ОПТИМИЗАЦИЯ ЗАПРОСОВ К БАЗЕ ДАННЫХ С ПОМОЩЬЮ АЛГОРИТМОВ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА

Шараев Е. В.
Кафедра электронных вычислительных вычислительных машин, Белорусский государственный университет информатики и радиоэлектороники
Минск, Республика Беларусь
E-mail: eugen.sharaev@gmail.com

В наше время существует большое число решений в области систем управления базами данных. Несмотря на растущую сложность реализации и большое обилие алгоритмов, обеспечивающих большую производительность, по-прежнему существует некоторое число проблем, связанных с производительностью запросов. В то время, как одна часть проблем с производительностью решается с помощью увеличения технических характеристик машин, на которых развернуты базы данных, создания реплик, тонкой настройки конфигураций СУБД либо других оптимизаций, существует так же возможность использования алгоритмов искусственного интеллекта для выбора наиболее оптимальных стратегий для взаимодействия с хранящимися в базе данными.

Введение

Для минимизации времени выполнения запроса к базе данных в современных РСУБД применяется модуль, который называется планировщик (иногда планировщик/оптимизатор), задачей которого является поиск наиболее оптимального плана выполнения запроса. Планы различаются между собой примененными стратегиями доступа и манипулирования извлекаемыми данными. Примером таких стратегий могут быть seq scan (сканирование каждой записи в таблице по очереди), index only scan (когда существует покрывающий индекс, при котором в индексе достаточно данных для формирования возвращаемой коллекции и нет необходимости дополнительно обращаться к данным в таблице), различные стратегии объединения – hash join, merge join и так далее. Таким образом, различия в планах заключаются в различиях использованных алгоритмах в соответствующих планах.

I. Стоимостная оптимизация

В большинстве РСУБД применяются принципы стоимостной оптимизации (CBO, Cost Based Optimizer). При стоимостной оптимизации оптимальность плана запроса определяется значением, которое именуется стоимостью плана. Данное значение рассчитывается как совокупная стоимость каждой операции в запросе и определяется в зависимости от планируемого числа записей, которые будут извлечены из базы (значение cardinality). Приблизительное значение числа записей извлекается из статистики, которую собирает и хранит база. При этом отдельно учитывается стоимость для различных операций: последовательное чтение страницы с диска (в PostgreSQL обозначается как seq_page_cost), стоимость выполнения операций над кортежами на процессоре (cpu_tuple_cost) и так далее. После рассчетов стоимости каждого из планов база сравнивает их и определяет самый производительный на основе полученных значений стоимости.

В некоторых базах так же существует возможность настраивать параметры стоимости различных операций посредством конфигураций, либо даже подсказок планировщику. Данная настройка необходима по причинам того, что база данных может быть развернута на устройствах с различными аппаратными характеристиками. Например, стоимость чтения с диска будет выше, если используется hdd, и, соответственно, меньше, если используется ssd, стоимость операции на процессоре может быть ниже установленной по умолчанию, если используется процессор нового поколения, либо наоборот – возрастать в случае того, если заведомо известно, что используется непроизводительный процессор.

Таким образом, различие в рассчитанной стоимости каждого из планов обусловлено различием операций, которые производит база в ходе выполнения запроса.

II. Генетический оптимизатор запросов

При большом количестве таких планов время подбора оптимального плана может так же сильно возрастать (так как данная задача относится к классу NP), что может привести к тому, что суммарное время подбора оптимального плана и его выполнение будет неоптимальным. Для предотвращения таких ситуаций в некоторых базах применяются алгоритмы искусственного интеллекта, позволяющие минимизировать суммарное время запроса. Так, например, база данных PostgreSQL использует генетический алгоритм [1] для поиска оптимального плана, когда количество использованных инструкций join превышает указанное значение (по умолчанию 12).

Данный алгоритм является более приемлемой альтернативой перебору всех возможных планов при большом их количестве, поскольку при меньшем числе итераций алгоритма (по сравнению с перебором) мы получаем промежуточное решение, которое наиболее вероятно будет являться более оптимальным.

