ПРИМЕНЕНИЕ ИМИТАЦИОННОГО МОДЕЛИРОВАНИЯ К ОПТИМИЗАЦИИ ИНДЕКСОВ БАЗ ДАННЫХ

И.И. Труб (Москва)

Битовые (*bitmap*) индексы достаточно хорошо известны в мире баз данных, т.к давно уже стали стандартным средством оптимизации запросов в таких промышленных СУБД как Oracle, Cache[2] и других. Эффективность их использования зависит от особенностей базы данных и обслуживаемых ею запросов. Битовые индексы на протяжении около двадцати лет являются темой научных исследований и экспериментов со сложившейся классификацией направлений. Главная идея заключается в том, чтобы не использовать их в чистом виде, а создавать модификацию, наиболее эффективную в каждом конкретном случае. И затраты на ее реализацию вполне оправдываются повышением производительности обработки запросов и, соответственно, рыночного рейтинга созданного приложения. Выбор способа применения битовых индексов определяетсяв том числе такими факторами как:

- особенности индексируемого атрибута – каков его тип, сколько уникальных значений он может принимать (это свойство атрибута называют кардинальностью), распределение значений атрибута – равновероятны ли они или описываются более сложной функцией распределения;
- особенности запросов к сущности (таблице) базы данных по этому атрибуту: запросы ли это на равенство, односторонние или двусторонние; как распределены по частоте запрашиваемые значения и границы интервалов; поисковые ли это запросы или запросы на построение статистических (OLAP) отчетов;
- можно ли рассматривать данные в таблице как неизменный контент (что характерно для систем data warehouse) или же частотой обновлений, а значит и необходимостью перестраивать индексы, нельзя пренебречь (что характерно для транзакционных систем);
- ограничения на ресурсы системы по оперативной памяти и объему дискового пространства.

Отметим, что развитие теории и практики идет тремя основными путями: сжатие (*compression*), кодирование (*encoding*) и группирование (*binning*)[7]. В работе рассматривается последний из перечисленных. Идея *binning*проста и понятна. Весь диапазон значений атрибута делится на интервалы (*bins*), и, наряду с битовыми строками для индивидуальных значений, отдельная битовая строка создается и для каждого интервала. При этом наибольший выигрыш в производительности достигается в том случае, если диапазон запроса по атрибуту полностью накрывает одну или несколько групп. Тогда количество дизъюнкций битовых строк сокращается с количества входящих в группу значений до одного. В общем же случае план запроса строится как комбинация битовых строк, соответствующих групповым и индивидуальным индексам с применением *OR* (логическое ИЛИ) и *XOR* (исключающее ИЛИ) операций. Задача заключается в том, чтобы подобрать такое значение крупного индекса, чтобы количество таких операций, в среднем приходящееся на один запрос, было как можно меньше.

Работа построена следующим образом. В первой части задача ставится для случая, когда тип атрибута – время, и совокупность его значений можно интерпретировать как случайный поток событий. Во второй части рассмотрены принципы построения собственно имитационной модели для поставленной задачи. В третьей части описана модификация исходной постановки для случая районирования значений атрибута. В четвертой части рассмотрена постановка задачи для иной интерпретации атрибута – как обычной выборки некоей случайной величины, где тип атрибута – числовой. В заключении сформулированы выводы и итоги работы.

Индексируемое свойство как стационарный входной поток

Предположим, что анализ входных данных [6, глава 9] позволяет поставить задачу следующим образом. Занесение записей в таблицу образует поток событий, описываемый функцией распределения $F_1(t)$ случайной величины X_1, которую определим как интервал между двумя последовательными событиями потока. Длина интервала времени, задаваемого в запросе на

выборку записей, является случайной величиной X_2 с распределением $F_2(t)$. Шкала времени разбита на равные интервалы единичной длины. Начало интервала для X_2всегда совмещено с началом единичного интервала. Назовем полуоткрытый единичный интервал $[i;i+1)$*помеченным*, если он содержит хотя бы одно событие из потокаX_1. Случайную величину Y определим как количество помеченных интервалов, полностью накрываемых случайным интерваломX_2. Тогда Yкак раз и является количеством индексов (дизъюнкций), используемых для обслуживания запроса. Эту величину следует сделать как можно меньше.

