Homework #4. Solr

Table of contents

История развития Solr

В 2004 году Solr был создан Йоником Сили в CNET Networks в качестве внутреннего проекта для добавления возможности поиска на сайте компании.

В январе 2006 года CNET Networks решила открыто опубликовать исходный код, передав его в фонд Apache Software Foundation. Как и любой новый проект Apache, он вступил в инкубационный период, который помог решить организационные, юридические и финансовые вопросы.

В январе 2007 года Solr вышел из статуса инкубатора в самостоятельный проект верхнего уровня (TLP) и постоянно развивался, наращивая свои возможности, привлекая тем самым пользователей, участников и коммиттеров. Несмотря на то, что Solr был довольно новым публичным проектом, он обеспечил работу нескольких веб-сайтов с высоким трафиком.

В сентябре 2008 года была выпущена версия Solr 1.3, включающая возможности распределенного поиска и улучшения производительности.

В январе 2009 года Йоник Сили вместе с Грантом Ингерсоллом и Эриком Хэтчером присоединился к Lucidworks (ранее Lucid Imagination), первой компании, предоставляющей коммерческую поддержку и обучение для поисковых технологий Apache Solr. С тех пор предложений по поддержке Solr стало много.

В ноябре 2009 года была выпущена версия Solr 1.4. В этой версии были представлены усовершенствования в индексировании, поиске и фасетировании, а также многие другие улучшения, такие как обработка документов (PDF, Word, HTML), кластеризация результатов поиска на основе Carrot2 и улучшенная интеграция с базами данных. Релиз также включает множество дополнительных плагинов.

В марте 2010 года произошло слияние проектов Lucene и Solr. Раздельные загрузки продолжались, но продукты теперь разрабатывались совместно одним набором коммиттеров.

В 2011 году схема номеров версий Solr была изменена, чтобы соответствовать схеме Lucene. После Solr 1.4 следующий релиз Solr был обозначен как 3.1, чтобы сохранить одинаковый номер версии Solr и Lucene.

В октябре 2012 года была выпущена версия Solr 4.0, включающая новую функцию SolrCloud.В 2013 и 2014 годах был выпущен ряд релизов Solr в линейке 4.x, в которых постоянно расширялся набор функций и повышалась надежность.

В феврале 2015 года был выпущен Solr 5.0, первый релиз, в котором Solr упакован как отдельное приложение, прекратив официальную поддержку развертывания Solr как войны. В Solr 5.3 была встроена подключаемая система аутентификации и авторизации.

В апреле 2016 года был выпущен Solr 6.0. Добавлена поддержка выполнения параллельных SQL-запросов через коллекции SolrCloud. Включена поддержка StreamExpression и новый драйвер JDBC для SQL-интерфейса.

В сентябре 2017 года был выпущен Solr 7.0. В этом выпуске, помимо прочего, добавлена поддержка нескольких типов реплик, автомасштабирование и математический движок.

В марте 2019 года был выпущен Solr 8.0, включающий множество исправлений и обновлений компонентов. Узлы Solr теперь могут прослушивать и обслуживать запросы HTTP/2. Имейте в виду, что по умолчанию внутренние запросы также отправляются с использованием HTTP/2. Кроме того, был добавлен вход в администраторский UI с поддержкой BasicAuth и Kerberos. А построение математических выражений в Apache Zeppelin теперь возможно.

В ноябре 2020 года Bloomberg передал Solr Operator проекту Lucene/Solr. Solr Operator помогает развертывать и запускать Solr в Kubernetes.

В феврале 2021 года Solr был создан как отдельный проект Apache (TLP), независимый от Lucene.

В мае 2022 года был выпущен Solr 9.0, как первый релиз, независимый от Lucene, требующий Java 11 и включающий такие основные моменты, как "нейронный" поиск KNN, лучшая модульность, больше плагинов безопасности и многое другое.

