Коннов Д.В. Использование RESTful сервисов в NAS устройствах на примере SYNCOVERY // Международный журнал информационных технологий и энергоэффективности. – 2023. –
Т. 8 № 7(33) с. 48–60

Международный журнал информационных технологий и энергоэффективности

Сайт журнала:
http://www.openaccessscience.ru/index.php/ijcse/

УДК 004

ИСПОЛЬЗОВАНИЕ RESTful СЕРВИСОВ В NAS УСТРОЙСТВАХ НА ПРИМЕРЕ SYNCOVERY

Коннов Д.В.
Университет Центральной Флориды, Орландо, США (32816, Орландо, Бульвар Центральная Флорида, 4000), e-mail: konnov72@knights.ucf.edu

В статье рассматриваются современные тенденции и подходы в области резервного копирования и синхронизации данных, дан обзор архитектуры NAS устройств и сопутствующего программного обеспечения, на примере программного сервиса Syncovery работающего с железом NAS и QNAP. В данном проекте обсуждаются вопросы унификации доступа к облачным хранилищам, создание сервиса, берущего на себя рутинную работу по синхронизации данных между различными хранилищами, резервное копирование и восстановление данных, обеспечение платформенной независимости данного сервиса, реализация избыточности хранения данных, пути повышения надежности хранения данных. Обсуждаются аспекты параллельной и последовательной передачи файлов. Рассмотрено внутреннее устройство программного обеспечения синхронизации файлов с использованием REST запросов на примере взаимодействия с GoogleDrive.

Ключевые слова: NAS, Synology, Syncovery, REST, GoogleDrive, облачные хранилища, односторонняя-двусторонняя синхронизация, версионность, синхронизация в реальном времени, блочное копирование, сжатие, архивирование, шифрование, параллелизм.

USING RESTful SERVICES IN NAS DEVICES ON THE EXAMPLE OF SYNCOVERY

Konnov D.V.
University of Central Florida, Orlando, USA (4000 Central Florida Blvd., Orlando, 32816), e-mail: konnov72@knights.ucf.edu

The article discusses current trends and approaches in the field of data backup and synchronization, gives an overview of the architecture of NAS devices and related software, using the Syncovery software service working with NAS and QNAP hardware as an example. This project discusses the issues of unifying access to cloud storage, creating a service that takes over the routine work of synchronizing data between different storages, backing up and restoring data, ensuring platform independence of this service, implementing data storage redundancy, and ways to improve data storage reliability. The aspects of parallel and serial file transfer are discussed. The internal structure of the file synchronization software using REST requests is considered on the example of interaction with GoogleDrive.

Keywords: NAS, Synology, Syncovery, REST, GoogleDrive, cloud storage, One-way-Two-way synchronization, Versioning, Real-time synchronization, Block copy, compression, archiving, encryption, parallelism.

Введение

В связи с ростом количества облачных хранилищ от провайдеров мировой облачной IT индустрии, таких как Amazon S3, Google Drive, Microsoft Azure, OneDrive, SharePoint, DropBox, Box, Backblaze B2 интеграция облачных хранилищ данных становится одной из наиболее востребованных задач современного пользователя. Каждое облачное хранилище имеет свою специфику, свой интерфейс взаимодействия с пользователем, свой API. Идеальным с точки зрения пользователя является сетевое устройство, которое берет на себя функцию резервного копирования файлов и будучи полностью автономным модулем, в фоновом режиме занимается синхронизацией локального контента с облаком. В таком случае стирается грань различия между локальным диском и облаком. Одно плавно перетекает в другое и наоборот. Пользователь получает возможность не заботиться о сохранности данных, возлагая всю рутину на автоматизированные процессы и устройства, которые этим занимаются. Эволюция развития средств хранения данных явила миру класс устройств под названием NAS.

Сетевое хранилище (NAS) — это компьютерный сервер хранения данных, подключенный к компьютерной сети, предлагающий хранилище на уровне файлов и предоставляющий доступ к данным различным клиентам в сети [1, 2]. Термин «NAS» может относиться как к используемым технологиям и системам, так и к специальному устройству, специально разработанному для этой цели. В отличие от тесно связанных технологий, таких как локальные сети, устройство NAS обычно представляет собой автономную единицу, выступающую в качестве специализированного и единственного решения для хранения данных. Сетевая топология архитектуры NAS показана на Рисунке 1.

