рисунок 1.12 – Принцип работы хранилища Amazon S3 Intellegent-Tiering

Объекты, загруженные в данное хранилище, автоматически сохраняются на уровне для частого доступа. S3 Intelligent-Tiering работает путем отслеживания сценариев доступа и последующего перемещения объектов, доступ к которым не осуществлялся в течение 30 дней подряд, на уровень нечастого доступа. После

активации одного или обоих уровней доступа к архивным данным S3 Intelligent-Tiering перемещает объекты, доступ к которым не осуществлялся в течение 90 последовательных дней, на уровень Archive Access, а затем по истечении 180 последовательных дней – на уровень Deep Archive Access. Если после этого запрашивается доступ к этим перемещенным объектам, S3 Intelligent-Tiering перемещает их обратно на уровень для частого доступа.

Поддерживает такие функции как [12]: архивное хранение, дедупликация данных на уровне файлов, георепликация данных, управления доступом к данным, шифрования данных, кэширования, самовосстановления, балансировки нагрузки.

Для расчета стоимости использования хранилища, компания Amazon создала онлайн-калькулятор, который расположен на их сайте. Для того, чтобы узнать конкретную стоимость, необходимо заполнить требования к хранилищу. Сперва необходимо выбрать регион, для нас подходит Франкфурт (самый близкий к РФ), далее выбрать тип хранилища – S3 Intelligent-Tiering, определить объем хранилища – 20 ТБ. Amazon требует выбрать планируемое количество запросов на добавления данных и получения, для тестового примера выбрано значение в 1 000 000 запросов каждого типа. Результаты расчета изображены на рисунке 1.13.

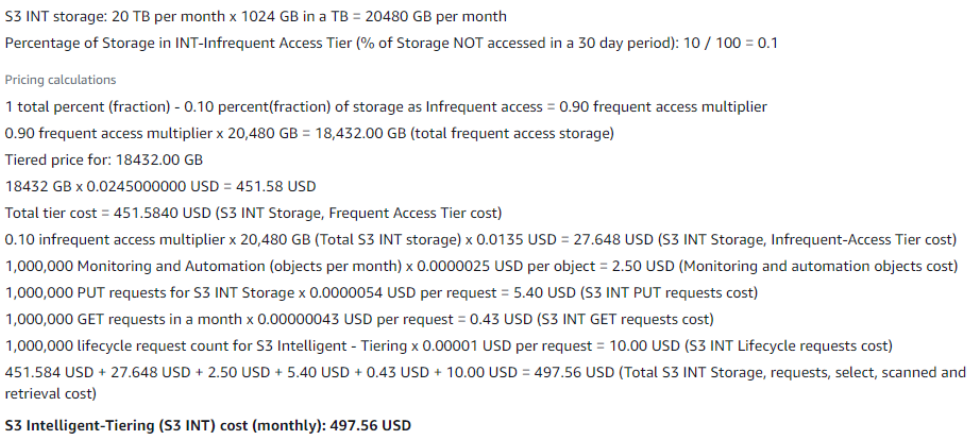

S3 INT storage: 20 TB per month x 1024 GB in a TB = 20480 GB per month
Percentage of Storage in INT-Infrequent Access Tier (% of Storage NOT accessed in a 30 day period): 10 / 100 = 0.1
Pricing calculations
1 total percent (fraction) - 0.10 percent(fraction) of storage as Infrequent access = 0.90 frequent access multiplier
0.90 frequent access multiplier x 20,480 GB = 18,432.00 GB (total frequent access storage)
Tiered price for: 18432.00 GB
18432 GB x 0.0245000000 USD = 451.58 USD
Total tier cost = 451.5840 USD (S3 INT Storage, Frequent Access Tier cost)
0.10 infrequent access multiplier x 20,480 GB (Total S3 INT storage) x 0.0135 USD = 27.648 USD (S3 INT Storage, Infrequent-Access Tier cost)
1,000,000 Monitoring and Automation (objects per month) x 0.0000025 USD per object = 2.50 USD (Monitoring and automation objects cost)
1,000,000 PUT requests for S3 INT Storage x 0.0000054 USD per request = 5.40 USD (S3 INT PUT requests cost)
1,000,000 GET requests in a month x 0.00000043 USD per request = 0.43 USD (S3 INT GET requests cost)
1,000,000 lifecycle request count for S3 Intelligent - Tiering x 0.00001 USD per request = 10.00 USD (S3 INT Lifecycle requests cost)
451.584 USD + 27.648 USD + 2.50 USD + 5.40 USD + 0.43 USD + 10.00 USD = 497.56 USD (Total S3 INT Storage, requests, select, scanned and retrieval cost)
S3 Intelligent-Tiering (S3 INT) cost (monthly): 497.56 USD

Рисунок 1.13 – Расчет стоимости месячного обслуживания хранилища для Amazon S3 Intelligent-Tiering

На момент написания данной работы, стоимость составила 497,56 USD (37 894,90 рублей).

1.2.2 Mail Cloud Storage

Mail Cloud Storage – является российским аналогом рассмотренного Amazon S3 [15]. Облачное хранилище (или S3) является объектного типа, SAN-архитектуры.

Данные хранятся в виде объектов в бакетах. Объект – это файл и любые дополнительные метаданные, описывающие файл. Чтобы сохранить файл его необходимо загрузить в бакет. Когда загружается файл как объект, можно установить разрешения доступа [16].

Бакеты – это контейнеры для объектов. Может создавать несколько. Доступ к каждому бакету можно контролировать, решая, кто может создавать, удалять и перечислять объекты в нем. Дополнительно можно просматривать журналы доступа для бакета и его объектов [16].

В отличии от Amazon S3, отсутствует понятие регионов, это обусловлено тем, что на сегодняшний день Mail Cloud Storage работает только на территории Российской Федерации. Заявлено о наличии 400 центров обработки данных (ЦОД) по всей территории РФ. Отсутствует версионность объектов.

Возможно хранить файлы размером от 0 байт до 32 ГБ. Общий размер хранилища не ограничен. Возможно хранить файлы любого формата. Для работы с хранилищем предоставлен простой веб-интерфейс, который имеет совместимость с веб-интерфейсом Amazon S3. Надежность хранилища составляет 99,5%.

Mail Cloud Storage отвечает всем требованиям закона «О хранении персональных данных» № 152-ФЗ [17].

Существуют следующие типы хранилищ [16]:

– Hotbox (горячее хранилище). Подходит для хранения данных для хранения часто используемых данных, обеспечивая низкую задержку и высокую пропускную способность;

– Icebox (холодное хранилище). Подходит для хранения файлов нечастого использования, таких как: архивы, резервные копии. Создано для хранения редко используемых данных, к которым при необходимости можно получить быстрый доступ;

– Backup. Используется для хранения файлов резервных копий виртуальных машин и баз данных.

Для расчета стоимости обслуживания используется онлайн-калькулятор, расположенный на сайте владельца облачного хранилища. Для расчета использован тот же пример, что и в случае Amazon S3. Результаты расчета представлены на рисунке 1.14.

Детализация расчета

Параметр	Значение	Цена за минуту (в том числе НДС)	Цена за месяц (в том числе НДС)
	Горячие данные		
Объем данных	2048 ГБ	0,083 ₽	3 584 ₽
Исходящий трафик	1000 ГБ	0,019 ₽	800 ₽
Запросы 1 типа	1000000 шт	0,007 ₽	295 ₽
Запросы 2 типа	1000000 шт	0,001 ₽	29,5 ₽
	Холодные данные		
Объем данных	18432 ГБ	0,683 ₽	29 491,2 ₽
Исходящий трафик	1000 ГБ	0,037 ₽	1 600 ₽
Запросы 1 типа	1000000 шт	0,007 ₽	295 ₽
Запросы 2 типа	1000000 шт	0,001 ₽	59 ₽
Итог за 1 сервис(ов)		**0,837 ₽**	**36 153,7 ₽**

Рисунок 1.14 – Расчет стоимости месячного обслуживания хранилища для Mail Cloud Storage

Под запросами 1 типа следует понимать запросы, на добавление и обновление данных. Запросы 2 типа – запросы на получение файлов из архива.

Также необходимо указать размер исходящего трафика. Исходящий трафик – ограничивает объем данных, который может быть загружен с хранилища за месяц. На момент написания данной работы цена месячного обслуживания составляет 36 153,70 рублей.

1.2.3 Yandex Object Storage

Сервис Yandex Object Storage – это универсальное масштабируемое решение для хранения данных. Оно подходит как для высоконагруженных сервисов, которым требуется надежный и быстрый доступ к данным, так и для проектов с невысокими требованиями к инфраструктуре хранения [18].

Yandex Object Storage можно отнести также к объектному типу хранилища с SAN-архитектурой. Основные понятия следующие:

– бакет: логическая сущность, которая помогает организовать хранение объектов;

– объект: данные произвольного формата, загруженные пользователем.

Можно сказать, что данное хранилище также является аналогом Amazon S3. В отличии от Mail Cloud Storage присутствует механизм версионности объектов. Понятие регионов отсутствует, все данные хранятся на территории Российской Федерации. Ко всем объектам и бакетам можно настроить режимы доступа.

Возможно хранить файлы размером от 0 байт до 5 ТБ. Общий размер хранилища не ограничен. Возможно хранить файлы независимо от их формата. Для работы с хранилищем предоставлен простой веб-интерфейс, который имеет совместимость с веб-интерфейсом Amazon S3.

Yandex Object Storage также отвечает всем требованиям закона «О хранении персональных данных» № 152-ФЗ [17].

У Yandex Object Storage представлено два типа хранилищ [18]:

– стандартное хранилище: предназначено для активной работы с объектами;

– холодное хранилище: предназначено для длительного хранения объектов с редкими запросами на чтение.

Для определения стоимости месячного обслуживания хранилища был использован калькулятор стоимости, представленный на сайте Yandex Cloud Storage. Воспользуемся тем же примером. Расчет месячного обслуживания представлен на рисунке 1.15.

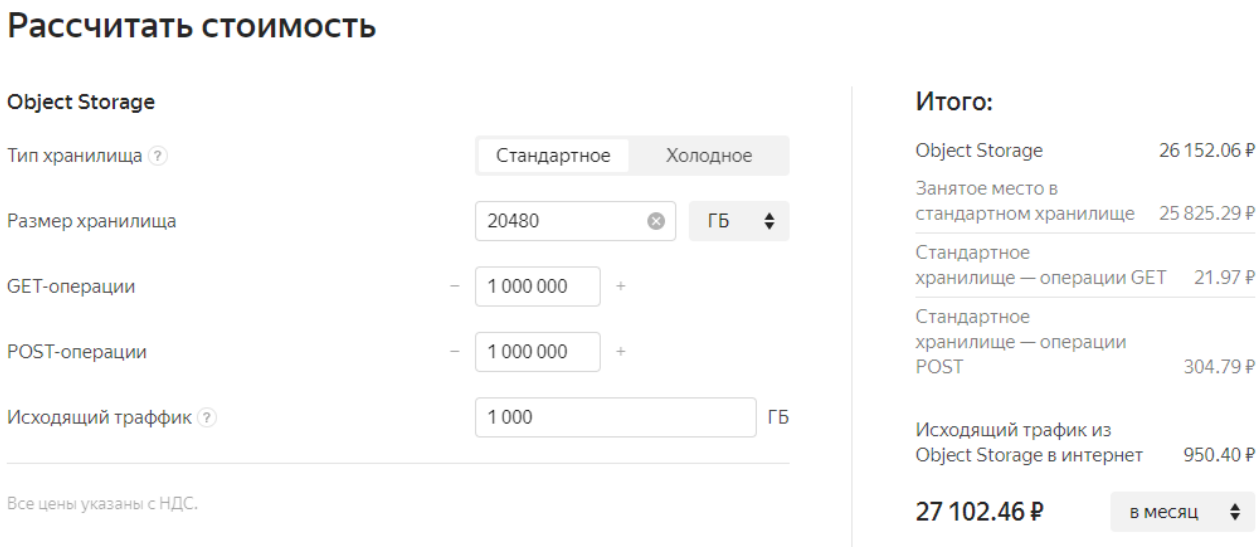


Рисунок 1.15 – Расчет стоимости месячного обслуживания для Yandex Cloud Storage

Таким образом, рассмотренные облачные хранилища обладают схожим функционалом для хранения данных большого объема. Каждое из решений предоставляет различные типы хранилищ. Самым интересным хранилищем – является решение от компании Amazon с «умным» менеджментом расположения данных на разных хранилищах с различной скоростью доступа.

1.3 Информационные процессы хранения данных в системах электронного документооборота

Электронный документооборот (ЭДО) – это система автоматизированных процессов обработки электронных документов, реализующая концепцию «безбумажного делопроизводства».

Система электронного документооборота (СЭД) – это специальное приложение, обеспечивающее участникам обмен электронными документами, имеющими юридическую значимость. Все системы электронного документооборота могут быть классифицированы по следующим признакам [19]:

– СЭД с развитыми системами хранения и поиска информации (электронные архивы);

– СЭД с развитыми системами маршрутизации, обеспечивающие движения документов по заданными маршрутам;

– СЭД с системой поддержки управления организацией и накопления знаний. Обычно системы данного типа сочетают в себе свойства двух предыдущих. Как правило используются в крупных компаниях и государственных структурах;

– СЭД с поддержкой совместной работы сотрудников. Основная цель таких систем – организация коллективной работы сотрудников, даже если они разделены территориально. Предоставляют возможность поиска информации, обсуждений и назначений встреч, включая реальные и виртуальные, а также сервисы хранения и публикации документов;

– СЭД с дополнительными сервисами: управление проектами, электронная почта, биллинг, сервис CRM (Customer Relationship Management).

Наиболее востребованными функциями СЭД являются [19]:

– хранение и поиск документов;

– поддержка делопроизводства;

– маршрутизация и контроль исполнения документов: составление маршрутов документов, поддержка действий во время маршрутов, уведомление сотрудников о поступлении нового документа, автоматический контроль сроков исполнения;

– составление аналитических отчетов, таких как отчет о текущей занятости, о выполнении работ по документам и о просроченных поручениях;

– обеспечение информационной безопасности, включая аутентификацию пользователей, поддержку электронной цифровой подписи, шифрования документов и писем, аудит работы в системе.

Сегодня существует большое количество СЭД, которые решают, как индивидуальные задачи отдельно взятых предприятий, так и задачи на уровне целого государства. Целью данной работы является оптимизация хранилища данных для оператора электронного документооборота.

Оператор электронного документооборота – организация, обладающая достаточными технологическими, кадровыми и правовыми возможностями для обеспечения юридически значимого документооборота счетов-фактур в электронном виде с использованием электронной подписи [20].

В первую очередь, операторы оказывают услуги по организации обмена электронными документами по сделкам, такими, как договоры, первичные бухгалтерские документы, счета-фактуры, между организациями, являющимися юридическими лицами, индивидуальными предпринимателям, государственными органами.

Функции оператора ЭДО закреплены в Приказе ФНС РФ от 23.10.2020 № ЕД-7-26/775@ [20]. Основными функциями в рамках электронного обмена является:

– соответствия требованиям норм законодательства РФ, регулирующих область электронного взаимодействия с использованием электронной подписи;

– организация обмена счетами-фактурами в электронном виде между контрагентами;

– организация обмена документами по сделкам между организациями (договоры, первичные бухгалтерские документы, счета-фактуры);

– фиксация дат выставления и получения документов;

– техническая поддержка программного обеспечения, реализующего электронный документооборот клиента;

– учет компаний в качестве участников электронного взаимодействия и присвоения им соответствующих идентификационных реквизитов;

– доведение реквизитов идентификации участников обмена до налоговых органов.

На рисунке 1.16 представлена обще-типовая схема электронного документооборота.

Рисунок 1.16 – Обще-типовая схема электронного документооборота

Операторы ЭДО предлагают целый спектр услуг, которые так или иначе связаны с обменными процессами. Пользуясь данными услугами, компании получают видимые преимущества в работе. К таким услугам можно отнести:

– обмен любыми документами по сделкам в электронном виде между контрагентами – это и договоры, товарные накладные, акты и прочие документы;

– различные варианты интеграционных решений позволяют клиентам отправлять и получать документы, не выходя из привычной в работе учетной, информационной или иной системы, не запуская при этом дополнительных программ и приложений;

– хранение электронных документов, переданных через сервис, в соответствии со всеми требованиями действующего законодательства РФ;

– квалифицированная поддержка, мгновенно реагирующая на запросы пользователей;

– отслеживания изменений нормативно правовых актов РФ, регулирующих область электронного взаимодействия и своевременное информирования об этом пользователей системы.

1.3.1 Требования к операторам электронного документооборота

Согласно Приказу ФНС РФ от 23.10.2020 № ЕД-7-26/775@ [20] определены следующие требования к хранению документов операторами ЭДО:

– хранение следующих видов документов: договоры, накладные, платежки, регистры бухучета – подлежат хранению на протяжении 5 лет. Счета-фактуры – 4 года. Бухгалтерская отчетность хранится до тех пор, пока оператор ЭДО осуществляет деятельность (не закрылся);

– документы, как минимум, должны быть сохранены в двух экземплярах на разных физических носителях информации;

– обеспечение надежного режима хранения документов, так, чтобы они не были утрачены, несанкционированным образом распространены, уничтожены или искажены;

– соблюдение Закона № 152-ФЗ «О персональных данных» [17]. Документы должны храниться на серверах, которые находятся на территории Российской Федерации. Допускается арендование хранилищ данных у сторонней организации, при наличии договора с указанной организацией, в которых описаны условия пользования и зоны ответственности каждой из сторон.

1.3.2 Описание процесса подачи налоговой отчетности в рамках стандарта IDEF0

При помощи методологии функционального моделирования процессов IDEF0 рассмотрены процессы, происходящие в системе электронного документооборота при подаче налоговой отчетности.

На рисунке 1.17 представлена IDEF0-диаграмма основного процесса, построенная на основе описания процессов модели, приведенных в таблице 1.2.

На вход системы поступает первичный бухгалтерский документ, ответ ведомства, запрос на получение истории документооборота на выходе происходит отправка транспортного пакета в налоговый орган, а также история документооборота.

