Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н. Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н. Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ

НА ТЕМУ:

«Метод резервного копирования с контролируемым размером избыточной информации в распределённом файловом хранилище»

Студент ИУ7-86Б (Группа) _____ (Подпись, дата) Д. В. Варин (И. О. Фамилия)

Руководитель ВКР _____ (Подпись, дата) Д. А. Кузнецов (И. О. Фамилия)

Нормоконтролер _____ (Подпись, дата) Д. Ю. Мальцева (И. О. Фамилия)

2023 г.

РЕФЕРАТ

Расчетно-пояснительная записка 70 с., 14 рис., 17 табл., 21 источн., 1 прил.

В работе представлена разработка метода резервного копирования с контролируемым размером избыточной информации в распределённом файловом хранилище.

В аналитическом разделе проведена классификация резервного копирования по способу, а также по месту хранения данных. Описаны основные понятия предметной области резервного копирования, такие как распределенное файловое хранилище, избыточность информации, контрольная сумма и целостность файла. Сформулирован разрабатываемый метод.

В конструкторском разделе приведены схемы алгоритмов, диаграммы последовательности, а также используемые структуры данных.

В технологическом разделе были выбраны средства реализации, а также разработано программное обеспечение, реализующее метод.

В исследовательском разделе проведено исследование эффективности разработанного метода. С целью сокращения размера избыточной информации исследовано влияние алгоритмов сжатия на резервные копии.

КЛЮЧЕВЫЕ СЛОВА

резервное копирование, распределенное файловое хранилище, избыточность, контрольная сумма, избыточные блоки данных, кодирование с использованием XOR, суперблок, алгоритмы сжатия

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В современном мире, где информационные технологии занимают все более важное место, защита данных становится критически важным вопросом.

Резервное копирование является одним из ключевых инструментов для обеспечения надежности и безопасности данных. Оно позволяет восстановить данные в случае их потери или повреждения.

Существует множество методов резервного копирования, отличающихся друг от друга по способу организации резервных копий, использованию избыточной информации и т.д.

В рамках работы предстоит разработать метод резервного копирования с контролируемым размером избыточной информации. Этот метод позволяет контролировать размер такой информации, что повышает эффективность использования ресурсов хранения.

Важным элементом резервного копирования является выбор хранилища для резервных копий. Распределенные файловые системы предоставляют множество преимуществ в этом отношении, в том числе возможность распределения данных по нескольким узлам, обеспечения отказоустойчивости и т.д.

1 Аналитический раздел

1.1 Резервное копирование

Резервное копирование — это процесс создания копии данных на устройства хранения, который используется для предотвращения потери данных вследствие их случайного удаления или повреждения носителя информации, например, выхода из строя жесткого диска [1].

Основная цель резервного копирования — обеспечить защиту от потери данных, вызванную сбоями оборудования, программными сбоями, вредоносными программами или человеческими ошибками. Копирование может быть осуществлено на различных уровнях — от отдельных файлов и папок до целых систем и баз данных. Это необходимо для обеспечения надежности, целостности данных и их доступности в случае непредвиденных обстоятельств. В процессе резервирования данных создается копия данных, которая хранится отдельно от основных данных. Резервные копии могут быть созданы на различных носителях, таких как диски, ленты, хранилища.

Существует несколько классификаций типов резервного копирования. Ниже представлены их примеры.

1. По способу хранения и обновления данных [2].
2. По месту хранения данных [3].

Рассмотрим подробнее данные классификации.

1.1.1 Классификация по способу хранения и обновления данных

Полное копирование

При этом типе резервирования копируются все данные и файлы с исходного устройства на резервное устройство. Этот тип копирования является наиболее надежным, но также и наиболее затратным по времени и месту на хранение.

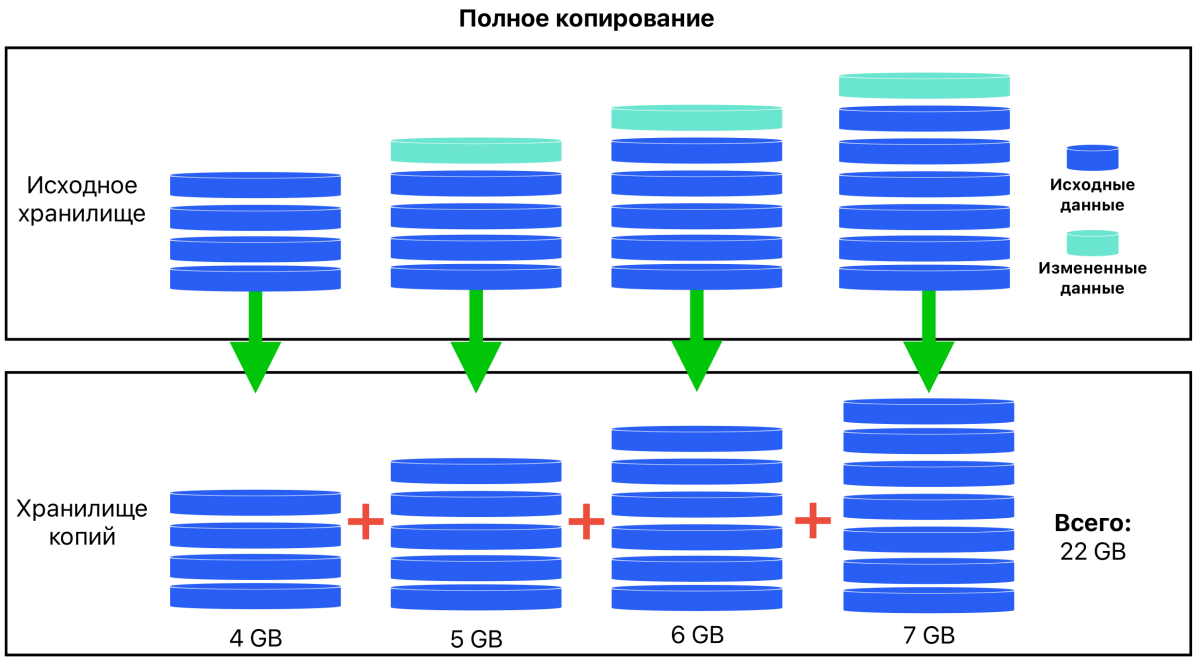

Рисунок 1.1 – Процесс полного резервного копирования

Инкрементное копирование

При этом типе резервирования копируются только измененные данные с момента последнего копирования. Это позволяет сократить время и место на хранение, но также увеличивает риск потери данных в случае отказа системы.

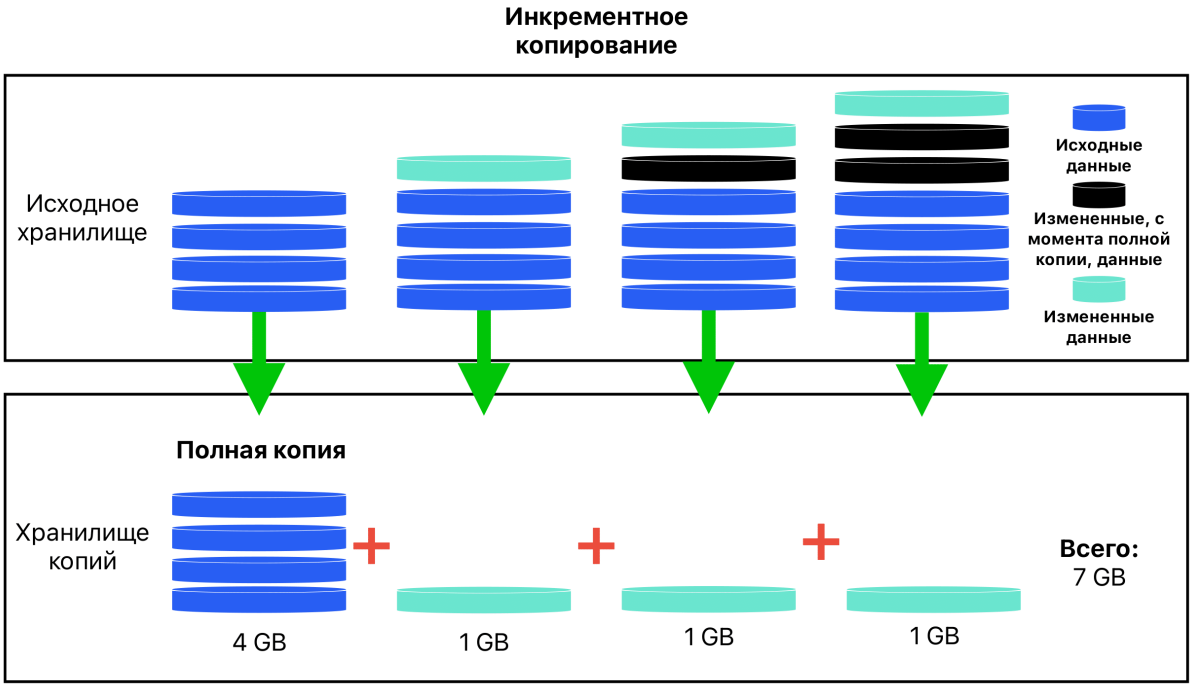

Рисунок 1.2 – Процесс инкрементного резервного копирования

Дифференциальное копирование

При этом типе резервирования копируются только измененные данные с момента последнего полного копирования. Это позволяет сократить время и место на хранение, но также увеличивает риск потери данных в случае отказа системы.

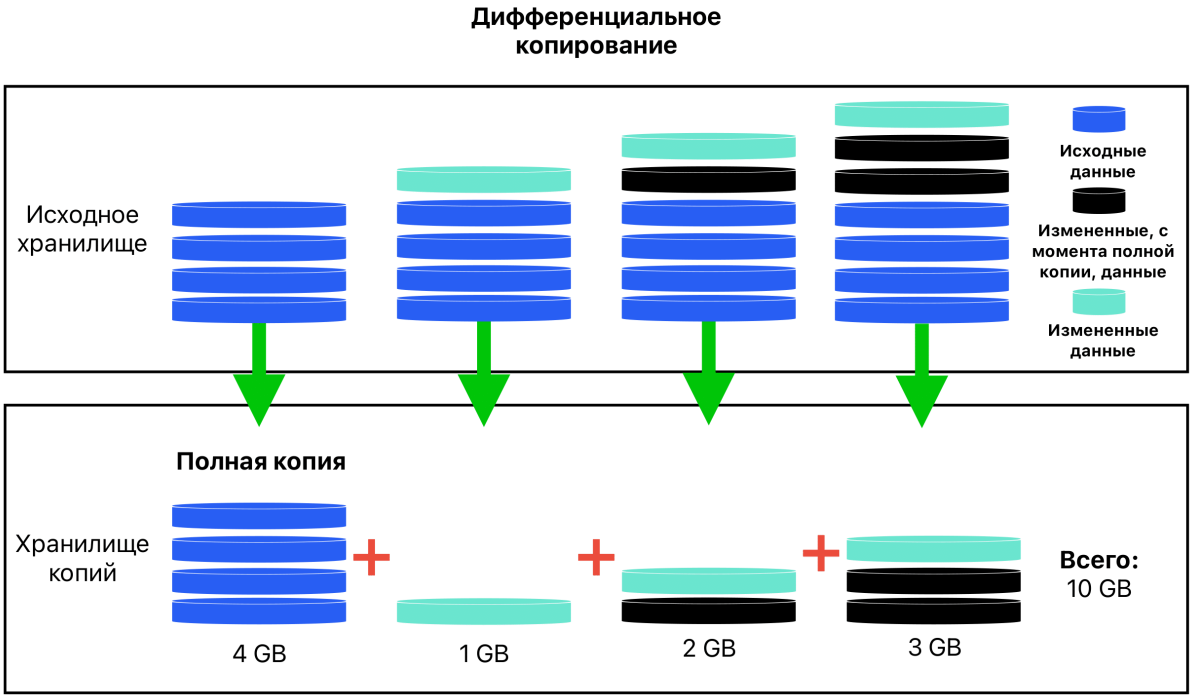

Рисунок 1.3 – Процесс дифференциального резервного копирования

Сравнение методов

Сведем разницу между методами резервного копирования по способу хранения и обновления данных в таблицу.

