Санкт-Петербургский Государственный Университет

Математико-механический факультет

Кафедра системного программирования

Оптимизация процесса сборки документов системой nutch

Курсовая работа студента 445 группы

Волкова Сергея Андреевича

Научный руководитель

ассистент кафедры АСОИУ СПбГЭТУ ЛЭТИ _____ Л. С. Выговский

Санкт-Петербург

2010

Оглавление

Введение

Цель

Целью данной работы является оптимизация процесса сборки документов системой nutch.

Nutch

Nutch - свободное программное обеспечение для интернет поиска. Nutch основан на поисковом движке Lucene[3], с набором средств для работы с web, таких как поисковый робот, база ссылок, парсеры для html и других форматов и.т.п. Nutch работает поверх фреймворка hadoop.

Hadoop

Hadoop является свободным Java фреймворком, поддерживающим выполнение распределённых приложений, работающих на больших кластерах, построенных на обычном оборудовании. Hadoop прозрачно предоставляет приложениям надёжность и быстродействие операций с данными. В Hadoop реализована вычислительная парадигма, известная как MapReduce.

MapReduce

Общая идея MapReduce состоит в том, чтобы представить алгоритм в виде набора последовательных этапов, из которых каждый состоит из двух шагов - map и reduce (возможны дополнительные шаги - combine и partition). Система разбивает входные данные на пары $\langle key, value \rangle$, над каждой парой выполняется функция map на выходе которой, должен получится набор пар $\langle key, value \rangle$. Сгенерированные данные система

реорганизует и для каждого из них выбирает узел на котором исполнится процедура reduce. Данные для одного reduce узла группируются по одинаковым ключам и ключи сортируются. После чего данные, в виде $\langle key, value* \rangle$ опять передаются пользователю, который производит над ними операцию reduce (свёртка) на каждом узле. Получившиеся на выходе пары $\langle key, value \rangle$ передаются в выходной поток (Рис. 1).