Меньшее количество итераций обеспечивается различиями в критериях остановки. При переборе критерием остановки является конец цикла по каждому плану, в то время как генетический алгоритм может иметь различные критерии: остановка по прошествии времени либо по достижении указанного числа итераций алгоритма, достижение плато (дальнейшие итерации не дают более оптимальных результатов), а так же комбинации перечисленных.

В данный момент используется именно условие остановки по прошествии необходимого числа итераций. Несмотря на свою эффективность, реализация генетического алгоритма в PostgreSQL не является финальной и авторы указывают на то, что все еще необходима работа над поиском оптимальных параметров алгоритма, оптимизация методов скрещивания хромосом и так далее.

III. Адаптивная статистика в Oracle Database

Начиная с 12c в Oracle Database был реализован механизм адаптивной статистики [2], который по своему описанию (исходный код данной РСУБД закрыт) представляет собой метод обучения с учителем. Суть данного механизма заключается в оптимизации предсказания количества строк, которые будут извлечены из базы (cardinality). Значение cardinality далее используется при определении оптимальных планов по принципам CBO. После выполнения каждого из запросов база данных сравнивает изначальное значение количества строк, которые планировалось извлечь, и полученное в результате запроса. В случае, когда предсказанное и полученное значения cardinality различаются, база данных делает запись о том, какое число строк вернулось после выполнения и в будущем использует это значение в последующих запросах. Таким образом, сохраняя данные значения, база каждый раз оптимизирует время выполнения.

Так же база обрабатывает случаи потери актульности статистики. Например, если данные в таблице изменяются больше чем на 10% с момента последнего сбора статистики, то соответствующие значения перестают учитываться при построении планов запросов.

IV. Адаптивная оптимизация запросов в PostgreSQL

В PostgreSQL так же существует экспериментальная реализация адаптивной оптимизации [3]. Как и в Oracle Database здесь используется аналогичный метод подбора значения cardinality для схожих запросов.

В качестве алгоритма используется модифицированный метод KNN (K-ближайших соседей), способный обрабатывать ситуацию изменения данных в таблице. Хотя данная реализация способна справляться с различными изменениями в данных, авторы все же рекомендуют реинициализировать алгоритм после внесения обширных изменений в данные (в контексте KNN это означает удаление созданных объектов с информацией о cardinality) .

Тем не менее, данная реализация имеет несколько недостатков. В качестве признаков здесь используются значения селективности (процент кортежей, удовлетворяющих конкретному условию), извлекаемые из статистики, что может работать не совсем корректно, так как неэквивалентные условия в запросе могут превращаться в эквивалентные по значению результаты предсказания. Например, если положить, что условия age $<$ 15 и age $>$ 25 имеют одинаковое значение селективности 0.05, то условия age $<$ 15 AND reservist IS TRUE и age $>$ 25 AND reservist IS TRUE будут иметь эквивалентное предсказание значения cardinality, что не является верным, так как в действительности первый запрос, в отличие от второго, не вернет кортежей (предполагается, что не существует военнообязанных моложе 15 лет). Так же существует проблема с работой данного плагина на репликах, поскольку параметры алгоритма хранятся в таблице в базе (однако существует возможность использования данных с master-сервера).

Альтернативой данному решению может быть использование расширенной статистики [4], которая была реализована в 10 версии PostgreSQL. Она позволяет создавать дополнительную статистику, которая хранит информацию о связи одновременно между несколькими атрибутами в таблице. Основным недостатком данного метода оптимизации является крайне быстро возрастающий размер статистики в зависимости от количества атрибутов, указанных при её создании.

1. PostgreSQL 10.5 Documentation. – 2017 p.
2. Farooq, T. Oracle Exadata Expert's Handbook / T. Farooq, C. Kim, N. Vengurlekar. – 157 p.
3. Ivanov, O. Adaptive Cardinality Estimation / O. Ivanov, S. Bartunov // arXiv:1711.08330. – 2017.
4. PostgreSQL: Documentation: 10: 14.2. Statistics Used by the Planner [Electronic resource]. – Mode of access: https://www.postgresql.org/docs/10/static/planner-stats.html. – Date of access: 21.09.2018.