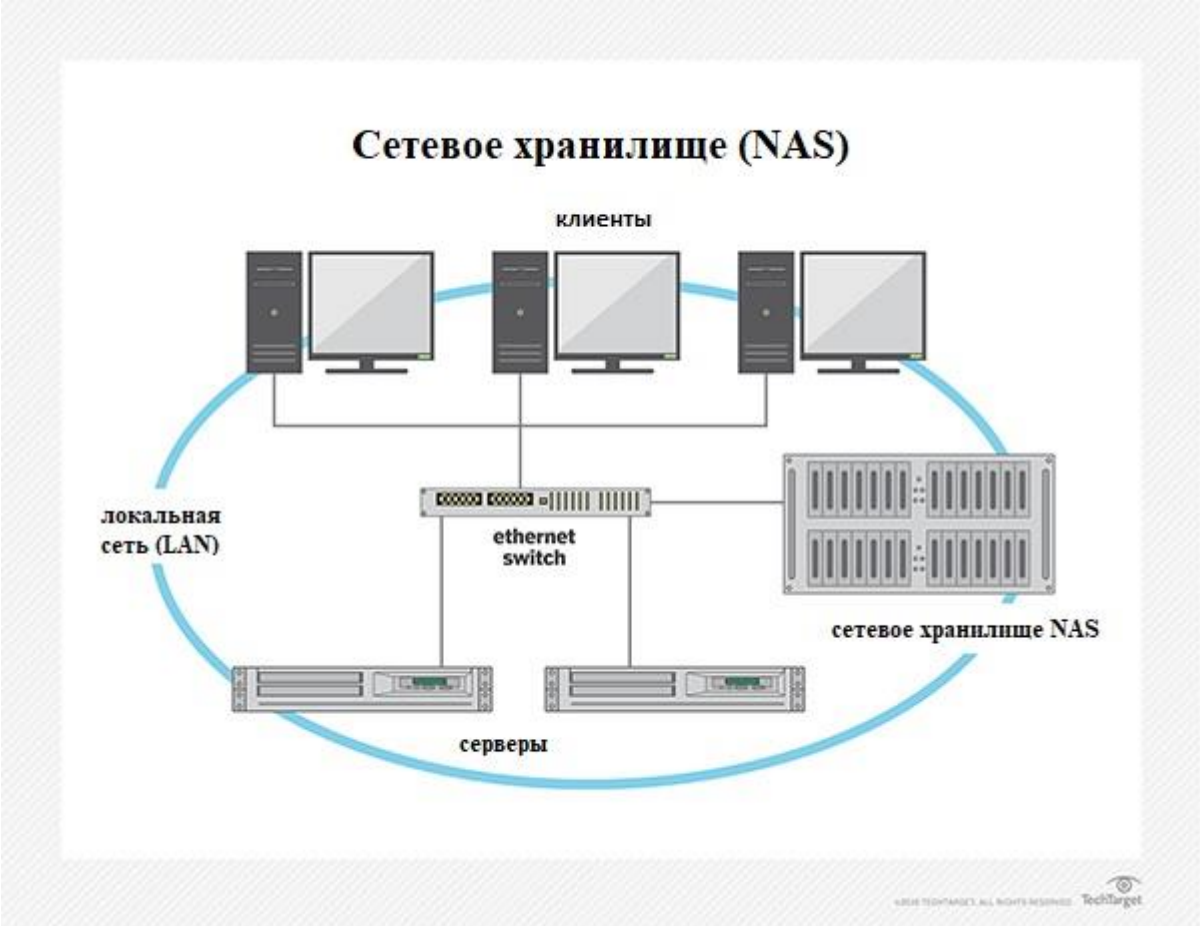

Рисунок 1 – Сетевая топология архитектуры NAS.

Устройство NAS специально разработано для эффективного обслуживания файлов с помощью своего оборудования, программного обеспечения или конфигурации. Он функционирует как сетевое устройство, часто напоминающее специально созданный компьютер, и содержит один или несколько дисков хранения, организованных в избыточные контейнеры хранения или конфигурации RAID [3]. Внутреннее устройство NAS представлено на Рисунке 2. Основной способ доступа к файлам в системах NAS осуществляется через сетевые протоколы обмена файлами, такие как NFS, SMB или AFP. С середины 1990-х годов устройства NAS приобрели популярность как удобное решение для обмена файлами между несколькими компьютерами и для снятия нагрузки по обслуживанию файлов с других серверов в сети. Таким образом, NAS обеспечивает более быстрый доступ к данным, более простое администрирование и более легкую настройку по сравнению с использованием серверов общего назначения для обслуживания файлов. Вместе с NAS поставляются специальные жесткие диски, которые похожи на диски, отличные от NAS, но могут иметь другую прошивку, виброустойчивость или рассеиваемую мощность в соответствии с RAID-массивами, обычно используемыми в настройках NAS. Например, некоторые версии дисков NAS поддерживают расширение команды для отключения расширенного восстановления после ошибок. В приложениях, отличных от RAID, дисковый накопитель может пойти на многое, чтобы успешно прочитать проблемный блок хранилища, даже если это займет несколько секунд. Однако в правильно сконфигурированном RAID-массиве один сбойный блок на одном диске можно полностью восстановить за счет избыточности, закодированной в RAID-наборе. Если диск тратит слишком много времени на повторные попытки, RAID-контроллер может считать диск «неработающим». С другой стороны, если диск быстро сообщает об ошибке контрольной суммы для блока данных, контроллер RAID может исправить ошибку, используя избыточные данные на других дисках, и продолжить работу без проблем. Такой жесткий диск SATA «NAS» также можно использовать в качестве внутреннего жесткого диска ПК без каких-либо проблем или необходимых настроек, поскольку он просто предлагает дополнительные опции и может быть построен в соответствии с более высоким стандартом качества (особенно если сопровождается более высоким значением MTBF(Mean time between failures) [4] и более высокой ценой) по сравнению с обычным потребительским диском.