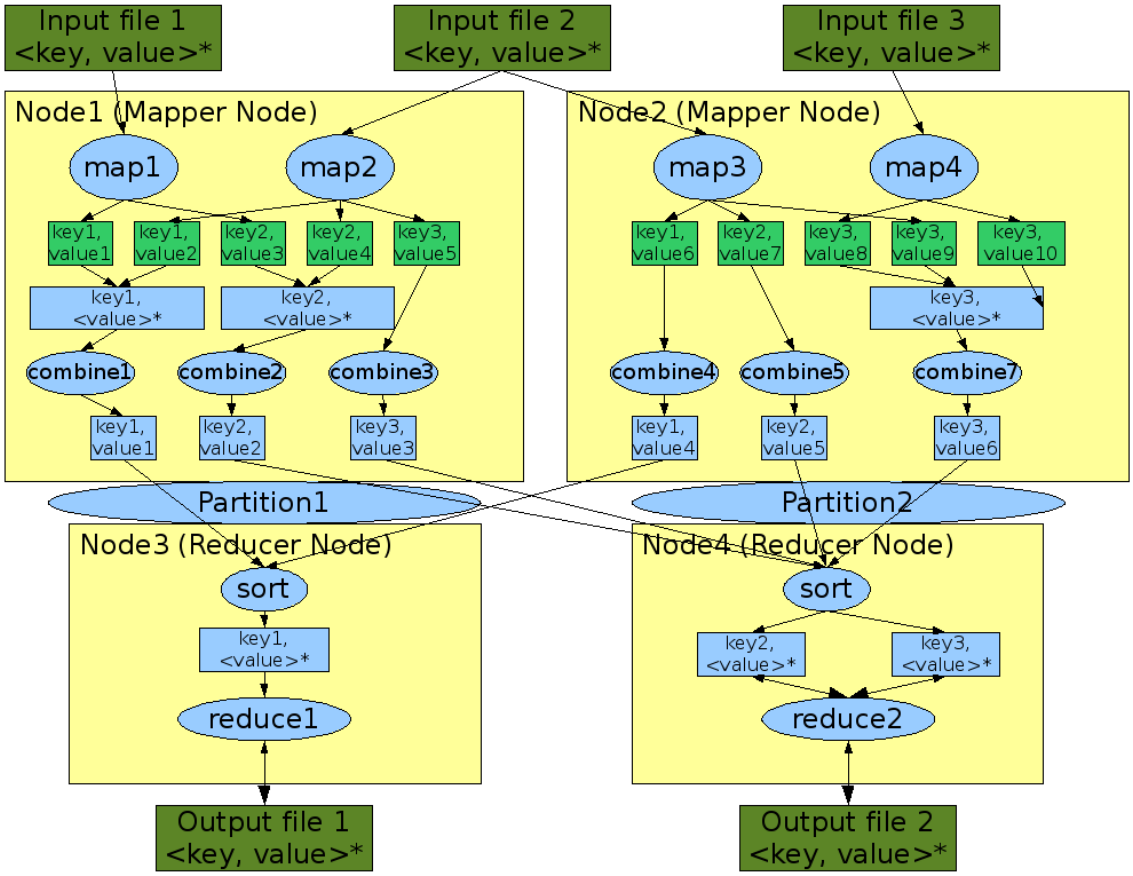

Рис. 1: Схема MapReduce[1]

Преимущество MapReduce заключается в том, что он позволяет распределенно производить операции предварительной обработки и свертки. Операции предварительной обработки работают независимо друг от друга и могут производиться параллельно. Аналогично, множество рабочих узлов могут осуществлять свертку — для этого необходимо только чтобы все результаты предварительной обработки с одним конкретным значением ключа обрабатывались одним рабочим узлом в один момент времени. Таким образом MapReduce может быть применен к большим объемам данных, которые могут обрабатываться большим количеством серверов.[5]

Работа nutch с точки зрения MapReduce

Сборка осуществляется итеративно - на каждой итерации осуществляется выбор url для загрузки, загрузка, индексация полученных данных, обновление базы ссылок и обновление базы обратных ссылок.

