МЕТОДЫ ОПТИМИЗАЦИИ РАБОТЫ С БОЛЬШИМИ ОБЪЕМАМИ ДАННЫХ В ОБРАЗОВАТЕЛЬНОЙ АНАЛИТИКЕ

BIG DATA OPTIMIZATION METHODS IN THE FIELD OF EDUCATION ANALYTICS

Харченко Александр Сергеевич, *аспирант кафедры информатики и вычислительной техники РГУ имени С.А. Есенина (г. Рязань).*

a.harchenko@rsu.edu.ru

Страшнов Юрий Владиславович, *кандидат физико-математических наук, инженер отдела телекоммуникаций РГУ имени С.А. Есенина (г. Рязань).*

j.strashnov@rsu.edu.ru

В статье рассмотрены методы и примеры оптимизации реляционных СУБД для повышения производительности при работе с большими объемами данных образовательной аналитики.

The article presents education analytics Big Data optimization methods and practical examples.

Ключевые слова: **большие данные, SQL, базы данных, разбиения, ИТ, образовательная аналитика, индексы, PostgreSQL.**

Keywords: **big data, SQL, databases, partitioning, IT, education analytics, indexes, PostgreSQL.**

Приход в образовательную сферу Больших Данных означает существенное усиление роли информационных технологий и делает их двигателем развития образовательных институтов и структур. Развитие концепции образовательной аналитики предполагает создание систем, функциональность которых охватывает задачи извлечения данных из разнородных источников, моделирование образовательных процессов, накопление баз педагогических практик.

Данные, накопленные в образовательной системе в течение многих лет, носят преимущественно структурированный характер и представлены, как правило, в форме отчетов и статистики. В большей степени они относятся к так называемой академической аналитике, ориентированной на интегральную оценку образовательных учреждений [1].

Не существует универсальных способов анализа или алгоритмов, пригодных для любых случаев и любых объемов информации. Методы анализа данных существенно отличаются друг от друга по производительности, качеству результатов, удобству применения и требованиям к данным.

Оптимизация может производиться на различных уровнях:

- оборудование – наращивание вычислительных мощностей;
- базы данных – оптимизация хранимых данных (индексы, разбиения);
- аналитическая платформа – оптимизация или разработка специализированных алгоритмов выборки данных;

– исходные данные – предварительная обработка данных перед помещением в хранилище [2].

Поскольку по результатам исследований корпорации IBM в настоящее время рост объема данных в мире опережает рост производительности вычислительных систем, представляется разумным искать пути увеличения скорости обработки данных в области программного обеспечения.

В данной статье мы рассмотрим возможности оптимизации данных, расположенных на сервере баз данных PostgreSQL. Выбор PostgreSQL 9.3 в качестве СУБД выходит за рамки данной статьи.

Современные системы управления базами данных (СУБД) содержат различные механизмы, применение которых, при определенных условиях, позволяет значительно увеличить скорость аналитической обработки. Одним из них является разбиение таблиц на разделы и табличные пространства: можно размещать на отдельных дисках данные, индексы и вспомогательные таблицы. Это позволяет СУБД параллельно считывать и записывать информацию на диски [3]. Кроме того, таблицы могут быть разбиты на разделы (разбиения) и в ситуациях, когда наиболее интенсивно используемая область таблицы является отдельным разбиением или небольшим количеством разбиений, производительность запроса может быть очень существенно увеличена. Такое разбиение и его индексы лучше размещаются в памяти, чем индекс всей таблицы [4]. Редко используемые данные можно перенести на более дешёвые и медленные носители.

Например, если нам необходимо произвести анализ данных помесячно, то можно логически использовать одну таблицу с историческими данными, но физически разбить ее на несколько разделов, чтобы при обращении к данным считывался единственный раздел, и не было обращений ко всем историческим данным.