Для связи NAS использует файловые протоколы, такие как NFS (обычно используется в системах UNIX), SMB (блок сообщений сервера) (используется в системах Microsoft Windows), AFP (используется в компьютерах Apple Macintosh) или NCP (используется в OES и Novell NetWare). Устройства NAS часто поддерживают несколько протоколов одновременно и не ограничиваются одним. Synology, Netgear ReadyNAS, QNAP QTS, Zyxel FW или TrueNAS Core — это далеко не полный список ведущих поставщиков NAS-устройств.

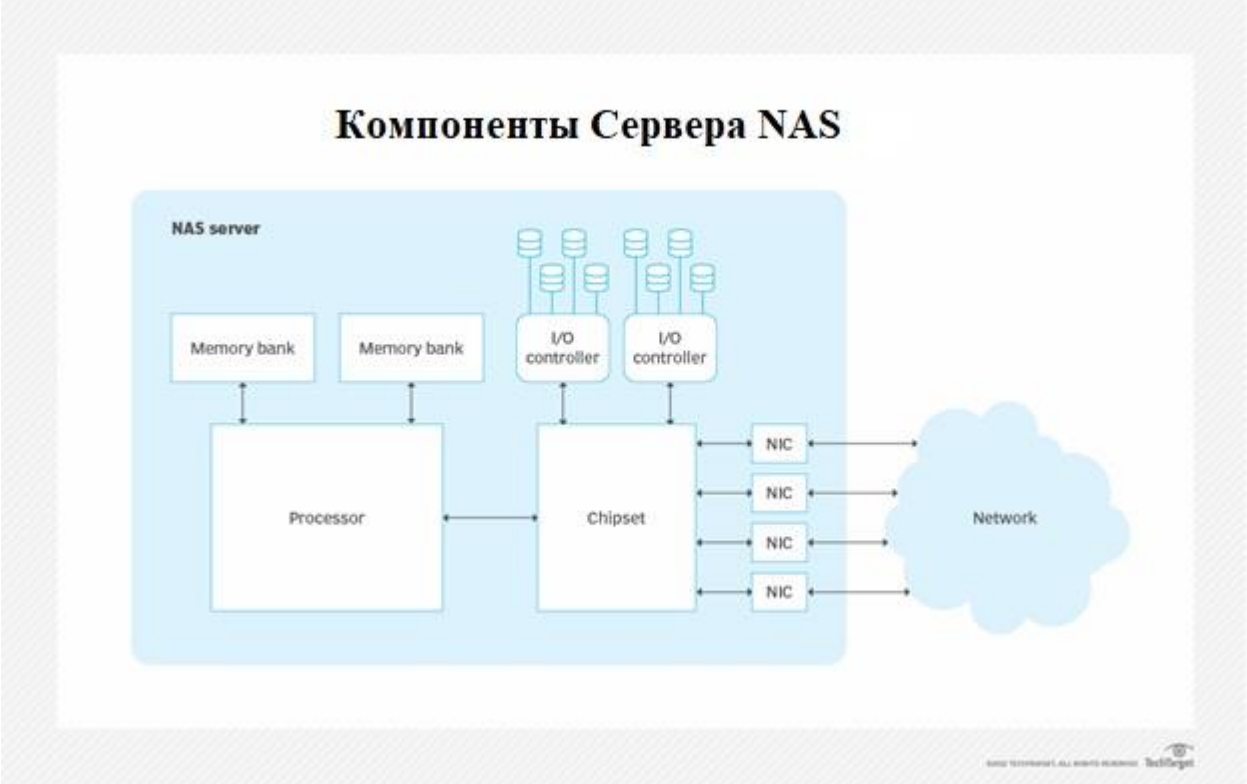

Рисунок 2 – Внутреннее устройство NAS.

Syncovery и ее возможности

Основой каждого NAS устройства является операционная UNIX-подобная система. В последнее время распространение в области устройств NAS получили такие ОС как FreeBSD, XigmaNAS [5, 6] и FreeNAS/TrueNAS. В рамках описываемого проекта был разработан универсальный слой программного обеспечения, позволяющий обеспечить синхронизацию локального и облачных хранилищ напрямую из устройства NAS. Данный сервис носит название Syncovery, который в последствии переродился в настоящий коммерческий продукт и в данное время используется на сотнях тысяч хостов по всему миру. Идея создания Syncovery принадлежит Тобиасу Гессену — программисту, музыканту, владельцу маленькой софтверной компании в Германии [7]. Автор данного сообщения непосредственно принимал участие в разработке продукта. Первая версия приложения, написанного на Delphi, была выпущена под названием Super Flexible File Synchronizer в 2003 году. В 2012 году продукт был переименован в Syncovery. Delphi как основной инструмент разработки, в данном случае для встраиваемых устройств, хорошо зарекомендовал себя и показал непревзойденную гибкость в разработке данного кроссплатформенного продукта [8]. В 2021 году на конкурсе Delphi Showcase Challenge, посвященном 26-летию языка программирования Delphi, инструмент Syncovery был удостоен четвертой премии как отличное программное обеспечение для синхронизации рабочего стола, которое демонстрирует гибкость среды разработки Delphi. Поскольку компилятор Delphi поддерживает компиляцию для Windows, Linux и Mac, удалось создать превосходное кроссплатформенное, дружественное к пользователю решение. В наши дни Syncovery стало неотъемлемым профессиональным инструментом каждого системного администратора.

