АЛГОРИТМ ФИЛЬТРАЦИИ В СИСТЕМЕ ПУБЛИКАЦИИ/ПОДПИСКИ[1]

Шагал А.А., магистрант второго курса кафедры компьютерных технологий Университет ИТМО, shagal@rain.ifmo.ru

Аннотация

Системы публикации/подписки — это принцип используемый при проектировании распределенных систем. Вычислительные узлы делятся на источников информации и подписчиков. Подписчик формирует критерии, относительно которых фильтруется информации перед отправкой. В зависимости от сложности этих критериев, фильтрация информации является критическим местом в системах публикации/подписки. В данной работе описываются свойства языка подписок и алгоритм фильтрации, показывающий наилучшую производительность на критическом пути по сравнению с существующими решениями.

Введение

Язык подписок в системе публикации/подписки — это язык, с помощью которого подписчик формирует критерии отбора информации. В данной работе речь пойдет о content-based языках. В таких языках критерии формируется относительно содержимого сообщения, и в большинстве случаев задаются логическим выражением. Примером логического выражения может быть “stock == IBM && MIC == XEUR”. Фильтруемая информации задается сообщением. Сообщение является набором пар: атрибут и его значение. Примером сообщения, которое должно быть отправлено подписчику, может быть “stock = IBM, MIC = XEUR, bid = 239”. Критическим местом большинства content-based систем является фильтрация. Задача фильтрации — поиск всех подписок, которым должно быть отправлено сообщение. Существует множество подходов по решению задачи фильтрации, каждый из которых преследует свои цели и обладает теми или иными отличительными качествами. Далее будут рассмотрены основные направления, по которым идет развитие

[1] Данная работа поддержана компанией ITIVITI (http://www.itiviti.com), описанный алгоритм применяется внутри системы алгоритмической торговли — основном продукте компании.

алгоритмов фильтрации в content-based системах.

Первое самое важное свойство, относительно которого строится большинство алгоритмов — форма фильтра. Подписчик формирует фильтр исходя из своих правил. Проверка, удовлетворяет ли очередное сообщение фильтру или нет, может использовать фильтр в той форме, которую сформировал подписчик, но большинство алгоритмов работает только с ДНФ фильтрами. Проблема ДНФ фильтра заключается в экспоненциально размере фильтра, а также в нивелировании логики подписчика.

В зависимости от сложности фильтров можно осуществлять более или менее точный отбор информации. Различные языки могут поддерживать различные операции сравнения, различные типы данных и т.д. Анализ существующих решений показывает, что чем сложнее язык, тем сложнее алгоритм фильтрации и тем хуже его производительность.

Выделяется два типа алгоритмов фильтрации: counting алгоритмы и алгоритмы использующие деревья. Эти алгоритмы имеют принципиально разные подходы к уменьшению времени обработки очередного события. Counting алгоритм ориентирован на минимизацию количества предикатов, которые необходимо проверить. Два наиболее известных алгоритма: Propagation[1] и k-index[2]. Алгоритмы, использующие деревья, ориентированы на ускорение обработки каждого отдельного предиката и отбросе заведомо не нужных подписок. Наиболее востребованным алгоритмом на сегодняшний день является BE-tree[3]. Так же существуют более экзотические алгоритмы, использующие автоматы для описания подписок. Далее будут более подробно описаны алгоритмы BE-tree и k-index.

BE-tree, k-index, BDD

BE-tree(Boolean expression tree) имеет множество преимуществ по сравнению с другими алгоритмами. Алгоритм работает с динамическим множеством подписок и большим числом различных атрибутов относительно которых может осуществляться фильтрация. Подписки хранятся в дереве в формате конъюнкций. Каждая вершина дерева может быть либо leaf-node, либо p-node, либо c-node. В leaf-node хранится список подписок. Изначально дерево состоит из единственного leaf-node. В случае, если среди подписок можно выделить большое количество подписок с одним и тем же атрибутом, дерево модифицируется. Все такие подписки выделяются в p-node. Каждая p-node имеет несколько c-node, где c-node организованы в деревья отрезков. Таким образом дерево модифицируется в процессе добавления подписок. Алгоритм

ориентирован на то, чтобы отметать заведомо не подходящие подписки. Так же алгоритм динамически определяет, по какому атрибуту следует отметать подписки в первую очередь.

K-index — типичный пример counting алгоритма. Каждая подписка/фильтр приведена к ДНФ форме. Далее фильтр рассматривается как конъюнкция. В процессе проверки очередного события для каждой подписки считается число предикатов, которые проведены из рассматриваемого фильтра. Если рассчитанное число совпадает с числом предикатов в фильтре, он считается проверенным. Особенностью k-index является использование инвертированного индекса. Ключом является предикат целиком. Значение — список подписок содержащих предикат. Counting алгоритм выигрывает в том, что один и тот же предикат будет проверен только один раз.

BDD — диаграммы решений. BDD является одним из самых популярных подходов, так как не требует ДНФ формы. Принцип алгоритма заключается в слиянии подписок. Алгоритм слияния подробно описан в статье о BDD[4].

Далее будут рассмотрены текущие задачи и нагрузка, относительно которых необходимо разработать новый алгоритм.

Анализ нагрузки и решение

Особенности языка

В фильтре могут быть предикаты на атрибуты, которых нет в сообщении. Примером такого фильтра может быть фильтр на параметры инструмента. Параметры и инструменты являются различными сущностями и передаются в двух сообщениях.