Массовые загрузки и удаления могут быть реализованы с помощью простого удаления или добавления разбиений, если такое заложено при планировании структуры разбиений. Команды ALTER TABLE NO INHERIT и DROP TABLE значительно быстрее, чем любые операции массовой загрузки или удаления. Эти команды также полностью снимают необходимость в выполнении VACUUMM, в отличие от массового удаления командой DELETE [5].

Эти выгоды обычно проявляют себя, когда таблица будет очень большой. Выиграет таблица от применения разбиения или нет, в конечном счете, зависит от приложения, но есть эмпирическое правило, что это произойдёт, если размер таблицы превышает физическую память сервера СУБД.

В СУБД PostgreSQL могут быть реализованы следующие виды разбиений:

Диапазонное разбиение

Таблица разбивается по "диапазонам", заданным ключевой колонкой или списком колонок без перекрывания диапазонов значений, предназначенных для разных разбиений. Например, это может быть диапазоны дат или диапазоны идентификаторов отдельных бизнес-объектов.

Списочное разбиение

Таблица разбивается по явным спискам, ключевые значения которых, имеются в каждом разбиении.

Для демонстрации применения разбиений был развернут испытательный стенд на базе CentOS 6.5 и PostgreSQL 9.3 из официального репозитория PostgreSQL с таблицей, содержащей 50 млн. записей в 70 разбиениях.

Диапазонное разбиение таблицы по полю «Дата» при помощи pg_partman и перенос существующих данных в разбиения без остановки работы.

Рассмотрим диапазонное разбиение существующей таблицы по полю «Дата» при помощи расширения pg_partman, которое призвано упростить работу с партиционированием таблиц в PostgreSQL. Подробное описание производимых действий содержится в документации на pg_partman [6, 7].

Установим необходимое ПО:

```
yum -y install pg_jobmon93 pg_partman93 \
python-argparse python-psycopg2
```

Для ускорения работы с таблицами, содержащими разбиения, в конфигурационном файле postgresql.conf должен быть включен параметр планирования запросов constraint_exclusion [8].

В интерфейсе psql (таблица hugetable в БД exampledb):

Создадим схему (SCHEMA) “partman” для pg_partman:

```
CREATE SCHEMA partman;
CREATE EXTENSION pg_partman SCHEMA partman;
```

Создадим отдельную схему (SCHEMA) “st” для большой таблицы. Работа pg_partman возможна только для таких таблиц:

```
CREATE SCHEMA st;
ALTER TABLE hugetable SET SCHEMA st;
```

После завершения данных операций к hugetable можно будет обратиться только по st.hugetable.

Создадим необходимые разбиения при помощи вызова специальной функции на языке PL/SQL из состава pg_partman. В данном примере в таблице hugestat есть поле «timestamp», содержащее информацию о дате, которым мы и воспользуемся:

```
SELECT partman.create_parent('st.hugetable',
’timestamp', 'time-static', 'daily');
```

Функция создаст необходимые дочерние таблицы и индексы, идентичные родительской таблице. Теперь необходимо перенести туда все существующие данные. Для этого в pg_partman есть специальный скрипт. Он производит миграцию данных в прозрачном режиме без блокировки таблиц, позволяя параллельно работать с ними.

Запускать его необходимо от имени пользователя postgres:

```
su - postgres
/usr/pgsql-9.3/bin/partition_data.py \
-c "host=localhost dbname=exampledb" \
-p st.hugetable -t time -w 1 &
```

Поместим вызов функции partman.run_maintenance() в CRON для того, чтобы она автоматически создавала необходимые новые дочерние таблицы и, если это необходимо, удаляла старые:

```
vi /etc/cron.d/pg_partman_maint

05 02 * * * postgres /usr/pgsql-9.3/bin/psql ex-
ampledb -c "select partman.run_maintenance()"
```

Возможности PostgreSQL для оптимизации запросов: constraint_exclusion, отказ от to_timestamp, индексы на несколько полей и функции.

Для получения прироста производительности недостаточно лишь произвести разбиение таблицы. Необходимо произвести аудит SQL-запросов аналитического приложения.