В основе продукта прежде всего лежит простота понимания концепции синхронизации данных. Для этого было предложено предоставить пользователю интуитивно понятный графический интерфейс в виде Левый – Правый, как изображено на Рисунке 3.

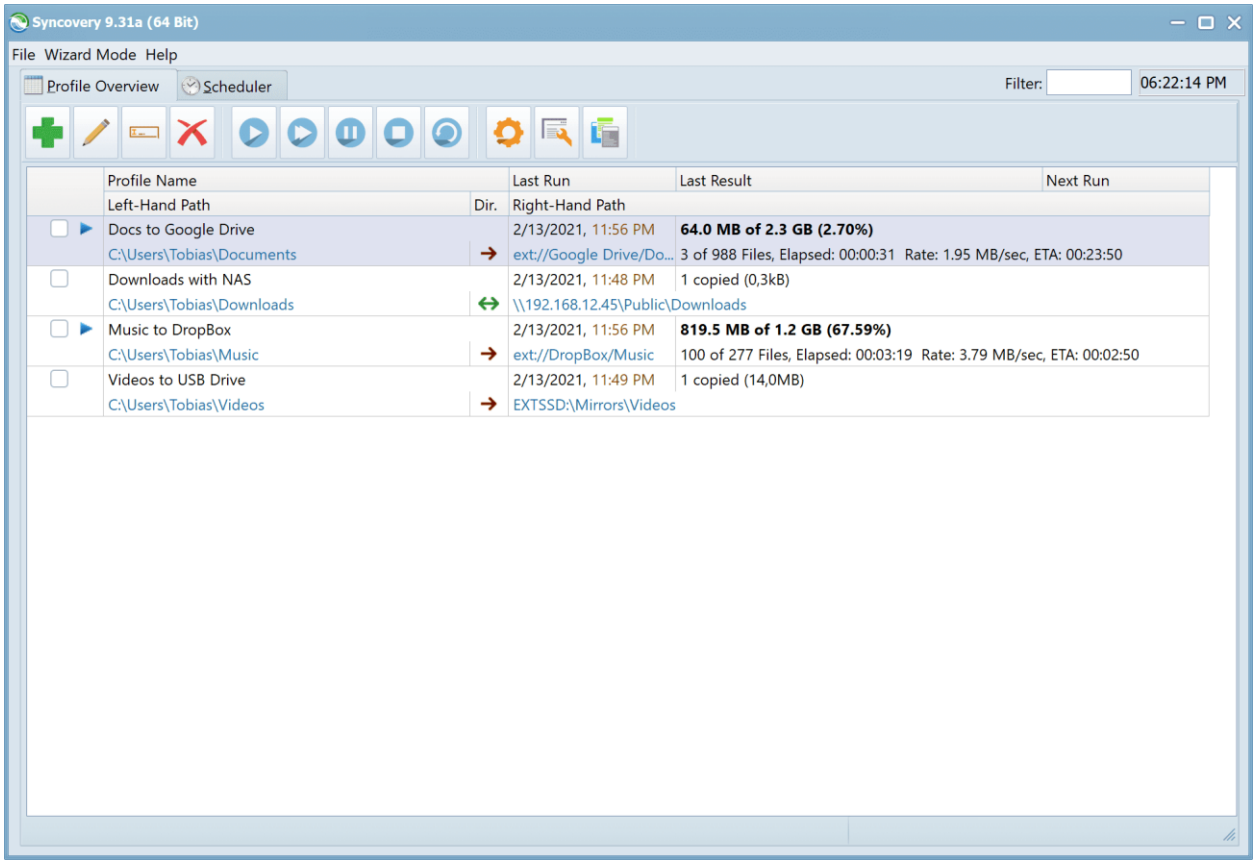

Рисунок 3 – Интерфейс пользователя Syncovery.

Левый – Правый путь – это локация файлов, которая может быть как локальной директорией, так и облачной. Пользователь настраивает точки(папки) синхронизации своих файлов, и встроенный Планировщик занимается запуском заданий синхронизации.
Рассмотрим ключевые возможности файловой синхронизации, реализованные в программном обеспечении:

- **Односторонняя, Двусторонняя синхронизация**

Файлы можно синхронизировать как в одну, так и в две стороны. Односторонняя синхронизация предполагает отслеживание изменений файла только на одном конце и распространение этих изменений только на другом конце, двусторонняя синхронизация, напротив вызовет распространение файла с обоих концов.

- **Версионность**

Syncovery также может хранить несколько версий файла в одной папке в резервной копии.