Таблица 1.1 – Сравнение полного, инкрементного и дифференциального резервного копирования

Хар-ка	**Полное копирование**	**Инкр-ое копирование**	**Дифф-ное копирование**
Объем данных	Большой	Постепенно растет	Постепенно растет
Время копирования	Длительное время	Быстрое время	Среднее
Затраты на хранилище	Высокие	Низкие (после полного копирования)	Низкие (после полного копирования)
Восстановление данных	Быстрое	Затруднительно без полного копирования	Затруднительно без полного копирования
Доступность в управлении	Нет	Да	Да
Частота выполнения	Единоразово	Периодически	Периодически
Затраты на сетевой трафик	Высокие	Низкие	Низкие

Рассмотрим каждую характеристику отдельно.

Объем данных:

— полное копирование имеет большой объем данных, так как копирует все файлы и папки каждый раз;

— инкрементное и дифференциальное копирование имеют постепенно растущий объем данных, так как они копируют только измененные или добавленные данные, при этом самая первая копия — полная.

Время копирования:

— полное копирование требует длительного времени, так как каждый раз выполняется полное сканирование и копирование всех файлов;

— инкрементное копирование обычно быстрое, так как копируются только новые или измененные файлы;

— дифференциальное копирование занимает среднее время, так как копируются только измененные файлы с момента последнего полного копирования.

Затраты на хранилище:

— полное копирование требует высокие затраты на хранилище, так как каждый раз создает полную копию данных;

— инкрементное и дифференциальное копирование имеют низкие затраты на хранилище, так как они сохраняют только измененные или добавленные данные.

Восстановление данных:

— полное копирование обеспечивает быстрое восстановление данных, так как все данные находятся в одном месте;

— инкрементное и дифференциальное копирование могут быть затруднительными при восстановлении данных без полного копирования, так как они зависят от последовательности и наличия предыдущих копий.

Доступность в управлении:

— полное копирование обычно менее сложное в управлении, так как требует полного сканирования и копирования всех файлов каждый раз;

— инкрементное и дифференциальное копирование более простые в управлении, так как копируют только измененные или добавленные данные.

Затраты на сетевой трафик:

— полное копирование требует высокие затраты на сетевой трафик, так как каждый раз передает все данные;

— инкрементное и дифференциальное копирование имеют более низкие затраты на сетевой трафик, так как передают только измененные или добавленные данные;

— однако, в случае инкрементного и дифференциального копирования, для полного восстановления данных может потребоваться передача большого количества инкрементных или дифференциальных копий, что может увеличить общий сетевой трафик.

Гибкость восстановления:

— полное копирование обеспечивает полную гибкость восстановления данных, так как все данные находятся в одной копии;

— инкрементное копирование обеспечивает гибкость восстановления данных только до момента последнего полного копирования, требуя последовательного восстановления каждой инкрементной копии;

— дифференциальное копирование обеспечивает гибкость восстановления данных до момента последнего полного или дифференциального копирования, требуя последовательного восстановления каждой дифференциальной копии.

Для метода используем механизм полного копирования, так как он более защищен от ошибок и повреждений — другие типы должны иметь доступ к нескольким последовательным копиям, что увеличивает риск появления ошибок при восстановлении.

1.1.2 Классификация по месту хранения данных

Локальное копирование

Локальное резервное копирование представляет собой тип резервного копирования, при котором данные копируются с одного устройства хранения на другое на локальном уровне. Это может быть копирование данных с жесткого диска на внешний носитель, на другой компьютер или на локальный сервер.

При локальном копировании данные остаются в пределах локальной сети и не передаются через общедоступную (интернет).

Копирование в удаленное хранилище

Копирование в удаленное хранилище представляет собой тип резервного копирования, при котором данные копируются с одного устройства в удаленное хранилище с использованием сети. В этом случае данные копируются на удаленный сервер или хранилище, находящееся в другом физическом или логическом месте.

Различия сетей

Локальная сеть и удаленная сеть — это два разных понятия, связанных с организацией и обеспечением сетевого соединения.

Локальная сеть (от англ. Local Area Network) представляет собой сеть, ограниченную географически, которая охватывает небольшую географическую область, такую как офис, университет или здание [4].

Локальная сеть строится для обеспечения связи и обмена данными между устройствами, находящимися в пределах этой области. Обычно она использует проводные или беспроводные технологии связи, такие как Ethernet или Wi-Fi. Локальная сеть может быть настроена для обмена файлами, печати, доступа к общим ресурсам и других совместных задач.

С другой стороны, глобальная сеть (от англ. Wide Area Network) — это сеть, которая объединяет компьютеры и устройства на больших расстояниях, как правило, через Интернет.

Удаленная сеть обеспечивает связь между географически разделенными

локальными сетями. Она может использовать различные технологии, например, виртуальные частные сети (VPN). Удаленная сеть позволяет расширить границы локальных сетей и обеспечить доступ к общим ресурсам и услугам, несмотря на физическое расстояние.

Таким образом, *разница между локальной и удаленной сетью* заключается в их географической области охвата и способе организации соединения. Локальная сеть предназначена для связи устройств внутри ограниченной области, в то время как удаленная сеть предоставляет связь между удаленными локациями или сетями через общедоступные сети. Однако в реальных сетевых сценариях часто используется комбинация локальных и удаленных сетей, чтобы обеспечить эффективную связь и доступность между удаленными и локальными устройствами.

Таблица 1.2 – Сравнение локального и удаленного резервного копирования

Характеристика	**Локальное копирование**	**Удаленное копирование**
Скорость доступа	Высокая	Ограничена интернет-соединением
Надежность	Зависит от носителей	Высокий уровень надежности
Масштабируемость	Ограничена объемом хранилища	Поддерживает большие объемы
Управление данными	Полный контроль и доступ	Доверенность и управление
Восстановление данных	Быстрое с локальных носителей	Восстановление через интернет
Стоимость	Высокие затраты на оборудование	Ежемесячная плата

Рассмотрим каждую характеристику отдельно.

Скорость доступа:

— локальное копирование обеспечивает высокую скорость доступа к данным, поскольку они хранятся на локальных носителях, доступ к которым осуществляется через локальную сеть;

— удаленное копирование ограничено скоростью интернет-соединения, что может влиять на время доступа к данным, особенно при крупных объемах данных или медленном интернет-соединении.

Надежность:

— локальное копирование зависит от надежности физических носителей, на которых хранятся резервные копии и в случае повреждения или отказа носителей, данные могут быть потеряны;

— удаленное копирование предоставляет высокий уровень надежности, так как облачные провайдеры обеспечивают резервное копирование и репликацию данных на удаленных серверах, что обеспечивает сохранность данных даже в случае сбоя оборудования или физической гибели данных.

Масштабируемость:

— локальное копирование ограничено объемом доступного физического хранилища, для увеличения масштабируемости необходимо приобретать и устанавливать дополнительное оборудование;

— удаленное копирование позволяет масштабировать хранилище данных в зависимости от потребностей, так как поставщики услуг предлагают гибкую масштабируемость, позволяющую увеличивать или уменьшать объем хранилища по требованию.

Стоимость:

— локальное копирование включает высокие затраты на приобретение и обслуживание оборудования для хранения и обработки данных;

— удаленное копирование предлагает модель ежемесячной платы за использование облачного хранилища, что может быть более экономически выгодным, особенно для небольших и средних предприятий.

Для метода объединим локальное копирование и удаленное, то есть сделаем возможность использования обоих способ копирования. Таким образом удастся взять преимущества обоих методов.

Резервное копирование является неотъемлемой частью обеспечения безопасности данных в любой информационной системе. Однако, с ростом объемов данных и требований к их доступности, возникает необходимость в использовании распределенных файловых хранилищ. Такие хранилища позволяют

хранить большие объемы данных на нескольких устройствах, обеспечивая высокую доступность и отказоустойчивость системы. При этом, для обеспечения безопасности данных, необходимо применять методы резервного копирования.

1.2 Распределенное файловое хранилище

Распределенное файловое хранилище (РФХ) — это способ организации хранения данных, при котором файлы разбиваются на части и хранятся на различных узлах (серверах) в сети [5].

В рамках работы используется распределенное файловое хранилище, которое работает не с файлами, а с его блоками.

1.2.1 Блочное файловое хранилище

При запросе на доступ к файлу, клиентское приложение обращается к одному из узлов, который предоставляет запрошенную часть файла (блок). В РФХ обычно применяются технологии, позволяющие обеспечить надежность и доступность данных, такие как репликация [6], шардинг [7], избыточность информации и т.д.

Кроме того, в распределенных хранилищах широко используется понятие избыточности информации.

Избыточность информации обеспечивает сохранность данных даже в случае сбоев в системе или потери части информации. Она достигается путем создания дополнительных копий данных или использованием специальных алгоритмов, таких как кодирование с избыточностью.

На рисунке ниже показана высокоуровненвая архитектура такого хранилища.

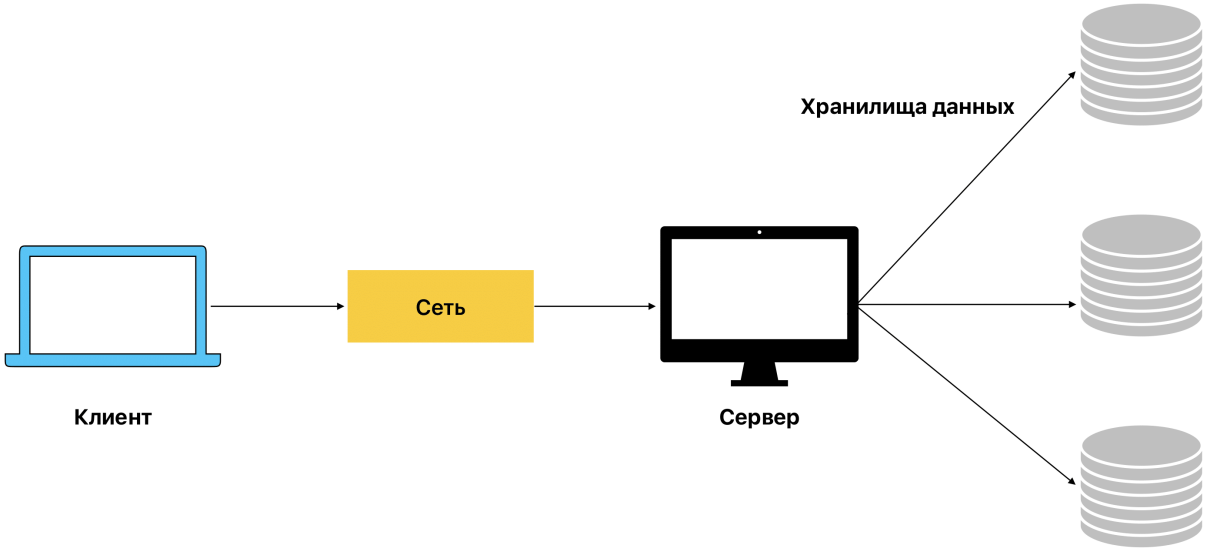

Рисунок 1.4 – Схема работы распределенного файлового хранилища

Клиент, посредством сети, отправляет файл в хранилище. Файл разбивается на блоки и распределяется сервером между различными хранилищами данных. Каждое такое хранилище может содержать несколько копий блоков данных или их фрагментов.

Это обеспечивает избыточность данных, так как в случае отказа одного хранилища, данные остаются доступными через другие копии, размещенные на других узлах.

При запросе файла клиентом, сервер собирает файл из блоков, на которые был разделен и отдает их клиенту.

Существует большое количество распределенных файловых хранилищ, например, HDFS [8], Ceph [9].

1.2.2 Избыточность информации

Избыточность информации — это концепция хранения дополнительных дублирующих данных в целях повышения надежности системы. В информационных системах, избыточность может быть реализована через различные методы, например, резервирование данных.