Инструменты для взаимодействия с Solr

Есть несколько инструментов взаимодействия с Solr:

1. Solr CLI. Терминал в Linux/Командная строка в Windows.
2. Solr Admin UI. Solr имеет веб-интерфейс, который обеспечивает онлайн доступ ко многим параметрам и функциям конфигурации Solr.
3. SolrJ. SolrJ, API для работы с Java-приложениями.
4. JavaScript. Клиенты JavaScript.
5. Python. Python и JSON responses.
6. Ruby. Solr с приложениями на Ruby.

Database engine в Solr

Solr работает как автономный сервер полнотекстового поиска. В его основе лежит поисковая библиотека Lucene Java для полнотекстового индексирования и поиска, а также REST-подобные API HTTP/XML и JSON, которые позволяют использовать его из большинства популярных языков программирования. То есть, Solr является search engine СУБД.

Язык запросов в Solr

TODO!!

Распределение файлов БД по разным носителям

Solr поддерживает распределение файлов бд по разным носителям, так как построен на проверенном в боях Apache Zookeeper.

На каких языках программирования написана Solr

Solr написана на языке Java. Она использует в своей основе поисковую библиотеку Lucene, которая тоже написана на Java.

Идексы в Solr

TODO!!!

Процесс выполнения запросов в Solr?

Когда пользователь запускает поиск в Solr, поисковый запрос обрабатывается обработчиком запроса. Обработчик запроса - это подключаемый модуль Solr, который определяет логику, используемую Solr при обработке запроса. Solr поддерживает множество обработчиков запросов. Некоторые из них предназначены для обработки поисковых запросов, другие управляют такими задачами, как репликация индекса.

Для обработки поискового запроса обработчик запроса вызывает парсер запроса, который интерпретирует условия и параметры запроса. Различные парсеры запросов поддерживают разный синтаксис. Парсер запросов по умолчанию в Solr известен как Standard Query Parser, или более распространенный парсер запросов "lucene". Solr также включает парсер запросов DisMax и расширенный парсер запросов DisMax (eDisMax).

Кроме того, существуют общие параметры запросов, принимаемые всеми анализаторами запросов. (Изображенны на картинке)

Входные данные для анализатора запросов могут включать:

- поисковые строки - то есть термины для поиска в индексе
- параметры для точной настройки запроса путем увеличения важности определенных строк или полей, применения булевой логики среди поисковых терминов или исключения содержимого из результатов поиска
- параметры для управления представлением ответа на запрос, например, указание порядка представления результатов или ограничение ответа определенными полями схемы поискового приложения.

Чтобы помочь пользователям найти нужный контент, Solr поддерживает два специальных способа группировки результатов поиска для облегчения дальнейшего поиска: фасетирование и кластеризация.

Фасет - это распределение результатов поиска по категориям (которые основаны на индексированных терминах). Внутри каждой категории Solr сообщает о количестве совпадений с соответствующим термином, который называется ограничением фасета. Фасетное ограничение облегчает пользователям изучение результатов поиска на таких сайтах, как сайты фильмов и обзоры товаров, где существует множество категорий и множество элементов внутри категории.

Кластеризация группирует результаты поиска по сходству, обнаруженному при выполнении поиска, а не при индексировании содержимого. Результаты кластеризации часто не имеют четкой иерархической организации, которая присутствует в результатах фасетного поиска, но, тем не менее, кластеризация может быть полезной. Она может выявить неожиданные общие черты в результатах поиска и помочь пользователям отсеять контент, не относящийся к тому, что они действительно ищут.

Компонент Solr, называемый response writer, управляет окончательным представлением ответа на запрос. Solr включает в себя различные составители ответов, в том числе XML Response Writer и JSON Response Writer.

На приведенной ниже схеме кратко представлены некоторые ключевые элементы процесса поиска.

План запросов в Solr

Да, в Solr есть план выполнения под названием "explain", который предоставляет подробное описание того, как Solr пришел к результатам поиска по определенному запросу. Функциональность "explain" предназначена для того, чтобы помочь разработчикам понять, почему определенные документы были возвращены для конкретного запроса, и как был применен алгоритм оценки Solr.