- **Синхронизация в реальном времени**

Синхронизация в реальном времени — это метод отслеживания папок на предмет изменений и копирования новых или измененных файлов с очень короткой задержкой после внесения изменений. Поскольку папки отслеживаются, их не нужно сканировать — шаг «Построение списка файлов» не выполняется. Синхронизация в реальном времени может быть очень

удобной и значительно ускоряет процесс синхронизации. Однако он будет копировать только новые и измененные файлы и может не подходить для всех ситуаций. Эта функция основана на уведомлениях об изменениях в режиме реального времени, которые она получает от операционной системы. Поэтому он работает только тогда, когда хранилище, из которого вы копируете, полностью поддерживает такие уведомления об изменении. Например, локальные жесткие диски обычно полностью поддерживаются. Большинство файловых систем также отправляют уведомления об изменениях по локальной сети. Но многие устройства NAS не отправляют уведомления об изменениях должным образом. А иногда, даже если уведомления отправляются, операционная система может их игнорировать (например, Mac OS). Кроме того, уведомления об изменениях не отправляются серверами, использующими интернет-протоколы, такие как FTP, SSH, WebDAV и т. д.

- **Блочное копирование (частичное обновление файла)**

Syncovery может определить, какие части большого файла были изменены, и скопировать только измененные блоки, а не весь файл. Эта функция работает аналогично Rsync (но это не то же самое) [9]. Копирование только измененных блоков может сэкономить полосу пропускания и время, особенно при медленном соединении [10]. При копировании между локальными дисками или в локальной сети это также может сэкономить полосу пропускания, но это не всегда может сэкономить больше времени на копирование, потому что исходный файл необходимо считывать каждый раз целиком, чтобы определить измененные блоки.

В этом режиме для копирования подходят только блочные типы файлов. К ним относятся файлы базы данных, такие как SQL или Outlook PST, а также образы дисков и образы виртуальных жестких дисков (VMs). Потоковые файлы, с другой стороны, обычно вызывают изменение всех блоков всякий раз, когда они модифицируются (например, текстовые документы, электронные таблицы, zip-файлы и фотографии). Таким образом, копирование на уровне блоков не сможет значительно сэкономить пропускную способность при использовании потоковых файлов.

В Syncovery копирование на уровне блоков иногда также называют «частичным обновлением файлов». Программа должна иметь быстрый доступ хотя бы к одной из сторон синхронизации. Другая сторона может быть соединением с низкой пропускной способностью. Для интернет-протокола только SSH/SFTP поддерживает непосредственное обновление на уровне блоков. Другие протоколы можно использовать только с синтетическим резервным копированием.

- **Сжатие/архивирование/шифрование**

Файлы также могут быть сжаты и зашифрованы. Доступно множество вариантов. Файлы могут быть сжаты как по папкам, так и по отдельности в сочетании с кодировкой имени файла. Поддерживается надежное шифрование файлов AES-256[11].

Особенности синхронизации с использованием параллелизма

Каждое облачное хранилище такое как Amazon S3, Google Drive, Microsoft Azure, OneDrive, SharePoint, DropBox, Box, Backblaze B2 предоставляет разработчику свой API доступа или REST интерфейс, включая методы аутентификации при подключении такие как OAuth/OAuth2. Внутренняя архитектура программного кода Syncovery многопоточна. Как правило на один синхронизируемый файл создается один поток приема – передачи файла, однако при передаче больших файлов данные могут нарезаться в chunks(кусочки) и передаваться на сторону облачного хранилища и загружаться последовательно или

параллельно [12]. В этом случае обрывы связи становятся некритичными, а производительность процесса повышается. При приеме chunk(кусочка) из облачного хранилища удаленному серверу, на примере GoogleDrive используется HTTP Header ‘Range’ а также начало и конец запрашиваемого кусочка. Например:

```
IdHTTP.Request.CustomHeaders.Values['Range'] := Format('bytes=%d-%d', [StartByte, EndByte]);
```

Из приведенного выше фрагмента кода на Delphi видно, что библиотека Indy [13] является неотъемлемой частью современных систем работы с облаком. При загрузке файла фрагментами скорость загрузки потенциально может повыситься при использовании параллельных загрузок, особенно при работе с большими файлами и достаточно быстрым подключением к Интернету. Однако фактическое увеличение скорости загрузки за счет параллельных загрузок зависит от различных факторов, таких как мощность сервера, состояние сети и количество параллельных подключений, разрешенных сервером. Чтобы понять, как параллельные загрузки могут улучшить скорость загрузки, рассмотрим некоторые основные математические рассуждения:

Предположим, мы хотим загрузить файл размера S в N chunks (параллельных соединениях). Для простоты предположим, что размер файла делится без остатка на количество фрагментов, т. е. S можно разделить на N частей одинакового размера.

- Последовательная загрузка. При последовательной загрузке файл загружается за одно соединение, а время загрузки может быть представлено как $T_{\text{sequential}}$.

$$T_{\text{sequential}} = S / R_{\text{sequential}} \quad (1)$$

Где $R_{\text{sequential}}$ — скорость загрузки для одного соединения.

- Параллельная загрузка. При параллельной загрузке файл делится на N фрагментов, и каждый фрагмент загружается одновременно с использованием N параллельных соединений. Время загрузки можно представить как T_{parallel}.

$$T_{\text{parallel}} = S / (N * R_{\text{parallel}}) \quad (2)$$

Где R_{parallel} — скорость загрузки для каждого параллельного соединения.