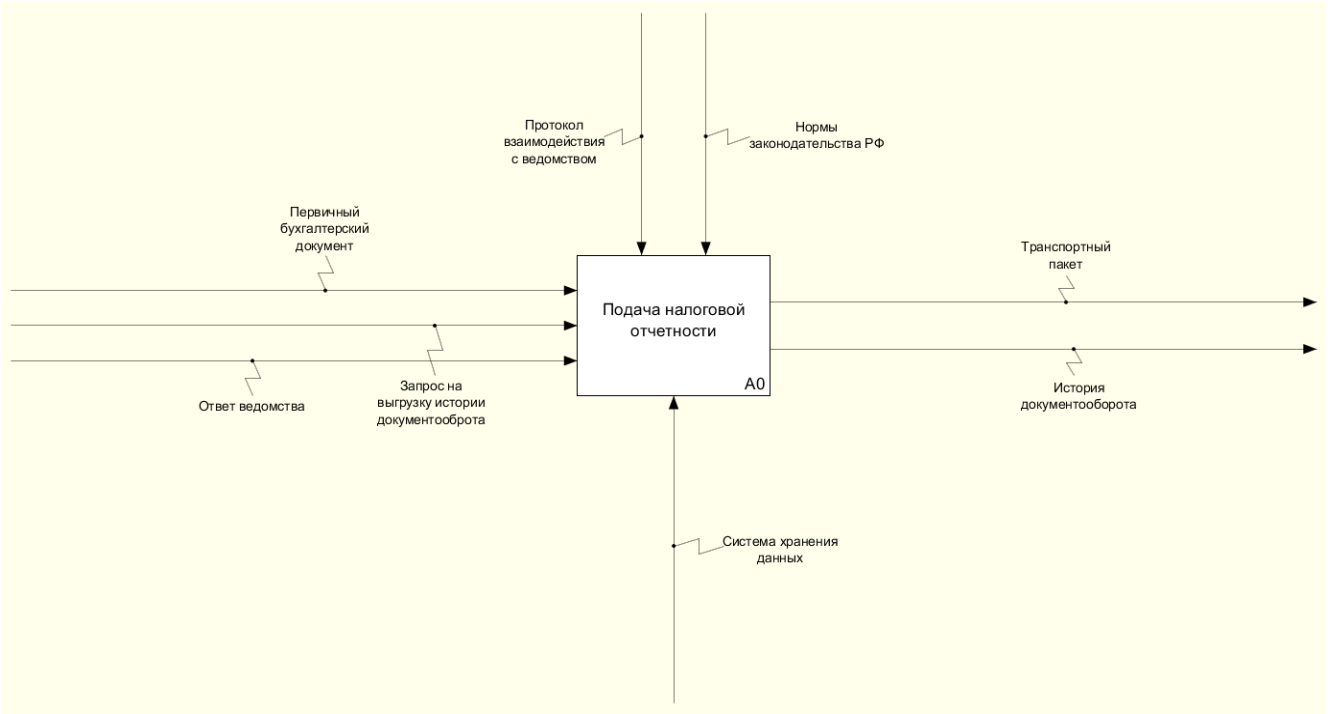


Рисунок 1.17 – IDEF0-диаграмма основного процесса

Таблица 1.2 – Описание процессов оператора ЭДО основного процесса

Шифр	Название процесса	Входные данные	Управляющие данные	Механизм	Результат процесса
A0	Подача налоговой отчетности	Первичный бухгалтерский документ	Протокол взаимодействия с ведомством	Система хранения данных	История документооборота
		Ответ ведомства			
		Запрос на выгрузку истории документооборота	Нормы законодательства РФ		Транспортный пакет

Как следует из рисунка 1.17 процесс подачи налоговой отчетности может быть декомпозирован на следующие процессы:

– создания нового документооборота;

– перевода документооборота на следующий этап;

– формирования истории документооборота по требованию.

Результат декомпозиции основного процесса представлена на рисунке 1.18. Описание процессов декомпозиции содержится в таблице 1.3.

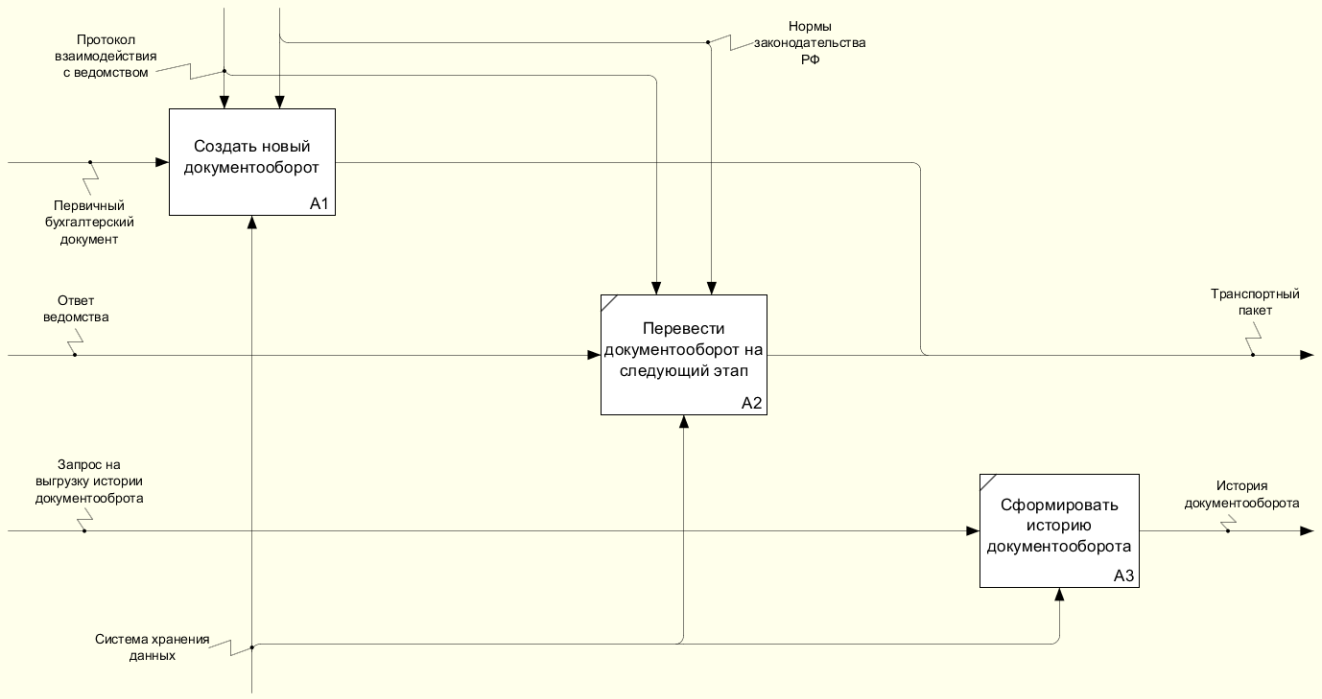


Рисунок 1.18 – IDEF0-диаграмма декомпозиции основного процесса

Таблица 1.3 – Описание декомпозиции основного процесса

Шифр	Название процесса	Входные данные	Управляющие данные	Механизм	Результат процесса
A1	Создать новый документооборот	Первичный бухгалтерский документ	Протокол взаимодействия с ведомством Нормы законодательства РФ	Система хранения данных	Транспортный пакет

А2	Перевести документооборот на следующий этап	Ответ ведомства	Протокол взаимодействия с ведомством Нормы законодательства РФ	Система хранения данных	Транспортный пакет

Продолжение таблицы 1.3

Шифр	Название процесса	Входные данные	Управляющие данные	Механизм	Результат процесса
А3	Сформировать историю документооборота	Запрос на выгрузку истории документооборота		Система хранения данных	История документооборота

При загрузке в систему первичного бухгалтерского документа, создается новый документооборот. Документооборот – это регламентированный процесс обмена документами с налоговым органом. В целях безопасности, каждый передаваемый или получаемый документ дополнительно подписывается электронно-цифровой подписью (ЭЦП), который однозначно идентифицирует отправителя документа. Также в целях безопасности перед отправкой документы шифруются открытым ключом получателя. Транспортный пакет – контейнер для документов и ЭЦП.

В общем случае документооборот состоит из следующих этапов:

1. Первичная валидация отчета (проверка на наличие синтаксических ошибок заполнения отчета).

2. Создание ЭЦП для отчета, оборачивание отчета в транспортный пакет, шифрование транспортного пакета, отправка транспортного пакета в ведомство.

3. Получение ответа от ведомства, расшифровка ответа, получение документа подтверждающий получение отчета ведомством и ЭЦП к нему.

4. Получение ответа от ведомства, расшифровка ответа, получение документа подтверждающего прием отчета ведомством или (в случае логических ошибок) документа отказа приема отчета ведомством и ЭЦП к нему.

5. Формирование документа подтверждения получения ответа от ведомства.

6. Создание ЭЦП для сформированного документа, создание транспортного пакета, шифрование транспортного пакета, отправка транспортного пакета в ведомство.

7. Получение ответа от ведомства, расшифровка ответа, получение документа, подтверждающий успешное либо неуспешное завершение документооборота и ЭЦП к нему.

Переход между этапами документооборота занимает длительное время от одного часа до двух суток. Каждый сформированный, полученный от ведомства документ и данные ЭЦП в рамках документооборота должны быть сохранены и храниться длительное время. История документооборота включает в себя все документы документооборота, все ЭЦП к документам и визуализацию состояния документооборота.

Исходя из IDEF0-модели процесса подачи налоговой отчетности – следует, что операторы электронного документооборота, используют систему хранения данных на каждом этапе функционирования. Также следует, что к некоторым документам, а именно тем, у которых документооборот не завершен – обращения происходят довольно часто. К документам, у которых документооборот завершен, в свою очередь, обращения происходят в «исключительных» ситуациях.

1.3.3 Особенности данных в системе оператора электронного документооборота

В системе оператора электронного документооборота основным типом данных являются – файлы в формате XML (eXtensible Markup Language), небольшого размера. В среднем файл занимает от 2 до 8 Кбайт информации. Формат XML является стандартом для описания первичных бухгалтерских отчетов и счет-фактур. Все сопутствующие документы документооборота также используют формат XML. Данные ЭЦП имеют формат SIG и занимают 2 Кбайт информации.

Также операторы ЭДО предоставляют возможность обмена неформализованными документами (произвольными документами) между абонентами системы. Как правило это файлы в формате: .pdf, .docx, .xlsx, .jpeg, .png, .tiff. Максимально-допустимый размер таких файлов ограничен 100 МБ информации.

1.4 Обзор методов оптимизации хранения данных

Ниже определены критерии, по которым будет проводиться оптимизация. Исходя из требований к операторам электронного документооборота, для системы хранения данных важны следующие критерии: объем данных, надежность, производительность и безопасность.

1.4.1 Методы оптимизации объема данных

Целью оптимизации по критерию объема данных является увеличение объема хранимых данных и уменьшение занимаемого дискового пространства на запоминающих устройствах. К таким методам можно отнести: методы сжатия данных, методы дедупликации данных.

Сжатие данных – алгоритмическое преобразование данных, производимое с целью уменьшения занимаемого ими объема. Сжатие основано на устранении избыточности, содержащийся в данных. Все методы сжатия делятся на два основных класса: сжатие без потерь, сжатие с потерями. Для разрабатываемой системы интерес представляют только методы сжатия без потерь.

Существует большое количество алгоритмов сжатия без потерь, выделим два самых популярных: алгоритм Хаффмана, который основан на энтропийном сжатии; алгоритм LZW (назван в честь создателей Lempel, Ziv, Welch), основанный на словарном методе. Согласно произведенному сравнению алгоритмов [21] следует, что для текстовых данных больший коэффициент сжатия имеет алгоритм LZW и он в среднем уменьшает объем данных в 4 раза, в свою очередь алгоритм Хаффмана в 2 раза.

Сам по себе процесс сжатия вне зависимости от выбранного алгоритма – ресурсоемкий процесс, требующий большого количества процессорного времени и значительный объем оперативной памяти, под промежуточные данные. Для наших целей, сжатия необходимо производить перед сохранением, а восстановление каждый раз, когда происходит обращение к данным.

Дедупликация данных – специализированный метод сжатия данных, использующий в качестве алгоритма сжатия исключение, дублирующийся копий повторяющихся данных. Методы делятся на: дедупликацию на уровне файлов, нацелена на устранение копий файлов; дедупликацию на уровне блоков, нацелена на устранение копий блоков фиксированного размера (chunks). В отличии от методов сжатия, поиск избыточности производится по хранилищу в целом. При нахождении копии, данные заменяются на ссылку уже сохраненных данных.

При добавлении новых данных в хранилище, необходимо производить поиск, дублирующийся копий повторяющихся данных и производить их замену на ссылку, данная операция требует больше ресурсов, чем алгоритмы сжатия, но операция выполняется лишь при добавлении данных. В свою очередь дедупликация на уровне блоков производится дольше, чем дедупликация на уровне файлов, но позволяет сэкономить больше места. Методы дедупликации данных позволяют уменьшить объем занимаемого пространства, в некоторых предметных областях до 95%, но на практике может достигать только несколько процентов [22].

1.4.2 Методы оптимизации надежности хранения данных

Целью оптимизации по критерию надежности хранения данных является уменьшение риска потери данных, в случаи нештатной ситуации (отказ запоминающего устройства, стихийные и техногенные бедствия). К методам оптимизации надежности хранения данных можно отнести: репликацию данных, резервного копирование данных, помехоустойчивое кодирования данных.

Репликация – механизм синхронизации нескольких копий данных, между различными запоминающими устройствами. Для определения количества копий данных, используется коэффициент репликации. Таким образом в случае отказа

одного из ЗУ, данные не теряются, а доступны на одном или нескольких других устройствах. Репликация может быть синхронной или асинхронной.

Синхронная репликация подразумевает синхронное добавление данных на все запоминающие устройства (в соответствии с коэффициентом репликации).

Асинхронная репликация подразумевает распространение данных спустя некоторое время. До того момента как копии данных сохраняться на всех необходимых ЗУ, нельзя гарантировать высокую надежность данных.

Репликацию можно производить на аппаратном уровне (RAID 1, RAID 10), и на программном уровне (NAS, SAN, SDS системы).

Резервное копирование – процесс создания копии данных на ЗУ, предназначенном для восстановления данных в случае нештатной ситуации. Параметры резервного копирования являются: RPO (Recovery Point Objective), RTO (Recovery Time Objective). RPO определяет точку восстановления (состояние хранилища в прошлом), RTO определяет время, необходимое на восстановление. При использовании данных методов, в отличии от методов репликации, восстановление данных происходит с задержкой по времени (RTO) и некоторые данные могут быть утраченными (RPO).

Существуют следующие виды резервного копирования:

– полное резервное копирование (full backup) – подразумевает создание полной копии всех данных;

– дифференциальное резервное копирование (differential backup) – копируется только те данные, которые были изменены с момента последнего полного резервного копирования;

– инкрементного резервного копирования (incremental backup) – такой же, как и дифференциальное резервное копирование, но данные, которые изменились или добавились не замещают старые данные.

Все перечисленные виды резервного копирования производятся через определенные промежутки времени, могут происходить как каждый час, так и каждый месяц, от этого напрямую зависит RPO.

Помехоустойчивое кодирование данных – процесс добавления к данным избыточной информации (контрольное число) с помощью которой, возможно восстановление данных при нештатных ситуациях. При использовании такого кодирования, добавляемые данные, делятся на блоки фиксированной длины, добавляется избыточная информация и затем блоки распределяется по различным запоминающим устройствам. При обращении к данным, блоки собираются обратно, в случае если по какой-либо причине, блок недоступен, то он восстанавливается с помощью избыточной информации.

Существуют большое количество различных видов алгоритмов помехоустойчивого кодирования, такие как: LRC-коды (коды с локальной четностью), XOR-коды, RS-коды (коды Рида-Соломона). Методы помехоустойчивого кодирования широко применяются в RAID-массивах (RAID 2, RAID 3, RAID 4, RAID 5).

При использовании помехоустойчивого кодирования использование объемов данных достигает 64-96%, в то время как при репликации данных он равняется 50% постоянно [6].

1.4.3 Методы оптимизации производительности систем хранений данных

Целью оптимизации по критерию производительности является уменьшение времени доступа к данным, добавления новых данных. К таким методам относятся следующие: кэширование данных, шардинг данных.

Кэш – это память с высокой скоростью доступа. Кэширование данных – это процесс размещения данных в кэше. Характеризуется понятием «уровень попаданий», а именно насколько часто данные обнаруживаются в кэше. Чем выше «уровень попаданий» тем меньше время доступа к данным. В силу того, что запоминающие устройства с высокой скорость не обладают большими объемами памяти, были разработаны различные алгоритмы вытеснения данных (алгоритмы кэширования), которые призваны освобождать память. Существуют следующие алгоритмы [23]:

– LRU (Least Recently Used) – в первую очередь, вытесняется неиспользованные данные дольше всех;

– MRU (Most Recently Used) – в первую очередь вытесняется последние использованные данные;

– LFU (Least-Frequently Used) – вытесняются те данные, к которым реже всего обращаются;

– SLRU (Segmented LRU).

Для алгоритма SLRU кэш делится на несколько (от двух до трёх) упорядоченных сегментов данных, данные добавляются в первый сегмент, если к данным обратились, то данные перемещаются в следующий сегмент. Вытеснение происходит также с первого сегмента и далее по списку сегментов. При такой организации кэша, наиболее используемые данные как можно дольше остаются в кэше.

Шардинг – процесс равномерного распределения данных между узлами распределенной системы хранения данных. В отличии от репликации, распределяются не копии данных. При таком подходе увеличение производительности происходит за счет распределении обращений к данным на разные узлы распределенной СХД. Как правило шардинг различается по двум способам распределения данных:

– равномерное распределение;

– географическое распределение – распределение данных с целью уменьшения дистанции между пользователем и данным (данные пользователя сохраняются в «ближайший» узел).

1.4.4 Методы оптимизации безопасности данных

Целью оптимизации, с точки зрения безопасности данных, является уменьшение рисков несанкционированного доступа к данным. К таким методам относятся: методы шифрования данных, введение аудита доступа к данным, ограничение сетевого доступа.

Шифрование – обратимое преобразование данных в целях сокрытия данных от несанкционированного доступа. Если все-таки по каким-либо причинам будет осуществлен несанкционированный доступ, то зашифрованные данные не несут какой-либо полезной информации. Для шифрования необходим ключ, который утверждает выбор конкретного преобразования из совокупности возможных. При добавлении данных – данные шифруются и сохраняются уже в зашифрованном виде. При извлечении данных, данные необходимо расшифровать. Операции шифрования и расшифровки очень ресурсоемкие.

Различают симметричное и асимметричное (с открытым ключом) шифрование. Операции шифрования и расшифровки происходят быстрее при использовании симметричных алгоритмов.

При симметричном – один и тот же ключ используется как для шифрования, так и для расшифровки данных. Самый распространенный алгоритм симметричного шифрование – AES (Advanced Encryption S).

При ассиметричном, существует два ключа – открытый (который может свободно распространяться) и закрытый (который должен быть хорошо спрятан). При шифровании используется открытый ключ, при расшифровке – закрытый. Самый распространенный алгоритм ассиметричного шифрования – RSA (в честь создателей Rivest, Shamir, Adleman).

Аудит доступа – сбор всевозможной информации о совершенных действий над данными, в нашем случае действиями с данными. Данный метод косвенно влияет на безопасность данных, так как в отличии от шифрования не производит никаких действий над данным. Аудит доступа служит для выявления попыток несанкционированного доступа. Различаются по способу сбора действии и типу информации. Для хранения необходимо хранение действий над каждым файлом.

Методы ограничения сетевого доступа направлены на ограничение доступа к системе хранения данных по сети. Различают следующие методы: изоляции системы, ограничения по спискам доступа (белым и черным).

Изоляция системы подразумевает закрытие доступа из глобальной сети, при таком методе какое-либо внешнее взаимодействие невозможно. Ограничения по

белым спискам – предоставляет возможность работы с СХД только конкретным IP-адресам. В свою очередь ограничения по черным спискам запрещает работу с СХД конкретным IP-адресам.