Проверить, как исполняются SQL-запросы и какие разбиения при этом используются можно при помощи команд:

- EXPLAIN <текст запроса> - план запроса
- EXPLAIN ANALYZE <текст запроса> - план запроса + время выполнения

Следует обратить внимание, что constraint exclusion по дате не сработает, если в SELECT-запросах есть указания даты в виде TIMESTAMP() – запрос пройдет все разбиения. Необходимо указывать дату в явном виде [9].

Другими функциями в запросах пользоваться можно имея ввиду, что SELECT-запрос, включающий в себя функцию преобразования значения поля не будет использовать индексы. Данное ограничение можно обойти, создав специальный индекс на основе результата выполнения данной функции [10].

Например, для функции LOWER:

```
CREATE INDEX idx_<indexname>_lcase
ON "<tablename>" (LOWER(<columnname>));
```

для UPPER() ситуация аналогична.

Так же рекомендуется использовать общие индексы по группам полей в случаях, когда в SELECT-запросах происходит выборка по нескольким полям или функциям от них. Например, для запроса, осуществляющего выборку по LOWER(hostname), timestamp. разумно создать отдельный индекс:

```
CREATE INDEX idx_<indexname>_comb_lcase
ON "<tablename>" (LOWER(hostname), timestamp);
```

Если новые индексы создаются в таблице, содержащей разбиения, необходимо перестроить индексы дочерних таблиц. Для этого в pg_partman необходимо выполнить специальный скрипт:

```
su - postgres
/usr/pgsql-9.3/bin/reapply_indexes.py \
-c "host=localhost dbname=exampledb" \
-p st.hugetable --concurrent --dryrun
```

Миграция с PgSQL 8.4 на 9.3

В PostgreSQL более старых версии отсутствуют удобные механизмы работы с разбиениями, в связи с этим их необходимо разрабатывать самостоятельно [4, 6].

Рассмотрим процесс миграции PostgreSQL с популярной устаревшей версии 8.4 на текущий стабильный релиз 9.3.

PostgreSQL устроен таким образом, что позволяет одновременно установить на одном сервере несколько разных версий. Вкупе с развитыми утилитами по миграции можно произвести все операции на одном сервере с минимальным временем простоя [11].

Установим PostgreSQL 9.3 при помощи yum:

```
yum install
http://yum.postgresql.org/9.3/redhat/rhel-6-
x86_64/pgdg-redhat93-9.3-1.noarch.rpm
yum install postgresql93-server postgresql93-
contrib pgtune
```

Инициализируем новую БД:

```
service postgresql-9.3 initdb
```

При необходимости можно запустить утилиту pg_tune для базовой оптимизации параметров PostgreSQL на основе аппаратных характеристик сервера:

```
mv /var/lib/pgsql/9.3/data/postgresql.conf \
/var/lib/pgsql/9.3/data/postgresql.conf.orig

pgtune -i
/var/lib/pgsql/9.3/data/postgresql.conf.orig \
```

```
-o /var/lib/pgsql/9.3/data/postgresql.conf

chown postgres:postgres
/var/lib/pgsql/9.3/data/postgresql.conf
```

Затем вручную редактируем все необходимые настройки (autovacuum, constraint_exclusion и т.д.) и настраиваем авторизацию:

```
cp /var/lib/pgsql/data/pg_hba.conf
/var/lib/pgsql/9.3/data/pg_hba.conf
```

Останавливаем pgsql 8.4:

```
service postgresql stop && chkconfig postgresql
off
```

Миграция данных осуществляется при помощи специальной утилиты pg_upgrade. Мигрировать данные можно двумя способами:

– Копией (предпочтительно, медленно, в случае возникновения проблем доступ к старой БД сохранится):

```
su - postgres
/usr/pgsql-9.3/bin/pg_upgrade --jobs \
--old-datadir "/var/lib/pgsql/data" \
--new-datadir "/var/lib/pgsql/9.3/data" \
--old-bindir "/usr/bin" \
--new-bindir "/usr/pgsql-9.3/bin"
```

