Санкт-Петербургский государственный университет

Математическое обеспечение и администрирование информационных систем

Семенов Александр Сергеевич

Транзакционное хранилище для версионирования бинарных данных в PostgreSQL

Производственная практика

Научный руководитель:
доц. каф. СП, к.ф.-м.н. Луцив Д.В.

Санкт-Петербург
2023

Оглавление

Введение

Сегодня сложно представить себе организацию, деятельность которой обходится без информационных систем. Одной из основных составляющих любой информационной системы являются данные. Для их хранения и управления используются различные технологии, например, базы данных. Для настраивания и администрирования баз данных применяется набор программных средств, называемый системой управления базами данных (СУБД).

Одной из самых популярных объектно-реляционных СУБД является PostgreSQL, которая широко используется в различных областях, включая финансы, науку и образование, а также имеет открытый исходный код, поддерживает репликации, хранимые процедуры и прочие полезные функции, а также совместима с набором требований ACID. Одним из требований ACID является атомарность, которая гарантирует, что транзакция либо будет выполнена полностью, либо не выполняется совсем. Это позволяет сохранять согласованность данных в базе, что может быть критично для многих информационных систем, например, для банковских или систем, взаимодействующих с рынком ценных бумаг.

Для хранения данных внутри базы в PostgreSQL предусмотрено множество типов, таких как integer, text, boolean и прочие. Эти типы данных позволяют эффективно хранить и обрабатывать структурированные данные, однако они не подходят для неструктурированных данных, например, бинарных. Основное отличие этих типов данных в том, что неструктурированные данные не соответствуют заранее определенной структуре данных, из-за этого подходы к хранению и обработке такого рода информации должны отличаться от соответствующих подходов работы со структурированными данными. Примерами бинарных данных, которые может быть полезно хранить информационной системе, могут быть pdf-документы, картинки, архивы, электронные письма и тп.

Если бинарные данные имеют небольшой размер, то часто для их

версионирования заводят отдельную таблицу, в которой хранят метаданные, такие как дата последнего изменения файла и ссылка на конкретную версию файла. Для такого подхода необходимо хранить все версии файлов отдельными записями. Проблемы возникают, когда появляется необходимость хранить большие файлы, так как из-за множества их версий разрастается сама база данных, а также это будет сказываться на её производительности.

В данной работе будут рассмотрены подходы к версионированию бинарных файлов внутри СУБД PostgreSQL, предложен подход, позволяющий делать это эффективно с точки зрения занимаемого дискового пространства и времени, а также поддерживающий транзакционность операций над бинарными данными.

1. Постановка задачи

Данная работа выполняется в рамках проекта, над которым работают два человека. Целью именно этой работы является написание библиотеки для версионирования бинарных данных, которая будет использоваться для написания расширения для PostgreSQL, позволяющее управлять данными при помощи синтаксиса SQL.

Для достижения цели были поставлены следующие задачи:

- Изучить существующие подходы к решению задачи версионирования бинарных данных в PostgreSQL.
- Определить подход к версионированию бинарных данных.
- Реализовать библиотеку для написания расширения PostgreSQL для версионирования бинарных данных.
- Провести тестирование полученного решения.

2. Обзор

2.1. Стандартные средства PostgreSQL

В СУБД PostgreSQL для хранения бинарных данных предусмотрены типы bytea и BLOB[1] [2]. Тип bytea является двоичной строкой переменной длины. Этот тип использует механизм TOAST[2], который позволяет разбивать большие файлы на множество мелких. Для этого заводится специальная таблица, которая хранит ссылки на части файла и с помощью которой впоследствии можно восстановить файл из частей [4]. Однако этот механизм не был разработан для поддержки обновлений данных. Поэтому в случае обновления данных sql-операцией UPDATE, механизм TOAST будет дублировать все части сохраненных данных, даже если изменения затрагивают какую-то одну часть из них [3]. Помимо этого, такой подход накладывает ограничения на производительность СУБД при работе с большими данными.

Помимо bytea, в PostgreSQL есть средство хранения больших объектов BLOB, который обеспечивает потоковый доступ к данным, однако для работы с ним нужно использовать отличающийся от стандартного интерфейс, и этот механизм не предусмотрен для версионирования данных.

Несмотря на то, что эти способы хранения данных поддерживают транзакционность, они не позволяют версионировать бинарные данные так, чтобы размер базы не увеличивался из-за количества их версий.