1.4.5 Влияние методов оптимизации на критерии

Таким образом, каждый из рассмотренных методов, помимо позитивного влияния на удовлетворения целей определенного критерия, также имеет влияния и на другие критерии. Влияние каждого рассмотренного метода на каждый из критериев, качественно оценены и сведены в таблице 1.4.

Таблица 1.4 – Влияние методов оптимизации хранения данных на критерии

Метод	Влияние на объем данных	Влияние на надежность данных	Влияние на производительность	Влияние на безопасность данных
Сжатия данных	Очень положительно	Не влияет	Очень отрицательно	Не влияет
Дедупликации данных на уровне файлов	Положительно	Не влияет	Отрицательно	Не влияет
Дедупликации данных на уровне блоков	Очень положительно	Не влияет	Отрицательно	Не влияет
Синхронной репликации данных	Очень отрицательно	Очень положительно	Очень отрицательно	Не влияет
Асинхронной репликации данных	Очень отрицательно	Очень положительно	Отрицательно	Не влияет
Резервного копирования данных	Очень отрицательно	Положительно	Не влияет	Не влияет
Помехоустойчивого кодирования данных	Отрицательно	Положительно	Отрицательно	Не влияет

Кэширования данных	Отрицательно	Положительно	Очень положительно	Не влияет
Шардинга данных	Не влияет	Отрицательно	Положительно	Не влияет

Продолжение таблицы 1.4

Метод	Влияние на объем данных	Влияние на надежность данных	Влияние на производительность	Влияние на безопасность данных
Симметричного шифрования данных	Не влияет	Не влияет	Отрицательно	Очень положительно
Асимметричного шифрования данных	Не влияет	Не влияет	Очень отрицательно	Очень положительно
Аудита доступа	Отрицательно	Не влияет	Не влияет	Положительно
Ограничения сетевого доступа	Не влияет	Не влияет	Не влияет	Очень положительно

Анализируя таблицу 1.4 можно сделать вывод, что для критерия производительности – очень положительное влияние имеет только метод кэширования данных.

Аналогичная ситуация имеется среди методов оптимизирующие критерий безопасности. Метод ограничения сетевого доступа очень положительно влияет на безопасность и никак более на другие критерии. Также очень положительно на безопасность влияют методы шифрования данных, но в отличии от метода ограничения сетевого доступа имеют отрицательное влияние на производительность.

Для методов оптимизации хранения данных для критериев объема данных и надежности хранения, противоположная ситуация, выбор метода затруднителен. Для выбора будет использован вариантный анализ на основе метода иерархий и нечетких критериев.

Можно выделить следующие критерии для оценки альтернатив решений:

q_j^1 – критерий объема данных;

q_j^2 – критерий надежности хранения;

q_j^3 – критерий производительности.

Для оптимизации хранения данных по критерию объема данных имеются следующие альтернативы:

c_1 – метод сжатия данных;

c_2– метод дедупликация данных на уровне файлов;

c_3– метод дедупликации данных на уровне блоков.

Вариантный анализ альтернатив на основе метода иерархий и нечетких критериев представляет собой сравнение вариантов при помощи следующих высказываний (1.1 – 1.2):

$$\text{Критерий } q_j^1: \begin{cases} \text{существенное преимущество } c_1 \text{ над } c_2, \\ \text{явное преимущество } c_1 \text{ над } c_3, \\ \text{существенное преимущество } c_3 \text{ над } c_2. \end{cases} \quad (1.1)$$

$$\text{Критерий } q_j^3: \begin{cases} \text{явное преимущество } c_2 \text{ над } c_1 \text{ и } c_3, \\ \text{явное преимщуество } c_3 \text{ над } c_1. \end{cases} \quad (1.2)$$

По критерию q_j^2 все альтернативы равнозначны. Данные высказывания соответствуют следующим матрицам парных сравнений на основе 9-балльной шкала Саати (1.3 – 1.4):

$$A(q_j^1) = \begin{array}{c} \\ c_1 \\ c_2 \\ c_3 \end{array} \begin{array}{c} \begin{array}{ccc} c_1 & c_2 & c_3 \end{array} \\ \begin{pmatrix} 1 & 9 & 7 \\ 1/9 & 1 & 1/5 \\ 1/7 & 5 & 1 \end{pmatrix} \end{array}. \quad (1.3)$$

$$A(q_j^3) = \begin{array}{c} \\ c_1 \\ c_2 \\ c_3 \end{array} \begin{array}{c} \begin{array}{ccc} c_1 & c_2 & c_3 \end{array} \\ \begin{pmatrix} 1 & 1/5 & 1/3 \\ 5 & 1 & 3 \\ 3 & 1/3 & 1 \end{pmatrix} \end{array}. \quad (1.4)$$

На основе матриц парных сравнений были определены нечеткие множества (1.5 – 1.6):

$$q_j^{\square 1} = \left\{ \frac{0.7}{c_1}, \frac{0.05}{c_2}, \frac{0.25}{c_3} \right\}. \quad (1.5)$$

$$q_j^{\square 3} = \left\{ \frac{0.1}{c_1}, \frac{0.61}{c_2}, \frac{0.29}{c_3} \right\}. \quad (1.6)$$

На основе принципа Беллмана-Заде было определено нечеткое множество, с помощью которого можно определить наилучшую альтернативу (1.7):

$$D = \left\{ \frac{0.1}{c_1}, \frac{0.05}{c_2}, \frac{0.25}{c_3} \right\}. \quad (1.7)$$

Отметим, что, исходя из выражения (1.5), метод сжатия данных наилучшим для оптимизации объема данных и он подходит для систем хранения данных с невысокими требованиями производительности. В свою очередь метод дедупликации данных на уровне файлов, исходя из выражения (1.7), подойдет для высокопроизводительных хранилищ. Метод дедупликации данных на уровне блоков – является золотой серединой между описанными выше методами, что и показывает множество *D*.

Далее аналогичным способом будет выбран метод оптимизации надежности данных. Можно выделить ранее определенные критерии, получаются следующие альтернативы:

r_1– метод синхронной репликации данных;

r_2– метод асинхронной репликации данных;

r_3– метод резервного копирования данных;

r_4– метод помехоустойчивого кодирования.

Сравнение альтернатив по критериям выполняется при помощи следующих высказываний (1.8 – 1.10).

$$\text{Критерий } q_j^1 : \{\text{явное преимущество } r_4 \text{ над } r_1, r_2, r_3. \tag{1.8}$$

$$\text{Критерий } q_j^2 : \begin{cases} \text{слабое преимущество } r_1 \text{ над } r_2, \\ \text{существенное преимущество } r_1 \text{ над } r_3, \\ \text{явное преимущество } r_2 \text{ над } r_3, \\ \text{слабое преимущество } r_4 \text{ над } r_2, \\ \text{явное преимущество } r_4 \text{ над } r_3. \end{cases} \tag{1.9}$$

$$\text{Критерий } q_j^3 : \begin{cases} \text{явное преимущество } r_1 \text{ над } r_4, \\ \text{явное преимущество } r_2 \text{ над } r_1, \\ \text{существенное преимущество } r_2 \text{ над } r_4, \\ \text{слабое преимущество } r_3 \text{ над } r_1, r_2, r_4. \end{cases} \tag{1.10}$$

Данные высказывания соответствуют следующим матрицам парных сравнений (1.11 – 1.13).

$$A(q_j^1) = \begin{array}{c} \\ r_1 \\ r_2 \\ r_3 \\ r_4 \end{array} \begin{array}{c} \begin{array}{cccc} r_1 & r_2 & r_3 & r_4 \end{array} \\ \begin{pmatrix} 1 & 1 & 1 & 1/5 \\ 1 & 1 & 1 & 1/5 \\ 1 & 1 & 1 & 1/5 \\ 5 & 5 & 5 & 1 \end{pmatrix} \end{array}. \tag{1.11}$$

$$A(q_j^2) = \begin{array}{c} \\ r_1 \\ r_2 \\ r_3 \\ r_4 \end{array} \begin{array}{c} \begin{array}{cccc} r_1 & r_2 & r_3 & r_4 \end{array} \\ \begin{pmatrix} 1 & 3 & 7 & 1 \\ 1/3 & 1 & 5 & 1/3 \\ 1/7 & 1/5 & 1 & 1/5 \\ 1 & 3 & 5 & 1 \end{pmatrix} \end{array}. \quad (1.12)$$

$$A(q_j^3) = \begin{array}{c} \\ r_1 \\ r_2 \\ r_3 \\ r_4 \end{array} \begin{array}{c} \begin{array}{cccc} r_1 & r_2 & r_3 & r_4 \end{array} \\ \begin{pmatrix} 1 & 1/5 & 1/3 & 5 \\ 5 & 1 & 1/3 & 7 \\ 3 & 3 & 1 & 3 \\ 1/5 & 1/7 & 1/3 & 1 \end{pmatrix} \end{array}. \quad (1.13)$$

На основе матриц парных сравнения были определены следующие нечеткие множества (1.14 – 1.16).

$$q_j^{\square 1} = \left\{ \frac{0.16}{r_1}, \frac{0.16}{r_2}, \frac{0.16}{r_3}, \frac{0.52}{r_4} \right\}. \quad (1.14)$$

$$q_j^{\square 2} = \left\{ \frac{0.4}{r_1}, \frac{0.22}{r_2}, \frac{0.05}{r_3}, \frac{0.33}{r_4} \right\}. \quad (1.15)$$

$$q_j^{\square 3} = \left\{ \frac{0.21}{r_1}, \frac{0.42}{r_2}, \frac{0.32}{r_3}, \frac{0.05}{r_4} \right\}. \quad (1.16)$$

С помощью принципа Беллмана-Заде определено нечеткое множество, на основе которого можно определить наилучшую альтернативу для методов оптимизации надежности хранения данных (1.17).

$$D = \left\{ \frac{0.16}{r_1}, \frac{0.16}{r_2}, \frac{0.05}{r_3}, \frac{0.05}{r_4} \right\}. \quad (1.17)$$

Исходя из множества *D* выходит, что синхронный и асинхронный метод репликации данных являются лучшими методами оптимизации надежности хранения данных. Стоит отметить, что синхронный метод подойдет лучше системам хранения данных, для которых критерием производительности можно пренебречь. Для высокопроизводительных систем – следует выбирать метод асинхронной репликации данных.

1.5 Постановка задачи создания оптимизированного хранилища данных для оператора электронного документооборота

Исходя их требований, представляемых со стороны законодательства, к оператору электронного оборота, нельзя использовать существующие облачные хранилища данных. В силу того, что Amazon S3 не имеет центров обработки данных на территории РФ, а отечественные аналоги хоть и находятся на территории РФ, но не предоставляют возможности заключения договора, требуемого законом. Таким образом встает необходимость создания собственной системы хранения данных, удовлетворяющей требования системы оператора электронного документооборота.

Разработка оптимизированной системы хранения данных для оператора электронного документооборота включает в себя следующие задачи:

1. Разработка архитектуры системы хранения данных.
2. Проектирование системы хранения данных.
3. Определение критериев оценки эффективности системы хранения данных.
4. Выбор принципов верификации и тестирования системы хранения данных.
5. Выбор технологических средств.
6. Разработка системы хранения данных.
7. Тестирование разработанной системы хранения данных.

Следующим этапом выполним формализованную постановку задачи создания оптимизированной системы хранения данных для системы оператора электронного документооборота.

1.6 Формализованная постановка задачи оптимизированного хранилища данных для оператора электронного документооборота

Выполним формализацию системы хранения данных для оператора электронного документооборота, для этого опишем систему как совокупность параметров, которые дают оценку удовлетворения критериям. Будет использовано следующие обобщенное описание системы по формуле (1.18).

$$DS = (C, N, V, R, S). \tag{1.18}$$

где C – объем хранилища;

N – количество информации;

V – скорость доступа к данным;

R – надежность хранения данных;

S – безопасность хранения данных.

Интегральный критерий эффективности системы можно представить следующим образом (1.19):

$$\Phi = \frac{N * V * R * S}{C}. \tag{1.19}$$

Тогда, цель работы сводится к максимизации эффективности Φ (1.20):

$$\Phi \rightarrow \max. \qquad (1.20)$$

Таким образом, задача оптимизации системы хранения данных для оператора электронного документооборота сводится к следующим задачам:

1. Уменьшению необходимого объема запоминающих устройств.

2. Увеличению количества информации при фиксированном объеме памяти запоминающих устройств.

3. Увеличению скорости доступа к данным.

4. Уменьшению рисков потери данных.

5. Уменьшению рисков несанкционированного доступа к данным.

Выполнение вышеописанных задач свидетельствует об успешном оптимизации хранилища данных.

Выводы по разделу 1

В данном разделе была проанализирована теория систем хранений данных, их разновидности. Рассмотрены виды запоминающих устройств, их характеристики. Рассмотрена технология объединения запоминающих устройств RAID. Дано определение распределенным системам хранения данных, выделены их преимущества и недостатки.

Далее был произведен обзор готовых решений для хранения данных, изучено их устройство и предоставляемые функции. Произведено описание предметной области для решаемой задачи, определены требования к операторам электронного документооборота со стороны законодательства. Рассмотрен процесс, подачи налоговой отчетности в налоговые органы.

Рассмотрены существующие методы оптимизации хранилищ данных по различным критериям, определены влияния на каждый из критериев. Произведен вариантный анализ для методов.

Поставлена и формализована задача создания оптимизированной системы хранения данных для оператора электронного документооборота.

2 ПРОЕКТИРОВАНИЕ СИСТЕМЫ ХРАНЕНИЯ ДАННЫХ ДЛЯ ОПЕРАТОРА ЭЛЕКТРОННОГО ДОКУМЕНТООБОРОТА

2.1 Архитектура системы хранения данных для оператора электронного документооборота

Поставленная задача в первом разделе подразумевает разработку системы хранения данных, отвечающей следующим требованиям оператора электронного документооборота:

1. Разрабатываемая система хранения данных должна быть надежной, риск потери данных должен быть сведен к минимуму.

2. Разрабатываемая система хранения данных должна быть безопасной, риск несанкционированного доступа должен быть сведен к минимуму.

3. Разрабатываемая система хранения данных должна обладать достаточно большим объемом памяти и обладать возможностью расширения объемов, так как количество данных возрастает с течением временем.

4. Разрабатываемая система хранения данных должна быть высокопроизводительной, потому что по сути является «сердцем» системы оператора электронного документооборота.

Учитывая каждое из вышеописанных требований была разработана архитектура распределенной система хранений данных. Основными преимуществами данной архитектуры являются:

- простое и не дорогое увеличение объемов памяти;
- использование различных по характеристикам, производителям, стоимости запоминающих устройств;
- высокий уровень надежности хранения данных;
- высокий уровень производительности.

Исходя из описания процессов, происходящих в системе оператора электронного документооборота было выявлено, что к данным в зависимости от времени с момента добавления, происходит различное количество обращений. Так для новых данных количество обращений больше, чем для старых. Было введено понятие «температуры» данных, которое отражает востребованность данных. Чем выше «температура», тем чаще происходят обращения к данным. Соответственно, чем ниже «температура», тем реже обращение к данным.

При таком разделении данных по «температуре» система хранения данных была разделена на следующие типы хранилища:

1. «Горячее» хранилище.
2. «Теплое» хранилище.
3. «Холодное» хранилище.

Каждое их хранилищ в свою очередь может содержат от одного и более устройств хранения данных.

«Горячее» хранилище – это хранилище для данных, которые имеют самую высокую «температуру», данные к которым обращения происходят чаще всего. Для этого типа хранилища, самым главным критерием является – производительность. Использование каких-либо описанных методов оптимизации нецелесообразно. По сути данное хранилище представляет собой кэш для всей системы хранения в целом. С течением времени объем памяти для «горячих» данных не растет, так как ограничены количеством одновременно активных документооборотов. Для данного хранилища возможно использование дорогостоящих запоминающих устройств с низким средним временем доступа к данным и как правило малым объемом памяти.

«Теплое» хранилище – это хранилище для данных, которые имеют среднюю «температуру», к таким данным обращения происходят, но с меньшей интенсивностью. В силу того, что к «теплым» данным происходит меньше обращений и количество этих данных больше, по сравнению с «горячими» данными, то для данного хранилища возможно использование запоминающих

устройств с более низкими показателями среднего времени доступа и большими объемами памяти.

«Холодное» хранилище – это хранилище для данных, к которые имеют самую низкую «температуру», к таким данным обращений практически не происходит. Для данного хранилища данных важным критерием является объем памяти. В отличии от «горячих» и «теплых» данных – с течением времени объем «холодных» данных будет увеличиваться. Так как к таким данным обращения практически не происходят, то возможно использование недорогостоящих запоминающих устройств с низким средним временем доступа и большим объемом памяти.

Для обеспечения надежности хранения данных, вне зависимости от «температуры» данных – копия или несколько копий данных обязательно размещаются в «холодном» хранилище. Распределение копий данных осуществляется с помощью метода асинхронной репликации данных. Использование «горячего» и «теплого» хранилища для обеспечения надежности невозможно, так как эти хранилища ограничены в объемах данных.

Для «горячего» и «теплого» хранилища применение методов объема памяти недопустимо, так как методы очень отрицательно влияют на производительность.

В силу того, что «холодные» хранилища со временем увеличивают объемы данных и обращений к данным практически не происходит, то возможно применение самого эффективного метода оптимизации объемов памяти – метода сжатия данных.

Для обеспечения безопасности хранения данных, разрабатываемая система хранения данных должна будет располагаться в изолированной сети отдельно от системы оператора электронного документооборота. Взаимодействие оператора электронного документооборота и системы хранения данных будет осуществляться через ACL (Access Control List) настроенном на маршрутизаторе сети системы хранения данных. В свою очередь взаимодействие между подсистемами хранилища внутри изолированной сети не ограничено.

Предложенная архитектура системы хранения данных позволяет использовать дорогостоящие запоминающие устройства с высокими показателями среднего времени доступа небольшого объема, для «горячих» и «теплых» данных. Для «холодных» данных, в свою очередь, количество хранимых данных возрастает, следовательно, требуется периодическое увеличение объемов памяти. Для «холодного» хранилища можно использовать недорогостоящие запоминающие устройства большого объема, что дает значительную экономию денежных средств при высокой производительности и надежности системы.

2.2 Проектирование системы хранения данных

В качестве продолжения идеи распределения данных по «температуре», была разработана архитектура, которая состоит из подсистем доступа к «горячим», «теплым» и «холодным» данным, а также общей (на всю систему хранения данных) подсистемы доступа к данным, и подсистему управления данных.

Архитектура системы хранения данных представлена на рисунке 2.1.