Типы данных, используемые в некоторых предикатах, невозможно индексировать. Примером таких данных могут быть динамические группировки. В случае, если проверка предиката подразумевает проверку иерархической структуры, индексирование является затруднительным. Предикаты отличаются скоростью проверки. Скорость варьируется от 10 до 500 наносекунд. Данные особенности заведомо отметают BE-tree. BE-tree рассчитан на то, что предикаты можно индексировать.

Counting алгоритмы так же плохо адаптируются. Рассмотрим стандартный фильтр: “deleted = false && smart condition” Очередное сообщение содержит атрибут deleted = true. В случае couting алгоритма будет осуществляться проверка smart condition. Несмотря на то, что она

занимает много времени и заведомо бессмысленна.

Анализ фильтров

Фильтры и предикаты можно классифицировать. Каждый предикат либо отметает большое число объектов, либо задает сложный критерий. Так же существуют предикаты, по которым легко осуществлять фильтрацию. К таким предикатам относятся те, которые можно индексировать хэш-таблицами, операция должна быть “==“ и тип данных должен хорошо индексировать подписки. Зачастую используются атрибуты с уникальным идентификатором.

Фильтры формируются клиентами. Клиент системы публикации/подписки — трейдер или стратегия, написанная программистом. Рассмотрим фильтр взятый из реальной системы: deleted = false && (CFI == OXXXX || CFI == FXXXX) && group == (American || Europian) && Trading enabled == true

Большая часть рассматриваемых объектов не пройдет deleted фильтр. 50% всех объектов не пройдут фильтр по CFI, 30% попадает в группу, 30% торгуется, скорость проверки каждого предиката возрастает экспоненциально.

Для данного языка нельзя использовать counting алгоритм. В лучшем случае для 10% сообщений необходима проверка сложных условий.
Для данного языка нельзя использовать BE-tree алгоритм, так как приведение фильтра к ДНФ форме увеличит количество фильтров, которые необходимо проверить.

BDD — алгоритм так же сложно адаптировать из-за структуры фильтров. Сложные предикаты имеют сложное распределение значений, в результате чего вершина диаграммы имеет большое число дочерних узлов. Таким образом BDD алгоритм сводится к полному перебору.

Алгоритм

Разделим фильтры на два типа. Первый тип содержит предикат, который можно индексировать хэш-таблицей. Для каждого атрибута поддержим хэш-таблицу. Второй тип — все остальные фильтры. Заметим, что большую часть фильтров можно представить в виде “Предикат && Фильтр”, при этом нет необходимости реорганизовывать фильтр,разработчик или пользователь уже сделал это, открыв подписку. Примеров может быть фильтр “deleted = false && (CFI == OXXXX || CFI == FXXXX) && group == (American || Europian) && Trading enabled ==

true”. Предикатом будет deleted = false. Выделим несколько таких групп, что каждая группа соответствует стартовому предикату. Статистика показывает, что таких групп не более 100. Алгоритм заключается в том, чтобы пройдя по каждой группе и проверив простые предикаты, соответствующие группе, выделить только те группы, которые необходимо проверить. Далее асинхронно проверить все подписки в каждой группе. Если предикат сложно проверить, то имеет смысл поддерживать информацию о том, не проверили ли этот предикат заранее. Для этого необходимо поддерживать множество проверенных сложных предикатов. В случае, если проверка предиката занимает дольше 100 наносекунд, проверка имеет смысл. Преимуществом данного алгоритма является его простота. В случаи если производительность измеряем в микросекундах сложность алгоритма имеет большое значение. Также существующие алгоритмы плохо адаптированы для многоядерных систем. Реализация данного алгоритма на языке C++ занимает меньше 100 строк, а проверка групп подписок является независимой, и распараллеливание не требует синхронизации. Достаточно выделить по потоку на каждую группу.

Результаты

Для замеров производительности использовались данные из промышленной системы алгоритмической торговли. Указанные ниже данные – стандартная нагрузка и техника используемая в промышленности:

- 500000 подписок(фильтров)
- 10000 сообщений в минуту
- 128 GB RAM, 16 cores

	50%	80%	90%	99%
k-index	2356 µs	3578 µs	4295 µs	4493 µs
BE-tree	1866 µs	2876 µs	3293 µs	3429 µs
BDD	2065 µs	3220 µs	3822 µs	3974 µs
new	1083 µs	1716 µs	2039 µs	2111 µs

Таблица 1: Результаты измерения времени работы алгоритмов фильтрации

Заключение

Предложен алгоритм фильтрации, обеспечивающий лучшую производительность для описанного языка. Алгоритм может быть применен для языков со свойствами такие как:

- Сложные типы, которые нельзя индексировать
- Сложные предикаты, проверка которых занимает больше 100 наносекунд

Литература

1. F. Fabret, H.-A. Jacobsen, F. Llirbat, J. Pereira, K. A. Ross, and D. Shasha. Filtering algorithms and implementation for fast pub/sub systems. SIGMOD'01.
2. S. Whang, C. Brower, J. Shanmugasundaram, S. Vassilvitskii, E. Vee, R. Yerneni, and H. Garcia-Molina. Indexing Boolean expressions. In VLDB'09.
3. M. Sadoghi and H.-A. Jacobsen. BE-Tree: An index structure to efficiently match boolean expressions over high-dimensional discrete space. SIGMOD' 11.
4. Campailla, S. Chaki, E. Clarke, S. Jha and H. Veith. Efficient filtering in publish-subscribe systems using binary decision diagrams. In International Conference on Software Engineering, 2001.