Типичным примером из практики является генерация отчетов, в которых фильтр накладывается на атрибут, являющийся временем занесения записи в таблицу. Например, в системе, обслуживающей call-центр, имеется типовая сущность «Входящие звонки», хранящаяся в отдельной таблице (Oracle) или глобале (Cache). Основным атрибутом сущности является время поступления звонка с точностью до секунды. Интенсивность поступления звонков в масштабах всей компании, высока, т.е. объем таблицы велик. Типовым сервисом для аналитиков компании является генерация ad-hocотчетов с помощью некоей формы задания ограничений, причем основным фильтром служит диапазон времени, за который нужно вывести все звонки. Полный поиск по таблице крайне неэффективен, поэтому по данному атрибуту строятся битовые индексы, где каждая битовая строка в случае большого количества записей возможно состоит из нескольких подстрок (*chunks*). Берутся все индексы для значений, входящих в заданный диапазон, и путем их дизъюнкции вычисляется результирующая битовая строка, единицы которой соответствуют ID записей, удовлетворяющих запросу и выводимых в итоговый отчет. Однако, практика показывает, что и это для промышленных объемов данных не дает приемлемой для потребителя производительности – отчеты генерируются слишком медленно в виду большого количества дизъюнкций. Это происходит при большой ширине диапазона и высокой интенсивности занесения в БД новых записей. Здесь то и приходит на помощь *binning*.

Заметим, что сама методология *binning*наряду с *OR*-операцией предполагает при работе с битовыми строками использование также и *XOR*. Поясним это примером. Пусть размер крупного индекса равен одной минуте, и начало фильтра запроса совпадает с границей минуты, а длина равна 50 секунд. Если вычислять результат только с помощью дизъюнкций, понадобится50 битовых строк – от первой до 50-й секунды. Но их количество можно сократить до 11, если осуществить *XOR*-операцию минутного индекса с секундными от 46-й до 60-й. Таким образом, вместо «складывания» событий, произошедших в заданном диапазоне, из всех событий, произошедших в течение минуты, «вычитаются» те события, которые в этот диапазон не вошли.

Итак, идея в самом деле проста, но ее использование требует обоснованной методики расчета показателей эффективности. В частности, необходимо с той или иной точностью дать ответ на следующие вопросы:

- как вычислить распределение случайной величины Y для заданного значения крупного индекса k ?
- как вычислить оптимальное значение k, минимизирующее Ycp ?

В [3] поставленная задача решена с помощью аналитической модели методами теории вероятностей при наличии двух упрощающих предположений: распределение$F_1(t)$является экспоненциальным, и левая граница фильтра запроса совпадает с границей крупного индекса. Однако, на практике они могут и не выполняться, особенно первое из них, т.к. потоки в Интернет имеют бОльшую тенденцию к *heavytailed* распределениям. Отказ от любого из предположений разрушает модель из [3], решение же задачи в общей постановке возможно только средствами имитационного моделирования, которое позволяет реализовать любые входные условия и внутреннюю логику системы.

Имитационная модель

Принципы построения модели для решения поставленной оригинальной задачи описаны в [4]. Укажем, следуя этой работе, в чем же состоит оригинальность, и каким образом она учтена в модели. При традиционном моделировании СМОмы генерируем две последовательности

случайных величин – для входного потока заявок и длительности их обслуживания. Обе эти последовательности являются runtime, т.е. в момент прихода заявки генерируется и интервал до прихода следующей, и длина этой заявки. Здесь же логика иная. Когда генерируется интервал для запроса необходимо, чтобы значения атрибута уже были назначены всем записям таблицы. Таким образом, сначала полностью должно быть реализовано распределение$F_1(t)$ (назовем эту реализацию *поле*), и только после этого реализуется распределение $F_2(t)$. Очередное значение выходной величины Yвычисляется для каждого элемента реализации$F_2(t)$. Иными словами, имея поле, значение крупного индекса и интервал запроса, мы по определенному алгоритму вычисляем, сколько же всего индексов понадобится для обслуживания этого запроса. Имитационный эксперимент сводится к вычислению Y_{cp} как среднего значения последовательности $Y_{cp,1}$,$Y_{cp,2}$,…, где $Y_{cp,i}$–среднее значение для реализации Y, вычисленной на одном поле. Эту последовательность назовем *глобальной*, в то время как последовательность реализаций Yна одном и том же поле для каждого i–*локальной*.