Избыточность информации играет важную роль в распределенных системах, где различные компоненты могут находиться на разных узлах сети и

могут быть недоступны в случае сбоев в сети или отказов оборудования.

При использовании избыточности данных в распределенных системах, дополнительные копии данных хранятся на разных узлах, что повышает доступность данных и уменьшает риск потери данных.

Однако использование избыточности данных может повлечь за собой дополнительные расходы на хранение и передачу данных, а также на управление и синхронизацию дополнительных копий. Поэтому необходим баланс между уровнем избыточности и экономическими затратами на его реализацию и поддержание.

Рассмотрим механизмы, которые создают избыточность в РФХ.

Репликация данных

Этот механизм заключается в создании нескольких копий одних и тех же данных и их распределении по разным серверам в сети. Если один сервер становится недоступным или данные на нем повреждены, то данные могут быть восстановлены из других копий. Репликация обеспечивает высокую доступность данных и устойчивость к сбоям.

Разделение информации

Этот механизм предполагает разбиение данных на части и их распределение по нескольким серверам. При чтении или записи данных они обрабатываются параллельно на нескольких серверах, что увеличивает производительность. Если один сервер становится недоступным, данные всё еще доступны на других серверах. Полосное размещение также позволяет увеличить пропускную способность системы.

Избыточные блоки данных

Этот механизм заключается в добавлении дополнительных избыточных блоков данных, которые могут использоваться для восстановления потерянных или поврежденных данных.

Избыточные блоки (суперблоки) могут быть созданы путем применения операций кодирования, таких как XOR или RAID [10]. Эти блоки содержат дополнительную информацию, которая позволяет восстановить данные, если один или несколько блоков становятся недоступными.

Таблица 1.3 – Сравнение механизмов избыточности в распределенном файловом хранилище

Механизм	**Плюсы**	**Минусы**
Репликация	Увеличение доступности данных Быстрый доступ к данным	Высокие затраты на оборудование
Разделение информации	Экономия места Высокая надежность	Зависимость от точности фрагментации Увеличение нагрузки на сеть
Избыточные блоки данных	Экономия места Высокая надежность	Вычислительная сложность Увеличение нагрузки на процессор

Рассмотрим плюсы и минусы отдельно.

Репликация

Плюсы:

— увеличение доступности данных — наличие нескольких копий данных обеспечивает доступ к информации, даже если одна из копий недоступна;

— быстрый доступ к данным — каждая реплика содержит полные данные, что позволяет быстро получать данные без дополнительных операций.

Минусы:

— высокие затраты на оборудование — требуется выделение дополнительного пространства и ресурсов для хранения и поддержки нескольких копий данных.

Разделение информации

Плюсы:

— экономия места — разделение информации на фрагменты позволяет сохранять данные в более компактном формате и эффективно использовать доступное пространство;

— высокая надежность — при потере или повреждении одного фрагмента данных, остальные фрагменты могут быть использованы для восстановления полной информации.

Минусы:

— зависимость от точности фрагментации — некорректное разделение информации на фрагменты может привести к потере данных или затруднить их восстановление;

— увеличение нагрузки на сеть — при передаче и обработке фрагментов данных возникает дополнительная нагрузка на сеть, что может снизить производительность.

Избыточные блоки данных

Плюсы:

— экономия места — кодирование данных с помощью XOR позволяет сократить объем хранимых данных, используя математические операции для создания избыточной информации;

— высокая надежность — при потере или повреждении одного или нескольких фрагментов данных, можно использовать оставшиеся фрагменты и XOR-операции для восстановления полной информации.

Минусы:

— вычислительная сложность — выполнение XOR-операций для кодирования и восстановления данных требует вычислительных ресурсов;

— увеличение нагрузки на процессор — при выполнении операций кодирования и декодирования с использованием XOR возникает дополнительная нагрузка на процессор, что может повлиять на производительность системы.

Анализируя плюсы и минусы, для метода предпочтительнее использовать механизм избыточных блоков, так как он позволяет сэкономить объем занимаемой памяти, а также обеспечить высокую надежность.

Избыточность данных с помощью XOR

Основная идея метода заключается в применении операции XOR (исключающее ИЛИ) для создания резервных копий блоков файлов. При этом для каждого блока файлов выбирается некоторое количество других блоков, с которыми производится операция XOR. Результат операции сохраняется в отдельный блок, который может использоваться для восстановления данных в случае потери или повреждения исходных блоков.

Преимущества использования XOR для создания резервных копий:

— XOR позволяет сократить объем хранимых данных, так как резервные блоки не являются полными копиями исходных блоков, а представляют собой только изменения в данных;

— при использовании XOR можно восстановить исходные данные путем выполнения операции XOR над доступными блоками и резервными блоками.

Недостатки использования XOR для создания резервных копий:

— если несколько блоков данных повреждены или потеряны, восстановление может быть невозможным.

Для обеспечения целостности данных при использовании XOR для создания резервных копий также часто используются контрольные суммы.

1.2.3 Проверка целостности файлов

Для проверки целостности файлов можно использовать механизмы контрольных сумм или хеширования.

Контрольная сумма — это значение, рассчитанное по набору данных путём применения определённого алгоритма и используемое для проверки целостности данных при их передаче или хранении.

Хеширование — это процесс преобразования входных данных произвольной длины в фиксированную строку фиксированной длины, называемую хешем. Хеш-функции обладают свойством, что даже незначительное изменение входных данных приведет к существенному изменению выходного хеша.

Хеширование широко используется для хранения паролей, проверки целостности данных, поиска дубликатов.

Сравнение контрольной суммы (хеша), полученной из исходной версии файла, с контрольной суммой (хешом), находящейся в резервной копии файла, помогает убедиться, что исходная копия файла является подлинной и не содержит ошибок.

Для вычисления контрольной суммы используют математические алгоритмы, которые отображают данные произвольного размера в массив фиксированного.

Существуют различные способы вычисления контрольных сумм и хешей. Ниже представлены их примеры.

1. CRC (Циклический избыточный код) — CRC-алгоритмы основаны на делении полиномов с двоичными коэффициентами. CRC вычисляет контрольную сумму путем деления данных на заданный полином. Полученный остаток является контрольной суммой [11].

2. MD5 — является одним из алгоритмов хеширования, который принимает произвольный объем данных и вычисляет 128-битную контрольную сумму [12].

3. SHA — семейство алгоритмов SHA, которое используется для вычисления контрольных сумм, например, SHA-256 [13].

1.3 Выводы

Определим метод резервного копирования, который будет реализован в рамках дипломного проекта.

Данный метод основан на использовании распределенного файлового хранилища и контролируемом размере избыточной информации. Он объединяет преимущества различных методов, рассмотренных ранее.

По способу хранения и обновления данных будет использоваться механизм полного копирования, по месту хранения данных - облачное копирование. Контролируемый размер избыточной информации будет обеспечиваться с помощью механизма создания избыточных блоков данных (кодирование с использованием XOR).

Цель работы — разработать метод резервного копирования с контролируемым размером избыточной информации в распределённом файловом хранилище.

Для достижения поставленной цели потребуется:

— описать основные понятия предметной области резервного копирования;

— провести анализ существующих методов;

— разработать метод резервного копирования с контролируемым размером избыточной информации в распределенном файловом хранилище;

— разработать программный комплекс, реализующий интерфейс для взаимодействия с разработанным методом;

— провести исследование эффективности разработанного метода и выявить количество избыточной информации, хранящейся при резервировании файлов.

2 Конструкторский раздел

2.1 IDEF0-диаграмма разрабатываемого метода

На рисунке ниже представлена диаграмма IDEF0 метода резервного копирования.

Рисунок 2.1 – IDEF0–диаграмма уровня A0

На вход метода подаются:

— место хранения суперблоков — расположение директорий в ФС;

— место хранения блоков файлов — расположение директории в ФС;

— файлы, для которых делается резервная копия.

Выходными данными являются резервные копии файлов (суперблоки).

Механизмы, используемые методом — файловая система (ФС) и хранилища данных (директории для хранения).

2.2 Схемы алгоритмов работы метода

Разрабатываемый метод резервного копирования с контролируемым размером избыточной информации в распределенном файловом хранилище имеет следующую схему работы.

1. Разделение файла на блоки — исходный файл разбивается на блоки фиксированного размера. Размер зависит от количества серверов, на которые сохраняется суперблок.

2. Вычисление контрольных сумм — для каждого блока данных вычисляется контрольная сумма.

3. Создание избыточной информации — вычисление резервных блоков при помощи операции XOR, применяемой к блокам файла.

4. Распределение избыточной информации — суперблоки распределяются по разным серверам или узлам в распределенном файловом хранилище.

5. Восстановление данных — в случае потери или повреждения блока данных, на основе избыточной информации можно восстановить потерянные или поврежденные данные.

Такие слова, как суперблок, избыточная информации и резервная копия в работе являются словами синонимами.

Рассмотрим детальнее эти шаги.

2.2.1 Алгоритм создания блоков

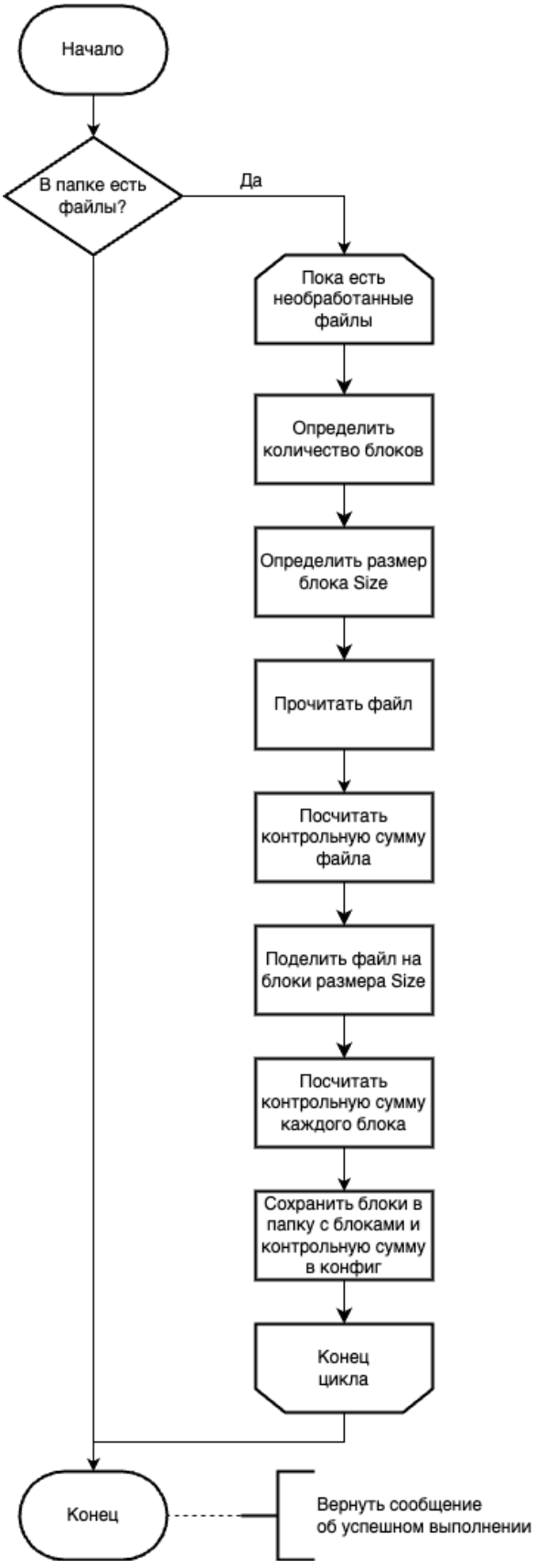

Рисунок 2.2 – Алгоритм создания блоков файлов

Алгоритм, для одного файла, можно описать следующим образом.

1. Определить количество блоков — количество блоков равно количеству используемых серверов, чтобы на каждом сервере хранилась часть файла.