2.2. Система контроля версий Git

Когда речь заходит о версионировании, нельзя не упомянуть о системе контроля версий Git. Она позволяет хранить историю изменений, поддерживает ветвления и работу с конфликтами. Наибольшую популярность эта система обрела среди разработчиков, потому что позволяет эффективно работать с изменением текстовых файлов с исходными

[1]Binary Large Object

[2]The Oversized Attribute Storage Technique

кодами программ. Помимо текстовых файлов, Git позволяет версионировать и бинарные файлы. Можно было бы представить себе решение поставленной задачи следующим образом: завести внешнее хранилище данных, настроить там Git и подключить его к СУБД. Однако здесь возникает существенный недостаток: Git умеет определить, что изменился бинарный файл, но не умеет определять, в каком именно месте произошло изменение. Из-за этого любое изменение файла, даже самое небольшое, будет приводить к созданию копии файла, а если мы имеем дело с большими файлами, то размер нашего хранилища будет существенно расти.

3. Предлагаемое решение

Существуют два основных способа решения поставленной задачи: написание расширения для PostgreSQL, которое реализовывало бы логику версионирования бинарных файлов внутри базы данных, или хранение файлов в другом месте, например, в файловом хранилище. Первый вариант предполагает, что внутри СУБД файлы хранятся в одном из типов bytea или BLOB, однако операции с такими данными были бы медленее, чем если бы файлы хранились в файловом хранилище [1]. С другой стороны, хранение файлов в файловой системе не гарантирует атомарность операций над ними.

3.1. Описание решения

Однако проблему можно решить, совместив два подхода. В данной работе предлагается решить проблему хранения и версионирования бинарных данных в СУБД PostgreSQL следующим образом: заводится сетевое файловое хранилище с файловой системой, которая позволяет на её основе реализовать необходимый для отслеживания версий файлов функционал. Предполагается, что у сервера базы данных есть доступ к этому сетевому хранилищу.

На основе файловой системы на сетевом хранилище будет реализован API, позволяющий взаимодействовать с файлами через стандартный синтаксис SQL, то есть поддерживающий стандартные операции SELECT, INSERT, UPDATE и DELETE. Далее будет написано расширение для PostgreSQL, использующее этот API и позволяющее сохранять и версионировать бинарные файлы из СУБД. Благодаря этому будет возможность взаимодействовать с файлами стандартными средствами СУБД, не заботясь о том, как эти файлы на самом деле хранятся.

3.2. Выбор файловой системы

Были выделены основные критерии отбора файловой системы для написания алгоритма версионирования бинарных данных и API для поддержки написания расширения для PostgreSQL:

- Наличие библиотеки для взаимодействия с файловой системой из программного кода;
- Поддержка создания снапшотов для реализации алгоритма версионирования;
- Поддержка транзакций через механизм copy-on-write.

Снапшоты – это специальные снимки, которые хранят полное состояние диска или файла в определенный момент времени. Благодаря этому механизму, появляется возможность хранить несколько снимком определенного файла или директории и возвращаться к любому снимку в случае необходимости.

Механизм copy-on-write позволяет при изменении данных не менять их напрямую, а создает копию данных в другом месте и изменяет её. Это позволяет избежать ошибок в данных при неудачных изменениях. Также эта технология позволяет увеличить скорость записи, так как она стирает старые данные только когда появляется в этом необходимость.

Выбор происходил между файловыми системами EXT4, BTRFS и ZFS, так как эти файловые системы чаще всего используются для сетевых хранилищ, а также у всех них есть библиотеки для взаимодействия с ними.

Несмотря на то, что EXT4 является самой стабильной из рассматриваемых файловых систем, а также по умолчанию используется на большинстве операционных систем семейства Unix, она не поддерживает механизм copy-on-write, поэтому дальше эта файловая система рассматриваться не будет.

Файловые системы BTRFS и ZFS обе поддерживают механизм copy-on-write и создание снапшотов. Однако, при работе с большими объё-

мами данных, BTRFS показывает более низкую эффективность в сравнении с ZFS [5].

Поэтому в дальнейшем для решения поставленной задачи будет использоваться файловая система ZFS. Это хороший выбор для хранения и версионирования бинарных данных в PostgreSQL, так как помимо того, что она поддерживает атомарные операции над файлами и создание снапшотов за константное время, в этой файловой системе реализовано сжатие данных, что позволит эффективнее использовать дисковое пространство [6].