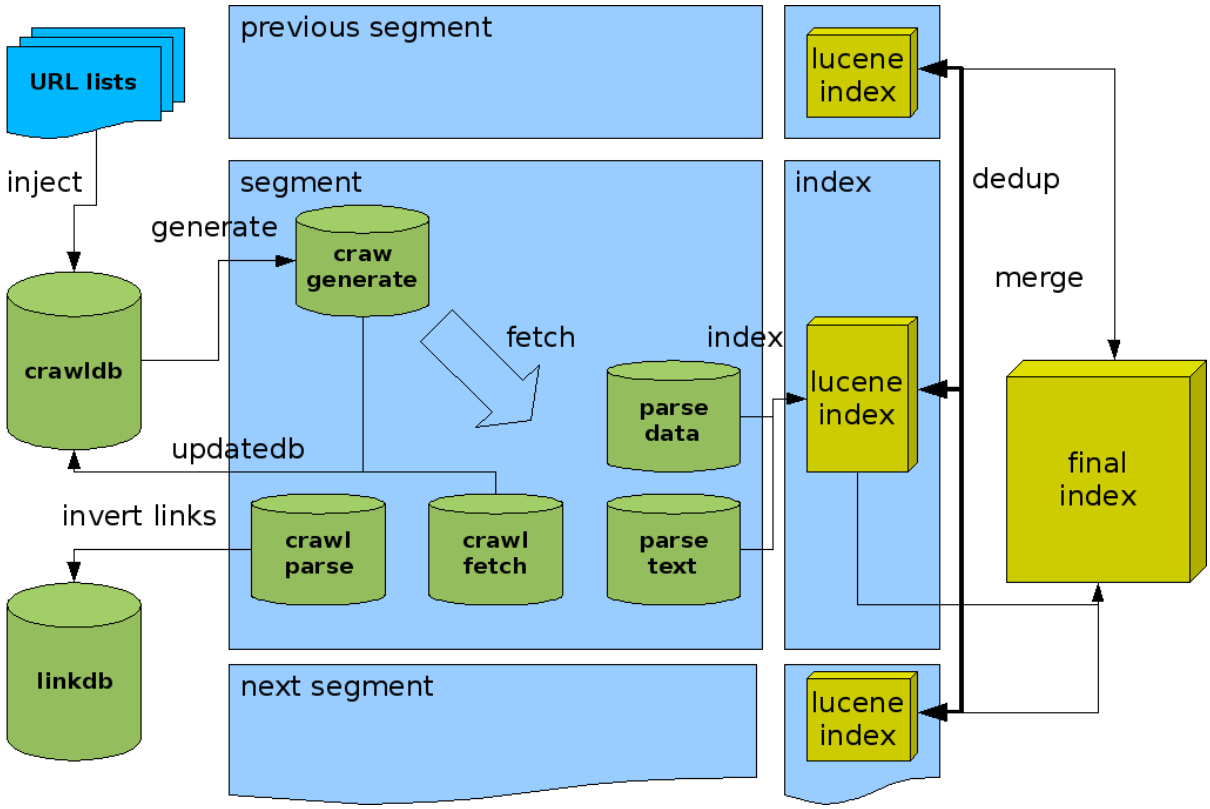

Рис. 2: Работа Nutch[1]

На рис. 2 представлена схема работы nutch, при этом приняты следующие обозначения:

- `generate` — выбор url из общей базы (crawldb)
- `fetch` — загрузка документов по url
- `index` — создание индекса по скачанным документам
- `merge` — слияние индекса по сегменту с основным
- `dedup` — удаление дубликатов из индекса
- `updatedb` — обновление crawldb
- `invert links` — создание и обновление базы обратных ссылок (linkdb)

Условия

Задача web сборки заключается в нахождении нужных url и скачивания документов по ним, большинство url появляется из уже скачанных документов. Иногда непосредственно по url можно определить что документ является “полезным”. Например, для того, что бы скачать все новости с ресурса lenta.ru, достаточно скачивать документы по url который можно задать регулярным выражением 1

$$http://lenta\backslash.ru/news/\backslash d\{4\}/\backslash d\{2\}/\backslash d\{2\}/\backslash w+/ \tag{1}$$

Постановка задачи

Необходимо изменить nutch для работы с учетом информации о “полезности” ссылок.

Основные требования:

- минимальные изменения ядра nutch
- возможность обновления информации о ссылках без остановки сборки
- возможность работы без вмешательства администратора

Актуальность

Nutch является активно развивающимся проектом, на его основе сейчас создан один из крупнейших поисковиков по исходному коду Krugle. Одна из основных проблем web-поиска это чрезмерное количество данных. Скачать все даже с одного ресурса крайне ресурсоемкая задача, и даже не всегда выполнимая - т.к. как многие документы создаются динамически, и возможно “зацикливание”.

Не смотря на то, что практически каждый ресурс содержит информацию по сборке (robots.txt), не редко огромное количество совершенно бесполезных ссылок и дубликатов не попадает под фильтры, отсюда вытекает необходимость контроля хода сборки в соответствии с дополнительной информацией о url.

Глава 1

Обзор средств и подходов для оптимизации сборки nutch

Плагины nutch

Система плагинов nutch подобна плагинам к Eclipse. Основная логика nutch описывается плагинами. Все что относится к разбору документов, индексации и поиску реализовано плагины. Подгружаемый модуль nutch предоставляет одно или более расширение так называемых точек расширения (extension-points). Один подгружаемый модуль может относиться к нескольким точкам расширения, так же как несколько подгружаемых моуелй могут относиться к одной и той же точке расширения. Каждая точка расширения предоставляет интерфейс, который необходимо реализовать в плагине. Для каждого плагина используется собственный загрузчик классов (classloader), который в нужный момент загружает плагин в систему.[2]

Ранжирование

Поскольку в момент поступления url в систему мы сразу можем определить её полезность, возникает желание сначала скачивать именно “полезные” url. База ссылок nutch устроена таким образом что с каждым url связан ранг, который образуется с помощью плагина nutch “scoring filter”. Достаточно написать свой плагин, который бы увеличивал ранг полезных url. Поскольку плагин контролирует ранг ссылки на каждой стадии сборки, нет необходимости изменять код ядра nutch.

Ранжирование в nutch

Как уже говорилось, ранжирование в nutch основано на плагинах. Плагин ранжирования является расширением точки “ScoringFilter” и реализует соответствующий интерфейс. ScoringFilter руководит определением ранга ссылки в момент попадания её в систему, модифицирует его после скачивания и разбора документа. Так же ScoringFilter отвечает непосредственно за генерацию “sort value” в момент выборки ссылок для скачивания.