Указанные особенности затрудняют использование стандартных прикладных пакетов, поэтому для построения модели был использован подход, описанный в [5] – собственная реализация на языке C/C++, позволяющая учесть логические нюансы в полном объеме. Программная структура модели представлена на рис.1. Наличие направленной связи обозначает вызов из одной функции другой. Опишем семантику этих функций:

- cycleGlo – цикл, на каждой итерации которого вычисляется очередное значение Ycp,i;
- batchingGlo, estimationGlo – функции анализа выходных данных, используемые для завершения моделирования;
- spectralVarAnalysis – функция, лежащая в основе анализа выходных данных;
- leastSquares – реализует интерполяцию методом наименьших квадратов;
- getSingleDataGlo – реализует вычисление одного значения Ycp,i ;
- getGround – вычисляет новое поле как последовательность целочисленных случайных величин, интервалы между которыми описываются функцией F1(t). Генерация поля продолжается до достижения значения 86400 (количество секунд в сутках), но может быть выбрано и другое значение;
- cycleLoc – цикл, в котором моделируется запросы на одном и том же поле;
- batchingLoc и estimationLoc – анализ выходных данных в локальном цикле. Реализация та же, что и для глобальных функций. Выделены в отдельные блоки, чтобы еще раз подчеркнуть двухфазную структуру модели;
- getSingleDataLoc – движок моделирования. Генерирует очередной запрос на выборку данных и вычисляет количество индексов для его обслуживания;
- getReqBegin – генерирует левую границу интервала запроса по заданному разработчиком модели алгоритму;
- getReqLength – генерирует длину запроса в соответствии с распределением F2(t);
- randomNumbers – генератор случайных чисел. Реализован с помощью метода обратной функции. Многочисленные примеры приведены в [5];
- Integration – реализация численного интегрирования (метод Симпсона), необходимого для вычисления некоторых неэлементарных функций распределения.

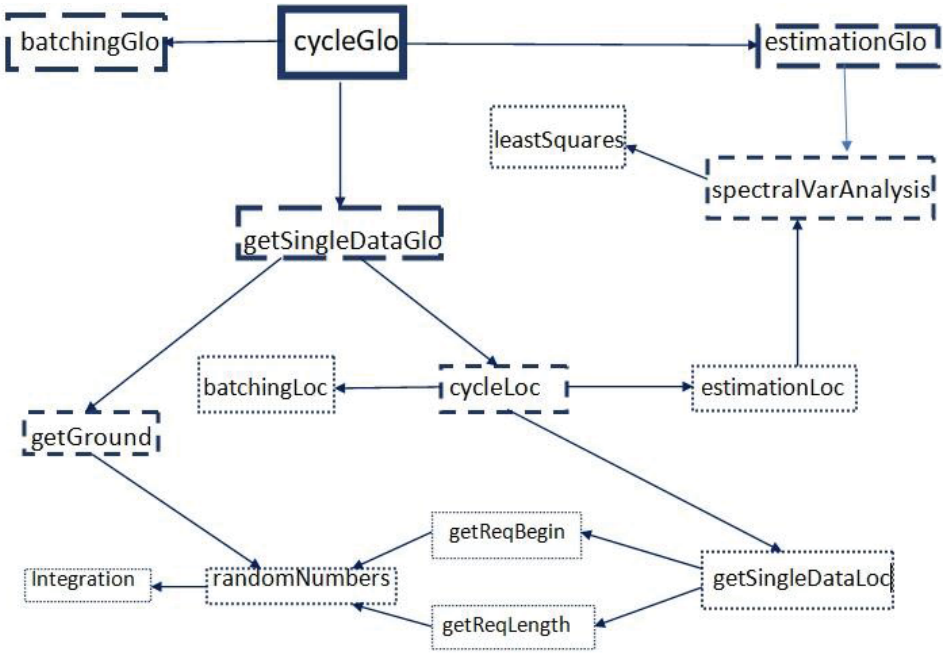

Рис.1. Программная структура имитационной модели

Целью анализа выходных данных является определение момента, когда моделирование следует прекратить, т.е. доверительный интервал для величины Y_{cp} с требуемым уровнем достоверности уже получен. Этот анализ нужно проводить на обеих фазах – локальной (моделирование на одном поле) и глобальной (последовательность результатов на разных полях). Основы анализа выходных данных (*outputanalysis*) изложены в [6, глава 11]. В отличие от классических моделей для СМО, где анализируется, например, количество заявок в системе, этот анализ в нашем случае упрощается благодаря следующим обстоятельствам:

- нет необходимости отделять период «разогрева» и определять момент установления стационарного режима, т.к. все наблюдения анализируемой случайной величины – количества индексов для обслуживания запроса – являются одинаково репрезентативными;
- все наблюдения выходной случайной величины являются независимыми от предыдущих, поэтому не требуется дополнительных вычислений, чтобы отбросить влияние фактора автокорреляции.

Для реализации в модели был выбран *метод групповых средних* (*batchingmeans*). Его доступное изложение с псевдокодом реализации приведено в обзорной статье [8].Этот метод анализа выходных данных достаточно прост и удобен в реализации и показал свою практическую работоспособность.

Верификация модели выполнена путем сравнения ее результатов с результатами уже упомянутой аналитической модели из [3] при тех же входных данных и ограничениях на них.Это сравнение полностью подтвердило корректность имитационной модели – расхождения между значениями Y_{cp}в обеих моделях составило не более 0.5% в относительном исчислении и не более 0.1 в абсолютном.

На рис.2 (нижняя кривая) показана зависимостьY_{cp} от размера крупного индекса k для $F_1(t)=1-e^{-\lambda t}$, $\lambda=1/30$, $F_2(t)=1-e^{-\mu t}$, $\mu=0.001$. Это соответствует появлению новой записи в таблице примерно каждые 30 секунд и средней длине интервала запроса примерно 17 минут. Значение левой границы интервала запроса разыгрывалось таким образом, что оно может с равной вероятностью приходиться на границу секунды, минуты или часа. Поведение построенной зависимости совпадает с интуитивно ожиданием – она имеет минимум. В самом деле, при слишком малых значениях k эффект укрупнения индекса проявляется незначительно, при слишком больших – не проявляется совсем, т.к. размер крупного индекса может оказаться больше, чем средняя длина запроса. Поэтому логично предположить, что при некотором оптимальном

значении достигается минимум, и модель позволяет его вычислить. При заданных исходных данных минимум достигается приблизительно в диапазоне 320-340 секунд и равен 6.5-6.6 индексов. С ростом kY_{cp} асимптотически стремится к предельному значению, соответствующему случаю, когда крупного индекса нет. Из рисунка видно, что наличие крупного индекса размером 5 или 6 минут позволяет ускорить обработку запросов приблизительно в 5 раз, т.к. при k=1 значение Y_{cp} лежит в диапазоне 30-35 индексов.

Входной поток с расслоением

Укажем на одну особенность входных данных, часто встречающуюся на практике и требующую учета при использовании модели в целях повышения точности ее результатов. До сих пор интервалы между событиями случайного потока предполагались одинаково распределенными на всем протяжении временнОй оси. Это вполне может быть справедливым, если речь идет о технологических событиях какого-либо непрерывного производственного цикла. Однако, для систем, в которых участвуют люди, например, тех же CRM (*CustomerRelationshipManagement*) со входящими звонками, это, скорее всего, будет уже не так. Действительно, на протяжении рабочего дня интенсивность поступления звонков может меняться, будучи в утренние часы одной, в обеденные - другой, в послеобеденные – третьей, а в ночные – вообще нулевой (или очень низкой, если call-центр компании работает в круглосуточном режиме). Поэтому для получения практически эффективных результатов важно при анализе входных данных выделить промежутки статистической однородности со своими входными данными и для каждого из них получить собственное решение с помощью имитационной модели. В противном случае решение получится слишком «размытым» и далеким от оптимального. С точки зрения проектировщика БД это означает, что для разных диапазонов данных размеры крупных индексов будут отличаться, что является вполне реализуемым с точки зрения разработки приложений БД.