2. Определить размер блока — для этого используется размер файла и количество блоков. Берется минимальное число X, которое больше размера файла, а также кратно количеству блоков и делится на количество блоков.

3. Чтение файла — прочитать исходный файл.

4. Вычисление контрольной суммы файла — для файла вычисляется контрольная сумма, чтобы проверять целостность файла.

5. Создание блоков — разделить прочитанные данные на блоки заданного размера.

6. Вычисление контрольной суммы — для каждого созданного блока данных вычисляется контрольная сумма, чтобы проверять целостность каждого блока в отдельности.

7. Сохранение блоков и контрольных сумм — сохранить каждый блок данных на диск, а также сохранить контрольные суммы в конфигурационный файл.

Метод позволяет подготовить данные к созданию резервной копии. Контрольные суммы обеспечивают возможность проверки целостности файлов и его блоков.

2.2.2 Алгоритмы создания и восстановления суперблока

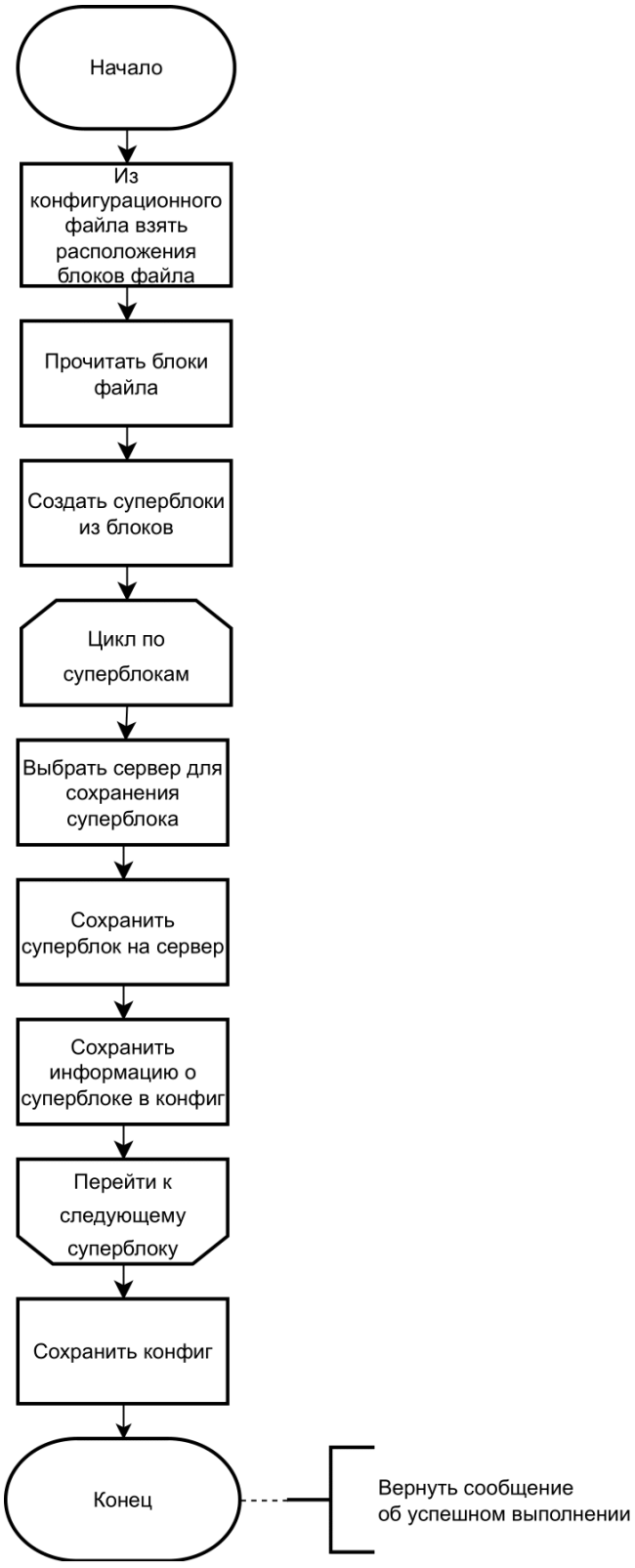

Рисунок 2.3 – Алгоритм создания суперблока

Алгоритм можно описать следующим образом.

1. Выбор блоков данных — выбрать блоки данных для включения в суперблок. Эти блоки являются блоками файла, для которого делается резервная копия.

2. Выбрать сервер для сохранения суперблока — для сохранения суперблока нужно выбрать сервер. Для этого можно использовать алгоритм Round-Robin.

3. Создание суперблока — создать суперблок, объединив выбранные блоки данных. С помощью суперблока можно восстановить исходный файл.

4. Сохранение суперблока и информации о его местоположении — сохранить созданный суперблок на выбранный сервер, а также информацию о местоположении.

   Выбор сервера осуществляется следующим образом.

Алгоритм распределения суперблоков по серверам

1. Инициализация переменных и структур данных.

2. Получение списка доступных серверов, на которых можно хранить суперблоки. Этот список определен предварительно.

3. Получение списка суперблоков, которые нужно распределить по серверам.

4. Сохранить суперблок на сервере, используя для выбора сервера алгоритм Round-Robin [14].

Рассмотрим алгоритм восстановления суперблока — операцию, при которой создается файл, позволяющий восстановить исходный файл при его повреждении.

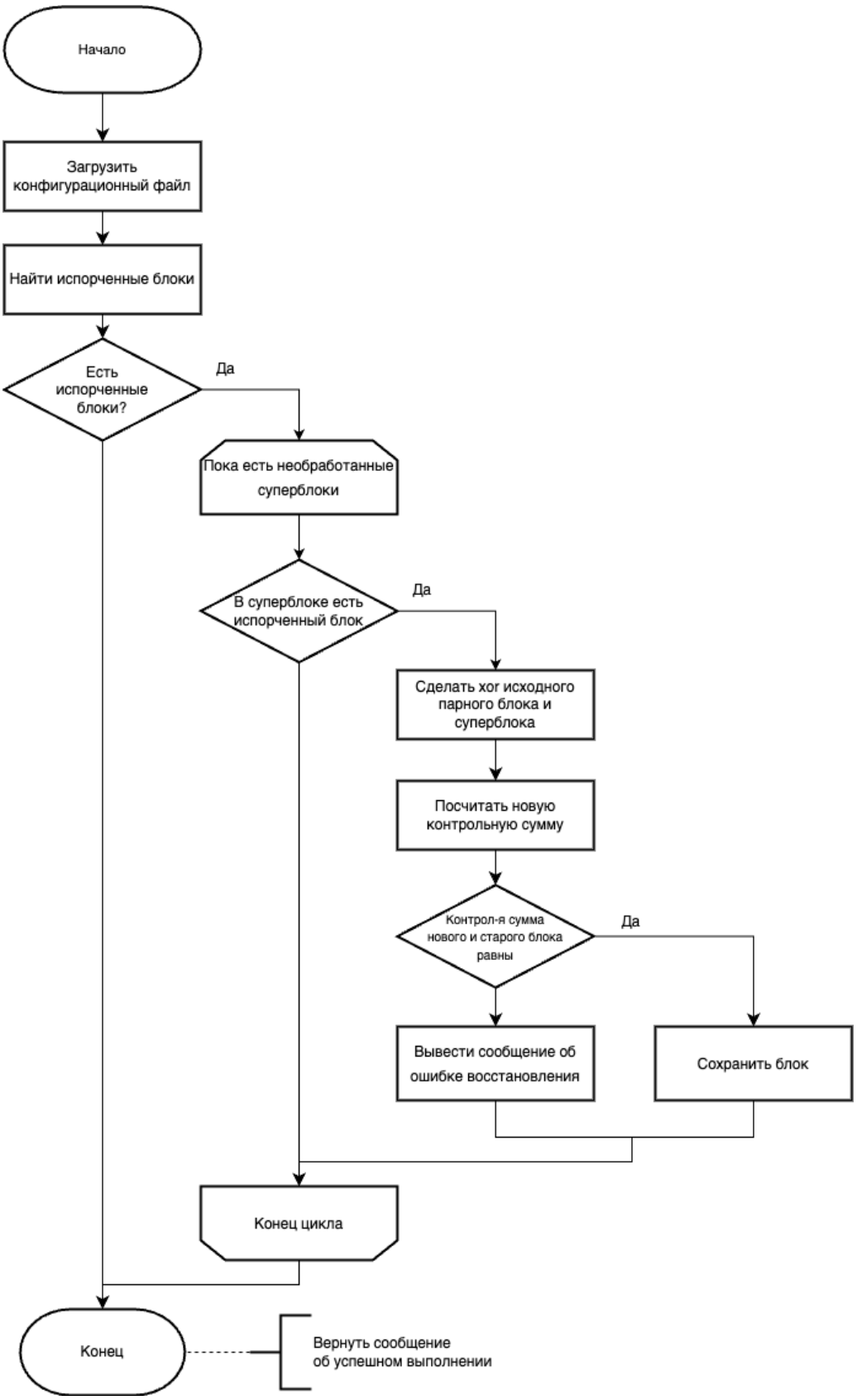

Рисунок 2.4 – Алгоритм восстановления суперблока

Алгоритм может быть описан следующим образом.

1. Получить суперблок восстанавливаемого файла — проверяются все файлы из конфигурационного файла, если на диске нет блока, который есть в конфиге, или у блока не сходится контрольная сумма с той, которая была получена при его создании, то блок помечается как испорченный.
2. В цикле по всем суперблокам ищем такой, у которого один из блоков является испорченным.
3. Делаем XOR этого суперблока с парным блоком — блоком, с которым была проведена операция XOR при создании суперблока.
4. Считаем и сравниваем контрольную сумма восстановленного блока. Если она не равна контрольной сумме, хранимой в конфигурационном файле, то выходим с ошибкой.
5. Если контрольные суммы сошлись, сохраняем восстановленный блок на место старого блока.

Алгоритм восстановления файла

Алгоритм может быть описан следующим образом.

1. Из конфигурационного файла извлечь информацию о местоположении блоков файла.
2. Последовательно записать в файл все блоки.
3. Сохранить восстановленный файл.

2.2.3 Вычисление контрольной суммы

В методе необходимо предусмотреть проверку целостности файлов и блоков. Она позволит подтвердить, что данные являются верными и не подверглись непредвиденным изменениям.

Для этого используется механизм контрольных сумм. Рассмотрим процесс вычисления контрольной суммы для блока.

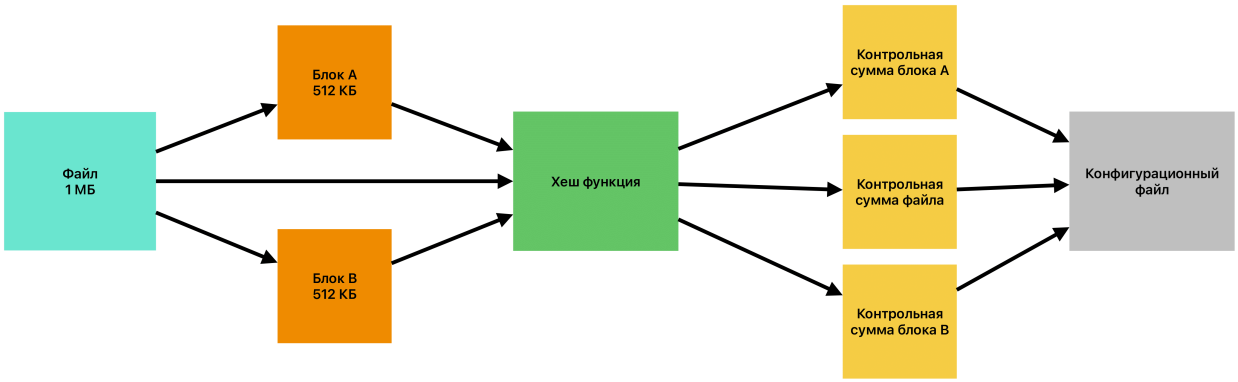

Рисунок 2.5 – Вычисление контрольной суммы файла и блоков

Например, у нас есть файл размером 1 МБ и мы хотим вычислить контрольную сумму для всего файла и его блоков.