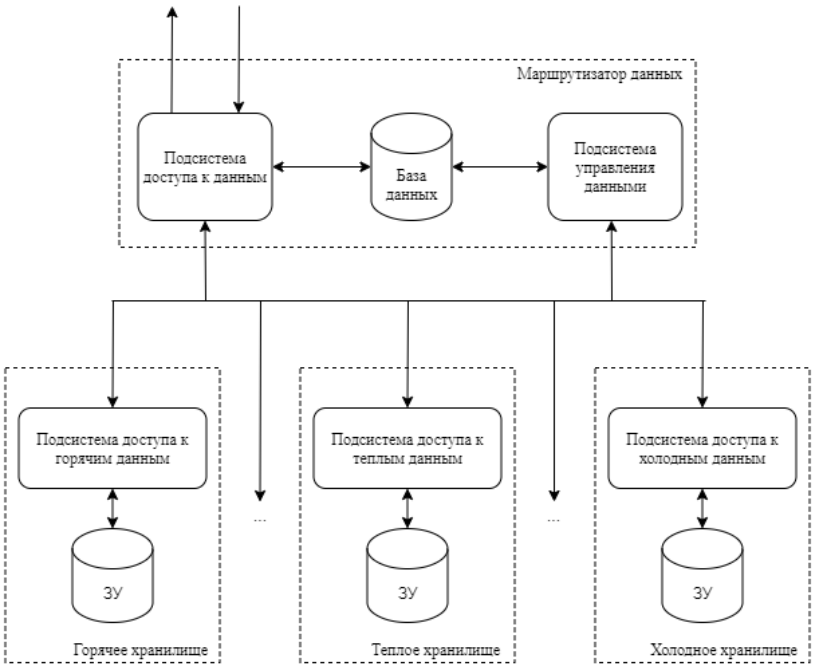


Рисунок 2.1 – Архитектура система хранения данных

Количество подсистем доступа к «горячим», «теплым» и «холодным» данным никоем образом не ограничивается.

2.2.1 Описание базы данных

Так как предложенная архитектура – распределенная, то появилась в сосредоточении информации о хранимых данных на разных хранилищах в одном месте. Данное решение – это хранение информации и поддержании этой информации в актуальном состоянии в единой базе данных.

Необходимо было хранить информацию о данных и хранилищах.

Для хранения информации о данных была выбрана структура – аналогичная рассмотренным готовым облачным хранилищам данных.

Все хранимые данные хранятся в объекте. Объект состоит из самих данных, метаданных и уникального ключа (который однозначно идентифицирующий объект). Метаданные в свою очередь разделяются на:

– системные: название, тип, размер, дата создания, дата последнего обращения;

– пользовательские: дополнительная информация в виде пар ключ-значения.

Каждый объект имеет список реплик, которые содержат информацию о том, на каком хранилище расположены данные объекта.

Все хранилища обязательно имеют уникальный идентификатор и дружелюбное к человеку название. Также хранилища описываются следующими характеристиками: тип (горячее, теплое, холодное), общий объем памяти, свободный объем памяти, режим работы (запись и чтение, только чтение, отключен).

2.2.2 Описание подсистемы доступа к данным

Данная подсистема – единственная, которая взаимодействует с внешней средой. Главное назначение этой подсистемы – предоставление удобного интерфейса для взаимодействия с системой оператора электронного

документооборота. Система оператора электронного документооборота ничего не должна знать об устройстве СХД.

В обязанности подсистемы доступа к данным входит:

– обработка входящих запросов на получение данных;

– обработка входящих запросов на добавление данных;

– обработка входящих запросов на удаление данных.

Обработка запросов на получение данных осуществляется по следующему алгоритму:

1. Получение списка-реплик требуемого объекта.

2. Выбор хранилища из списка. Приоритет отдается доступным хранилищам в порядке по типу хранилища: «горячее», «теплое», «холодное».

3. Получения данных из выбранного хранилища. В случае доступности хранилища – возвращение данных вызывающей стороне и уведомление подсистеме управления данных об обращении к данным. В случае недоступности хранилища – уведомление подсистемы управления данным о недоступности хранилища и повторное обращение к следующему по списку-реплик хранилищу.

Обработка запросов на добавление данных происходит по следующему алгоритму:

1. Получение информации о всех доступных хранилищах.

2. Выбор «горячего» и «холодного» хранилища для добавления. Хранилища должны быть в режиме «запись и чтение». Если на момент добавления данных нет подходящих «горячих» хранилищ, то выбирается «теплое», если доступных «теплых» хранилищ также нет, то выбор только «холодного» хранилища. Если хранилищ в итоге не найдено – то уведомление подсистемы управления данными о невозможности выбора хранилища для добавления данных.

3. Создание объекта, заполнение метаданных.

4. Отправка запроса на добавление данных в выбранные хранилища и ожидание подтверждения сохранения.

5. Заполнение списка реплик, сохранение объекта в базу данных и уведомление подсистемы управления данными о новом добавленном объекте.

6. Возвращение ключа объекта вызывающей стороне.

Обработка запросов на удаление данных происходит по следующему алгоритму: запрашивается список реплик для объекта и последовательно для каждого хранилища идет отправка запроса на удаление данных. В случае если какое-либо из хранилищ было не доступно – в обязательном порядке будет отправлено соответствующие уведомление в подсистему управления данными.

2.2.3 Описание подсистемы управления данными

Данная подсистема является «мозгом» системы хранения данных, все действия, которые происходят в СХД, должны в обязательном порядке быть переданы в данную подсистему.

Уведомления о всех действиях над объектами в системе необходимы для «умной» реакции на действия, по сути данный сервис собирает о всем статистику.

В обязанности данной системы входит:

- поддержание надежности хранения данных;
- перемещение данных в зависимости от их «температуры»;
- поддержание актуальной информации о хранилищах.

Для поддержания надежности хранения данных, данная подсистема с определенной периодичностью производит проверку всех объектов, на предмет наличия нескольких копий данных в различных хранилищах (в зависимости от коэффициента репликации). Коэффициент репликации – натуральное число, которое характеризует количество копий данных. Исходя из требований законодательства к операторам электронного документооборота, данный коэффициент не может быть меньше 2. В случае если условие надежности не исполняется, то данная система должна создать необходимые копии, используя следующий алгоритм:

1. Получить список доступных для записи «холодных» хранилищ.
2. Отправить запрос на добавление данных в хранилище.
3. Актуализировать список-реплик для объекта.

Для перемещение данных в зависимости от «температуры» специально была разработана формула определения «температуры» данных, которая зависит от анализа обращений к данным, а также времени, прошедшего с момента добавления данных.

Так как все хранимые данные системы оператора электронного документооборота обладают общей особенность, чем дольше они хранятся, тем меньше вероятность того, что к этим данным будет обращение. Данная особенность была выражена с помощью формулы (2.1).

$$A = \max\left(\frac{T_{\text{m}} - T_c}{T_{\text{m}}}, 0\right). \tag{2.1}$$

где T_{m} – максимальное время востребованности данных;

T_c – время, прошедшее с момента добавления файла.

A получило название – коэффициент старения данных. Коэффициент старения является безразмерной величиной и принимает значения в диапазоне [1; 0]. Регулируя T_{m}, изменяется время после которого, данные будут считаться старыми и следовательно обращение к данным не будет ожидаться.

Также было выявлено, что к разным данным происходит различное количество обращений в разные моменты времени (в зависимости от этапа документооборота). Данную особенность можно было выразить с помощью формулы (2.2).

$$P = \frac{V_t}{V}. \tag{2.2}$$

где V – общее количество обращений к данным;

V_t – количество обращений к данным за определенный промежуток времени.

P характеризует количество обращений к данным за конкретный промежуток времени и получило название – коэффициент востребованности данных. Коэффициент востребованности данных не имеет размерности и значения лежат в диапазоне [1; 0]. Регулируя временной промежуток, за который подсчитывается количество обращений к данным, регулируется инерция коэффициента на количество обращений. Чем шире промежуток, тем дольше длительность востребованности и наоборот.

С помощью коэффициентов старения данных и коэффициента востребованности данных была выражена «температура» данных (2.3).

$$T = P * A. \tag{2.3}$$

где *P* – коэффициент востребованности данных;

A – коэффициент старения данных.

«Температура» данных – безразмерная величина, принимает значения в диапазоне [1; 0]. Значения близкие к 1 – говоря о том, что данные имеют высокую «температуру», близкие к 0, наоборот – низкую.

Таким образом для перемещения данных в соответствии «температуры», подсистема управления данных производит определение данных для объектов в определенной периодичностью по следующему алгоритму:

1. Для объекта определяется «температура».

2. Если *T* лежит в диапазоне [1; 0.75) – то данные необходимо разместить на «горячем» хранилище.

3. Если *T* лежит в диапазоне [0.75; 0.25) – то данные необходимо разместить на «теплом» хранилище.

4. Если *T* лежит в диапазоне [0.25; 0] – то данные необходимо разместить на «холодном» хранилище.

Для поддержания актуальной информации о подключенных хранилищах, аналогично – с определенной периодичностью подсистема производит актуализацию подключенных хранилищ в соответствии со следующим алгоритмом:

1. Выбор всех подключенных хранилищ.

2. Последовательная отправка «пинг» запросов в хранилища.

3. Если хранилище доступно, то в ответ на запрос возвращается информация о хранилище: идентификатор, название, общий объем памяти, объем свободной памяти. В зависимости от объема свободной памяти, перевод хранилища в режим «запись и чтение» либо «чтение», также обновление информации о свободном объеме памяти.

4. Если хранилище недоступно, то переключение хранилища в режим «отключено».

Для того, чтобы хранилище было включено в систему, либо пере подключено, при запуске хранилища, оно должно наоборот отправить аналогичный запрос подсистеме управления данными, после чего оно будет включено в систему, в базу данных запишется актуальная информация.

Также, если в подсистему управления данными поступают уведомления о том, что при обработке запроса не хранилище было не доступно, то подсистема сразу же отправляет «пинг» запрос в «проблемное» хранилище и актуализирует информацию в базе данных.

2.2.4 Описание подсистем доступа к хранилищам

Данные подсистемы осуществляет доступ к запоминающим устройствам в соответствии с типом хранимых данных («температурой» данных), используют унифицированный интерфейс для взаимодействия с подсистемой доступа к данным и подсистемой управления данными.

У данных подсистем следующие обязанности:

– добавлять данные;

– передавать данные;

– удалять данные;

– отвечать на запрос доступности («пинг» запрос).

В зависимости от типа хранилища («горячее», «теплое», «холодное») немного различается реализация данных обязанностей.

Для всех типов хранилищ, должна использоваться локальная база данных, которая содержит информацию необходимую для осуществления манипуляции над данными.

Так при добавлении данных в случае «горячего» и «теплого» хранилища, данные просто сохраняются на запоминающем устройстве (различается только способ взаимодействия с ЗУ), а для «холодного» хранилища данные должны предварительно быть сжаты.

При передачи данных, ситуация аналогичная, реализация отличается в зависимости от типа ЗУ, но для «холодного» хранилища необходима предварительная распаковка данных.

При удалении данных – для всех типов хранилищ различается только реализация в зависимости типа запоминающего устройства.

2.3 Критерии оценки эффективности системы хранения данных

Для оценки эффективность разрабатываемой системы хранения данных воспользуемся критериями, которые были выделены в пункте 1.4 текущей работы, а именно:

– критерий объема памяти;

– критерий надежности хранения;

– критерий производительности.

Так для критерия объема памяти, целью является увеличение количество хранимой информации при уменьшении необходимого объема памяти для хранения. Для проверки данного критерия воспользуемся формулой (2.4).

$$K_c = \frac{N}{C}. \quad (2.4)$$

где K_c – оценка критерия объема памяти;

N – количество информации;

C – объем памяти

Под количеством информации следует понимать объем информации без какого-либо преобразования. Для систем хранения данных, в которых не применяются какие-либо методы оптимизации объема памяти K_c равняется единице, следовательно значения K_c, которые больше единицы, говорят о том, что система хранения данных оптимизирована по критерию объема памяти.

Так как разрабатываемая система хранения данных из-за требований надежности хранения данных хранит данные избыточно (хранит дополнительно копии данных), то при расчете оценки следует исключить объем памяти копий данных. В силу того, что весь объем памяти сосредоточен в «холодных» хранилищах, то оценку необходимо производить для «холодного» хранилища.

Под критерием надежности хранения данных подразумевается уменьшение рисков потери данных в случае форс-мажорных ситуаций, начиная от выхода из строя запоминающего устройства и заканчивая техногенными катастрофами.

Для оценки данного критерия воспользуемся формулой (2.5).

$$K_r = 1 - p^{R_f}. \quad (2.5)$$

где K_r – оценка критерия надежности хранения данных;

p – вероятность отказа запоминающего устройства за период эксплуатации;

R_f – коэффициент репликации.

В зависимости от типа, производителя запоминающего устройства мы можем обозначить вероятность выхода из строя одного из ЗУ. Согласно статистике компании «Backblaze», вероятность отказа жестких дисков в течении трёх лет составляет примерно от 3.1% до 26.5% [5]. Вероятность наступления техногенных катастроф прогнозировать затруднительно, поэтому они не учитываются. В разрабатываемой системе хранения данных будет содержаться более чем один жесткий диск, поэтому следует учитывать вероятность выхода из строя нескольких дисков. Для борьбы с отказами запоминающих устройств в разрабатываемой системе используется метод репликации данных, благодаря которому данные расположены сразу на нескольких ЗУ. Количество копий определяется с помощью коэффициента репликации.

В качестве СХД, по которой будем производить оценку критерия надежности хранения возьмем: вероятность выхода из строя жесткого диска равную 0.2; коэффициент репликации равен 1. Для данного примера оценка K_r равняется 0.8, следовательно при выходе из строя одного из запоминающего устройства возможно безвозвратное потеря данных.

Для критерия производительности целью является увеличение скорости передачи данных, а также увеличение количества операций записи и чтения (IOPS) в секунду. В случае простого хранения (не использования методов сжатия, шифрования) данные параметры напрямую зависят от характеристик запоминающего устройства, на котором расположены данные (операция чтения), либо куда располагаются данные (операция записи).

В случае разрабатываемой системы хранения данных, используются различные типы запоминающих устройств, которые объединяются в единую СХД. Так как данные распределяются в зависимости от «температуры» данных и как было определено, данные к которым обращаются чаще всего, расположены на самых дорогостоящих и высокопроизводительных хранилищах. Таким образом для сравнения необходимо брать характеристики запоминающего устройства «горячего» хранилища.

2.4 Принципы верификации и тестирования системы хранения данных

В первую очередь, для разрабатываемой системы хранения данных необходимо произвести верификацию критериев эффективности определенных в пункте 2.3.

Для определения оценки по критерию объему памяти, следует подготовить набор тестовых данных, состоящий из файлов различного размера и типа, и подать их на вход разработанной системы хранения данных.

Так в соответствии с пунктом 1.3.3, было определено, что для систем операторов электронного документооборота примерно половину объема памяти системы, занимают небольшие по размеру XML файлы, вторую половину – занимают данные различных размеров и типов.

После добавления тестового набора в систему хранения данных, следует произвести расчеты в соответствии с формулой (2.4). В качестве параметра N следует взять общий размер тестового набора данных, в качестве параметра C занимаемый объем данных в «холодном» хранилище.

Если оценка K_c приняло значение большее 1, то следует считать, что система хранения данных является оптимизированной со стороны критерия объема памяти.

Для определения оценки по критерию надежности хранения данных следует воспользоваться формулой (2.5). В случае если оценка K_r превышает значение рассчитанного примера, то следует считать, что система хранения данных оптимизирована со стороны критерия объема памяти.

Также необходимо провести тестирование системы в случае выхода из строя (недоступности) одного из подключенных хранилищ. Для этого, необходимо намерено выключить одно из хранилищ. В такой ситуации система обязана продолжить свою работу, все данные должны быть доступны, новые данные также должны добавляться, допускается небольшая задержка в работе системе. Подсистема управления данных, должна осуществить попытку создания дополнительных копий данных, которые стали недоступны из-за отключения

хранилища, а также актуализировать информацию о подключенных хранилищах. После необходимо включить, ранее отключенное, хранилище и проверить, что хранилище вновь стало активным.

Следующим шагов, является тестирование механизма перераспределения данных в соответствии с «температурой». Для этого необходимо предварительно уменьшить параметры, необходимые для определения «температуры», и увеличить частоту проверки «температуры» данных для подсистемы управления данных. Добавить данные и наблюдать, как с течением времени данные будут перемещаться с «горячего» на «теплое» и затем на «холодное» хранилище.

Выводы по разделу 2

В данном разделе была предложена распределенная архитектура системы хранения данных, отвечающей требованиям оператора электронного документооборота. Введено понятие «температуры» данных, дано описание преимуществ предложенной архитектуры.

Далее было произведено проектирование системы хранения данных, выделены подсистемы их обязанности, а также произведено описание взаимодействий подсистем между собой и их устройство. Произведена формализация «температуры» данных.

Следующим этапом стало определение критериев эффективности, благодаря которым возможно оценить эффективность разрабатываемой системы хранения данных с точки зрения различных критериев эффективности.

В завершении раздела были описаны принципы проведение верификации и тестирования разрабатываемой системы.

3 РАЗРАБОТКА СИСТЕМЫ ХРАНЕНИЯ ДАННЫХ ДЛЯ ОПЕРАТОРА ЭЛЕКТРОННОГО ДОКУМЕНТООБОРОТА

3.1 Выбор средств разработки системы хранения данных

Так как разрабатываемая распределенная система хранения данных может включать в себя множество запоминающих устройств различных характеристик, то необходимо решить вопрос о способе объединения запоминающих устройств. Исходя из архитектуры системы хранения данных – объединение происходит за счет сети. Для взаимодействия подсистем внутри сети был выбран протокол прикладного уровня HTTP, из-за наличия механизма параллельной загрузки данных и большей производительностью по сравнению с FTP [24].

3.1.1 Выбор языка программирования

В силу того, что разрабатываемая система хранения данных может включать в себя множество хранилищ с различными характеристиками, аппаратным обеспечением и операционной системой, то главным критерием выбора языка программирования – является наличие кроссплатформенности.

На сегодня одними из наиболее популярных кроссплатформенных языков программирования «enterprise» уровня – являются объектно-ориентированные языки C# и Java. Язык программирования C++ не будет рассматриваться, хоть он и считается кроссплатформенным, но в зависимости от аппаратного и операционного обеспечения, возможно наличие необходимости доработки исходного кода. В свою очередь C# и Java не имеют данного недостатка, так как модули компилируются в промежуточный код, который при исполнении транслируется в машинные команды с помощью программ исполнения (CLR для C# и JVM для Java).

C# и Java обладают схожим функционалом, схожим синтаксисом. C# в отличии от Java содержит в себе атрибуты, делегаты (указатель на функцию) и события [25]. Также C# содержит в себе библиотеку LINQ (Language Integrated

Query), которая предоставляет простой и мощный язык запросов к различным источникам данных. Отметим, что на языке C# гораздо проще и удобней работать с асинхронным кодом, благодаря операторам async/await, основанным на обещаниях (Promise), которые позволяют писать оптимальный асинхронный код в виде синхронного [25]. В свою очередь Java имеет больше готовых фреймворков и библиотек кода в открытом доступе.

Таким образом для реализации системы хранения данных для оператора электронного документооборота был выбран язык C# и платформа .NET 5.

Так как для разработки системы хранения данных была выбрана платформа .NET, то для реализации API-интерфейсов подсистем в виде веб-сервисов будем использовать фреймворк с открытым исходным кодом ASP.NET Core, разработанный компанией Microsoft [26].

