Реализация эффективного выполнения поисковых запросов по множеству полей для колоночно-ориентированной базы данных, хранящейся в памяти.

Жаворонков Эдгар Андреевич
группа 4501
руководитель Дельгядо Ф.И

Университет ИТМО

22 июня 2015

Актуальность разработки

1. Создание Open Source in-memory NoSQL базы данных, позволяющей решать задачу быстрого многокритериального поиска
2. Существующие решения либо очень дороги, либо ориентированы больше в сторону аналитических приложений

Актуальность разработки

1. Создание Open Source in-memory NoSQL базы данных, позволяющей решать задачу быстрого многокритериального поиска
2. Существующие решения либо очень дороги, либо ориентированы больше в сторону аналитических приложений
3. Либо медленные, так как работают с диском

Анализ предметной области

Колоночно-ориентированные базы данных:

1. Появились ещё в 70-80ых гг. прошлого века;
2. Бурно развились в начале 2000ых годов;
3. Хранят данные не построчно друг за другом, а наоборот - колонка за колонкой;
4. Это даёт возможность сжатия данных, но накладывает ограничения на решаемые задачи;

Обзор аналогов

Product	in-memory	NoSQL	OLAP	Price
Oracle TimesTen	✓	✓	✓	€19,969.00
VoltDB	✓	✓	✓	$3500/month
SAP HANA	✓	✓	✓	$3595/month

1. Все три решения - очень мощные аналитические инструменты, использующие поколоночное хранение данных в оперативной памяти.
2. Как видно, их основной недостаток - цена. Вендоры, кроме всего прочего, предоставляют сервера, на которых разворачиваются эти решения.

Цель и задачи ВКР

Цель работы - разработать компонент, обеспечивающий быстрое выполнение поисковых запросов по нескольким полям для in-memory базы данных. Кроме того, требуется обеспечить возможность атомарного добавления записей в такую базу.
Задачи:

1. Разработать и описать алгоритмы выполнения поисковых запросов;
2. Реализовать алгоритмы;
3. Ввести формальные показатели для сравнения и сравнить алгоритмы между собой;
4. Разработать, описать, реализовать и сравнить между собой алгоритмы атомарного добавления записей;

Требования к решению

Функциональные требования:

1. Планирование порядка выполнения запроса;
2. Учет сложности запросов к сложным хранилищам (полнотекстовый поиск и т.п.);
3. Возможность автоматической подстройки модуля;
4. Реализация корректного выполнения запросов на массированных изменениях данных;

Формальные показатели:

1. Уменьшение времени выполнения отдельных запросов в два и более раз

Нефункциональные требования:

1. Низкие накладные расходы на обеспечение эффективности выполнения запросов;
2. Тестирование правильности выполнения запросов

Архитектура системы. Общий обзор

1. Основная сущность - документ с уникальным номером. Документ - набор полей(атрибутов) и их значений;
2. Значения атрибутов лежат в индексах - хранилищах. Хранилище - набор сжатых битовых массивов;
3. Каждое хранилище предназначено для выполнения запросов определенного типа;
4. Отдельный модуль, исполняющий запросы;