– Ссылками (быстро и опасно - старая БД не запустится):

```
su - postgres
/usr/pgsql-9.3/bin/pg_upgrade --jobs –link \
--old-datadir "/var/lib/pgsql/data" \
--new-datadir "/var/lib/pgsql/9.3/data" \
--old-bindir "/usr/bin" \
--new-bindir "/usr/pgsql-9.3/bin"
```

Производим запуск pgsql 9.3:

```
chkconfig postgresql-9.3 on && service post-
gresql-9.3 start
```

От имени postgres выполняем необходимые послемиграционные скрипты:

```
su - postgres
/usr/pgsql-9.3/bin/psql --username postgres --
file pg_largeobject.sql
/analyze_new_cluster.sh
```

и, при необходимости, удаляем старые данные:

```
./delete_old_cluster.sh
```

Заключение

Мы показали, каким образом можно оптимизировать работу с большими объемами данных на сервере SQL и рассмотрели на конкретных примерах соответствующий инструментарий, имеющийся в арсенале СУБД PostgreSQL.

Таким образом, можно прийти к выводу, что оптимальным методом работы с SQL-таблицами большого объема является использование разбиений (партиционирования) и специализированных индексов, что позволяет:

- увеличить скорость доступа к актуальным данным;
- исключить фрагментирование таблиц и индексов при массовых операциях удаления устаревших данных;
- максимально эффективно использовать имеющиеся в наличии аппаратные ресурсы.

📖 БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Мальцева С. В. Большие Данные в образовании: новые возможности и новые вызовы [электронный ресурс] // Конференция Большие Данные в национальной экономике Тезисы докладов. 2013. URL: http://www.ospcon.ru/files/uploads/file/BigData_thesis.pdf (дата обращения 10.11.2014).
2. Анализ больших объемов данных// BaseGroup.ru:: Анализ больших объемов данных. 2014. URL: http://www.basegroup.ru/library/methodology/very_large_data/ (дата обращения 06.11.2014).
3. Travis Parker, Paul RitcheyHigh Capacity Single Table Performance Design Using Partitioning in Oracle or PostgreSQL [электронный ресурс] // ICF INTERNATIONAL INC FAIRFAX VA. 2012. URL: http://oai.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=ADA559376 (дата обращения 30.10.2014).
4. Partition Table In PostgreSQL (Create Partition) // Partition Table In PostgreSQL (Create Partition) – Part 1 2008. URL: http://www.mkyong.com/database/partition-table-in-postgresql-create-partition-part-1/.
5. Разбиение // Разбиение | PostgreSQL. 2014. URL: http://postgresql.ru.net/manual/ddl-partitioning.html (дата обращения 11.11.2014).
6. Partition management extention for PostgreSQL // keithf4/pg_partman GitHub 2014. URL: https://github.com/keithf4/pg_partman.
7. Pg partman // Pg partman – Imre kasutab arvutit. 2014. URL: http://www.auul.pri.ee/wiki/Pg_partman (дата обращения 30.10.2014).
8. Query Planning // PostgreSQL: Documentation: 9.3: Query Planning 2014. URL: http://www.postgresql.org/docs/9.3/static/runtime-config-query.html (дата обращения 30.10.2014).

9. CONSTRAINT EXCLUSION WHEN IT FAILS TO WORK // CONSTRAINT EXCLUSION WHEN IT FAILS TO WORK 2008. URL: http://www.postgresonline.com/journal/archives/39-Constraint-Exclusion-when-it-fails-to-work.html (дата обращения 02.11.2014).
10. Efficient Use of PostgreSQL Indexes // Efficient Use of PostgreSQL Indexes | Heroku Dev Center 2014. URL: https://devcenter.heroku.com/articles/postgresql-indexes (дата обращения 04.11.2014).
11. Upgrading a PostgreSQL Cluster // PostgreSQL: Documentation: 9.3: Upgrading a PostgreSQL Cluster 2014. URL: http://www.postgresql.org/docs/9.3/static/upgrading.html (дата обращения 04.11.2014)