Для взаимодействия подсистем с базой данных используется ORM EF Core (EntityFramework Core). ORM (Object-Relational Mapping) – технология программирования, которая связывает базы данных с концепциями объектно-ориентированных языков программирования, создавая «виртуальную объектную базу данных». EF Core представляет собой объектно-ориентированную, легковесную расширяемую технологию от компании Microsoft для доступа к данным, которые хранятся в реляционной базе данных [27]. EF Core поддерживает работу со следующими СУБД:

- MS SQL Server;
- SQLite;
- PostgreSQL;
- MySQL.

3.1.2 Выбор базы данных

Так как система хранения данных в первую очередь предназначена для хранения данных в виде файлов, представленных в различных форматах (.docx, .pdf, .xml, .jpeg, .tiff) то встает необходимость хранения информации о файлах,

которые сохранены внутри разрабатываемой системы хранения данных. Исходя из проектирования, необходимо хранение и поиск информации о расположении данных, метаданных файлов и состояние хранилищ системы. Для хранения таких данных очень хорошо подойдут реляционные базы данных.

На сегодня широкое распространение получили следующие СУБД:

– Oracle Database;
– Microsoft SQL Server;
– PostgreSQL;
– SQLite.

Все перечисленные СУБД поддерживают стандарт SQL (Structured Query Language). Oracle Database и MS SQL Server являются платными и обладают хорошими показателями производительности, как правило используются в больших информационных системах, где количество записей в таблицах превышает 50 миллионов записей [28].

PostgreSQL это мощная кроссплатформенная объектно-реляционная система управления базами данных с открытым исходным кодом. PostgreSQL как и Oracle Database, MS SQL полностью поддерживает ACID (Atomicity, Consistency, Isolation, Durability) принципы. Главное преимущество PostgreSQL – это поддержка большого количества типов данных и поддержка управления параллельным доступом посредством версионности данных (MVCC) [28].

SQLite – легко встраиваемая в приложения база данных. При работе с этой СУБД обращения происходят напрямую к файлам, вместо сетевого взаимодействия, поэтому данная СУБД имеет высокие показатели скорости. SQLite использует урезанный стандарт SQL, некоторые функции отсутствуют (например ролевое разделение доступа), но основные все-таки поддерживаются. Данная СУБД не предназначена для больших объемов информации.

Исходя из предложенной архитектуры системы хранения данных, для хранения информации расположении данных, их метаданных, состояний хранилищ, хорошим выбором будет использовать ОРСУБД PostgreSQL. Для

функционирования подсистем доступа к «горячим», «теплым» и «холодным» данных воспользуемся SQLite.

Для того, чтобы подсистемы имели возможность обмениваться сообщениями (событиями) между собой, воспользуемся брокерами сообщений. Самые распространенные из брокеров сообщений – это RabbitMQ от компании SpringSource, реализующий протокол AMQP (Advanced Message Queuing Protocol) и Apache Kafka от компании Apache Software Foundation. RabbitMQ в отличии от Apache Kafka имеет механизмы гибкой настройки маршрутизации сообщений, но производительность хуже чем у Apache Kafka [29]. Так как обмен сообщений между подсистемами не постоянный, то подходит RabbitMQ.

Так как подсистема управления данными подразумевает выполнение действий по расписанию, таких как расчет температур для объектов и проверка доступности хранилищ, то необходима библиотека для планирования задач.

Для платформы .NET существуют следующие планировщики задач: Hangfire, Quartz.NET. Quartz.NET является портом планировщика Quartz для Java. Hangfire – это многопоточный и масштабируемый планировщик задач с открытым исходным кодом. Представленные выше планировщики предоставляют одинаковый функционал по планированию и исполнению задач, из преимуществ Hangfire над Quartz.NET стоит выделить более простой API-интерфейс и наличия «из коробки» веб-интерфейса [30].

Для запуска системы хранения данных будем использовать технологию контейнеризации. Docker – это программное обеспечение с открытым исходным кодом, применяемое для разработки, тестирования, доставки и запуска приложений в изолированное среде. Docker нужен для более эффективного использования системы и ресурсов, быстрого развертывания готовых программных продуктов, а также для масштабирования и переноса в другие среды с гарантированным сохранением стабильной работы.

Основной принцип работы Docker – контейнеризация приложений. Этот тип виртуализации позволяет упаковывать программное обеспечение в изолированную среду – контейнер. Каждый из контейнеров содержит все необходимое для работы

приложения. Это дает возможность одновременного запуска большого количества контейнеров на одной машине.

Преимущества использования Docker:

– минимальное потребление ресурсов – контейнеры потребляют намного меньше ресурсов, чем виртуальная машина;

– скорость развертывания – установка подсистем отсутствует так как используется заранее готовый и настроенный образ подсистемы;

– изолированность – каждый из контейнеров изолирован друг от друга;

– простое масштабирование – для масштабирования необходимо просто запустить дополнительный контейнер.

3.2 Разработка структуры данных

При разработке структуры данных для разрабатываемой системы хранения данных был взят термин «объект» по аналогии с рассмотренными ранее облачными хранилищами. Каждый объект имеет уникальный идентификатор и данные (метаданные) характеризующие его: название файла, размер файла, расширение файла и другие. Для хранения были выделены сущности Objects и ObjectMetadata, которые связаны отношением один ко многим.

Так как система хранения данных состоит из множества распределенных хранилищ различного типа, была выделена сущность Storages, которая описывает состояние хранилищ системы.

Так как копии объектов могут быть сохранены на различных хранилищах, то для этого хранения данной информация была создана сущность ObjectReplicas, которая хранит информацию о размещении всех копий объектов. Данная сущность связана отношениями один ко многим с Objects и Storages.

Для функционирования подсистемы управления данными необходим сбор статистики действий над объектом, для этого была выделена сущность ObjectEvents, которая связана отношением одни ко многим с Objects.

Схема база данных в нотации IDEF1X изображена на рисунке 3.1.

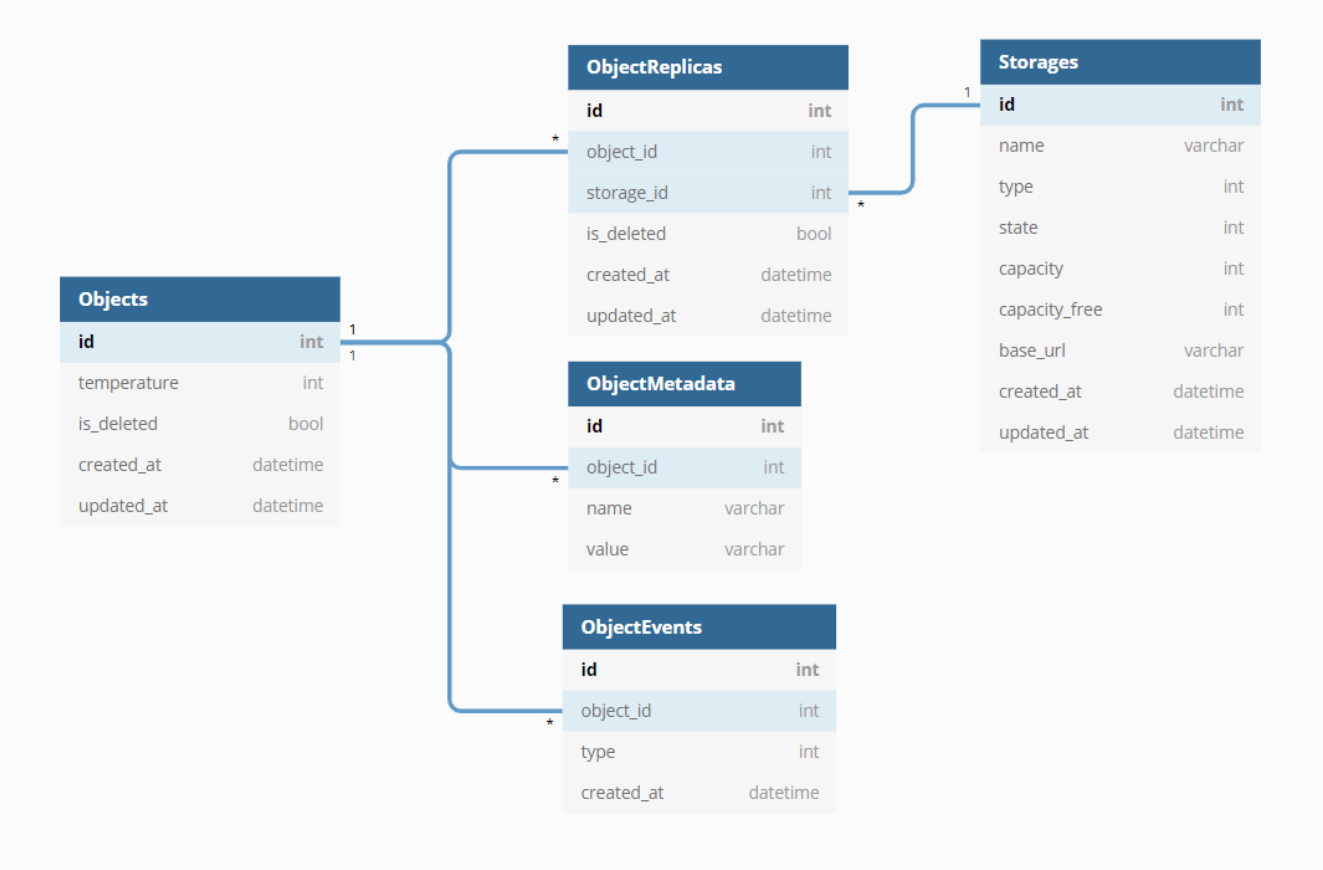


Рисунок 3.1 – Структура базы данных

Сущность Objects состоит из следующих атрибутов:

– id – уникальный идентификатор объекта в базе данных;

– temperature – температура объекта, принимает следующие значения: 1 (горячая), 2 (теплая), 3 (холодная);

– created_at – временная метка добавления объекта в систему;

– updated_at – временная метка изменения объекта;

– is_deleted – признак удаленности данных, при удалении объект помечается как удаленный.

Сущность Storages состоит из следующих атрибутов:

– id – уникальный идентификатор хранилища;

– name – дружелюбное имя хранилища;

– type – тип хранилища, принимает следующие значения: 1 (hot), 2 (warm), 3(cold);

– state – состояние хранилища, принимает следующие значения: 1 (чтение и запись), 2 (только чтение), 3 (отключен);

– capacity – общий объем памяти хранилища, измеряется в килобайтах;

– capacity_free – свободный объем памяти хранилища, измеряется в килобайтах;

– base_url – базовый URL-адрес хранилища;

– created_at – временная метка добавления хранилища в систему;

– updated_at – временная метка изменения состояния хранилища.

Сущность ObjectMetadata состоит из следующих атрибутов:

– id – уникальных идентификатор метаданных;

– object_id – идентификатор объекта к которым относятся метаданные;

– name – имя метаданных;

– value – значение метаданных.

Сущность ObjectReplicas состоит из следующих атрибутов:

– id – уникальный идентификатор реплики объекта;

– object_id – идентификатор объекта к которым относятся реплики;

– storage_id – идентификатор хранилища на котором размещены данные;

– is_deleted – признак удаленности данных;

– created_at – временная метка добавления реплики объекта;

– updated_at – временная метка изменения реплики объекта.

Сущность ObjectEvents состоит из следующих атрибутов:

– id – уникальный идентификатор события;

– object_id – идентификатор объекта к которым относятся события;

– type – тип события, принимает следующие значения: 1 (добавление объекта), 2 (обращение к объекту), 3 (удаление объекта);

– created_at – временная метка создания события.

Разработанная структура данных позволит в полном объеме реализовать задуманную систему хранения данных.

3.3 Реализация системы хранения данных

Для реализации системы хранения данных была выбрана архитектура REST (Representational State Transfer). Требования к архитектуре REST [31]:

1. Модель клиент-сервер.

2. Отсутствие состояния. В период между запросами клиента никакая информация о состоянии клиента на сервере не храниться (Stateless protocol). Все запросы от клиента должны быть составлены так, чтобы сервер получил всю необходимую информацию для выполнения запроса.

3. Кэширование. Ответы сервера, в свою очередь должны иметь явное или неявное обозначение как кэшируемые или некэшируемые с целью предотвращения получения клиентами устаревших или неверных данных.

4. Единообразие интерфейса. Наличие унифицированного интерфейса является фундаментальным требованием REST архитектуры. Все предоставляемые ресурсы обязательно уникально идентифицируются.

5. Слои. Клиент не должен быть способен определить, взаимодействует он напрямую с сервером или же с промежуточным узлом.

Исходный код разработанных модулей и подсистем содержится в приложении А.

3.3.1 Реализация модуля доступа к базе данных

В силу того, что доступ к общей базе данных, где храниться информация о всех объектах системы хранения данных, необходимо двум подсистемам: подсистеме доступа к данным и подсистеме управления данным, то необходимо было создание общего модуля.

Для реализации данного модуля, был использован фреймворк EF Core и система управления базой данных PostgreSQL. Для каждой сущности, описанной в пункте 3.2 настоящей работы, был создан соответствующий класс языка программирования C#. Для каждого созданного класса с помощью FluentApi были настроены связи между классами, определены типы данных для атрибутов, определены названия столбцов. Для создания базы данных было использован механизм миграций EF Core, который позволил в автоматическом режиме без ручного написания SQL-скриптов создать необходимые таблицы в базе данных.

3.3.2 Реализация подсистемы доступа к данным

Подсистема доступа к данным является – единственная подсистема, которая должна взаимодействовать с системой оператора электронного документооборота. Для взаимодействия подсистемы и системы ЭДО используется протокол HTTP, подсистема реализована в виде веб сервиса, который предоставляет API (Application Programming Interface) для работы с хранилищем.

Для реализации веб-сервиса используется фреймворк ASP.NET Core, для работы с базой данных – модуль доступа к данным.

Подсистема доступа к предоставляет простой интерфейс, предоставляющий CRUD (Create, Read, Update, Delete) операции над хранимыми объектами, имеет следующие конечные точки доступа:

– http://host/api/objects/ POST – добавление нового объекта в систему;

– http://host/api/objects/{id}/meta GET – получение только метаданных объекта;

– http://host/api/objects/{id}/data GET – получение только данных;

– http://host/api/objects/{id} DELETE – удаление объекта из системы.

Id – уникальный идентификатор объекта, который присваивается объекту при добавлении его в хранилище. Алгоритмы, в соответствии с которыми, выполняются операции описаны во 2-ой части настоящей работы. Опишем интерфейсы вышеописанных операций.

Для добавления нового объекта, необходимо отправить POST-запрос, указав HTTP-заголовок «Content-Type» как «multipart/form-data» и прикрепив загружаемый файл к параметру «file», все остальные параметры будут рассматриваться подсистемой как метаданные.

При успешном добавлении объекта в систему, в ответ на запрос будет возвращен статус 200 (Ok) и JSON-объект, который будет содержать идентификатор объекта и метаданные объекта, пример ответа представлен на рисунке 3.2.


```
{
  "id": 2805,
  "metadata": {
    "name": "some.xml",
    "size": 154,
    "mime-type": "application/xml"
  }
}
```


Рисунок 3.2 – Успешный результат добавления объекта

В случае, если при добавлении объекта происходит ошибка, то будет возвращен статус 400 (Bad request) и дружелюбное описание произошедшей ошибки.

Операции чтения объектов разделены на две: получение метаданных и получение ранее загруженного файла.

Для получения метаданных объекта необходимо отправить GET-запрос на api/objects/{id}/meta, где id – уникальный идентификатор объекта, полученного ранее при добавлении объекта. В случае успешной обработки запроса, будет возвращен статус 200 и JSON-объект аналогичный рисунку 3.2. Если по идентификатору отсутствует какой-либо объект в базе данных (не был добавлен, либо удален) будет возвращен ответ со статусом 404 (Not found), в другом случае, будет возвращен ответ со статусом 400 и описанием ошибки.

Для получения данных необходимо отправить GET-запрос на api/objects/{id}/data. В случае успеха, будут возвращены ранее загруженные данные. При возникновении ошибок обработки запроса на получение содержимого, ответы сервера аналогичны, ответам сервера при ошибках обработки запроса на получение метаданных.

Для удаления объектов из системы необходимо отправить DELETE-запрос на api/object/{id}. При успешном удалении объекта из системы, сервер в ответ на запрос отправить статус 200, при попытках обращении к удаленному объекту, подсистема будет возвращать 404 ошибку. Реакция на ошибки аналогична операциям чтения объектов.

Так как подсистема доступа к данным должна взаимодействовать с «горячими», «теплыми» и «холодными» хранилищами данных, то для этого был разработан модуль-клиент интерфейса хранилища. Так как со всеми хранилищами данных взаимодействие происходит по HTTP и все хранилища данных имеют общий для всех API-интерфейс, то с помощью библиотеки Flurl был разработан универсальный клиент для всех хранилищ.

Для уведомлений подсистемы управления данными о событиях, которые происходят с хранимыми объектами – был создан интеграционный модуль с брокером сообщений RabbitMQ. Для разработки модуля использовалась библиотека RabbitMQ.Client. Данный модуль позволяет отправлять сообщения в очереди RabbitMQ.

3.3.3 Реализация подсистемы управления данными

Подсистема управления данными выполняет важную роль в работе системы хранения данных. В её обязанности входит:

– поддержание актуальной информации о подключенных хранилищах;

– обеспечение надежности хранения данных (объектов);

– перемещение данных (объектов) в зависимости от их температуры.

Для выполнение всех описанных обязанностей, данная подсистема выполняет следующие задачи: проверяет доступность каждого хранилища;

проверяет количество копий объектов; рассчитывает температуру для каждого объекта. Каждая задача является повторяющейся и выполняется с определенной периодичностью. Алгоритмы выполнения задач описаны во второй части настоящей работы.

Подсистемы управления данными реализована в виде сервиса-службы с помощью фреймворка Worker Service платформы .NET. Так как данная подсистема не предоставляет API для других подсистем, а выполняет повторяющиеся задачи и прослушивает очереди, то реализация данной подсистемы в виде веб-сервиса – избыточна.

Подсистема для своей работы использует раннее описанный модуль доступа к базе данных и также универсальный клиент к хранилищам данных.

В качестве планировщика задач используется фреймворк Hangfire, по умолчанию данный фреймворк использует СУБД (в нашем случае PostgreSQL) для хранения расписания запуска задач на выполнение. Были созданы следующие задачи в виде классов языка C#:

– CheckStoragesTask – задача, которая проверяет доступность всех подключенных хранилищ к системе хранения данных, в случае если какое-либо хранилище недоступно, то производит отключение данного хранилища из системы и планирует создание копий всех объектов, которые находились в отключаемом хранилище, задача выполняется через каждые 10 минут;

– CalculateTemperaturesTask – задача, которая пересчитывает температуру всех «горячих» и «теплых» объектов (для «холодных» объектов перерасчет температуры бессмыслен), в случае если температура изменилась, то производит перемещение задачи в соответствии с температурой, задача выполняется один раз в сутки.