Архитектура системы. Классы и сущности

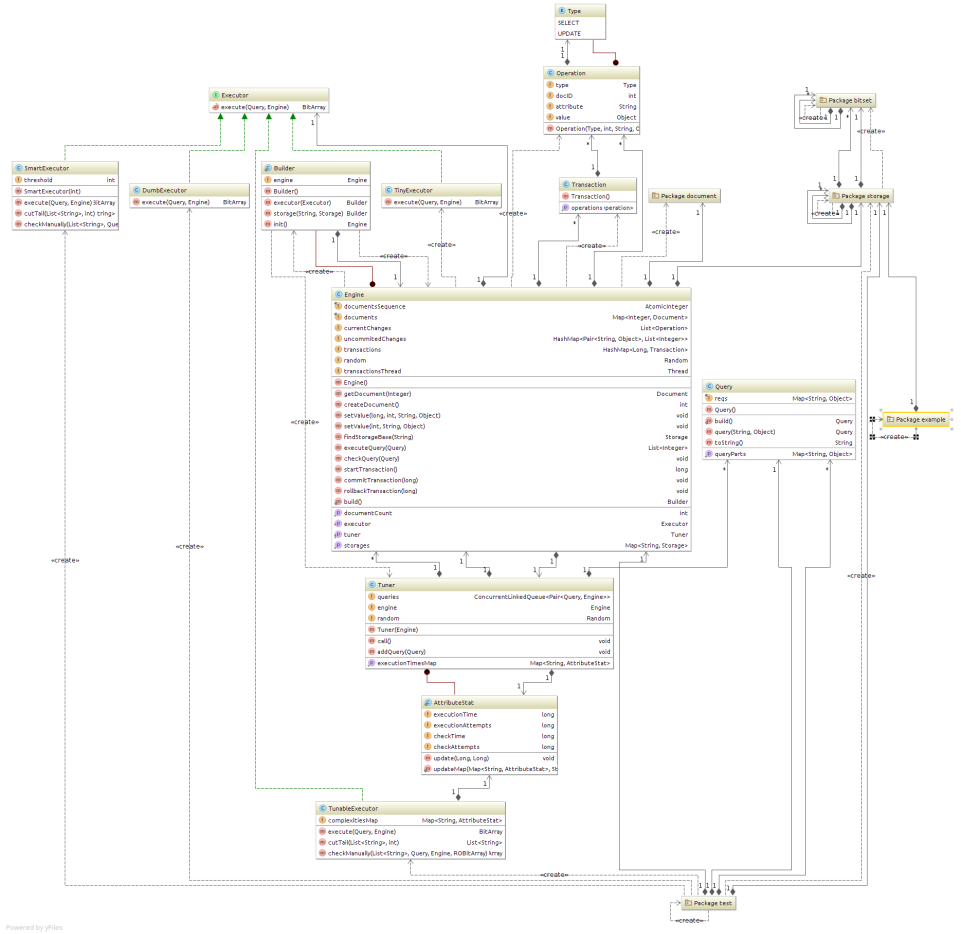

Рис.: Диаграмма классов системы Vindur.

Специфика решения. Процесс выполнения запроса

Формально, запрос - это множество пар (a, v), где a - атрибут по которому осуществляется поиск, а v - искомое значение. Нестрого говоря - несколько этапов:

1. Проверка на корректность;
2. Получение битовых массивов из хранилищ;
3. Пересечение между собой;
4. Преобразование в список с номерами документов;

Специфика решения. Стратегии выполнения запросов

На текущий момент реализованы три стратегии:

1. Тривиальная (DumbExecutor);
2. На основе весов хранилищ (SmartExecutor);
3. На основе статистики по атрибутам (TunableExecutor);

Специфика решения. Тривиальная стратегия

1. Для каждого атрибута, получаем ассоциированное с ним хранилище;
2. Извлекаем оттуда битовый массив документов, имеющих соответствующее значение;
3. Все битовые массивы ”сворачиваются” операцией логического умножения(AND);
4. Это соответствует пересечению множеств между собой;

Специфика решения. Стратегия с весами

Вес хранилища - эмпирическая мера относительной трудоёмкости извлечения информации из хранилища.

1. Множество пар (a, v) сортируется по возрастанию веса хранилищ, ассоциированных с атрибутами;
2. Снова получаем хранилища извлекаем битовые массивы с документами и пересекаем их;
3. Если на очередном шаге получили пустой битовый массив - прекратили выполнение;
4. Если в процессе выполнения результирующая выборка стала содержать мало документов, то оставшиеся части запроса проверяются напрямую, минуя хранилища;

Специфика решения. Стратегия со статистикой

Вместо весов теперь - среднее время обращения к хранилищу и среднее время проверки в обход него.

1. Преподсчитывается примерное время выполнения запроса и время проверки одного документа;
2. Множество пар (a, v) сортируется по возрастанию среднего времени обращения к хранилищам, ассоциированных с атрибутами;
3. Если в процессе выполнения время проверки оставшихся частей в запросе для результирующей выборки стало меньше, чем время обращения к оставшимся хранилищам, то оставшиеся части запроса проверяются напрямую;

Специфика решения. Атомарное добавление 1

1. ”Добавляем - не ищем. Ищем - не добавляем”;
2. В ходе сеанса все изменения записываются в журнал;
3. По подтверждению - БД блокируется и все изменения фиксируются;
4. Блокировка означает, что ни один параллельный поток не имеет в это время доступа к базе;

Специфика решения. Атомарное добавление 2

1. Изменения снова записываются в журнал;
2. Однако по подтверждению - не применяются, а начинают учитываться при поиске документов(записываются в ещё один журнал);
3. Применение происходит небольшими порциями, параллельно выполнению запроса;

Тестирование

Задачи тестирования:

1. Определить среднее время выполнения запроса для разных алгоритмов;
2. Выявить достоинства и недостатки алгоритмов атомарного добавления записей;

Результаты тестирования алгоритмов выполнения запросов

Один миллион документов, три атрибута, пять тысяч запросов

Таблица: Результаты тестирования алгоритмов выполнения поисковых запросов

Алгоритм	Среднее время выполнения запроса	Дисперсия	Стандартное отклонение
DumbExecutor	179.71ms	506.8ms^2	22.51ms
SmartExecutor	10.31ms	9.04ms^2	3.01ms
TunableExecutor	10.08ms	7.91ms^2	2.81ms