- Общее улучшение скорости загрузки. Чтобы понять улучшение скорости загрузки, мы можем сравнить общее время, затрачиваемое на последовательную и параллельную загрузку:

$$\text{Speed\_Improvement} = T_{\text{sequential}} / T_{\text{parallel}} = R_{\text{parallel}} / (N * R_{\text{sequential}}) \quad (3)$$

Фактор Speed_Improvement зависит от отношения скорости загрузки для одного параллельного соединения $R_{parallel}$ к скорости загрузки для одного последовательного соединения $R_{sequential}$ и количества параллельных соединений N.

Если сервер и сеть могут эффективно обрабатывать несколько параллельных подключений, фактор Speed_Improvement может быть значительным, что приводит к более быстрому времени загрузки. Однако существуют практические ограничения, которые следует учитывать:

- Емкость сервера.

Некоторые серверы могут ограничивать количество параллельных подключений, разрешенных для каждого клиента, чтобы предотвратить злоупотребление и обеспечить справедливое использование [14].

- Сетевые условия.

Высокий уровень параллелизма может привести к перегрузке и снижению производительности, если сеть не сможет справиться с возросшим трафиком.

- Размер файла.

Для небольших файлов накладные расходы на настройку нескольких подключений могут перевешивать преимущества параллельных загрузок.

- Накладные расходы CPU и диска.

При работе с очень высокоскоростными соединениями и небольшими файлами накладные расходы на управление несколькими параллельными соединениями могут потреблять больше ресурсов, чем фактический процесс загрузки [15]. В целом, параллельные загрузки могут повысить скорость загрузки, но степень улучшения зависит от упомянутых выше факторов. В реальных сценариях можно добиться значительного повышения скорости за счет параллельных загрузок, особенно при загрузке больших файлов по-быстрому и надежному соединению. Однако важно учитывать практические ограничения и соответствующим образом сбалансировать количество параллельных подключений.

Взаимодействие с облаком при помощи REST

В современных API взаимодействия с облаком XML формат обмена данными вытесняется форматом JSON, что во многом облегчает разработку. Рассмотрим взаимодействие с облачным файл хранилищем на примере Google Drive. Для взаимодействия с удаленным хранилищем по http вначале необходимо получить endpoints. В контексте REST (передача репрезентативного состояния) конечная точка относится к определенному URL-адресу (унифицированный указатель ресурса) или URI (унифицированный идентификатор ресурса) на веб-сервере, который соответствует определенному ресурсу или функции, предоставляемой сервером.

Конечные точки — важная концепция в дизайне RESTful API [16]. Каждая конечная точка представляет отдельный ресурс или операцию, к которым клиент может получить доступ через API. Этими ресурсами могут быть объекты данных, такие как файлы. Когда клиентское приложение хочет взаимодействовать с RESTful API, оно делает это, отправляя HTTP-запросы на соответствующие конечные точки. Метод HTTP (например, GET, POST,

PUT, DELETE), используемый в запросе, указывает тип действия, которое клиент хочет выполнить с ресурсом, связанным с этой конечной точкой [17].

Например, рассмотрим простой RESTful API для управления сообщениями в блогах. API может иметь следующие конечные точки:

Получить метаданные файла:

GET https://www.googleapis.com/drive/v3/files/{fileId}

позволяет получать метаданные (информацию) об определенном файле на Google Диске. Замените {fileId} идентификатором файла, к которому вы хотите получить доступ.

Получить список файлов на диске:

GET https://www.googleapis.com/drive/v3/files

получает список всех файлов на Google Диске пользователя. Для фильтрации и настройки результатов необходимо использовать параметры запроса.

Загрузить файл на сервер:

POST https://www.googleapis.com/upload/drive/v3/files

используется для загрузки нового файла на Google Диск. Содержимое файла включается в тело запроса, а метаданные (имя файла, родительская папка и т. д.) могут быть указаны в параметрах запроса.

Обновление Файла:

PATCH https://www.googleapis.com/drive/v3/files/{fileId}

Используется для обновления метаданных существующего файла на Google Диске. Позволяет изменять такие атрибуты, как имя файла, описание и родительская папка.

Загрузка файла с сервера:

GET https://www.googleapis.com/drive/v3/files/{fileId}?alt=media

позволяет получить содержимое файла с Google Диска. Параметр {fileId} – идентификатор файла, который необходимо загрузить. Параметр запроса alt=media используется для прямого извлечения содержимого файла.

Удаление файла:

DELETE https://www.googleapis.com/drive/v3/files/{fileId}

используется для удаления определенного файла с Google Диска. Параметр {fileId} идентификатор удаляемого файла.

Поиск файлов:

GET https://www.googleapis.com/drive/v3/files?q={searchQuery}

позволяет искать файлы на Google Диске с помощью поискового запроса. Параметр запроса q используется для указания критериев поиска.

Это всего лишь несколько примеров конечных точек, доступных в API Google Диска. API предлагает гораздо больше конечных точек и функций для работы с файлами, папками, разрешениями и многим другим на Google Диске. При использовании Google Диска API потребуется аутентифицировать свои запросы и получить необходимые токены доступа для взаимодействия с Google Диском пользователя от его имени.