Все плагины ранжирования организованы в цепь таким образом что полученный “sort value” одного плагина попадает на вход другого. В дальнейшем на основе “score value” формируется список скачиваемых страниц, при этом страницы с более высоким “score value” попадают в него раньше, чем страницы с меньшим (за создание списка отвечает набор работ generate в nutch). Порядок плагинов и количество url в списке определяется пользователем.[4]

Информация о полезности url

Необходимо выбрать способ, которым по url будет определяться её полезность. Поскольку можно использовать несколько плагинов в связке, достаточно определить полезность посредством задания ранга для “полезных” url, и для обычных, а различные промежуточные значения переложить на другие плагины.

Варианты задания полезных url:

1. при помощи задания префикса
2. при помощи задания регулярного выражения

Хотя первый вариант будет работать быстрее и более прост, он не дает достаточного контроля, и высокий ранг будут получать лишние url, поэтому было решено использовать регулярные выражения.

Самый простой способ хранения регулярных выражений — это добавить их в виде файла ресурса в конфигурацию nutch, но таким способом нельзя обновлять и изменять его в процессе сборки. Для этих целей было решено хранить все регулярные выражения в базе данных.

Генерация черных списков

Основная задача черных списков — уменьшение размера базы ссылок — crawldb.

Crawldb

Одна из основных составляющих nutch — база ссылок crawldb, которая представляет из себя записи вида $\langle url, crawldatum \rangle$, где $crawldatum$ это дополнительная информация об url, такая как:

- текущее состояние (например оно показывает была ли ссылка скачана, или является ли данная ссылка редиректом на другую страницу)
- время когда документ по ссылке был в последний раз скачан
- время когда документ была в последний раз обновлен
- сколько раз система пыталась скачать данный документ
- как часто надо обновлять документ
- дополнительные произвольные данные

Crawldb используется во многих задачах, время которое тратится на обработку crawldb линейно зависит от её размера. Время на выполнения всего цикла зависит от размера crawldb, и количества ссылок выбираемых для скачки. Хотя само время скачивания очень сильно зависит от канала и доменов с которых идет скачка, общее время можно представить в виде:

$$t_c = n_c * c_1 + n_g * c_2$$

Где n_c - число записей crawldb, n_g - число выбираемых ссылок, а t_c общее время цикла. В реальных условиях $c_1/c_2 \approx 0.0013$, таким образом при $n_c = 1000000$ и $n_g = 20000$ на работу с crawldb тратится порядка 30% времени.

Фильтры

Фильтрация ссылок в nutch реализована через плагины с точкой расширения URLFilter. Фильтры используются в момент добавления ссылки в базу ссылок и во время её обновления. В nutch реализовано несколько плагинов для фильтрации:

- PrefixURLFilter — фильтрация по префиксу url. Служит, как правило, для ограничения сборки определенными доменами.
- RegexURLFilter — фильтрация по суффиксу url. Используются для ограничения форматов файлов.

- SuffixURLFilter — фильтрация по регулярному выражению.

Каждый из плагинов допускает “пропускающие” и “исключающие” правила, которые берутся из файлов конфигурации. Так как система должна работать непрерывно, а фильтры должны изменятся в ходе работы, необходимо изменить плагины работающие с префиксами и регулярными выражениями для работы с базой данных. Поскольку суффиксные фильтры нацелены на то, что бы отлавливать не поддерживающиеся форматы данных (.jpeg .css .zip) их можно не изменять.

Создание фильтров

Фильтры ограничивающие домены и форматы тривиально создаются из списка доменов и расширений поддерживаемых форматов. Это дает самое общее ограничение области ссылок, при этом остается еще большое число ссылок не являющихся “полезными”. Например если мы хотим скачивать только новости то нам совершенно не интересно знать что на в том же домене располагается форум, или какие-либо статьи. Так же часто присутствует большое число технических ссылок, например некоторые сайты делают внешние ссылки как редиректы с собственного домена и.т.п.

Основную сложность для ручного создания фильтров представляют как раз технические разделы сайта, да и оценить в ручную размер любой другой части сайта не просто. Для того что бы эффективно находить подобные разделы необходима обратная связь от nutch в процессе работы. Так же возникает желание автоматически создавать фильтры.