Помимо выполнения повторяющихся задач, подсистема управления данными производит обработку событий, произошедших над объектами и хранилищами, в реальном времени. Для этого используется очереди сообщений RabbitMQ. Определены следующие очереди:

– Fs.ObjectEvents – в данную очередь поступают сообщения, которые связанны с объектами (добавление в систему, обращение к объекту, удаление объекта);

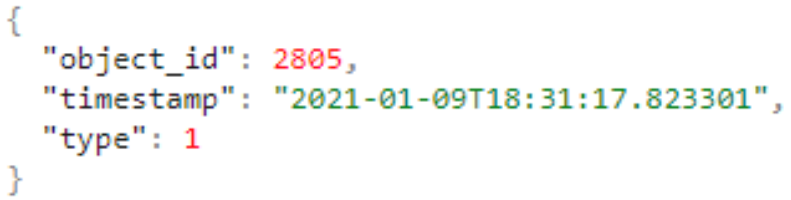
– Fs.StorageEvents – в данную очередь поступают все события связанные с хранилищами (подключение хранилища, события о недоступности хранилища);

– Fs.ObjectTransfer – в данную очередь поступают сообщения о необходимости создании копии объекта в другом хранилище, перемещении объекта из одного хранилища в другое.

При запуске подсистемы управления данными, в первую очередь происходит подключение (регистрация обработчиков сообщений) к вышеописанным очередям, подключение происходит за счет библиотеки RabbitMQ.Client. Далее при поступлении сообщений в очереди, они сразу же начнут свою обработку.

Для того, чтобы отправить сообщение в очередь RabbitMQ его необходимо предварительно сериализовать, а при обработке соответственно провести обратную операцию. Все сообщения имеют JSON формат, преобразованные в массив байт с помощью кодировки UTF-8. Также сообщение должно соответствовать конкретными типам сообщений (для того, чтобы обработчики могли правильно обрабатывать сообщения каждой очереди).

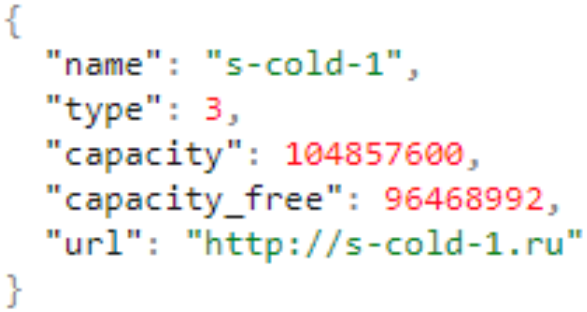
Так для очереди Fs.ObjectEvents все сообщения должны соответствовать JSON-объекту, который содержит идентификатор объекта, временную метку и тип события (1 – объект добавлен, 2 – обращение к объекту, 3 – удаление объекта), на рисунке 3.3 представлен пример сообщения.



```
{
  "object_id": 2805,
  "timestamp": "2021-01-09T18:31:17.823301",
  "type": 1
}
```


Рисунок 3.3 – Пример сообщения для очереди Fs.ObjectEvents

Очередь Fs.StorageEvents хранит в себе сообщения двух типов: подключение хранилища, событие о недоступности хранилища. Первый тип сообщений содержит информацию о подключаемой хранилище (уникальное дружелюбное имя, описание характеристик хранилища), пример сообщения изображен на рисунке 3.4.

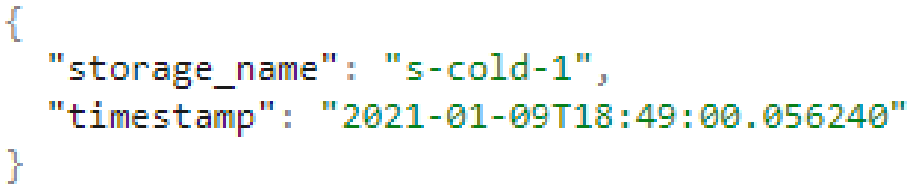


```
{
  "name": "s-cold-1",
  "type": 3,
  "capacity": 104857600,
  "capacity_free": 96468992,
  "url": "http://s-cold-1.ru"
}
```


Рисунок 3.4 – Пример сообщения подключения хранилища

При подключении хранилища к системе, подсистема проверяет доступ к хранилищу и в случае успеха, добавляет информацию в базу данных. Поле «name» задается хранилищу при конфигурации и обязательно должно быть уникально. Если хранилища с заданным именем уже включена в систему, то подсистема её повторно подключит (в случае если хранилище с таким именем было отключено) либо проигнорирует сообщение.

Второй тип сообщений содержит информацию о имени хранилища и временной метки, пример сообщений представлен на рисунке 3.5.



```
{
  "storage_name": "s-cold-1",
  "timestamp": "2021-01-09T18:49:00.056240"
}
```


Рисунок 3.5 – Пример сообщения о недоступности хранилища

В очередь Fs.ObjectTransfer поступают сообщений одного типа. Сообщение содержит информацию о текущем размещении объекта, тип операции (1 –

копирование, 2 – перемещение) и хранилище в которое будет отправлен объект. Пример сообщения изображен на рисунке 3.6.

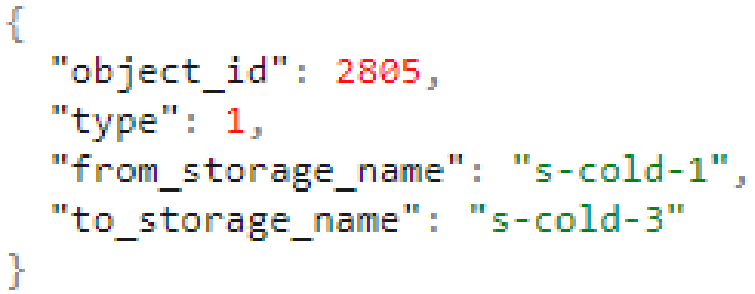


Рисунок 3.6 – Пример сообщения перемещения объекта

При копировании объекта происходит только создание реплики объекта, при перемещении – после успешного размещения объекта в хранилище-получателе, будет удален объект из хранилища-отправителя и обновлена информация в базе данных.

3.3.4 Реализация подсистем доступа к хранилищам

Подсистемы доступа к «горячим», «теплым» и «холодным» данным предоставляют собой API-интерфейс с CRUD операциями для данных объекта. Данные подсистемы в независимости от температуры хранимых данных предоставляют один и тот же интерфейс. Это необходимо, для того, чтобы при добавлении нового хранилища, не было необходимости доработки вышестоящих подсистем (подсистемы доступа к данным и подсистемы управления данными).

Подсистемы реализованы в виде веб-сервисов ASP.NET Core, аналогично подсистеме доступа к данным. Интерфейс подсистем состоит из следующих конечных точек:

– api/files POST – используется для добавления данных в хранилище;

– api/files/{id} GET – используется для извлечения данных из хранилища;

– api/files/{id} DELETE – используется для удаления данных их хранилища;

– api/storage/ping GET – используется для проверки доступности хранилища со стороны подсистемы управления данными.

Id в запросах на извлечение и удаления данных, является уникальным идентификатор, который присвоен подсистемой доступа к данным, при создании объекта в системе.

Для добавления данных в хранилище, необходимо отправить запрос на адрес api/files методом POST, с указанием заголовка запроса «Content-Type» как «multipart/form-data», прикрепив обязательно загружаемые данные к параметру «file», также обязательно необходимо указать уникальный идентификатор объекта под параметром «id».

В случае успешного добавления данных в хранилище возвращает результат со статусом 200 (Ok), в случае отсутствии хотя одного из обязательных параметров – возвращается результат со статусом 400 (Bad Request) и описанием ошибки.

При успешном извлечении данных из хранилища, как результат возвращаются хранимые данные в бинарном виде со статусом 200 (Ok), в случаях если, по переданному идентификатору объекта, данных не содержится – будет возвращен результат со статусом 404 (Not Found), аналогичное поведение в случае удаления данных.

На конечную точку api/storage/ping запрос отправляется без параметров, в качестве результат будет возвращено описание текущего состояния хранилища (общий размер памяти и объем свободной памяти), в случае если в ответ на запрос приходят ответы отличные от 200 (Ok), подсистема управления данными выводит хранилища из системы.

Для хранения информации о хранимых данных хранилища, используется СУБД SQLite, для подключения к СУБД используется EF Core.

Для подключения или повторного подключения хранилища в систему, при запуске хранилища, происходит отправка сообщения в очередь Fs.StorageEvents с помощью библиотеки RabbitMQ.Client.

«Горячие» хранилища располагают данные в оперативной памяти. Как известно оперативна память обладает высокими показателями скорости чтения и

записи, но также является энергозависимой. В силу того, что оперативная память энергозависимая, то при любой перезагрузке хранилища – все данные будут утеряны. Для борьбы с этим недостатком подсистема, сначала сохраняет данные на запоминающее устройство (должно быть не меньше чем объем оперативной памяти), а после с помощью MemoryStream загружает данные в оперативную память. В случае если происходит перезагрузка подсистемы, то при запуске подсистемы, все данные будут вновь загружены в оперативную память.

«Теплые» хранилища хранят свои данные на SSD-дисках. Данные диски обладают большим объемом памяти, чем оперативна память, а также хорошими показателями скорости чтения и записи.

«Холодные» хранилища используют в качестве запоминающих устройств HDD-диски. В отличии от предыдущих подсистем доступа, перед записью данных на запоминающие устройство, подсистема производит операцию сжатия данных, а для извлечения данных подсистема производит восстановления данных. Для описанных операций используется класс GZipStream, предоставляющий функционал сжатия и восстановления данных. Он основан на основе алгоритма DEFLATE, который основан на алгоритме LZ77 и алгоритма Хаффмана.

3.4 Тестирования, верификация системы хранения данных

Исходя из пунктов 2.3 и 2.4 настоящей работы, были разработаны оценки для критериев эффективности работы системы хранения данных, а также определены способы верификации системы.

Первым этапом, определим оценки критериев эффективности разрабатываемой системы хранения данных. Первым критерием эффективности, является критерий объема памяти. Для того, чтобы его определить нам необходимо создать тестовый набор файлов различного формата и объема. Описание тестового набора сведено в таблице 3.1.

Таблица 3.1 – Описание тестового набора данных

Формат	Количество (Шт)	Занимаемый объем (МБ)
.xml	1200	3,7
.pdf	30	95,3
.jpeg	30	32,6
.docx	150	111,3
.xslx	250	15,4

Общий объем данных составляет 258,3 МБ. Далее с помощью программного обеспечения Postman, загрузим эти данные в систему хранения данных. После загрузки всего тестового набора, занимаемый объем памяти на «холодном» хранилище составил 211,7 МБ. Будем использовать формулу (2.4) для расчета оценки критерия эффективности и получим следующий результат (3.1)

$$K_c = \frac{221,7}{258,3} = 0,86. \quad (3.1)$$

Так как оценка критерия эффективности меньше единицы, будем считать, что разработанная система хранения данных оптимизирована по критерию объема памяти.

Для получения оценки критерия надежности хранения, необходимо воспользоваться формулой (2.5). Вероятность отказа дискового носителя информации в течении первых трёх лет, имеет среднее значение 0,2, коэффициент репликации равняется двум. Получим следующий результат (3.2).

$$K_r = 1 - 0,2^2 = 0,96 \quad (3.2)$$

В силу того, что оценка критерия надежности хранения больше рассчитанного ранее, будем считать, что разработанная система хранения данных также оптимизирована по критерию надежности хранения.

Следующим этапом, необходимо проверить работу системы в случае выхода из строя одного из подключенных хранилищ. В таком случае, подсистема управления данными должна, при следующей проверке доступности хранилищ, пометить его как отключенное и разместить недостающие копии объектов на других доступных «холодных» хранилищах.

Для демонстрации работы системы, будем использовать программное обеспечение DBeaver (клиент баз данных), показывая, как меняются таблицы при работе.

На рисунке 3.7 представлен скриншот содержимого таблицы Storages.

	name	type	state	capacity	capacity_free	created_at	updated_at
1	cold-1	3	1	10 485 760	10 175 488	2021-01-05 10:33:35	2021-01-05 10:33:35
2	cold-2	3	1	10 485 760	10 281 984	2021-01-05 10:34:05	2021-01-05 10:34:05
3	cold-3	3	1	10 485 760	10 386 432	2021-01-05 10:34:09	2021-01-05 10:34:09
4	warm-1	2	1	5 485 760	4 122 612	2021-01-05 10:33:29	2021-01-05 10:33:29
5	hot-1	1	1	512 000	397 346	2021-01-05 10:33:28	2021-01-05 10:33:28

Рисунок 3.7 – Таблица Storages до отключения хранилища

Исходя из рисунка 3.7 видно, что все хранилища имеют состояние «запись и чтение», все они доступны. Также необходимо выбрать объект и наблюдать за его списоком реплик, так как он должен будет измениться после отключения хранилища. На рисунке 3.8 представлен скриншот списка реплик для объекта с идентификатором 743.

	name	object_id	is_deleted	created_at	updated_at
1	cold-1	743	[ ]	2021-01-05 11:07:01	2021-01-05 11:07:01
2	cold-3	743	[ ]	2021-01-05 13:07:10	2021-01-05 13:07:10
3	warm-1	743	[X]	2021-01-05 12:07:05	2021-01-05 13:07:05
4	hot-1	743	[X]	2021-01-05 11:07:01	2021-01-05 12:07:01

Рисунок 3.8 – Список реплик объекта до отключения хранилища

Данные объекта расположены на хранилищах с именем cold-1 и cold-3, для целей тестирования отключим хранилище cold-3. Подсистема управления данными

должна будет её исключить из системы. На рисунке 3.9 представлен скриншот содержимого таблицы Storages после отключения хранилища cold-3.

	name	type	state	capacity	capacity_free	created_at	updated_at
1	cold-1	3	1	10 485 760	10 175 488	2021-01-05 10:33:35	2021-01-05 14:02:17
2	cold-2	3	1	10 485 760	10 281 984	2021-01-05 10:34:05	2021-01-05 14:02:17
3	cold-3	3	3	10 485 760	10 386 432	2021-01-05 10:34:09	2021-01-05 14:02:17
4	warm-1	2	1	5 485 760	4 122 612	2021-01-05 10:33:29	2021-01-05 14:02:17
5	hot-1	1	1	512 000	397 346	2021-01-05 10:33:28	2021-01-05 14:02:17

Рисунок 3.9 – Таблица Storages после отключения хранилища

Из рисунка 3.9 видно, что хранилище cold-3 поменяло свое состояние на «отключено», следовательно, подсистема управления данным отработала корректно. На рисунке 3.10 представлен скриншот демонстрирующий список реплик для объекта с идентификатором 743, после отключения хранилища cold-3.

	name	object_id	is_deleted	created_at	updated_at
1	cold-1	743	[ ]	2021-01-05 11:07:01	2021-01-05 11:07:01
2	cold-2	743	[ ]	2021-01-05 14:02:33	2021-01-05 14:02:33
3	cold-3	743	[X]	2021-01-05 13:07:10	2021-01-05 13:07:10
4	warm-1	743	[X]	2021-01-05 12:07:05	2021-01-05 13:07:05
5	hot-1	743	[X]	2021-01-05 11:07:01	2021-01-05 12:07:01

Рисунок 3.10 – Список реплик объекта после отключения хранилища

Рисунок 3.10 демонстрирует, что после отключения хранилища cold-3 в немедленно порядке была создана копия объекта в хранилище cold-2.

Далее необходимо верифицировать механизм перераспределения данных в соответствии с их температурой. Для этого предварительно уменьшим коэффициент старения данных, будем считать, что данные считаются старыми после 10 минут пребывания в системе. Также уменьшим интервал проверки температуры для объекта до 1 минуты.

Для начала необходимо добавить новый объект в систему хранения данных, также воспользуемся программной Postman. На рисунке 3.11 представлен скриншот отправки запроса в подсистему доступа к данным.

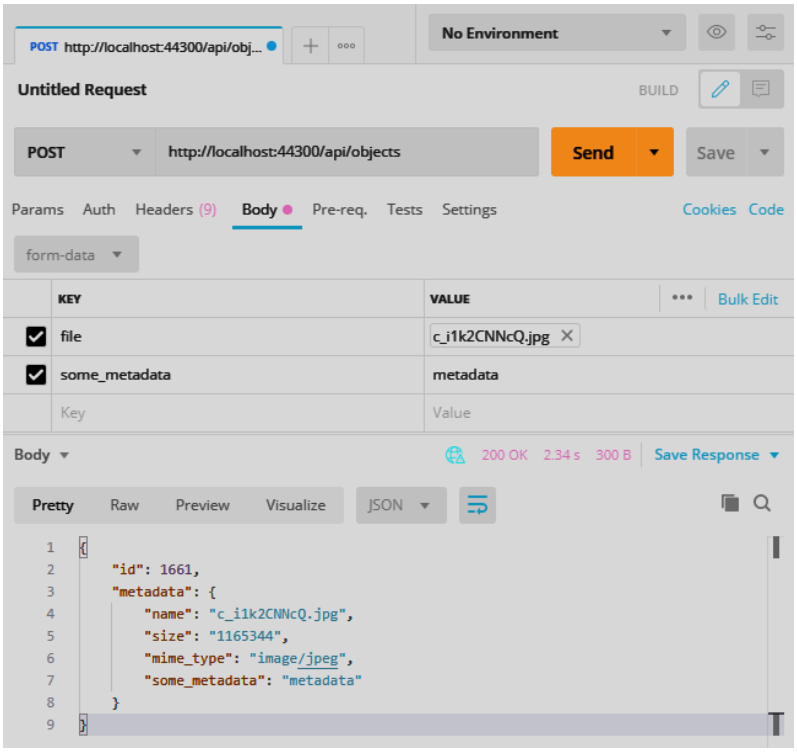


Рисунок 3.11 – Добавление нового объекта в систему

Рисунок 3.11 демонстрирует успешное добавление нового объекта в систему, новому объекту присвоен идентификатор 1661. На рисунке 3.12 представлен скриншот вывода таблицы Objects по идентификатору 1661.

id	temperature	created_at	updated_at	is_deleted
1 661	1	2021-01-05 16:21:55	2021-01-05 16:21:55	[ ]

Рисунок 3.12 – Содержимое таблицы Objects

Из рисунка 3.12 следует, что объект имеет «горячую» температуру. На рисунке 3.13 скриншот списка реплик для объекта с идентификатором 1661.

name	object_id	is_deleted	created_at	updated_at
hot-1	1 661	[ ]	2021-01-05 16:21:55	2021-01-05 16:21:55
cold-2	1 661	[ ]	2021-01-05 16:21:55	2021-01-05 16:21:55

Рисунок 3.13 – Список реплик объекта

Подсистема доступа к данным отработала корректно, новые объекты размещаются, как и полагается – в «горячем» и «холодном» хранилище. Необходимо подождать около 3-рех минут и также проверим состояния этих же таблиц. На рисунках 3.14 и 3.15 представлены скриншоты таблиц, по истечению 3-рех минут.

id	temperature	created_at	updated_at	is_deleted
1 661	2	2021-01-05 16:21:55	2021-01-05 16:25:11	[ ]

Рисунок 3.14 – Таблица Objects после 3-рех минут

name	object_id	is_deleted	created_at	updated_at
hot-1	1 661	[X]	2021-01-05 16:21:55	2021-01-05 16:25:11
cold-2	1 661	[ ]	2021-01-05 16:21:55	2021-01-05 16:21:55
warm-1	1 661	[ ]	2021-01-05 16:25:11	2021-01-05 16:25:11

Рисунок 3.15 – Список реплик объекта после 3-рех минут

По истечению 3-рех минут, как и ожидалось, объект «остыл» и переместился в «теплое» хранилище, освободив при этом место на «горячем». Будем ждать еще 7 минут, чтобы объект окончательно остыл. На рисунках 3.16 и 3.17 представлены скриншоты таблиц, по истечению 10-ти минут с момента добавления объекта в систему.

id	temperature	created_at	updated_at	is_deleted
1 661	3	2021-01-05 16:21:55	2021-01-05 16:32:27	[ ]

Рисунок 3.16 – Таблица Objects после 10-ти минут

name	object_id	is_deleted	created_at	updated_at
cold-2	1 661	[ ]	2021-01-05 16:21:55	2021-01-05 16:21:55
cold-3	1 661	[ ]	2021-01-05 16:32:27	2021-01-05 16:32:27
warm-1	1 661	[X]	2021-01-05 16:25:11	2021-01-05 16:32:27
hot-1	1 661	[X]	2021-01-05 16:21:55	2021-01-05 16:25:11

Рисунок 3.17 – Список реплик объекта после 10-ти минут

Анализируя рисунки 3.16 и 3.17 делаем вывод о том, что механизм распределения данных в соответствии с температурой работает исправно.