Аутентификация — это процесс проверки личности пользователя или приложения перед предоставлением доступа к определенным ресурсам или функциям [18]. В контексте API Google Диска аутентификация является важным шагом для обеспечения того, чтобы только авторизованные пользователи или приложения могли получать доступ и управлять файлами и данными, хранящимися на Google Диске пользователя. API Google Диска использует OAuth

2.0 в качестве механизма аутентификации, который является стандартным протоколом для безопасной авторизации. Протокол OAuth 2.0 включает несколько шагов для получения токена доступа, который затем используется для проверки подлинности последующих запросов API.

Упрощенное объяснение процесса аутентификации OAuth 2.0 для доступа к GoogleDrive API необходимо:

- Приложение, работающее с GoogleDrive должно быть зарегистрировано.

Это делается путем создания проекта в Google Developers Console и включением API Диска Google для этого проекта. На этом шаге создаются необходимые учетные данные (идентификатор клиента и секрет клиента), которые будут использоваться в процессе аутентификации.

- Получить авторизацию

Когда пользователь хочет использовать приложение для доступа к своему GoogleDrive, приложение Syncovery перенаправит его на страницу входа в учетную запись Google. Затем пользователь войдет в систему со своими учетными данными Google и предоставит приложению Syncovery разрешение на доступ к своим данным Google Диска.

- Код авторизации

После предоставления разрешения пользователь будет перенаправлен обратно в приложение Syncovery с кодом авторизации.

- Код авторизации Exchange для токена доступа:

Syncovery использует этот код авторизации для запроса маркера доступа с сервера авторизации Google. Этот токен представляет разрешение пользователя и действует как ключ для доступа к его данным на Google Диске.

- Доступ к GoogleDrive

С токеном доступа, полученным на предыдущем шаге, Syncovery может выполнять аутентифицированные запросы к API GoogleDrive от имени пользователя. API проверит токен и предоставит доступ к авторизованным ресурсам.

- Обработка токенов обновления

Токены доступа [19] обычно имеют ограниченный срок службы. Чтобы поддерживать постоянный доступ к данным пользователя на GoogleDrive токен обновления может быть запрошен во время первоначального процесса аутентификации. Маркер обновления можно использовать для получения новых маркеров доступа, не требуя от пользователя повторной авторизации приложения. Очень важно хранить учетные данные (идентификатор клиента и секрет клиента) в безопасности и не раскрывать их в общедоступных репозиториях кода или коде на стороне клиента. Правильная обработка токенов аутентификации и доступа обеспечивает безопасность и конфиденциальность данных пользователя. Чтобы реализовать аутентификацию OAuth 2.0 с API GoogleDrive в своем приложении, могут использоваться различные библиотеки и SDK, предоставляемые Google для разных языков программирования и платформ. Эти библиотеки упрощают процесс аутентификации и помогают сосредоточиться на создании функциональности вашего приложения, взаимодействующего с GoogleDrive. Важно тщательно проектировать конечные точки RESTful API, чтобы обеспечить ясность, согласованность и простоту использования для разработчиков клиентов. Значимые и хорошо

структурированные конечные точки способствуют созданию более интуитивно понятного и удобного для разработчиков API.

Заключение.

В статье рассмотрено использование сторонних RESTful-сервисов в качестве внешних источников данных в приложениях резервного копирования и синхронизации данных. Целью данного исследования была поставлена задача предложить развитие темы резервного копирования и синхронизации путем слияния устройств класса NAS с облачными хранилищами. Таким образом стирая границы между локальным и облачным сегментом данных. Новизна данного исследования заключается в предоставлении универсального кроссплатформенного слоя программного обеспечения, которое может работать как на устройствах NAS так и на локальных станциях под управлением Windows, Linux, Mac.

Список литературы

1. "An introduction to network attached storage". HWM Singapore. SPH Magazines. July 2003. pp. 90–92
2. Васильев А. QNAP: Продвижение NAS в России // Первая миля. 2013. Т. 39. № 6. С. 4-15.
3. Терентьев Д.И., Николаев А.Б., Остроух А.В. Исследование дисковых массивов RAID по параметрам надежности и быстродействия // Международный журнал экспериментального образования. 2015. № 3-4. С. 423-427.
4. Проскуряков Н.Е., Ануфриева А.Ю. Анализ и перспективы современных систем хранения цифровых данных // Известия Тульского государственного университета. Технические науки. 2013. № 3. С. 368-377.
5. Чусов П.А. Создание файлового хранилища на основе программного продукта NAS4FREE // Аллея науки. 2017. Т. 2. № 8. С. 714-719.
6. XigmaNAS is an Open Source Storage NAS (Network-Attached Storage) distribution based on FreeBSD. // https://xigmanas.com/xnaswp (дата обращения: 26.07.2023)
7. Tobias Giesen // https://ru.syncovery.com (дата обращения: 26.07.2023)
8. Шевцов А.Н., Бекен Б.К., Таласбаев А.А. Исследование параллельных вычислений на Delphi // Theoretical & Applied Science. 2013. № 5 (1). С. 27-35.
9. Ханенко А.А., Медведский А.Н. Синхронизация файлов и резервное копирование с помощью Rsync // Международный научный журнал. 2016. № 6-2. С. 28-30.
10. Архипов Д.Р. Метод удаленной разностной синхронизации файлов ориентированный на минимизацию передаваемой информации // «StudNet» №11/2020
11. Нурмухаметов Д.Р. Исследование стандартов шифрования AES-128 И AES-256 Научное Сообщество Студентов. Междисциплинарные Исследования. Сборник статей по материалам CX студенческой международной научно-практической конференции. Новосибирск, 2021. С. 37-42.
12. Dan C. Marinescu Cloud Computing: Theory and Practice // Elsevier Science 2022
13. Indy programming library for Delphi // https://www.indyproject.org
14. Зеленский М.Д. DDOS-Атаки: Типы атак, Устранение DDOS-Атак // Студенческая наука для развития информационного общества. сборник материалов IV Всероссийской научно-технической конференции: в 2-х томах. 2016. С. 241-243.