3.3. Архитектура библиотеки

Основная часть библиотеки состоит из нескольких модулей:

- ZFSVersionManager: Это основной компонент системы, который обеспечивает взаимодействие с файловой системой ZFS и системой версионирования. Он служит в качестве посредника между файловой системой и системой версионирования, обеспечивая координацию и управление всеми операциями,

- ZFSHandler: Этот компонент представляет собой обработчик для файловой системы ZFS, которая используется для хранения и управления файлами и снапшотами. Он обеспечивает функции для сохранения файлов и создания снапшотов, а также для взаимодействия с ними, например, saveFile(), createSnapshot() и restoreFile().

- VersionManager: Этот компонент отвечает за назначение версий и управление их ими посредством снапшотов. В этом модуле присутствуют такие методы, как getVersion () – возвращает определенную версию файла, и createVersion() – этот метод используется для создания новой версии файла. Каждая версия представляет собой уникальное состояние файла и содержит его снимок в определенный момент времени.

- File: Этот компонент представляет собой файл, который сохраняется в файловой системе ZFS. Он содержит информацию о содер-

жимом файла и его свойствах, например, название файла и дату последнего изменения.

- Snapshot: Этот компонент представляет собой снимок файла, созданные в файловой системе ZFS. Он содержит информацию о состоянии файла в определенный момент времени.

- VersionedFile: Этот компонент представляет собой версию файла, который отслеживаются системой версионирования. Он содержит информацию о версии файла и ссылку на соответствующий снапшот.

- ErrorHandler: Этот класс управляет всеми ошибками, которые могут возникнуть, и предоставляет информативные сообщения об ошибках.

- Logger: Этот класс отслеживает и записывать все действия, выполняемые в библиотеке, что может быть полезно для отладки и аудита.

Диаграмма классов описанного решения представлена на Рис.1.

3.4. Реализация

В качестве языка программирования для реализации был выбран язык C++, так как библиотека должна будет в дальнейшем использоваться для написания расширения для PostgreSQL, а расширения для этой СУБД пишутся в основном на C и C++. Для взаимодействия с файловой системой ZFS используется библиотека OpenZFS[3].

Класс ZFSHandler инкапсулирует взаимодействие с файловой системой ZFS. Этот модуль позволяет записать бинарный файл в файловой системе и сделать снапшот. Когда вызывается функция изменения файла, то после применения изменений генерируется новый снапшот. В случае, когда необходимо восстановить какую-либо версию файла, этот