Чтобы получить информацию о плане запроса нужно в запросе задать параметр "debug = results".

По умолчанию score explanations возвращаются в виде больших строковых значений, с использованием новых строк и отступов табуляции для структурирования и удобочитаемости, но можно указать дополнительный параметр debug.explain.structured=true, чтобы вернуть эту информацию в виде вложенных структур данных, соответствующих формату ответа, запрашиваемому wt. (Параметр wt выбирает программу написания ответов, которую Solr должен использовать для форматирования ответа запроса. Если вы не определите параметр wt в своих запросах, в качестве формата ответа будет возвращен JSON.)

Транзкации в Solr

Solr не является транзакицонной базой данных, поэтому она не поддерживает транзакции в полном смысле этого слова. Commit делает все отложенные изменения всех клиентов видимыми для новых запросов. Аналогично, rollback отменяет все ожидающие изменения всех клиентов. При этом не учитывается, какой клиент отправил команду commit/rollback.

По этой причине обработка ошибок не должна автоматически приводить к rollback. Потому что последствия могут быть гораздо шире, чем просто ошибочные данные. И в результате очистка может быть намного сложнее.

Руководство из документации Solr гласит, что следует использовать autocommit. Это особенно верно при выполнении массовых операций. Если вы выполняете массовое индексирование, возможно, с несколькими параллельными клиентами, то лучше выполнять autocommit каждые несколько раз (или через каждые несколько документов). Это приведет к созданию меньшего количества новых сегментов индекса, и в результате индекс будет менее фрагментированным.

Backup и Restore в Solr

Solr предоставляет два подхода к резервному копированию и восстановлению ядер или коллекций Solr, в зависимости от того, как вы используете Solr. Если вы используете кластер SolrCloud, вы будете использовать API Collections. Если вы используете user-managed cluster или a single-node installation, вы будете использовать обработчик репликации.

Поддержка резервного копирования в SolrCloud обеспечивается с помощью Collections API. Это позволяет создавать резервные копии на нескольких шардах и восстанавливать их на том же количестве шардов и реплик, что и исходную коллекцию. (Для SolrCloud Backup/Restore требуется общая файловая система, смонтированная по одному и тому же пути на всех узлах, или HDFS.)

В User-Managed Clusters и Single-Node Installations для резервного копирования и восстановления используется обработчик репликации Solr. Из коробки Solr включает неявную поддержку репликации, поэтому можно использовать этот API. Однако конфигурацию обработчика репликации можно настроить, определив свой собственный обработчик репликации в solrconfig.xml.

Solr предоставляет абстракцию репозитория, позволяющую пользователям создавать резервные копии и восстанавливать данные в различных системах хранения. Например, кластер Solr, работающий на локальной файловой системе (например, EXT3), может хранить резервные копии данных на том же диске, на удаленном сетевом диске, в HDFS или даже в некоторых популярных "облачных хранилищах", в зависимости от выбранной реализации "репозитория". Solr предлагает несколько различных реализаций репозиториев из коробки (LocalFileSystemRepository, HdfsBackupRepository, GCSBackupRepository и S3BackupRepository) и позволяет пользователям создавать плагины для своих собственных систем хранения данных по мере необходимости. Пользователи могут определить любое количество хранилищ в своем файле solr.xml.

Sharding в Solr

TODO!!!

Применимость терминов Data Mining, Data Warehousing и OLAP к Solr

TODO!!!

Методы защиты в Solr

TODO!!!

Какие сообщества развивают данную СУБД? Кто в проекте имеет права на коммит и создание дистрибутива версий?

TODO!!!

Демонстрация работы Solr на собственных данных

TODO!!!

Самостоятельное изучение языка запросов с помощью демобазы.

TODO!!!

Документация и обучение

TODO!!!

Как быть в курсе происходящего

TODO!!!