Алгоритм вычисления контрольной суммы

1. Размер файла — 1 МБ, размер блока файла — 512 КБайт.
2. Делим файл на два блока.
3. Для файла и каждого блока вычисляем контрольную сумму, применяя хеш-функцию. Получаем контрольные суммы.
4. Сохраняем контрольные суммы в конфигурационный файл.

Теперь для проверки целостности блока или всего файла достаточно еще раз вычислить контрольную сумму и сравнить с той, что лежит в конфигурационном файле. В конфигурационном файле находятся контрольные суммы, которые были получены при создании резервной копии. Если новая контрольная сумма не совпадает с суммой из конфига, то файл (или блок) нужно восстановить, он поврежден.

В качестве хеш функции используется алгоритм SHA-256 [13].

Формула, по которой вычисляется контрольная сумма для блока файла выглядит следующим образом:

$$h_i = \text{sha256}(\text{block}_i), i \in [1, n], \tag{2.1}$$

где h_i — контрольная сумма блока файла, sha256 — хеш-функция sha256, block_i — блок файла, n — количество блоков файла.

Формула, по которой вычисляется контрольная сумма для всего файла выглядит следующим образом:

$$H = \text{sha256}(\text{file}), \tag{2.2}$$

где H — итоговая контрольная сумма всего файла, file — исходный файл.

2.3 Структуры данных

Основными структурами данных являются:

— блок данных — это базовая структура, представляющая блок информации фиксированного размера, где каждый блок содержит фрагмент данных файла, которые могут быть сохранены на различных серверах.

— суперблок — это агрегированная структура, которая содержит информацию о блоках данных.

Конфигурационный файл

В контексте метода конфигурационный файл представляет собой место хранения информации о текущем состоянии хранилища: расположение блоков, блоков данных, контрольные суммы, связи между блоками и суперблоками. Всё это позволяет восстановить файлы в случае их повреждения.

Файл создается и обновляется в процессе работы метода, используется для восстановления данных.

Структура данных конфигурационного файла включает следующие поля:

Servers – Список серверов, на которых хранятся блоки файлов.

Таблица 2.1 – Описание полей поля `servers`

Поле	**Описание**
`id`	Идентификатор
`name`	Название сервера
`address`	Адрес
`port`	Порт

При использовании абстракции сервер — это папка, поле servers всегда равно **null**.

Поля **files** — Список файлов, которые были сохранены в хранилище. Рассмотрим структуру данных.

Таблица 2.2 – Описание полей списка файлов

Поле	**Описание**
`filename`	Имя файла
`size`	Размер файла в байтах
`block_size`	Размер блока в байтах
`blocks`	Список блоков, составляющих файл
`created_at`	Дата и время создания файла

Поле **blocks** выглядит следующим образом.

Таблица 2.3 – Описание полей блоков *blocks*

Поле	**Описание**
`id`	Идентификатор
`checksum`	Контрольная сумма файла
`location`	Местоположение

Superblocks — Список суперблоков, которые содержат информацию о блоках, принадлежащих определенным файлам.

Для списка суперблоков (`superblocks`) в конфигурационном файле можно использовать следующую таблицу:

Таблица 2.4 – Описание полей списка суперблоков

Поле	**Описание**
`id`	Уникальный идентификатор суперблока
`name`	Имя суперблока
`blocks`	Список блоков, входящих в суперблок

LastUsedServerIndex – индекс последнего использованного сервера. Это поле используется для определения следующего сервера, куда будет сохранен суперблок, чтобы обеспечить равномерное распределение суперблоков по серверам и увеличить надежность хранения данных.

Пример конфигурационного файла для метода приведен в листингах А.1 и А.2.

2.4 Диаграммы последовательности

Программно-алгоритмический комплекс, использующий разрабатываемый метод, имеет несколько команд. Основные из них описаны ниже: *add*, *backup*, *restore*.

Рассмотрим их в виде диаграмм последовательности.