Анализ crawldb

Для анализа crawldb в nutch используется утилита readdb которая умеет получать статистику по базе. Наибольший интерес представляет такой предоставляемый ею набор метрик, как число ссылок определенного статуса, разбитые по доменам. Что бы стало возможным нахождение бесполезных разделов, необходимо реализовать возможность получения статистики не только по доменам, но и по крупным их частям. К сожалению это не возможно эффективно сделать лишь за счет плагинов — необходимо изменять код ядра nutch. Так же, в отличии от стандартной утилиты, статистика должна поступать не на stdout а во внешнюю базу данных, для простоты дальнейшего анализа.

Автоматическая генерация фильтров

Основная сложность в автоматической генерации — создать достаточно эффективный метод с низкой вероятностью ошибок первого рода (отклонение разделов с полезной информации). Проблема заключается в том, что необходимо оставить не только все “полезные ссылки”, но и те страницы без которых не все “полезные” ссылки достижимы из корневой страницы домена.

Вернемся к примеру с lenta.ru. Все новости находятся по ссылкам вида 1.1, а архив новостей, без которого мы не сможем получить все новости, находится в разделе 1.2 который не должен попасть под фильтры.

$$http://lenta\backslash.ru/news/\backslash d\{4\}/\backslash d\{2\}/\backslash d\{2\}/\backslash w+/ \tag{1.1}$$

$$http://lenta\backslash.ru/\backslash d\{4\}/\backslash d\{2\}/\backslash d\{2\}/ \tag{1.2}$$

Формальные требования к автоматической генерации:

- ни одна “полезная” ссылка не должна попадать под фильтры
- после применения фильтров все “полезные” ссылки должны быть достижимы из корневой ссылки домена
- значительно число остальных ссылок должно попасть под фильтры

Т.к фильтры и статистика хранятся в базе данных, генератор фильтров может выступать как отдельное приложение.

Методы оценки эффективности

Под эффективностью сборки далее будет пониматься отношение n_i/t_i, где n_i — число скаченных ссылок за итерацию i, а t_i — время итерации. Стоит заметить, что если для ранних итерациях основное время используется для обработки сегмента, то на поздних большую часть времени система занимается обработкой базы ссылок (crawldb) и базой обратных ссылок (linkdb).

Глава 2

Описание алгоритмов и реализации

Повышение эффективности сборки, учитывая признак “полезности” при ранжировании

Алгоритм

В момент поступления url в систему, в случае её принятия хотя бы одним регулярным выражением, необходимо установить начальный ранг c_f.

Реализация

Данная функциональность была реализовано при помощи двух плагинов. Первый проверяет является ли url “полезной”, и добавляет необходимую информацию в *crawldatum*, а второй выставляет ранг согласно *crawldatum* и настроек приложения. Такое разбиение было сделано потому что информация о том, что url является “полезной”, представляет собой интерес и без использования данного ранжирования (например это было использовано при получении статистики).

SkaiScoringMeta

SkaiScoringMeta — плагин являющийся расширением к точке ScoringFilter, выделяющий “полезные” url. Для хранения дополнительной информации в *crawldatum* используется поле $metaData$, которое представляет из себя набор пар вида $\langle key, value \rangle$ Для сохранения индикатора “полезности” в мета данные *crawldatum* добавляется пара $\langle skai.scoring.fit, true \rangle$.

Для определения “полезности” url используется кэш с регулярными выражениями, загружаемыми из базы данных при инициализации.

SkaiScoring

SkaiScoring — плагин являющийся расширением к точке ScoringFilter, выставляющий ранг. Для ссылок только что попавших в систему, зависимости от наличия мета тега, выставляет значения ранга либо c_f, либо c_n, где коэффициенты c_f и c_n получаются из стандартной конфигурации nutch.

Результат

При использовании стандартных ScoringFilter'ов в среднем за цикл, из всех скачиваемых ссылок “полезных” скачивалось порядка 10% (В качестве пространса поиска использовалось 40 крупнейших СМИ, на каждой итерации выбиралось 20000 ссылок, “полезными” признавались непосредственно новости). А при использовании данных плагинов уже на ранней стадии сборки порядка 80% ссылок были “полезными” (Рис. 2.1). Таким образом было получено увеличение эффективности в восемь раз по сравнению с базовой реализацией nutch.