Теоретически такая задача анализа входных данных хорошо известна в математической статистике и носит название *расслоенной* или *районированной выборки (stratifiedsampling).* Основной вопрос, ответ на который следует найти, таков: сколько должно быть слоев (интервалов однородности данных) и где должны быть границы между ними? Наиболее полно эта и многие другие классические задачи, связанные с изучением расслоенных выборок, рассмотрены в известной монографии [1, глава 5], но концептуально основной принцип расслоения можно сформулировать так: дисперсия выборки в пределах одного отдельно взятого слоя должна значимо отличаться от дисперсии полной выборки по всем собранным данным. Основываясь на этом интуитивно понятном свойстве, специальные процедуры анализа данных пробуют выполнить гипотетическое разделение их на слои, а методы из [1] позволяют оценить то или иное разделение по математически точным критериям и остановиться на наиболее удачном из них. Алгоритмические подробности выполнения расслоения реальных данных для временнОго потока событий могут составить тему отдельной публикации и выходят за рамки данной работы.

Индексируемое свойство как совокупность данных

Предложенный подход к построению модели можно применить и в том случае, когда входные данные интерпретируются не как поток событий, а как некоторая реализация случайной величины с заданной функцией распределения. Это соответствует числовому типу атрибута (*float*или *integer*), где данные представляют собой множество собранных в разное время измерений, например, температуры, давления, влажности и т.д. Назовем такую модель *распределенной* в отличие от предыдущей – *потоковой*. При ее использовании следует иметь в виду отличия от потоковой модели, а именно:

- модель работает с упорядоченным по возрастанию полем, поэтому анализ входных данных следует производить только тогда, когда данные полностью собраны. Естественная упорядоченность значений атрибута по мере занесения новых записей в таблицу здесь отсутствует. Такая ситуация как раз и характерна для нетранзакционных систем;
- даже после упорядочения данных по возрастанию их интерпретация как потока событий будет заведомо неприемлемой, т.к. интервал между соседними значениями выборки не

является стационарной случайной величиной, а по определению зависит от *x*. В частности, если объем выборки равен *N*,легко получить формулу для функции распределения *Φ(x,t)*длины интервала *t* между элементом выборки со значением *x*и ближайшим по возрастанию следующим элементом:

$$\phi(x,t) = 1 - \sum_{i=0}^{N-2} (F(x))^i (1 - F(x+t))^{N-1-i} \qquad 1)$$

Формула (1) вычисляет вероятность события, дополнительного к следующему: в правой *t*-окрестности *x*не окажется элементов выборки. Верхняя граница для *i* отражает тот факт, что мы не рассматриваем случай, когда *x*является максимальным элементом выборки, т.е. когда все остальные *N-1*элементов лежат слева от *x*.

Наилучшим систематическим изложением аналитических результатов для задачи в такой постановке является фундаментальная работа известных специалистов по теории битовых индексов [9]. В ней выведены полезные оценочные формулы для большого количества вариантов использования групповых битовых индексов. Но в силу некоторых принятых авторами допущений эти формулы не всегда применимы на практике, а именно:

- предполагается заданной кардинальность атрибута C;
- распределение значений атрибута предполагается равномерным;
- распределение значений границ интервала запроса тоже предполагается равномерным.

По этой причине в результатах постоянно встречаются множители *1/C*и *1/C*^{2} . Снять эти ограничения может только имитационная модель. Основное и, пожалуй, единственное отличие этой модели от модели с потоком событий состоит в способе генерации поля. Т.к. в случае потока событий функция распределения описывает интервал между ними, генерация поля происходит «нарастающим итогом», т.е. последующее значение вычислялось рекуррентно из предыдущего прибавлением очередной реализации случайной величины. В данном случае реккуррентности нет, каждая точка поля генерируется сама по себе, поэтому итоговую совокупность данных, как уже отмечалось, следует упорядочить по возрастанию. На рис.2 показаны зависимости $Y_{cp}(k)$для обеих моделей при максимальном сохранении общности исходных данных. Так, интервалы для запросов генерируются абсолютно без изменений, распределение значений атрибута принято экспоненциальным с параметром λ=1/43200, чтобы точки поля были, как и прежде, разбросаны в диапазоне от 0 до 86400, количество генерируемых точек поля равно 86400/(1/0.033)=2880.

Из рисунка видно, что при различии количественных показателей качественное поведение зависимостей для обеих моделей сохраняется – в распределенной модели также существует точка минимума. Количественное же различие связано с тем, что для потоковой модели интервалы между точками поля независимы и одинаково распределены, а для распределенной модели, как уже было сказано, это не так.