Команды разрабатываемого метода

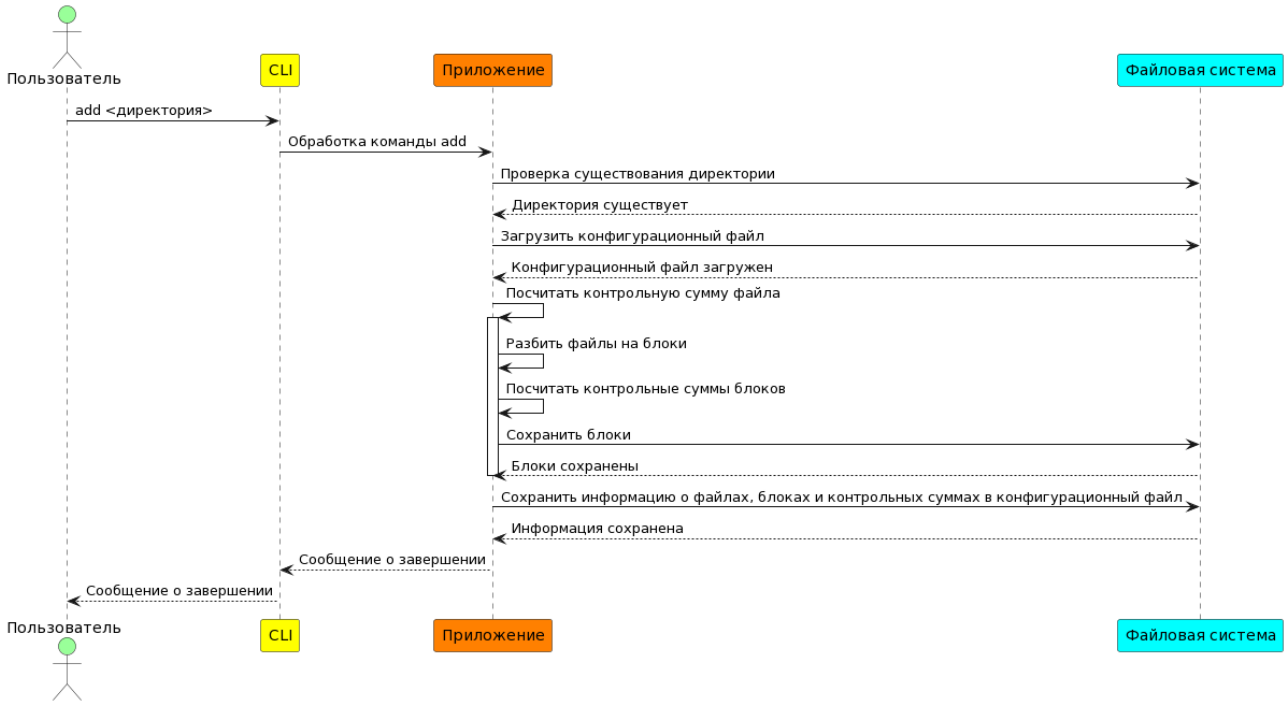

Рисунок 2.6 – Диаграмма последовательности для команды add

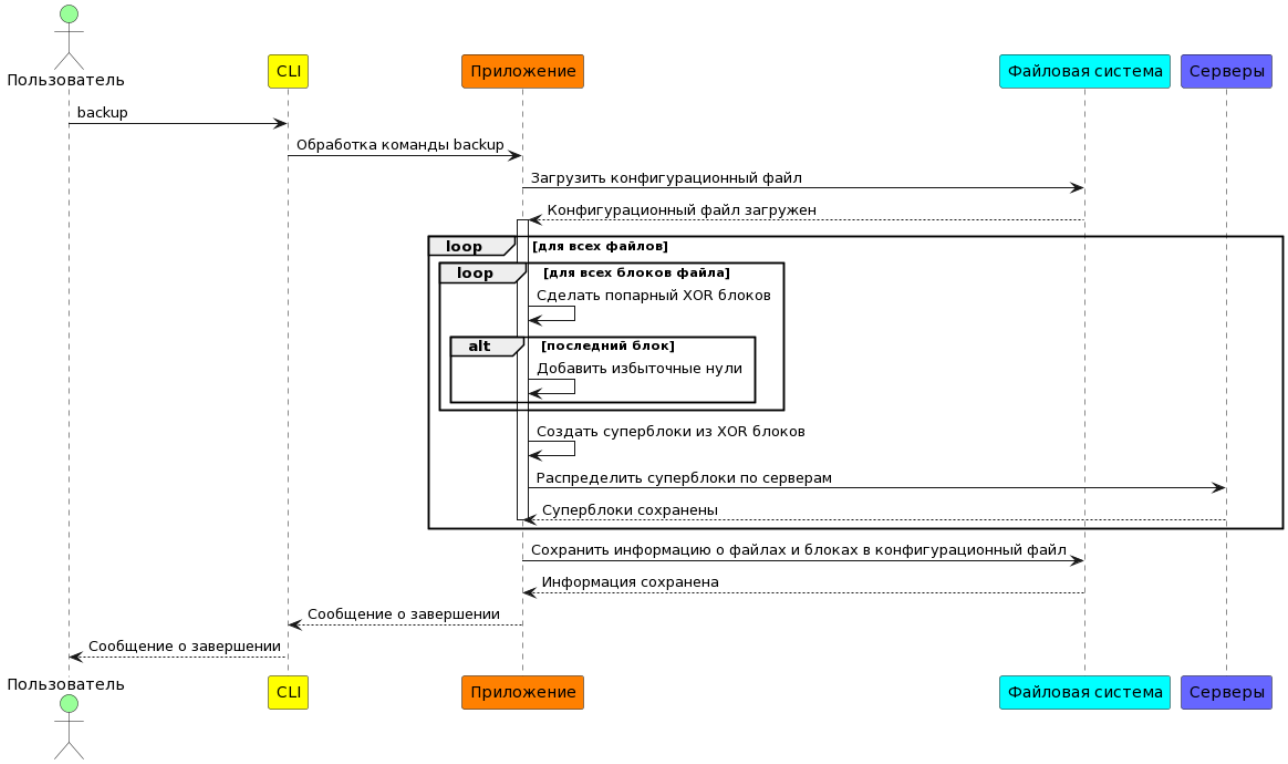

Рисунок 2.7 – Диаграмма последовательности для команды backup

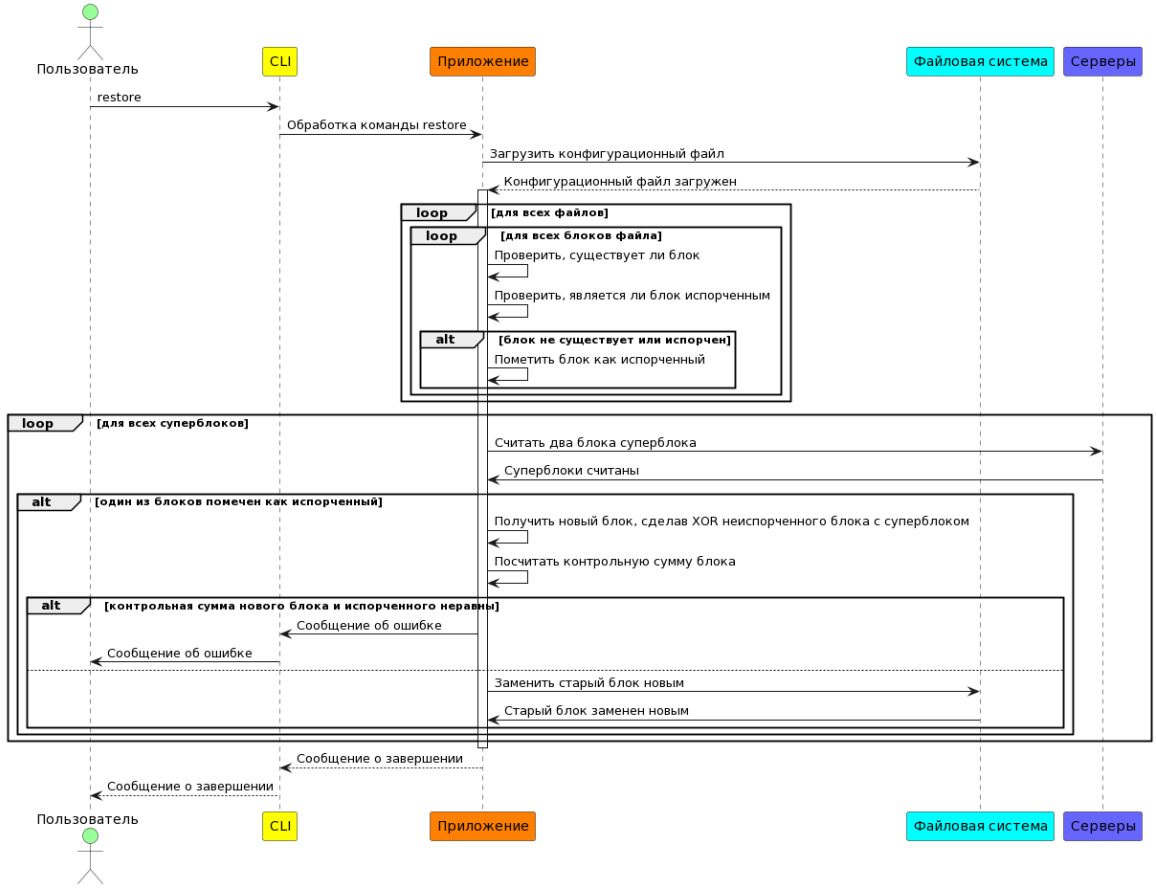

Рисунок 2.8 – Диаграмма последовательности для команды restore

2.5 Выводы

В данном разделе был разработан метод резервного копирования с контролируемым размером избыточной информации в распределенном файловом хранилище, детально рассмотрены алгоритмы:

— создания блоков;

— создания и восстановления суперблоков;

— распределения блоков по разным серверам;

— восстановления файла.

Разработаны структуры данных для хранения информации о файлах, блоках и суперблоках в виде конфигурационного файла.

3 Технологический раздел

3.1 Язык программирования

В качестве языка программирования для разработки было решено использовать язык Golang [15]. К преимуществам данного языка можно отнести статическую типизацию, компилируемость, большое разнообразие библиотек для документирования, тестирования и создания интерфейсов.

Статическая типизация обеспечивает:

— надежность и безопасность — статическая типизация позволяет обнаруживать ошибки во время компиляции, такие как несоответствие типов, отсутствие ожидаемых методов и неправильное использование переменных, что помогает предотвратить множество ошибок времени выполнения и повысить надежность и безопасность программы.

— читаемость и поддержку кода — статическая типизация делает код более читаемым и понятным, поскольку типы переменных и функций указывают на их предназначение и взаимодействие, кроме того, статическая типизация облегчает сопровождение кода, поскольку компилятор и инструменты разработчика могут предоставлять более точные подсказки и автодополнение кода.

Golang обладает малым размером исполняемых файлов и низким потреблением памяти. Это особенно важно для метода резервного копирования, где требуется обработка больших объемов данных. Go позволяет эффективно использовать ресурсы, что обеспечивает более быструю и эффективную обработку данных при резервном копировании.

3.2 Интерфейс приложения

Приложение будет представлять из себя консольную утилиту (CLI) [16].

Для этого будет использоваться библиотека cobra [17], которая предоставляет:

— механизмы обработки ошибок и валидации пользовательского ввода;

— API для создания команд и флагов;

— предоставляет встроенную поддержку генерации справки по командам и флагам;

— обеспечивает поддержку для тестирования.

Такой интерфейс имеет ряд преимуществ.

1. CLI обеспечивает легкость в автоматизации и скриптовании операций резервного копирования. Пользователь может создавать скрипты или автоматические задачи для выполнения резервного копирования по расписанию (cron [18]) или на основе определенных событий.

2. CLI-интерфейсы более гибкие и переносимые, поскольку они не привязаны к конкретной операционной системе или графической библиотеке.

Выбор CLI для метода резервного копирования обеспечивает возможность автоматизации, высокую производительность, гибкость и переносимость.

Рассмотрим реализации алгоритмов, которые используются разработанным приложением.

3.3 Создание блоков

Проверяем, если директория пустая, то завершаем алгоритм, иначе - начинаем обработку каждого файла.

Листинг 3.1 – Алгоритм создания блоков. Часть 1.

```
func addFiles(cmd *cobra.Command, args []string) {
    dir := args[0]
    files, err := ioutil.ReadDir(dir)
    if err != nil {
        fmt.Printf("Error reading directory: %v\n", err)
        os.Exit(1)
    }
    if len(files) == 0 {
        fmt.Printf("Backup directory is empty")
        return
    }
    numBlocks := calculateNumBlocks(fileInfo.Size(), len(cfg.
       Servers))
    blockSize := calculateBlockSize(fileInfo.Size(), numBlocks)
    for _, file := range files {
        if file.Mode().IsRegular() {
            addFile(filepath.Join(dir, file.Name()), &cfg,
               numBlocks, blockSize)
        }
    }
}
```

Считываем файл, затем делим его на блоки размера *blockSize*. Размер блока определяется функцией *calculateBlockSize*.

Листинг 3.2 – Алгоритм определения размера блока

```
func calculateBlockSize(fileSize int64, numBlocks int) int64 {
    return int64(RoundUpToNearestMultiple(int(fileSize),
       numBlocks) / numBlocks)
}

func RoundUpToNearestMultiple(n, k int) int {
    return ((n + k - 1) / k) * k
}
```

Для вычисления размера блока используется алгоритм округления вверх до ближайшего кратного.

Находится такое число X, которое больше числа N и кратно ему, но при этом минимально. В данном случае N — размер файла, k — количество серверов.

Такое вычисление размера позволяет гарантировать, что последний блок всегда будет меньше остальных. При создании суперблока это позволяет дописывать нули в конец последнего блока , чтобы размер блоков был одинаковый.

Затем считается чек-сумма, сохраняются блоки и информация о них в конфигурационный файл.

Листинг 3.3 – Алгоритм создания блоков. Часть 2.

```
func addFile(path string, cfg *config.Config, numBlocks int64,
   blockSize int64) {
    filePath := path
    file, err := os.Open(filePath)
    if err != nil {
        fmt.Printf("Error opening file: %v\n", err)
        os.Exit(1)
    }
    defer file.Close()
    fileInfo, err := file.Stat()
```

Листинг 3.4 – Алгоритм создания блоков. Часть 3.

```
    if err != nil {
        fmt.Printf("Error getting file information: %v\n", err)
        os.Exit(1)
    }
    checkSum, err := fileChecksum(file)
    if err != nil {
        fmt.Printf("Error getting file checksum: %v\n", err)
        os.Exit(1)
    }
    blocks := make([]config.FileBlock, 0)

    for i := 0; i < numBlocks; i++ {
        offset := int64(i) * blockSize
        blockSize := blockSize
        if i == numBlocks-1 {
            blockSize = fileInfo.Size() - offset
        }
        checkSum, err := blockChecksum(file, offset, blockSize)
        if err != nil {
            fmt.Printf("Error getting block checksum: %v\n", err)
            os.Exit(1)
        }
        blockPath := filepath.Join(blockPath, fmt.Sprintf("%s.%d
           ", fileName[len(fileName)-1], i))
        fileName := strings.Split(filePath, "/")
        block := config.FileBlock{
            Id:       i,
            Size:     int(blockSize),
            Checksum: checkSum,
            Location: blockPath,
        }
        blocks = append(blocks, block)
        err = os.MkdirAll(filepath.Dir(blockPath), 0755)
        if err != nil {
            fmt.Printf("Error creating directory: %v\n", err)
            os.Exit(1)
        }
```

Листинг 3.5 – Алгоритм создания блоков. Часть 4.

```
            err = saveBlockToFile(file, offset, blockSize, blockPath)
            if err != nil {
                fmt.Printf("Error saving block to file: %v\n", err)
                os.Exit(1)
            }
        }
        fileInf := config.FileInfo{
            Filename:  filepath.Base(filePath),
            Size:      fileInfo.Size(),
            BlockSize: blockSize,
            Checksum:  checkSum,
            Blocks:    blocks,
            Date:      time.Now().Format("2006-01-02 15:04:05"),
        }
        cfg.Files = append(cfg.Files, fileInf)
        err = saveConfig(*cfg)
        if err != nil {
            fmt.Printf("Error saving config file: %v\n", err)
            os.Exit(1)
        }
        fmt.Println("File info and blocks saved successfully.")
}
```

3.4 Создание и восстановление суперблока

Для создания суперблока проходим по всем файлам из конфигурационного файла, выбираем сервер для сохранения суперблока, создаем суперблок при помощи XOR блоков, добавляя нули в конец последнего блока, затем сохраняем информацию о суперблоке в конфиг, а сам суперблок на сервер.

Листинг 3.6 – Алгоритм создания и восстановления суперблока. Часть 1.

```
func Backup(configData *config.Config, isTest bool) error {
    for _, server := range configData.Servers {
        err := os.MkdirAll(server.Path, os.ModePerm)
        if err != nil {
            return fmt.Errorf("failed to create directory %s: %v
               ", server.Path, err)
        }
    }
    serverIndex := 0
    for _, file := range configData.Files {
        blocks := readBlocks(file.Blocks)
        superblockData := xorBlocks(blocks, int(file.BlockSize))
        for idx, val := range superblockData {
            serverPath := configData.Servers[serverIndex].Path
            backupDir := filepath.Join(serverPath)
            err := os.MkdirAll(backupDir, os.ModePerm)
            if err != nil {
                return fmt.Errorf("failed to create Backup
                   directory: %v", err)
            }

            superblock := superBlock{
                ID:     idx,
                Blocks: val.Data,
            }
            saveLocation := fmt.Sprintf("%s.%d", file.Filename,
               idx)
            err = writeSuperblock(backupDir, saveLocation, &
               superblock)
            if err != nil {
```

Листинг 3.7 – Алгоритм создания и восстановления суперблока. Часть 2.

```
                return fmt.Errorf("failed to write superblock for
                    %s: %v", file.Filename, err)
            }
            updateBlockLocationInConfig(configData, backupDir,
               file.Filename, superblockBlocks)
            serverIndex = (serverIndex + 1) % len(configData.
               Servers)
        }

    }
    configPath := configFilePath
    if isTest {
        configPath = configFileTestPath
    }
    err := saveConfig(*configData, configPath)
    if err != nil {
        return fmt.Errorf("failed to save configData: %v", err)
    }

    return nil
}
```

Функция для создания суперблока через XOR его блоков. Для последнего блока добавляются нули, выравнивающие размер блока.

Листинг 3.8 – Функция создания суперблока через XOR. Часть 1.

```
func xorBlocks(blocks []BlockInfo, fileBlockSize int) []XorData {
    if len(blocks) == 1 {
        return []XorData{
            {
                Size:        blocks[0].Size,
                CountBlocks: 1,
                Location: []string{
                    blocks[0].Location,
                },
                Data: blocks[0].Data,
            },
        }
    }
    xorBlocks := make([]XorData, 0, 1)
```

Листинг 3.9 – Функция создания суперблока через XOR. Часть 2.

```
    if len(blocks) > 1 {
        xorBlocks = make([]XorData, int(math.Ceil(float64(len(
            blocks))/2.0)))
    }
    for i := 0; i < len(blocks)-1; i++ {
        block1 := blocks[i]
        block2 := blocks[i+1]
        if block2.Size != fileBlockSize {
            block2.Data = addPadding(block2.Data, fileBlockSize)
        }
        xorBlock := XorData{
            Size:         block2.Size,
            CountBlocks: 2,
            Location: []string{
                block1.Location,
                block2.Location,
            },
            Data: make([]byte, fileBlockSize),
        }
        for j := 0; j < fileBlockSize; j++ {
            xorBlock.Data[j] = block1.Data[j] ^ block2.Data[j]
        }
        xorBlocks[i] = xorBlock
    }
    return xorBlocks
}
```

3.5 Восстановление файла

Для восстановления файла нужна информация о нем и о суперблоки. Такая информация берется из конфигурационного файла, после чего начинается процесс восстановления — выполняется обратный XOR с удаланием избыточных нулей из последнего блока.