15. Gibson G.A. Redundant disk arrays: Reliable, parallel secondary storage // 1992
16. А.С. Гусаренко, В.В. Миронов Использование RESTful-сервисов в ситуационно-ориентированных базах данных // Системы управления и информационные технологии 2015 г.
17. Строгонов В.Г. Краткий обзор протоколов HTTP/1.1 И HTTP/2 и сравнение их производительности // Математика, информационные технологии, приложения. Сборник трудов Межвузовской научной конференции молодых ученых и студентов. Воронеж, 2022. С. 188-191.
18. Д.Н. Вишневский OAuth и его использование в современных системах // Вестник магистратуры. 2019. № 6-2(93)
19. Ситдиков Р.Р. Токен. Инструмент ограниченного пользования в REST API // Информационные технологии обеспечения комплексной безопасности в цифровом обществе. сборник материалов IV Всероссийской молодежной научно-практической конференции с международным участием. Уфа, 2021. С. 208-209.

References

1. "An introduction to network attached storage". HWM Singapore. S.P.H. Magazines. July 2003. pp. 90–92
2. Vasiliev A. QNAP: Promotion of NAS in Russia // First Mile. 2013. V. 39. No. 6. S. 04-15.
3. Terentiev D.I. Nikolaev A.B., Ostroukh A.V. Research of RAID disk arrays in terms of reliability and speed // International Journal of Experimental Education. 2015. No. 3-4. pp. 423-427.
4. Proskuryakov N.E. Anufrieva A.Yu. Analysis and prospects of modern digital data storage systems. Bulletin of the Tula State University. Technical science. 2013. No. 3. pp. 368-377.
5. Chusov P.A. Creating a file storage based on the NAS4FREE software product // Alley of Science. 2017. V. 2. No. 8. pp. 714-719.
6. XigmaNAS is an Open Source Storage NAS (Network-Attached Storage) distribution based on FreeBSD. // https://xigmanas.com/xnaswp (date of access: 07/26/2023)
7. Tobias Giesen // https://ru.syncovery.com (Accessed: 07/26/2023)
8. Shevtsov A.N., Beken B.K. Talasbaev A.A. Investigation of parallel computing on Delphi // Theoretical & Applied Science. 2013. No. 5 (1). pp. 27-35.
9. Khanenko A.A., Medvedsky A.N. File synchronization and backup using Rsync // International Scientific Journal. 2016. No. 6-2. pp. 28-30.
10. Arkhipov D.R. The method of remote differential file synchronization focused on minimizing the transmitted information // "StudNet" No. 11/2020
11. Nurmukhametov D.R. Research on AES-128 AND AES-256 Encryption Standards // Scientific Community of Students. Interdisciplinary Research. Collection of articles based on the materials of the CX student international scientific and practical conference. Novosibirsk, 2021, pp. 37-42.
12. Dan C. Marinescu Cloud Computing: Theory and Practice // Elsevier Science 2022
13. Indy programming library for Delphi // https://www.indyproject.org
14. Zelensky M.D. DDOS Attacks: Types of Attacks, Elimination of DDOS Attacks // Student science for the development of the information society. collection of materials of the IV All-Russian scientific and technical conference: in 2 volumes. 2016. pp.. 241-243.

15. Gibson G.A. Redundant disk arrays: Reliable, parallel secondary storage // 1992
16. A.S. Gusarenko, V.V. Mironov Using RESTful services in situationally oriented databases // Control systems and information technologies 2015.
17. Strogonov V.G. Brief overview of HTTP/1.1 and HTTP/2 protocols and comparison of their performance // Mathematics, information technologies, applications. Proceedings of the Interuniversity Scientific Conference of Young Scientists and Students. Voronezh, 2022, pp. 188-191.
18. D.N. Vishnevsky OAuth and its use in modern systems // Bulletin of the Magistracy. 2019. No. 6-2(93)
19. Sitdikov R.R. Token. Limited use tool in REST API // Information technologies for ensuring integrated security in a digital society. collection of materials of the IV All-Russian Youth Scientific and Practical Conference with international participation. Ufa, 2021. pp. 208-209.