Выводы по разделу 3

В текущем разделе были выбраны средства разработки, необходимые для реализации системы хранения данных оператора электронного документооборота.

На основе ранее разработанной архитектуры распределенной системы хранения данных, была разработана структура данных, а также реализованы следующие программные модули: подсистема доступа к данным, подсистема управления данными, подсистемы доступа к «горячим», «теплым» и «холодным» данным. Для каждой подсистемы были описаны интерфейсы взаимодействия.

Помимо реализации системы хранения данных также была проведена верификация. В ходе верификации системы хранения данных, были рассчитаны оценки критериев эффективности, которые показали, что полученную систему можно считать оптимизированной. Помимо прочего было проведено тестирование системы хранения данных в нештатной ситуации, а именно – выход из строя одного из подключенных хранилищ. Результаты данного тестирования показали, что система хранения данных остается доступной при выходе из строя одного из хранилищ, хранимые данные не были утеряны. Также было проведено тестирование механизма распределения данных в соответствии с «температурой».

ЗАКЛЮЧЕНИЕ

В ходе выполнения выпускной квалификационной работы была решена задача, включающая в себя анализ методов оптимизации распределенного хранилища данных для систем электронного документооборота. В ходе ее решения были рассмотрены существующие подходы к организации систем хранения данных, рассмотрена предметная область систем электронного документооборота, а также были рассмотрены методы оптимизации хранения данных.

На основе проанализированной предметной области и методов оптимизации хранения данных была разработана распределенная архитектура системы хранения данных, позволяющая снизить денежные затраты для увеличения объема хранилища, а также отвечающей высоким требования надежности, безопасности и соответствия по требованиям производительности. Система хранения данных была декомпозирована на следующие подсистемы: подсистема доступа к данным, подсистема управления данными, подсистема доступа к хранилищам данных. Для каждой подсистем были подробно описаны алгоритмы функционирования и их интерфейсы.

Предложенная распределенная архитектура системы хранения данных позволило выполнить разработку подсистем и осуществить их верификацию и тестирование. Результаты верификации и тестирования убедительно показывают, что система хранения данных удовлетворяет заданным требованиям.

Таким образом, решение поставленных задач позволило достичь цели выпускной квалификационной работы, а именно создать оптимизированное распределенное хранилище данных для системы электронного документооборота.

Дальнейшее усовершенствование системы хранения данных может быть осуществлено путем добавления машинного обучения в работу подсистемы управления данными при распределении данных по хранилищам. Данное решение позволит эффективней использовать объемы памяти хранилища и повысит скорость работы системы.

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И УСЛОВНЫХ ОБОЗНАЧЕНИЙ

СЭД – Система Электронного Документооборота

СХД – Система Хранения Данных

ЭДО – Электронный Документооборот

ОС – Операционная Система

СУБД – Система Управления Базой Данных

DAS – Direct Attached Storage

NAS – Network Attached Storage

SAN – Storage Area Network

ЗУ – Запоминающее Устройство

ОЗУ – Оперативное Запоминающее устройство

HDD – Hard Disk Drive

SSD – Solid-State Drive

IOPS – Input/Output Per Second

RAID – Redundant Array of Independent Disks

ЦОД – Центр Обработки Данных

SDS – Software-Defined Storage

CRM – Customer Relationship Management

ЭЦП – Электронная Цифровая Подпись

XML – eXtensible Markup Language

RPO – Recovery Point Objective

RTO – Recovery Time Objective

LRU – Least Recently Used

MRU – Most Recently Used

LFU – Least-Frequently Used

SLRU – Segmented Least Recently Used

AES – Advanced Encryption Standard

RSA – Rivest Shalmir Adleman

ACL – Access Control List

CLR – Common Language Runtime

LINQ – Language Integrated Query

ORM – Object-Relational Mapping

SQL – Structured Query Language

ACID – Atomicity Consistency Isolation Durability

AMQP – Advanced Message Queuing Protocol

REST – Representational State Transfer

API – Application Programming Interface

CRUD – Create Read Update Delete

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Понятие системы хранения данных [Электронный ресурс]. URL: https://itglobal.com/ru-ru/company/glossary/shd-sistema-hraneniya-dannyh/ (дата обращения 14.01.2021);

2. Богановский А.В. Анализ методов построения систем хранения данных // Перспективы развития информационных технологий. Сборник материалов XXXV Международной научно-практической конференции. – Новосибирск, 2017. – С. 13-17.;

3. И.В. Савин. Анализ систем хранения данных // Журнал «Известия Тульского государственного университета. Технические Науки». – 2019. – С. 193-196;

4. Запоминающие устройство [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/Запоминающее_устройство (дата обращения 14.01.2021)

5. Джиблазде З.Г., Гасюк К.В., Садыков Д.К. SSD и HDD // Журнал «Вестник современных исследований». – 2020. – С. 4-6;

6. Исалёв А.С. Анализ надежности технологии хранения данных RAID // Журнал «Актуальные проблемы авиации и космонавтики». – 2016. – С. 959-961;

7. Шувалов Н.В. Технология RAID // Журнал «Аллея науки». – 2017. – С. 825-828;

8. Distributed Storage: What’s Inside Amazon S3? [Электронный ресурс]. URL: https://cloudian.com/guides/data-backup/distributed-storage/ (дата обращения 14.01.2021);

9. What is software-defined storage? [Электронный ресурс]. URL: https://www.redhat.com/en/topics/data-storage/software-defined-storage (дата обращения 14.01.2021)

10. Understanding distributed data storage [Электронный ресурс]. URL: https://www.googlinux.com/understanding-distributed-data-storage/ (дата обращения 14.01.2021);

11. Мазур Э.М. Распределенные системы хранения данных: анализ, классификация и выбор // Журнал «Перспективы развития информационных технологий». – 2015. – С. 33-60;

12. Облачные хранилища данных [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/Облачное_хранилище_данных (дата обращения 14.01.2021);

13. Margaret Rouse. Cloud Storage [Электронный ресурс]. URL: https://searchstorage.techtarget.com/definition/cloud-storage (дата обращения 14.01.2021);

14. Документация облачного хранилища Amazon S3 [Электронный ресурс]. URL: https://aws.amazon.com/ru/s3/ (дата обращения 14.01.2021);

15. Архитектура S3: 3 года эволюции Mail.ru Cloud Storage [Электронный ресурс]. URL: https://habr.com/ru/company/mailru/blog/513356/ (дата обращения 14.01.2021);

16. Документация облачного хранилища Mail Cloud Storage [Электронный ресурс]. URL: https://mcs.mail.ru/storage/ (дата обращения 14.01.2021);

17. Федеральный закон «О персональных данных» от 27.07.2005 № 152-ФЗ [Электронный ресурс]. URL: http://www.consultant.ru/document/cons_doc_LAW_61801/ (дата обращения 14.01.2021);

18. Документация облачного хранилища Yandex Object Storage [Электронный ресурс]. URL: https://cloud.yandex.ru/services/storage (дата обращения 14.01.2021);

19. Электронный документооборот как способ оптимизации бизнес-процессов [Электронный ресурс]. URL: https://www.kp.ru/guide/ielektronnyi-dokumentooborot-na-predprijatii.html (дата обращения 14.01.2021);

20. Приказ ФНС РФ от 23 октября 2020 г. № ЕД-7-26/775@ [Электронный ресурс]. URL: http://www.consultant.ru/cons/cgi/online.cgi?req=doc&base=LAW&n=366073&dst=100000000001&date=28.11.2020#09771917765017271 (дата обращения 14.01.2021);

21. Comparative data compression techniques and multi-compression result [Электронный ресурс]. URL: https://www.researchgate.net/publication/261014704_Comparative_data_compression_techniques_and_multi-compression_results (дата обращения 14.01.2021);

22. Введение в дедупликацию данных [Электронный ресурс]. URL: https://habr.com/ru/company/veeam/blog/203614/ (дата обращения 14.01.2021);

23. Эффективное кеширование. От теории к практике [Электронный ресурс]. URL: https://habr.com/ru/company/surfingbird/blog/306252/ (дата обращения 14.01.2021);

24. N. Martinez. HTTP Vs. FTP File Transfer [Электронный ресурс]. URL: https://smallbusiness.chron.com/http-vs-ftp-file-transfer-54226.html (дата обращения 14.01.2021);

25. A. Simran. C# vs Java: Differences you should Know [Электронный ресурс]. URL: https://hackr.io/blog/c-sharp-vs-java (дата обращения 14.01.2021);

26. D. Roth, R. Anderson, S. Luttin. Introduction to ASP.NET Core [Электронный ресурс]. URL: https://docs.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-5.0 (дата обращения 14.01.2021);

27. Entity Framework documentation [Электронный ресурс]. URL: https://docs.microsoft.com/en-us/ef/ (дата обращения 14.01.2021);

28. J. Training. Difference between Oracle, SQL Server, MySQL and PostgreSQL [Электронный ресурс]. URL: https://www.janbasktraining.com/blog/oracle-sql-server-mysql-and-postgresql/ (дата обращения 14.01.2021);

29. B. Mcclain. Understanding the Differences Between RabbitMQ vs Kafka [Электронный ресурс]. URL: https://tanzu.vmware.com/developer/blog/understanding-the-differences-between-rabbitmq-vs-kafka/ (дата обращения 14.01.2021);

30. И. Чумаков. Hangfire – планировщик задач для .NET [Электронный ресурс]. URL: https://habr.com/ru/post/280732/ (дата обращения 14.01.2021);