Листинг 3.10 – Алгоритм восстановления файла. Часть 1.

```
func Restore(cfg *config.Config, isTest bool) error {
    badBlocks := make(map[string]config.FileBlock, 2)
    for _, val := range cfg.Files {
        for _, block := range val.Blocks {
            _, err := os.Stat(block.Location)
            if os.IsNotExist(err) {
                badBlocks[block.Location] = block
            } else if err != nil {
                return fmt.Errorf("ошибка при проверке файла
                   блока: %v", err)
            } else {
                dir, fileName := filepath.Split(block.Location)
                err = VerifyBlockChecksum(cfg, dir, fileName,
                   int64(block.Size))
                if err != nil {
                    badBlocks[block.Location] = block
                }
            }
        }
    }
    if len(badBlocks) == 0 {
        fmt.Println("all blocks are correct")
        return nil
    }
    for _, val := range cfg.SuperBlocks {
```

Листинг 3.11 – Алгоритм восстановления файла. Часть 2.

```
        if len(val.Blocks) == 1 {
            if _, ok := badBlocks[val.Blocks[0].Location]; ok {
                blockBytes, err := readSuperblock(val.Location)
                if err != nil {
                    log.Println(fmt.Sprintf("error read log %s",
                       val.Location))
                    return err
                }
                err = SaveBytesToFile(blockBytes.Blocks, val.
                   Blocks[0].Location)
                if err != nil {
                    log.Println(fmt.Sprintf("error save recovered
                        block: %v", val.Blocks[0].Location))
                    return err
                }
            }
            continue
        }
        // New block save here
        var badBlockLocation string
        // XOR superblock with recoverBlock
        recoverBlock := config.FileBlock{}
        block1 := val.Blocks[0]
        block2 := val.Blocks[1]
        _, ok := badBlocks[block1.Location]
```

Листинг 3.12 – Алгоритм восстановления файла. Часть 3.

```
        if ok {
            badBlockLocation = block1.Location
            recoverBlock = block2
        } else {
            _, ok2 := badBlocks[block2.Location]
            if ok2 {
                badBlockLocation = block2.Location
                recoverBlock = block1
            } else {
                continue
            }
        }
        recoverBlockBytes, err := readBlock(recoverBlock.Location
           )
        if err != nil {
            log.Println(fmt.Sprintf("error read log %s",
               recoverBlock.Location))
            return err
        }
        xoredBlockBytes, err := readSuperblock(val.Location)
        if err != nil {
            log.Println(fmt.Sprintf("error read log %s",
               recoverBlock.Location))
            return err
        }
```

Листинг 3.13 – Алгоритм восстановления файла. Часть 4.

```
        xorRecoverBlockBytes := xorBlock(recoverBlockBytes,
           xoredBlockBytes.Blocks)[:val.BlockSize]
        newChecksum, err := bytesChecksum(xorRecoverBlockBytes)
        if err != nil {
            return fmt.Errorf("error get checksum: %w", err)
        }
        _, badBlockName := filepath.Split(badBlockLocation)
        expectedChecksum := getBlockChecksum(badBlockName, cfg)

        if expectedChecksum != newChecksum {
            return fmt.Errorf("checksum %s mismatch: expected = %
               s, actual = %s", badBlockName, expectedChecksum,
               newChecksum)
        }
        delete(badBlocks, badBlockLocation)

        err = SaveBytesToFile(xorRecoverBlockBytes,
           badBlockLocation)
        if err != nil {
            log.Println(fmt.Sprintf("error save recovered block:
               %v", badBlockLocation))
            return err
        }
    }
    return nil
}
```

3.6 Пример использования разработанного метода

CLI интерфейс имеет следующие команды:

— *add* — деление файлов на блоки;

— *backup* — создание резервных копий (суперблоков);

— *block-info* — проверка целостности блоков;

— *files-info* — проверка целостности файлов;

— *restore* — восстановление блоков файла;

— *build* — собрать файлы из блоков, восстановление файла.

Базовый сценарий использования предусматривает:

— добавление файлов и деление их на блоки — *backupcli add*;

— создание суперблоков — *backupcli backup*;

— восстановление блоков, например, если один блок был удален — *backupcli restore*;

— проверка целостности блоков до/после восстановления — b*ackupcli block-info*;

— восстановление файлов из блоков — *backupcli build*;

Работа с программой следующим образом.

```
➜  app-blocks git:(main) ✗ ./backupcli add
2023/06/05 01:24:08 init config
File src/bmstu.png blocks and info saved successfully.
File src/text.txt blocks and info saved successfully.
File src/video.mp4 blocks and info saved successfully.
File src/war-and-peace.txt blocks and info saved successfully.
➜  app-blocks git:(main) ✗ ./backupcli backup
Backup completed successfully.
➜  app-blocks git:(main) ✗ ./backupcli block-info
Checksum match for file bmstu.png.1
Checksum match for file bmstu.png.2
Checksum match for file text.txt.0
Checksum match for file text.txt.2
Checksum match for file video.mp4.0
Checksum match for file video.mp4.1
Checksum match for file war-and-peace.txt.0
Checksum match for file war-and-peace.txt.2
Block not found in blocks dir video.mp4.2
Block not found in blocks dir war-and-peace.txt.1
Block not found in blocks dir bmstu.png.0
Block not found in blocks dir text.txt.1
Check files completed successfully.
```

Рисунок 3.1 – Деление файлов на блоки, создание суперблоков и проверка целостности блоков

```
➜  app-blocks git:(main) ✗ ./backupcli restore
Restore completed successfully.
➜  app-blocks git:(main) ✗ ./backupcli block-info
Checksum match for file bmstu.png.0
Checksum match for file bmstu.png.1
Checksum match for file bmstu.png.2
Checksum match for file text.txt.0
Checksum match for file text.txt.1
Checksum match for file text.txt.2
Checksum match for file video.mp4.0
Checksum match for file video.mp4.1
Checksum match for file video.mp4.2
Checksum match for file war-and-peace.txt.0
Checksum match for file war-and-peace.txt.1
Checksum match for file war-and-peace.txt.2
Check files completed successfully.
➜  app-blocks git:(main) ✗ ./backupcli build
File build completed.
Build completed successfully.
```

Рисунок 3.2 – Восстановелние блоков, проверка целостности и сборка файла

3.7 Тестирование

Тестирование метода производилось несколькими способами:

— юнит тестирование - покрыть основные функции тестами;

— ручное тестирование.

Для ручного тестирования использовался следующий сценарий.

1. Создать несколько тестовых файлов, которые будут использоваться в процессе резервного копирования.

2. Создать резервные копии через команду *backup*.

3. Восстановить файлы через команду *restore*.

4. Проверить различия файлов (diff) визуально, или с использованием специальных программ, которые позовляют найти отличия в файлах.

Для юнит тестирования будут использоваться встроенные в язык программирования инструменты.

3.8 Выводы

В данном разделе:

— были выбраны средства реализации метода резервного копирования;

— было разработано программное обеспечение, реализующее разработанный метод;

— приведен пример использования разработанного метода;

— осуществлено тестирование метода, для которого использовано два типа тестов — юнит и ручные тесты.

4 Исследовательский раздел

4.1 Исследование эффективности разработанного метода

Метриками эффективности метода были выбраны размер избыточной информации и возможность восстановления файла. Избыточной информацией является суперблок.

В рамках исследования были выбраны несколько видов файлов:

— текстовые файлы;

— изображения;

— видеозаписи.

Для проверки следует сравнить, насколько размер суперблока больше или меньше размера исходного файла.

Для этого сравнения были сделаны резервные копии файлов, типы которых выделены выше.

По результатам анализа были сделаны следующие выводы:

— суммарный размер суперблоков равен размеру исходного файла;

— метод гарантирует восстановление файла при потере одного сервера, если количество серверов больше одного.

Рассмотрим пример с файлом, который разделен на три блока. Суперблоки распределены по трем серверам.

При отказе одного из серверов, файл можно восстановить при помощи суперблоков, сохраненных на другие сервера.

Чтобы добиться такого уровня отказоустойчивости в механизме полной копии нужно сохранить на все сервера копию файла, в результате чего количество избыточной информации будет в 3 раза больше, чем с использованием разработанного метода.

С целью сократить размер избыточной информации в разработанном методе исследуем, как изменится размер суперблока с применением к нему алгоритмов сжатия.

4.2 Исследование изменения размера избыточной информации от применения алгоритмов сжатия

Для исследования выберем несколько алгоритмов сжатия:

— GZIP [19];

— ZSTD [20];

— LZMA [21].

В исследовании будем использовать файлы, которые можно разбить на следующие группы:

— текстовые, с расширением *.txt*;

— документы, с расширением *.pdf*;

— изображения, с расширениями *.jpg* и *.png*;

— видеозаписи, с расширениями *.mov* и *.mp4*.

Рассмотрим каждую группу отдельно.

Текстовые

Таблица 4.1 – Текстовый файл, используемый в исследовании

Имя	Размер (в байтах)
war_and_peace.txt	3 202 332

Выполним операцию создания резервной копии. Затем применим алгоритмы сжатия и сравним результаты.

Попробуем сжать суперблок книги *«Война и Мир»* , которая сохранена в файл с расширением *.txt*.

Алгоритм сжатия позволяет уменьшить размер суперблока более чем в 2 раза.

Посчитаем, на сколько процентов размер сжатого суперблока меньше размера несжатого. Исходный размер измерялся в байтах.

Таблица 4.2 – Сравнение размеров текстовых файлов в разных форматах

Название	Исходный размер	% с LZMA	% с GZIP	% с ZSTD
war_and_peace.txt	3 202 332	64.76%	62.87%	63.67 %

По результатам можно сделать вывод — при сжатии суперблоков текстовых файлов размер избыточной информации существенно уменьшается: на десятки процентов от исходного.

Документы

Таблица 4.3 – Документы, используемые в исследовании

Имя	Размер (в байтах)
book2.pdf	63 950 013
book.pdf	136 542 190

Посчитаем, на сколько процентов размер сжатого суперблока меньше размера несжатого.

Таблица 4.4 – Сравнение размеров документов в разных форматах

Название	Исходный размер	% с LZMA	% с GZIP	% с ZSTD
book2.pdf	63 950 013	76.11%	75.93%	75.96%
book.pdf	136 542 190	88.72%	88.13%	88.46%

По результатам можно сделать вывод — при сжатии документов размер избыточной информации уменьшается более чем в 2 раза.

Изображения

Таблица 4.5 – Изображения, используемые в исследовании

Имя	Размер (в байтах)
img5.png	1 300 370
img1.jpg	7 054 172
img4.png	8 200 370
img2.jpg	14 733 613
img3.jpg	19 290 046

Посчитаем, на сколько процентов размер сжатого суперблока меньше размера несжатого.

Таблица 4.6 – Сравнение размеров изображений в формате jpg

Название	Исходный размер	% с LZMA	% с GZIP	% с ZSTD
img1.jpg	7 054 172	0.27%	0.20%	0.00%
img2.jpg	14 733 613	1.35%	0.10%	0.54%
img3.jpg	19 290 046	0.03%	0.01 %	0.04 %

По результатам можно сделать вывод — при сжатии суперблоков изображений в формате jpg размер файла существенно не меняется.

Рассмотрим изображение в формате png. В сравнении с png, jpg сжимается по алгоритму сжатия с потерями, в то время как для файлов формата png используют алгоритм сжатия без потерь.

Посчитаем, на сколько процентов размер сжатого суперблока меньше размера несжатого.