[3] `https://github.com/openzfs/zfs`

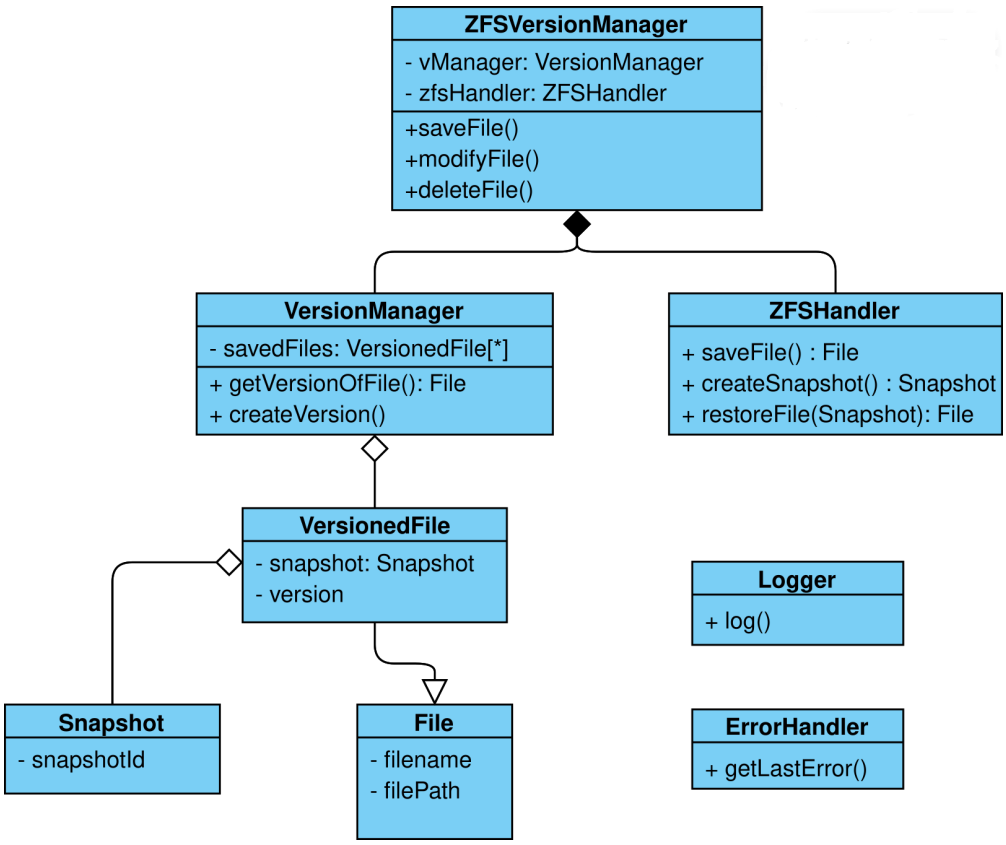

Рис. 1: Диаграмма классов UML

модуль восстанавливает состояние файловой системы по снапшоту и возвращает содержимое файла. Также в модуле реализованы функции по удалению снапшотов и удалению файлов.

4. План по тестированию

4.1. Тестирование функциональности

Предполагается следующий план по тестированию функциональности:

- Модульное тестирование: будут протестированы все основные функции, такие как сохранение файлов, присвоение версий, изменение файлов и возврат определенных версий файлов. Так же будет проверено, что тесты проверяют обработку ошибок. Для тестирования будет использоваться библиотека для юнит-тестирования на C++ Google Test.

- Интеграционное тестирование: будут проверены, как функции взаимодействуют друг с другом и с системой ZFS (сохраняются ли файлы корректно, корректно ли возвращаются различные версии файлов).

- Тестирование по интеграции с расширением для PostgreSQL: когда расширение будет написано, то будет проведено тестирование по выявлению полноты предоставляемого API для реализации логики работы расширения.

4.2. Тестирование производительности

Будет проведено тестирование производительности расширения, а именно тест скорости работы с файлами в СУБД: будут проведены эксперименты по сохранению, изменению и извлечению файлов из базы данных, используя встроенные механизмы PostgreSQL (bytea, BLOB), а также с использованием разработанного расширения, и будут замерены такие метрики, как время выполнения запроса и количество транзакций в секунду (TPS).

Заключение

В результате работы над производственной практикой были решены следующие задачи:

- Изучены существующие подходы к решению задачи версионирования бинарных данных в PostgreSQL.
- Определён подход к версионированию бинарных данных.
- Начата реализация библиотеки для написания расширения PostgreSQL для версионирования бинарных данных:
  - Разработана архитектура библиотеки;
  - Разработан модуль взаимодействия с файловой системой ZFS.

Также был продуман план по будущему тестированию как библиотеки, так и будущей системы, состоящей из библиотеки и расширения.

Планы для дальнейшей работы:

- Завершить реализацию библиотеки для написания расширения PostgreSQL для версионирования бинарных данных.
- Провести тестирование полученного решения.

Код проекта закрыт и принадлежит компании ООО “Датаджайл”.

Список литературы

[1] Albe Laurenz. Binary data performance in PostgreSQL // Cybertec, professional partner for PostgreSQL services, support and training. — 2020. — Access mode: https://www.cybertec-postgresql.com/en/binary-data-performance-in-postgresql/ (online; accessed: 24.05.2023).

[2] BinaryFilesInDB // PostgreSQL Wiki. — 2021. — Access mode: https://wiki.postgresql.org/wiki/BinaryFilesInDB (online; accessed: 25.11.2022).

[3] Fittl Lukas. 5mins of Postgres E3: Postgres performance cliffs with large JSONB values and TOAST // pgAnalyze Blog. — 2022. — Access mode: https://pganalyze.com/blog/5mins-postgres-jsonb-toast (online; accessed: 02.01.2023).

[4] TOAST // PostgreSQL Documentation. — 2015. — Access mode: https://www.postgresql.org/docs/current/storage-toast.html (online; accessed: 25.11.2022).

[5] Vondra Tomas. Postgres vs. File Systems: A Performance Comparison // EDB Blog. — 2022. — Access mode: https://www.enterprisedb.com/blog/postgres-vs-file-systems-performance-comparison (online; accessed: 02.01.2023).

[6] Меликов Георгий. ZFS: архитектура, особенности и отличия от других файловых систем // «Завтра облачно», журнал о цифровой трансформации от VK Cloud Solutions. — 2020. — Access mode: https://mcs.mail.ru/blog/zfs-arhitektura-osobennosti-i-otlichija (online; accessed: 02.01.2023).