Повышение эффективности сборки, отсечением страниц, не содержащих значимой информации

Реализация фильтров

Загрузка фильтров из базы данных является достаточно тривиальной задачей — во время конфигурации плагинов все фильтры загружаются не только из файлов (оставлено для совместимости с предыдущими версиями), но и из базы данных. Префиксные фильтры создаются непосредственно из описания домена, а регулярные выражения представляют из себя записи в базе данных.

Проблема с www

В ходе тестирования возникла проблема — большинство ресурсов по url вида 2.1 и 2.2 содержат один и тот же документ, как правило отличающийся тем что в первом

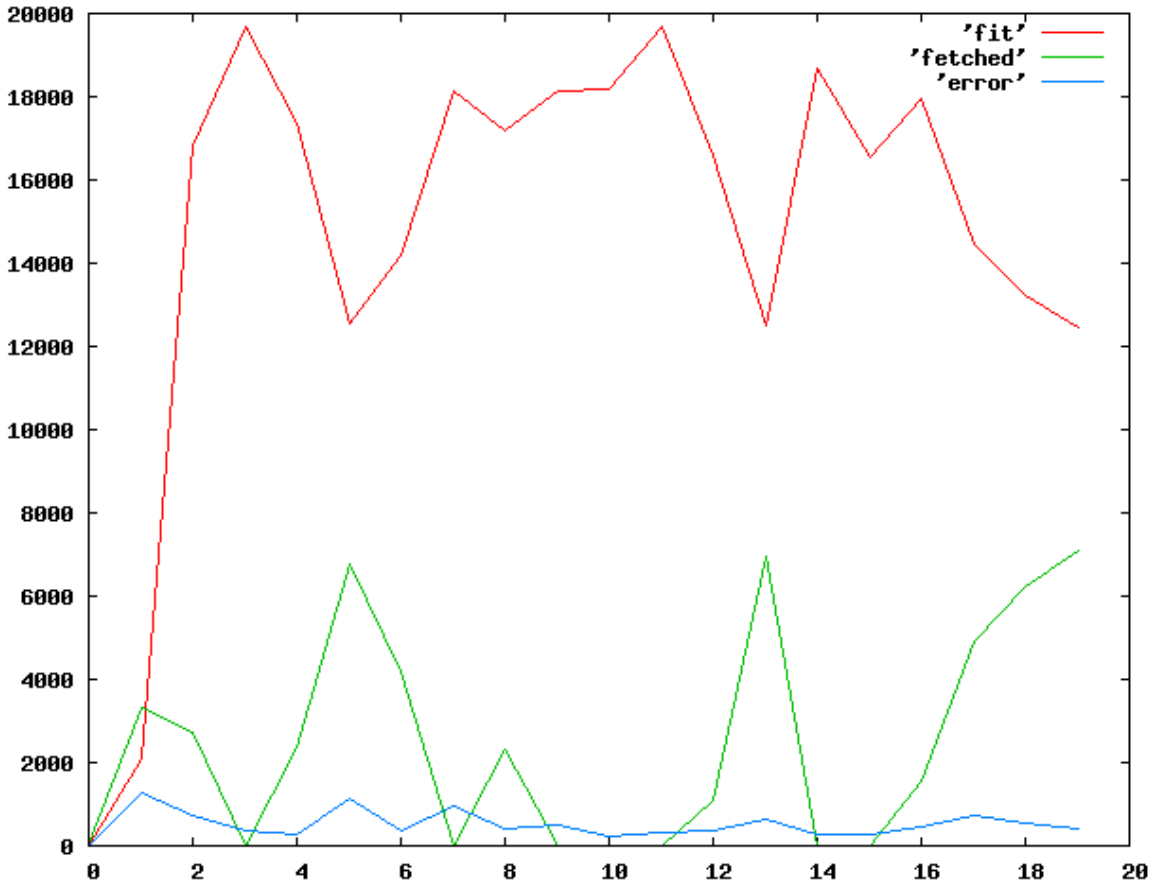

Рис. 2.1: График зависимости числа ссылок от номера итерации. fit—число полезных ссылок, fetched—число скачанных ссылок не являющихся полезными, error—число ошибок при скачивании

документе все исходящие url содержат www, а во втором нет.

$$http://domainname/documentname \tag{2.1}$$

$$http://www.domainname/documentname \tag{2.2}$$

Что бы избежать дублирования документов, разумно фильтровать либо ссылки с www, либо без www.