Таблица 4.7 – Сравнение размеров изображения в формате png

Название	Исходный размер	% с LZMA	% с GZIP	% с ZSTD
img5.png	1 300 370	10.01%	9.81%	5.02%
img4.png	8 200 370	32.10%	30.37%	27.41%

По полученным результатам можно сделать вывод, что сжатие позволяет уменьшить размер суперблоков для изображений, если формат этих файлов использует алгоритм сжатия без потерь. Таким форматом является png. Сравнивая с jpg, разница достигает более 10 % от размера исходного файла.

Видеозаписи

Таблица 4.8 – Видеозаписи, используемые в исследовании

Имя	Размер (в байтах)
video.mp4	3 891 031
video1.mov	63 639 323
video2.mp4	69 075 086

Посчитаем, на сколько процентов размер сжатого суперблока меньше размера несжатого.

Таблица 4.9 – Сравнение размеров видеозаписей в формате mp4

Название	Исходный размер	% с LZMA	% с GZIP	% с ZSTD
video.mp4	3 891 031	0.29%	0.24%	0.08%
video2.mp4	69 075 086	0.05%	0.02%	0.04%

По результатам можно сделать вывод — при сжатии суперблоков видеозаписей формата mp4 размер файла существенно не меняется.

Рассмотрим видеозапись в формате mov. В сравнении с mov, mp4 обычно более сжат и меньше по размеру, в то время как файлы mov часто имеют более высокое качество и больший размер.

Посчитаем, на сколько процентов размер сжатого суперблока меньше размера несжатого.

Таблица 4.10 – Сравнение размеров видеозаписи в формате mov

Название	Исходный размер	% с LZMA	% с GZIP	% с ZSTD
video1.mov	63 639 323	26.71%	3.25%	20.61%

По полученным результатам можно сделать вывод, что сжатие позволяет уменьшить размер суперблоков для видеофайлов, в случае если формат этих файлов имеет минимальный уровень сжатия. Таким форматом является mov. Сравнивая с mp4, разница достигает более 20 % от размера исходного файла.

4.3 Выводы

В данном разделе:

— проведено исследование эффективности разработанного метода — в результате анализа было получено, что суммарный размер суперблоков равен размеру исходного файла, а метод гарантирует восстановление файла при потере одного сервера, если количество серверов больше одного;

— проведено исследование изменения размера избыточной информации от применения алгоритмов сжатия;

— сжатие суперблоков позволяет уменьшить их размер: в худшем случае размер суперблока равен исходному, в лучшем — уменьшится на десятки процентов;

— сжатие хорошо работает на файлах, имеющих малый процент сжатия, таких как текстовые файлы, документы, некоторые форматы видео, например, mov;

— сжатие позволяет уменьшить размер изображений, форматы которых используют алгоритм сжатия без потерь, например, формат png;

— добавление в разработанный метод алгоритма сжатия позволит уменьшить размер суперблоков и сделает метод более эффективным по памяти.

ЗАКЛЮЧЕНИЕ

В ходе выполнения работы были выполнены следующие задачи:

— описаны основные понятия предметной области резервного копирования;

— разработан метод резервного копирования с контролируемым размером избыточной информации в распределенном файловом хранилище;

— разработан программный комплекс, реализующий интерфейс для взаимодействия с разработанным методом;

— проведено исследование эффективности разработанного метода;

— анализ показал, что разработанный метод гарантирует восстановление файла при потере одного сервера, если количество серверов больше одного, а также гарантирует восстановление файла при потере одного сервера.

Исследование показало, что добавление в разработанный метод алгоритма сжатия позволит уменьшить размер суперблоков и сделает метод более эффективным по памяти.

Таким образом, поставленная цель работы, разработать метод резервного копирования с контролируемым размером избыточной информации в распределенном файловом хранилище, была достигнута.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Acronics. Словарь терминов. Резервное копирование [Электронный ресурс]. — Режим доступа: http://www.acronis.ru/glossary/backup/backup — (Дата обращения: 2023-04-16).
2. Cloud Backup [Электронный ресурс]. — Режим доступа: https://www.acronis.com/en-us/blog/posts/cloud-vs-local-backup/ — (Дата обращения: 2023-04-16).
3. Backup types explained [Электронный ресурс]. — Режим доступа: https://www.nakivo.com/blog/backup-types-explained/ — (Дата обращения: 2023-04-16).
4. What is a lan [Электронный ресурс]. — Режим доступа: https://www.cloudflare.com/learning/network-layer/what-is-a-lan/ — (Дата обращения: 2023-05-23).
5. What is DFS? [Электронный ресурс]. — Режим доступа: https://www.nutanix.com/info/distributed-file-systems — (Дата обращения: 2023-05-08).
6. Репликация данных [Электронный ресурс]. — Режим доступа: https://highload.today/replikatsiya-dannykh/ — (Дата обращения: 2023-05-29).
7. Горизонтальный шардинг [Электронный ресурс]. — Режим доступа: https://highload.today/gorizontalniy-sharding/ — (Дата обращения: 2023-05-29).
8. Hadoop Distributed File System (HDFS) [Электронный ресурс]. — Режим доступа: https://www.databricks.com/glossary/hadoop-distributed-file-system-hdfs — (Дата обращения: 2023-05-08).
9. What is Ceph? [Электронный ресурс]. — Режим доступа: https://ubuntu.com/ceph/what-is-ceph — (Дата обращения: 2023-05-08).
10. What is RAID? [Электронный ресурс]. — Режим доступа: https://www.hellotech.com/blog/what-is-raid-0-1-5-10 — (Дата обращения: 2023-05-08).

11. Контрольная сумма CRC [Электронный ресурс]. — Режим доступа: https://soltau.ru/index.php/themes/dev/item/461-kak-poschitat-kontrolnuyu-summu-crc32-crc16-crc8 — (Дата обращения: 2023-05-29).

12. MD5 Hash Generator [Электронный ресурс]. — Режим доступа: https://www.md5hashgenerator.com/ — (Дата обращения: 2023-05-09).

13. SHA-256 Hashing Algorithm Overview [Электронный ресурс]. — Режим доступа: https://blog.boot.dev/cryptography/how-sha-2-works-step-by-step-sha-256/ — (Дата обращения: 2023-05-09).

14. Балансировка нагрузки: основные алгоритмы и методы [Электронный ресурс]. — Режим доступа: https://selectel.ru/blog/balansirovka-nagruzki-osnovnye-algoritmy-i-metody/ — (Дата обращения: 2023-05-14).

15. The Go Programming Language [Электронный ресурс]. — Режим доступа: https://go.dev/ — (Дата обращения: 2023-05-24).

16. Командная строка [Электронный ресурс]. — Режим доступа: https://help.ubuntu.com/community/CommandlineHowto — (Дата обращения: 2023-05-24).

17. A Commander for modern Go CLI interactions [Электронный ресурс]. — Режим доступа: https://github.com/spf13/cobra — (Дата обращения: 2023-05-24).

18. Crontab command [Электронный ресурс]. — Режим доступа: https://www.ibm.com/docs/en/aix/7.2?topic=c-crontab-command — (Дата обращения: 2023-05-24).

19. Gzip is a file format and a software application used for file compression and decompression [Электронный ресурс]. — Режим доступа: https://www.gzip.org/ — (Дата обращения: 2023-05-29).

20. Zstandard - Fast real-time compression algorithm [Электронный ресурс]. — Режим доступа: https://github.com/facebook/zstd — (Дата обращения: 2023-05-29).

21. 7-Zip is a file archiver with a high compression ratio [Электронный ресурс]. — Режим доступа: https://7-zip.org/ — (Дата обращения: 2023-05-29).

ПРИЛОЖЕНИЕ А
Конфигурационные файлы

В листингах А.1 – А.4 представлены примеры конфигурационного файла для метода.

Листинг А.1 – Конфигурационный файл метода. Часть 1.

```
{
    "servers": [{
            "id": "1",
            "hostname": "hostname",
            "username": "admin",
            "password": "qwerty",
            "path": "/Users/dvvarin/BMSTU/xoracle-bachelors-
              diploma/app-blocks/s1"
        },
        {
            "id": "2",
            "hostname": "hostname",
            "username": "admin",
            "password": "12345",
            "path": "/Users/dvvarin/BMSTU/xoracle-bachelors-
              diploma/app-blocks/s2"
        },
        {
            "id": "3",
            "hostname": "hostname",
            "username": "admin",
            "password": "12345Qwerty",
            "path": "/Users/dvvarin/BMSTU/xoracle-bachelors-
              diploma/app-blocks/s3"
        }
    ],
```

Листинг А.2 – Конфигурационный файл метода. Часть 2.

```
    "files": [{
        "filename": "bmstu.png",
        "size": 303268,
        "block_size": 101090,
        "checksum": "88ce9316b6742d6413d25ebe91fd8aa17a53",
        "blocks": [{
                "id": 0,
                "checksum": "9e8f15f5a29319bacebb8fbe2b87",
                "location": "/Users/dvvarin/BMSTU/xoracle-
                   bachelors-diploma/app-blocks/blocks/bmstu.png
                   .0",
                "size": 101090
            },
            {
                "id": 1,
                "checksum": "5b9617421dfce18e9fc8957353d6",
                "location": "/Users/dvvarin/BMSTU/xoracle-
                   bachelors-diploma/app-blocks/blocks/bmstu.png
                   .1",
                "size": 101090
            },
            {
                "id": 2,
                "checksum": "c88f7b711f18e680bc5d70a16c42",
                "location": "/Users/dvvarin/BMSTU/xoracle-
                   bachelors-diploma/app-blocks/blocks/bmstu.png
                   .2",
                "size": 101088
            }
        ],
        "created_at": "2023-06-08 10:45:15"
    }],
```

Листинг А.3 – Конфигурационный файл метода. Часть 3.

```
    "superblocks": [{
            "id": 0,
            "block_size": 101090,
            "checksum": "fvso48f7b7115d70a1c8f18e46c42c4f",
            "location": "/Users/dvvarin/BMSTU/xoracle-bachelors-
               diploma/app-blocks/s1/bmstu.png.0.sb",
            "blocks": [{
                    "id": 0,
                    "location": "/Users/dvvarin/BMSTU/xoracle-
                       bachelors-diploma/app-blocks/blocks/bmstu.
                       png.0",
                    "size": 101090
                },
                {
                    "id": 1,
                    "location": "/Users/dvvarin/BMSTU/xoracle-
                       bachelors-diploma/app-blocks/blocks/bmstu.
                       png.1",
                    "size": 101090
                }
            ]
        },
        {
            "id": 1,
            "block_size": 101090,
            "checksum": "e4fvsoc8890ecb9af8cf7b711f1845d7",
            "location": "/Users/dvvarin/BMSTU/xoracle-bachelors-
               diploma/app-blocks/s2/bmstu.png.1.sb",
            "blocks": [{
                    "id": 0,
                    "location": "/Users/dvvarin/BMSTU/xoracle-
                       bachelors-diploma/app-blocks/blocks/bmstu.
                       png.1",
                    "size": 101090
                },
                {
                    "id": 1,
                    "location": "/Users/dvvarin/BMSTU/xoracle-
                       bachelors-diploma/app-blocks/blocks/bmstu.
                       png.2",
```

Листинг А.4 – Конфигурационный файл метода. Часть 4.

```
                    "size": 101088
                }
            ]
        },
        {
            "id": 2,
            "block_size": 101090,
            "checksum": "8vn3yc88f7b711f18e4fvso456c42c9d",
            "location": "/Users/dvvarin/BMSTU/xoracle-bachelors-
               diploma/app-blocks/s3/bmstu.png.2.sb",
            "blocks": [{
                    "id": 0,
                    "location": "/Users/dvvarin/BMSTU/xoracle-
                       bachelors-diploma/app-blocks/blocks/bmstu.
                       png.0",
                    "size": 101090
                },
                {
                    "id": 1,
                    "location": "/Users/dvvarin/BMSTU/xoracle-
                       bachelors-diploma/app-blocks/blocks/bmstu.
                       png.2",
                    "size": 101088
                }
            ]
        }
    ],
    "lastUsedServerIndex": 1
}
```