31. REST [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/REST (дата обращения 14.01.2021).

ПРИЛОЖЕНИЕ А

Листинг реализованных подсистем

```
namespace Infrastructure.Database
{
    public class FsContext : DbContext
    {
        public FsContext(DbContextOptions<FsContext> options) : base(options) { }

        public DbSet<Object> Objects { get; set; }
        public DbSet<Storage> Storages { get; set; }
        public DbSet<ObjectReplica> ObjectReplicas { get; set; }
        public DbSet<ObjectMetadata> ObjectMetadata { get; set; }
        public DbSet<ObjectEvent> ObjectEvents { get; set; }

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            modelBuilder.ApplyConfigurationsFromAssembly(Assembly.GetExecutingAssembly());
        }
    }
}
namespace Infrastructure.Database.Entities
{
    public class BaseEntity
    {
        public long Id { get; set; }
    }
}
namespace Infrastructure.Database.Entities
{
    public interface IDeleteableEntity
    {
        bool IsDeleted { get; set; }
    }
}
namespace Infrastructure.Database.Entities
{
    public interface IEntityWithCreatedDate
    {
        DateTime CreatedAt { get; set; }
    }
}
namespace Infrastructure.Database.Entities
{
    public interface IEntityWithUpdatedDate
    {
        DateTime UpdatedAt { get; set; }
    }
}
namespace Infrastructure.Database.Entities
{
    public class Object : BaseEntity, IEntityWithCreatedDate, IEntityWithUpdatedDate,
IDeleteableEntity
    {
        public virtual ICollection<ObjectReplica> Replicas { get; set; }
        public virtual ICollection<ObjectMetadata> Metadata { get; set; }
```

```
        public virtual ICollection<ObjectEvent> Events { get; set; }
        public TemperatureType Temperature {get;set;}
        public DateTime CreatedAt { get; set; }
        public DateTime UpdatedAt { get; set; }


        public bool IsDeleted { get; set; }
    }
}
namespace Infrastructure.Database.Entities
{
    public class ObjectEvent : BaseEntity, IEntityWithCreatedDate
    {
        public long ObjectId { get; set; }
        public virtual Object Object { get; set; }

        public ObjectEventType Type { get; set; }

        public DateTime CreatedAt { get; set; }
    }
}
namespace Infrastructure.Database.Entities
{
    public class ObjectMetadata : BaseEntity
    {
        public long ObjectId { get; set; }
        public virtual Object Object { get; set; }

        public string Name { get; set; }
        public string Value { get; set; }
    }
}
namespace Infrastructure.Database.Entities
{
    public class ObjectReplica : BaseEntity, IDeleteableEntity, IEntityWithCreatedDate,
IEntityWithUpdatedDate
    {
        public long ObjectId { get; set; }
        public virtual Object Object { get; set; }

        public long StorageId { get; set; }
        public virtual Storage Storage { get; set; }

        public bool IsDeleted { get; set; }
        public DateTime CreatedAt { get; set; }
        public DateTime UpdatedAt { get; set; }
    }
}
namespace Infrastructure.Database.Entities
{
    public class Storage : BaseEntity, IEntityWithCreatedDate, IEntityWithUpdatedDate
    {
        public string Name { get; set; }
        public StorageType Type { get; set; }
        public StorageState State { get; set; }
        public int Capacity { get; set; }
        public int CapacityFree { get; set; }
        public string BaseUrl { get; set; }

        public virtual ICollection<ObjectReplica> Objects { get; set; }
```

```
        public DateTime CreatedAt { get; set; }
        public DateTime UpdatedAt { get; set; }
    }
}
namespace Infrastructure.Database.Enums
{
    public enum ObjectEventType
    {
        Created = 0,
        Read = 1,
        Deleted = 2
    }
}
namespace Infrastructure.Database.Enums
{
    public enum StorageState
    {
        Disabled = 0,
        ReadAndWrite = 1,
        OnlyRead = 2
    }
}
namespace Infrastructure.Database.Enums
{
    public enum StorageType
    {
        Unknown = 0,
        Hot = 1,
        Warm = 2,
        Cold = 3
    }
}
namespace Infrastructure.Database.Enums
{
    public enum TemperatureType
    {
        Unknown = 0,
        Hot = 1,
        Warm = 2,
        Cold = 3
    }
}

namespace Infrastructure.Database.EntityConfigurations
{
    public class ObjectConfiguration : IEntityTypeConfiguration<Object>
    {
        public void Configure(EntityTypeBuilder<Object> builder)
        {
            builder.HasKey(x => x.Id);
            builder.Property(x => x.Id).HasColumnName("id");
            builder.Property(x => x.IsDeleted).HasColumnName("is_deleted");
            builder.Property(x => x.CreatedAt).HasColumnName("created_at");
            builder.Property(x => x.UpdatedAt).HasColumnName("updated_at");
            builder.Property(x => x.Temperature).HasColumnName("temperature");


            builder.HasMany(x  =>  x.Replicas).WithOne(x  =>  x.Object).HasForeignKey(x  =>
x.ObjectId);
            builder.HasMany(x  =>  x.Metadata).WithOne(x  =>  x.Object).HasForeignKey(x  =>
x.ObjectId);
```

```
            builder.HasMany(x => x.Events).WithOne(x => x.Object).HasForeignKey(x => x.ObjectId);


        }
    }
}
namespace Infrastructure.Database.EntityConfigurations
{
    public class ObjectEventConfiguration : IEntityTypeConfiguration<ObjectEvent>
    {
        public void Configure(EntityTypeBuilder<ObjectEvent> builder)
        {
            builder.HasKey(x => x.Id);
            builder.Property(x => x.Id).HasColumnName("id");
            builder.Property(x => x.CreatedAt).HasColumnName("created_at");

            builder.Property(x => x.ObjectId).HasColumnName("object_id");
            builder.Property(x => x.Type).HasColumnName("type");
        }
    }
}
namespace Infrastructure.Database.EntityConfigurations
{
    public class ObjectMetadataConfiguration : IEntityTypeConfiguration<ObjectMetadata>
    {
        public void Configure(EntityTypeBuilder<ObjectMetadata> builder)
        {
            builder.HasKey(x => x.Id);
            builder.Property(x => x.Id).HasColumnName("id");

            builder.Property(x => x.ObjectId).HasColumnName("object_id");
            builder.Property(x => x.Name).HasMaxLength(1024).HasColumnName("name");
            builder.Property(x => x.Value).HasMaxLength(1024).HasColumnName("value");
        }
    }
}
namespace Infrastructure.Database.EntityConfigurations
{
    public class ObjectReplicaConfiguration : IEntityTypeConfiguration<ObjectReplica>
    {
        public void Configure(EntityTypeBuilder<ObjectReplica> builder)
        {
            builder.HasKey(x => x.Id);
            builder.Property(x => x.Id).HasColumnName("id");
            builder.Property(x => x.IsDeleted).HasColumnName("is_deleted");
            builder.Property(x => x.CreatedAt).HasColumnName("created_at");
            builder.Property(x => x.UpdatedAt).HasColumnName("updated_at");


            builder.Property(x => x.ObjectId).HasColumnName("object_id");
            builder.Property(x => x.StorageId).HasColumnName("storage_id");
        }
    }
}
namespace Infrastructure.Database.EntityConfigurations
{
    public class StorageConfiguration : IEntityTypeConfiguration<Storage>
    {
        public void Configure(EntityTypeBuilder<Storage> builder)
        {
```

```
            builder.HasKey(x => x.Id);
            builder.Property(x => x.Id).HasColumnName("id");
            builder.Property(x => x.CreatedAt).HasColumnName("created_at");
            builder.Property(x => x.UpdatedAt).HasColumnName("updated_at");

            builder.HasMany(x  =>  x.Objects).WithOne(x  =>  x.Storage).HasForeignKey(x  =>
x.StorageId);

            builder.Property(x => x.Name).HasMaxLength(1024).HasColumnName("name");
            builder.Property(x => x.Type).HasColumnName("type");
            builder.Property(x => x.State).HasColumnName("state");
            builder.Property(x => x.Capacity).HasColumnName("capacity");
            builder.Property(x => x.CapacityFree).HasColumnName("capacity_free");
            builder.Property(x => x.BaseUrl).HasMaxLength(1024).HasColumnName("base_url");
        }
    }
}
namespace Infrastructure.Database.Migrations
{
    public partial class Initial : Migration
    {
        protected override void Up(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.CreateTable(
                name: "Objects",
                columns: table => new
                {
                    id = table.Column<long>(type: "bigint", nullable: false)
                        .Annotation("Npgsql:ValueGenerationStrategy",
NpgsqlValueGenerationStrategy.IdentityByDefaultColumn),
                    temperature = table.Column<int>(type: "integer", nullable: false),

                    is_deleted = table.Column<bool>(type: "boolean", nullable: false)
                    created_at  =  table.Column<DateTime>(type:  "timestamp  without  time
zone", nullable: false),
                    updated_at  =  table.Column<DateTime>(type:  "timestamp  without  time
zone", nullable: false)

                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_Objects", x => x.id);
                });

            migrationBuilder.CreateTable(
                name: "Storages",
                columns: table => new
                {
                    id = table.Column<long>(type: "bigint", nullable: false)
                        .Annotation("Npgsql:ValueGenerationStrategy",
NpgsqlValueGenerationStrategy.IdentityByDefaultColumn),
                    name = table.Column<string>(type: "character varying(1024)", maxLength:
1024, nullable: true),
                    type = table.Column<int>(type: "integer", nullable: false),
                    state = table.Column<int>(type: "integer", nullable: false),
                    capacity = table.Column<int>(type: "integer", nullable: false),
                    capacity_free = table.Column<int>(type: "integer", nullable: false),
                    base_url   =   table.Column<string>(type:   "character   varying(1024)",
maxLength: 1024, nullable: true),
                    created_at  =  table.Column<DateTime>(type:  "timestamp  without  time
zone", nullable: false),
```

```
                    updated_at  =  table.Column<DateTime>(type:  "timestamp  without  time
zone", nullable: false)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_Storages", x => x.id);
                });

            migrationBuilder.CreateTable(
                name: "ObjectEvents",
                columns: table => new
                {
                    id = table.Column<long>(type: "bigint", nullable: false)
                        .Annotation("Npgsql:ValueGenerationStrategy",
NpgsqlValueGenerationStrategy.IdentityByDefaultColumn),
                    object_id = table.Column<long>(type: "bigint", nullable: false),
                    type = table.Column<int>(type: "integer", nullable: false),
                    created_at  =  table.Column<DateTime>(type:  "timestamp  without  time
zone", nullable: false)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_ObjectEvents", x => x.id);
                    table.ForeignKey(
                        name: "FK_ObjectEvents_Objects_object_id",
                        column: x => x.object_id,
                        principalTable: "Objects",
                        principalColumn: "id",
                        onDelete: ReferentialAction.Cascade);
                });

            migrationBuilder.CreateTable(
                name: "ObjectMetadata",
                columns: table => new
                {
                    id = table.Column<long>(type: "bigint", nullable: false)
                        .Annotation("Npgsql:ValueGenerationStrategy",
NpgsqlValueGenerationStrategy.IdentityByDefaultColumn),
                    object_id = table.Column<long>(type: "bigint", nullable: false),
                    name = table.Column<string>(type: "character varying(1024)", maxLength:
1024, nullable: true),
                    value   =   table.Column<string>(type:   "character   varying(1024)",
maxLength: 1024, nullable: true)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_ObjectMetadata", x => x.id);
                    table.ForeignKey(
                        name: "FK_ObjectMetadata_Objects_object_id",
                        column: x => x.object_id,
                        principalTable: "Objects",
                        principalColumn: "id",
                        onDelete: ReferentialAction.Cascade);
                });

            migrationBuilder.CreateTable(
                name: "ObjectReplicas",
                columns: table => new
                {
                    id = table.Column<long>(type: "bigint", nullable: false)
```

```
                        .Annotation("Npgsql:ValueGenerationStrategy",
NpgsqlValueGenerationStrategy.IdentityByDefaultColumn),
                    object_id = table.Column<long>(type: "bigint", nullable: false),
                    storage_id = table.Column<long>(type: "bigint", nullable: false),
                    is_deleted = table.Column<bool>(type: "boolean", nullable: false),
                    created_at  =  table.Column<DateTime>(type:  "timestamp  without  time
zone", nullable: false),
                    updated_at  =  table.Column<DateTime>(type:  "timestamp  without  time
zone", nullable: false)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_ObjectReplicas", x => x.id);
                    table.ForeignKey(
                        name: "FK_ObjectReplicas_Objects_object_id",
                        column: x => x.object_id,
                        principalTable: "Objects",
                        principalColumn: "id",
                        onDelete: ReferentialAction.Cascade);
                    table.ForeignKey(
                        name: "FK_ObjectReplicas_Storages_storage_id",
                        column: x => x.storage_id,
                        principalTable: "Storages",
                        principalColumn: "id",
                        onDelete: ReferentialAction.Cascade);
                });

            migrationBuilder.CreateIndex(
                name: "IX_ObjectEvents_object_id",
                table: "ObjectEvents",
                column: "object_id");

            migrationBuilder.CreateIndex(
                name: "IX_ObjectMetadata_object_id",
                table: "ObjectMetadata",
                column: "object_id");

            migrationBuilder.CreateIndex(
                name: "IX_ObjectReplicas_object_id",
                table: "ObjectReplicas",
                column: "object_id");

            migrationBuilder.CreateIndex(
                name: "IX_ObjectReplicas_storage_id",
                table: "ObjectReplicas",
                column: "storage_id");
        }

        protected override void Down(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.DropTable(
                name: "ObjectEvents");

            migrationBuilder.DropTable(
                name: "ObjectMetadata");

            migrationBuilder.DropTable(
                name: "ObjectReplicas");

            migrationBuilder.DropTable(
                name: "Objects");
```

```
            migrationBuilder.DropTable(
                name: "Storages");
        }
    }
}
namespace Infrastructure.Database.Migrations
{
    [DbContext(typeof(FsContext))]
    partial class FsContextModelSnapshot : ModelSnapshot
    {
        protected override void BuildModel(ModelBuilder modelBuilder)
        {
#pragma warning disable 612, 618
            modelBuilder
                .UseIdentityByDefaultColumns()
                .HasAnnotation("Relational:MaxIdentifierLength", 63)
                .HasAnnotation("ProductVersion", "5.0.1");

            modelBuilder.Entity("Infrastructure.Database.Entities.Object", b =>
                {
                    b.Property<long>("Id")
                        .ValueGeneratedOnAdd()
                        .HasColumnType("bigint")
                        .HasColumnName("id")
                        .UseIdentityByDefaultColumn();

                    b.Property<int>("Temperature")
                        .HasColumnType("integer")
                        .HasColumnName("state");

                    b.Property<bool>("IsDeleted")
                        .HasColumnType("boolean")
                        .HasColumnName("is_deleted");

                    b.Property<DateTime>("CreatedAt")
                        .HasColumnType("timestamp without time zone")
                        .HasColumnName("created_at");

                    b.Property<DateTime>("UpdatedAt")
                        .HasColumnType("timestamp without time zone")
                        .HasColumnName("updated_at");



                    b.HasKey("Id");

                    b.ToTable("Objects");
                });

            modelBuilder.Entity("Infrastructure.Database.Entities.ObjectEvent", b =>
                {
                    b.Property<long>("Id")
                        .ValueGeneratedOnAdd()
                        .HasColumnType("bigint")
                        .HasColumnName("id")
                        .UseIdentityByDefaultColumn();

                    b.Property<DateTime>("CreatedAt")
                        .HasColumnType("timestamp without time zone")
                        .HasColumnName("created_at");
```

```
            b.Property<long>("ObjectId")
                .HasColumnType("bigint")
                .HasColumnName("object_id");

            b.Property<int>("Type")
                .HasColumnType("integer")
                .HasColumnName("type");

            b.HasKey("Id");

            b.HasIndex("ObjectId");

            b.ToTable("ObjectEvents");
        });

    modelBuilder.Entity("Infrastructure.Database.Entities.ObjectMetadata", b =>
        {
            b.Property<long>("Id")
                .ValueGeneratedOnAdd()
                .HasColumnType("bigint")
                .HasColumnName("id")
                .UseIdentityByDefaultColumn();

            b.Property<string>("Name")
                .HasMaxLength(1024)
                .HasColumnType("character varying(1024)")
                .HasColumnName("name");

            b.Property<long>("ObjectId")
                .HasColumnType("bigint")
                .HasColumnName("object_id");

            b.Property<string>("Value")
                .HasMaxLength(1024)
                .HasColumnType("character varying(1024)")
                .HasColumnName("value");

            b.HasKey("Id");

            b.HasIndex("ObjectId");

            b.ToTable("ObjectMetadata");
        });

    modelBuilder.Entity("Infrastructure.Database.Entities.ObjectReplica", b =>
        {
            b.Property<long>("Id")
                .ValueGeneratedOnAdd()
                .HasColumnType("bigint")
                .HasColumnName("id")
                .UseIdentityByDefaultColumn();

            b.Property<DateTime>("CreatedAt")
                .HasColumnType("timestamp without time zone")
                .HasColumnName("created_at");

            b.Property<bool>("IsDeleted")
                .HasColumnType("boolean")
                .HasColumnName("is_deleted");
```

```
            b.Property<long>("ObjectId")
                .HasColumnType("bigint")
                .HasColumnName("object_id");

            b.Property<long>("StorageId")
                .HasColumnType("bigint")
                .HasColumnName("storage_id");

            b.Property<DateTime>("UpdatedAt")
                .HasColumnType("timestamp without time zone")
                .HasColumnName("updated_at");

            b.HasKey("Id");

            b.HasIndex("ObjectId");

            b.HasIndex("StorageId");

            b.ToTable("ObjectReplicas");
        });

    modelBuilder.Entity("Infrastructure.Database.Entities.Storage", b =>
        {
            b.Property<long>("Id")
                .ValueGeneratedOnAdd()
                .HasColumnType("bigint")
                .HasColumnName("id")
                .UseIdentityByDefaultColumn();

            b.Property<string>("BaseUrl")
                .HasMaxLength(1024)
                .HasColumnType("character varying(1024)")
                .HasColumnName("base_url");

            b.Property<int>("Capacity")
                .HasColumnType("integer")
                .HasColumnName("capacity");

            b.Property<int>("CapacityFree")
                .HasColumnType("integer")
                .HasColumnName("capacity_free");

            b.Property<DateTime>("CreatedAt")
                .HasColumnType("timestamp without time zone")
                .HasColumnName("created_at");

            b.Property<string>("Name")
                .HasMaxLength(1024)
                .HasColumnType("character varying(1024)")
                .HasColumnName("name");

            b.Property<int>("State")
                .HasColumnType("integer")
                .HasColumnName("state");

            b.Property<int>("Type")
                .HasColumnType("integer")
                .HasColumnName("type");

            b.Property<DateTime>("UpdatedAt")
                .HasColumnType("timestamp without time zone")
```

```
            .HasColumnName("updated_at");

        b.HasKey("Id");

        b.ToTable("Storages");
    });

modelBuilder.Entity("Infrastructure.Database.Entities.ObjectEvent", b =>
    {
        b.HasOne("Infrastructure.Database.Entities.Object", "Object")
            .WithMany("Events")
            .HasForeignKey("ObjectId")
            .OnDelete(DeleteBehavior.Cascade)
            .IsRequired();

        b.Navigation("Object");
    });

modelBuilder.Entity("Infrastructure.Database.Entities.ObjectMetadata", b =>
    {
        b.HasOne("Infrastructure.Database.Entities.Object", "Object")
            .WithMany("Metadata")
            .HasForeignKey("ObjectId")
            .OnDelete(DeleteBehavior.Cascade)
            .IsRequired();

        b.Navigation("Object");
    });

modelBuilder.Entity("Infrastructure.Database.Entities.ObjectReplica", b =>
    {
        b.HasOne("Infrastructure.Database.Entities.Object", "Object")
            .WithMany("Replicas")
            .HasForeignKey("ObjectId")
            .OnDelete(DeleteBehavior.Cascade)
            .IsRequired();

        b.HasOne("Infrastructure.Database.Entities.Storage", "Storage")
            .WithMany("Objects")
            .HasForeignKey("StorageId")
            .OnDelete(DeleteBehavior.Cascade)
            .IsRequired();

        b.Navigation("Object");

        b.Navigation("Storage");
    });

modelBuilder.Entity("Infrastructure.Database.Entities.Object", b =>
    {
        b.Navigation("Events");

        b.Navigation("Metadata");

        b.Navigation("Replicas");
    });

modelBuilder.Entity("Infrastructure.Database.Entities.Storage", b =>
    {
        b.Navigation("Objects");
    });
```

```
#pragma warning restore 612, 618
        }
    }
}

#Infrastructure.Database.csproj
<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    <TargetFramework>netcoreapp3.1</TargetFramework>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="Microsoft.EntityFrameworkCore" Version="5.0.1" />
    <PackageReference Include="Microsoft.EntityFrameworkCore.Tools" Version="5.0.1">
      <PrivateAssets>all</PrivateAssets>
      <IncludeAssets>runtime;      build;      native;      contentfiles;      analyzers;
buildtransitive</IncludeAssets>
    </PackageReference>
    <PackageReference Include="Npgsql.EntityFrameworkCore.PostgreSQL" Version="5.0.1" />
  </ItemGroup>

</Project>

namespace ColdStorageApi
{
    public class Program
    {
        public static void Main(string[] args)
        {
            CreateHostBuilder(args).Build().Run();
        }

        public static IHostBuilder CreateHostBuilder(string[] args) =>
            Host.CreateDefaultBuilder(args)
                .ConfigureWebHostDefaults(webBuilder =>
                {
                    webBuilder.UseStartup<Startup>();
                });
    }
}
namespace ColdStorageApi
{
    public class Startup
    {
        private readonly IConfiguration _configuration;

        public Startup(IConfiguration configuration)
        {
            _configuration = configuration;
        }

        public void ConfigureServices(IServiceCollection services)
        {
            services.AddControllers();
            services.AddDbContext<StorageContext>(opt =>

opt.UseSqlite(_configuration.GetConnectionString("SqliteConnectionString")));
        }

        public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
```

```
        {
            if (env.IsDevelopment())
            {
                app.UseDeveloperExceptionPage();
            }

            app.UseStatusCodePages("text/plain", "Error. Status code : {0}");

            app.UseRouting();

            app.UseEndpoints(endpoints =>
            {
                endpoints.MapControllers();
            });
        }
    }
}

#ColdStorageApi.csproj

<Project Sdk="Microsoft.NET.Sdk.Web">

  <PropertyGroup>
    <TargetFramework>net5.0</TargetFramework>
    <DockerDefaultTargetOS>Linux</DockerDefaultTargetOS>
           </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="Microsoft.EntityFrameworkCore" Version="5.0.1" />
    <PackageReference Include="Microsoft.EntityFrameworkCore.Sqlite" Version="5.0.1" />
    <PackageReference      Include="Microsoft.VisualStudio.Azure.Containers.Tools.Targets"
Version="1.10.9" />
    <PackageReference Include="Serilog" Version="2.10.0" />
    <PackageReference Include="Serilog.Sinks.Elasticsearch" Version="8.4.1" />
  </ItemGroup>

</Project>

namespace ColdStorageApi.Data.Entities
{
    public class StoredFile
    {
        [DatabaseGenerated(DatabaseGeneratedOption.None)]
        public long Id { get; set; }
        public string Path { get; set; }
        public DateTime CreatedAt { get; set; }
    }
}

namespace ColdStorageApi.Data
{
    public sealed class StorageContext : DbContext
    {
        public DbSet<StoredFile> StoredFiles { get; set; }

        public StorageContext(DbContextOptions<StorageContext> options) : base(options)
        {
            Database.EnsureCreated();
        }

        protected override void OnModelCreating(ModelBuilder modelBuilder)
```

```
        {
            base.OnModelCreating(modelBuilder);

            modelBuilder.Entity<StoredFile>().HasKey(x => x.Id);
            modelBuilder.Entity<StoredFile>().Property(x => x.Path)
                .HasMaxLength(2048)
                .IsRequired();
        }
    }
}

namespace ColdStorageApi.Models
{
    public class AddFileModel
    {
        public long Id { get; set; }
        public IFormFile File { get; set; }
    }
}

namespace ColdStorageApi.Controllers
{
    [Route("api/files")]
    [ApiController]
    public class FilesController : ControllerBase
    {
        private readonly StorageContext _storageContext;
        private readonly IConfiguration _configuration;
        private readonly ILogger _logger;

        public FilesController(StorageContext storageContext, IConfiguration configuration,
ILogger logger)
        {
            _storageContext = storageContext;
            _configuration = configuration;
            _logger = logger;
        }

        [HttpGet("{id}")]
        public async Task<IActionResult> Get(long id)
        {
            try
            {
                var storedFile = await _storageContext.StoredFiles.SingleOrDefaultAsync(x
=> x.Id == id);
                if (storedFile == null)
                {
                    return NotFound();
                }

                var fs = new FileStream(storedFile.Path, FileMode.Open, FileAccess.Read);

                return File(fs, "application/octet-stream");
            }
            catch (Exception ex)
            {
                _logger.Error(ex, ex.Message);
                return BadRequest(ex.Message);
            }
        }
```

```
        [HttpPost]
        public async Task<IActionResult> Put(AddFileModel model)
        {
            try
            {
                if (model == null)
                {
                    return BadRequest("Model is missing");
                }

                if (model.File == null || model.File.Length == 0)
                {
                    return BadRequest("File is missing");
                }

                var path = $"{_configuration["StoragePath"]}/{Guid.NewGuid():D}.bin";

                await using (var fileStream = System.IO.File.Create(path))
                {
                    await model.File.CopyToAsync(fileStream);
                }

                await _storageContext.StoredFiles.AddAsync(new StoredFile
                {
                    Id = model.Id,
                    Path = path,
                    CreatedAt = DateTime.UtcNow
                });

                await _storageContext.SaveChangesAsync();

                return Ok();
            }
            catch (Exception ex)
            {
                _logger.Error(ex, ex.Message);
                return BadRequest(ex.Message);
            }
        }

        [HttpDelete("{id}")]
        public async Task<IActionResult> Delete(long id)
        {
            try
            {
                var storedFile = await _storageContext.StoredFiles.SingleOrDefaultAsync(x
=> x.Id == id);
                if (storedFile == null)
                {
                    return NotFound();
                }

                _storageContext.Remove(storedFile);

                await _storageContext.SaveChangesAsync();
            }
            catch (Exception ex)
            {
                _logger.Error(ex, ex.Message);
                return BadRequest(ex.Message);
            }
```

```
            return Ok();
        }
    }
}

#Dockerfile ColdStorageApi
FROM mcr.microsoft.com/dotnet/aspnet:5.0-buster-slim AS base
WORKDIR /app
EXPOSE 80

FROM mcr.microsoft.com/dotnet/sdk:5.0-buster-slim AS build
WORKDIR /src
COPY ["ColdStorageApi/ColdStorageApi.csproj", "ColdStorageApi/"]
RUN dotnet restore "ColdStorageApi/ColdStorageApi.csproj"
COPY . .
WORKDIR "/src/ColdStorageApi"
RUN dotnet build "ColdStorageApi.csproj" -c Release -o /app/build

FROM build AS publish
RUN dotnet publish "ColdStorageApi.csproj" -c Release -o /app/publish

FROM base AS final
WORKDIR /app
COPY --from=publish /app/publish .
ENTRYPOINT ["dotnet", "ColdStorageApi.dll"]
```