Результаты тестирования атомарного добавления 1

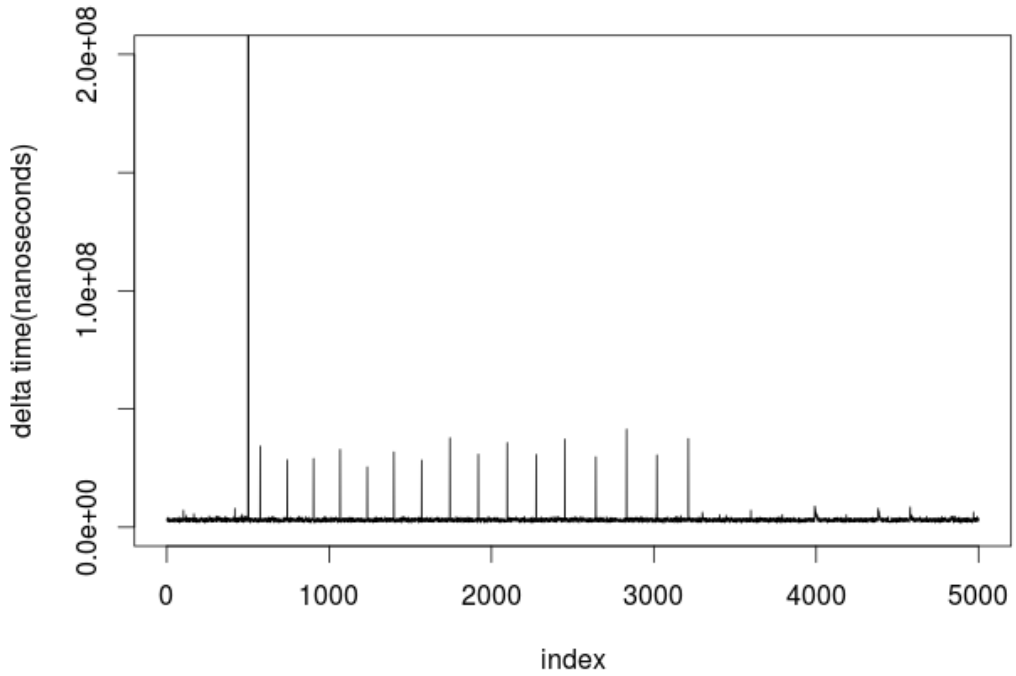

Рис.: График зависимости времени выполнения операций в ходе сеанса массовой загрузки(первый вариант)

Результаты тестирования атомарного добавления 2

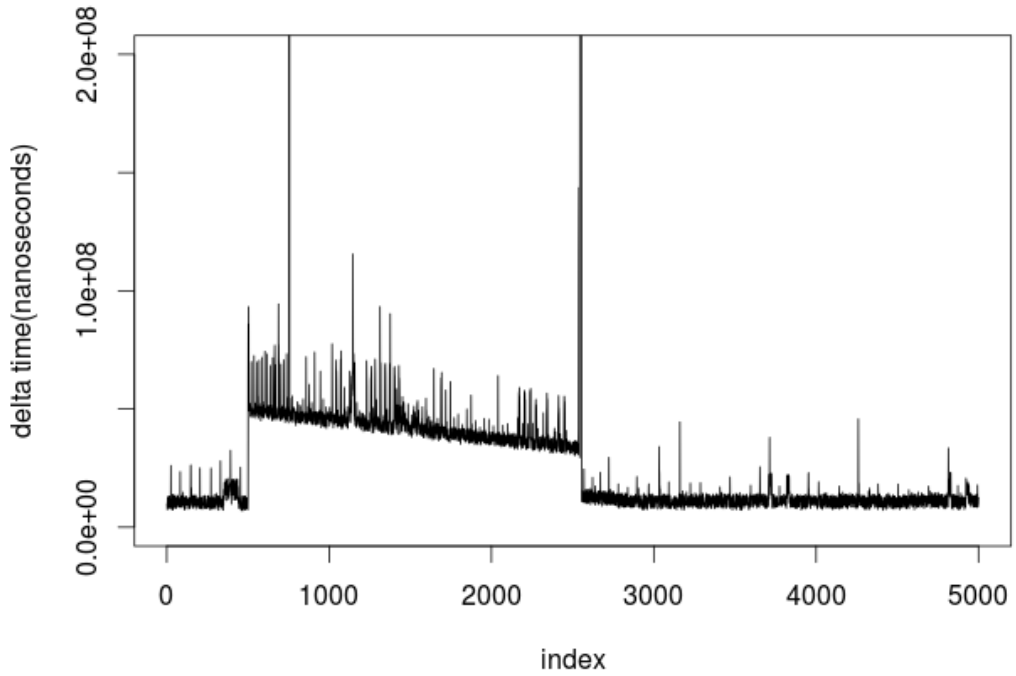

Рис.: График зависимости времени выполнения операций в ходе сеанса массовой загрузки(второй вариант)

Спасибо за внимание!
Вопросы?

Github repo:

https://github.com/cscenter/Vindur