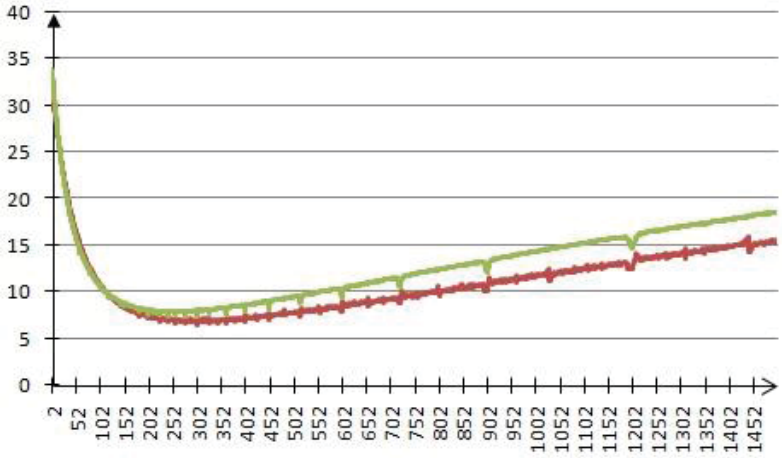

Рис. 2. Графики $Y_{cp}(k)$ для потоковой (нижний) и распределенной (верхний) моделей

Выводы

В работе впервые предложена идея применения имитационного моделирования к поиску оптимального размера крупного битового индекса для временнЫх и числовых свойств в таблицах реляционных баз данных, а также концепция реализации этой идеи в виде двухфазной модели. По результатам исследований можно сделать следующие выводы:

1) модель работоспособна, параметризуема и демонстрирует достижение требуемой точности на каждой из фаз моделирования;
2) использование модели требует правильной интерпретации и, соответственно, генерации входных данных – как стационарного входного потока (моделируются промежутки между случайными событиями) либо как совокупности значений некоей случайной величины (моделируются сами значения);
3) качественная зависимость средней трудоемкости обслуживания запроса как числа битовых операций от размера крупного индекса показывает наличие точки минимума, что совпадает с интуитивно ожидаемым поведением. Модель позволяет дать количественную оценку этого значения и оценить выигрыш в производительности.
4) модель может быть легко адаптирована для решения других исследовательских задач – таких как зависимость оптимального размера индекса и доставляемого им оптимального значения от параметров бизнес-процессов, от видов используемых распределений и от других условий.

Предложенная имитационная модель может служить полезным практическим инструментом, позволяющим архитектору баз данных подобрать размер битового индекса, обеспечивающий наибольшую производительность пользовательских запросов с учетом индивидуальных особенностей бизнес-процессов конкретного клиента, таких как интенсивность заполнения базы и длина запроса по индексируемому свойству.

Литература

1. Кокрен У. Методы выборочного исследования. - М.: Статистика, 1976. 440 с.
2. Мартынов Д. Bitmap-индексы в Cache на глобалах. URL: http:// habrahabr.ru/company/intersystems/blog/174657/.
3. Труб И.И. Вероятностная модель иерархических индексов баз данных // Программные системы и вычислительные методы. 2017. № 4. С. 15–31.
4. Труб И.И. Имитационное моделирование иерархических bitmap-индексов // Прикладная информатика. 2018 № 4 (76). С.53–69.
5. Труб И.И. Объектно-ориентированное моделирование на C++. СПб.: Питер, 2005. — 416 с.
6. Discrete-Event Simulation. 3rd ed. // Banks J., Carson J.S., Nelson B.L., Nicol D.M. Prentice Hall Englewood Cliffs, NS, 2000. — 594 p.
7. Otoo E., Wu K. Accelerating Queries on Very Large Datasets // Sceintific Data Management. Challenges, Technology and Deployment (edited by Arie Shoshani, Doron Rotem). Chapman & Hall/CRC, 2010.P. 183-224.
8. Pawlikowski K. Steady-State Simulation of Queueing Processes: a Survey of Problems and Solutions // ACM Computer Surveys. Vol. 22. Issue 2. June 1990. P. 123–170.
9. Wu K., Shoshani A., Stockinger K. Performance of multi-level and multi-component compressed bitmap indexes. ACM Transactions on Database Systems, Volume 35, Issue 1, February 2010 P.1-52.