Однако бывают ресурсы в которых из документа с url вида 2.1 присутствуют как ссылки с www так и без, таким образом часть нужных ссылок выбрасывается фильтрами. Для решения подобной проблемы пришлось модифицировать нормализатор url.

Нормализатор url

Нормализатор url (url normalizer) — точка расширения nutch (URLNormalizer), которая отвечает за преобразование url на различных стадиях работы. В nutch реализовано несколько нормализаторов, которые подобно ScoreFilter'ам обьеденены в цепь. Так же URLNormalizer предоставляет возможность использовать разные правила преобразования на разных этапах сборки.

Для решения возникшей проблемы за основу был взят стандартный RegexUrlNormalizer, который осуществляет преобразование url по средствам регулярных выражений. Было добавлено создание правил заменяющих url с www на url без www, если домен был помечен как "без www" и наоборот в противном случае.

Получение статистики

Основная идея заключается в том, что бы получить записи вида $\langle domain, prefix, metrics \rangle$ где:

- domain — домен к которому относится запись.
- prefix — префикс url к которому относится запись.
- metrics — набор пар вида $\langle state, count \rangle$, где *count* — это число документов с данным префиксом на данном домене, в состоянии *state* (например: новая ссылка, скачанная ссылка, "полезная ссылка").

По подобным записям в дальнейшем достаточно легко делать выводы о разделах ресурсов.

Алгоритм

Поскольку данная задача работает с crawldb, размер которой может быть очень большим, алгоритм необходимо представить в виде MapReduce задачи.

Map

На этапе map по *url* и *crawldatum* получается множество пар $\langle prefix, info \rangle +$, где $prefix$ это префикс url вместе с доменом, а $info$ это пара вида $\langle state, 1 \rangle$

$$\langle url, crawldatum \rangle \rightarrow \langle prefix, info \rangle +$$

Сначала из url неким способом получается множество префиксов S_p, например:

$$http://lenta.ru/2010/05/09/ \rightarrow$$

$$http://lenta.ru/2010/05/09,$$

$$http://lenta.ru/2010/05/,$$

$$http://lenta.ru/2010/,$$

$$http://lenta.ru/;$$

Далее в зависимости от свойств url описанных в $crawldatum$ создается множество метрик S_m, например если ссылка была "полезной" но не была еще скачана, то

$$S_m = \{\langle fit, 1\rangle, \langle unfetched, 1\rangle\}$$

Затем все пары из $S_p \times S_m$ попадают в выходной поток. Для эффективной работы необходимо, что бы url разбивалось на наименьшее число префиксов, но так, что бы основные разделы домена могли быть сопоставлены какому-нибудь префиксу. В идеале множество метрик может содержать только одно значение, если url не может находится сразу в нескольких состояниях.

Reduce

На этапе reduce метрики префиксов суммируются и отбрасываются незначительные префиксы

$$\langle prefix, \langle state, 1\rangle+\rangle \rightarrow \langle prefix, metricvector\rangle$$

$Metricvector$ представляет из себя вектор из пар $\langle state, nurl\rangle$, где $nurl$ — число url с состоянием $state$. Незначительными признаются метрики где сумма $nurl$ достаточно мала. Такие префиксы отбрасываются, а остальные сохраняются в базу данных.

Реализация

В ходе реализации был изменен класс CrawlDbReader, отвечающий за получение статистики. Написаны классы для map и reduce.

CrawlDbExtendedStatMapper

CrawlDbExtendedStatMapper — класс имплементирующий стандартный интерфейс Hadoop — Mapper, который служит для создания собственных этапов map. Для разбиения url на префиксы был использован UrlSplitter, который разбивал url на не более чем n_s префиксов по символу "/", так же как и в примере с lenta.ru. Таким образом $|S_p| \leqslant n_s$ Пожалуй это самая простая реализация которая требует доработки в будущем. В качестве метрик использовались следующие показатели:

- unfetched —документ по ссылке не был скачан.
- fetched — документ по ссылке была скачана.
- fit — ссылка была признана "полезной"

Первые два показателя брались непосредственно из статуса *crawldatum*, а последняя из мета данных, которые были проставлены с помощью SkaiScoringMeta. Таким образом $|S_m| \leqslant 2$. В результате число записей попадающих в выходной поток $n \leqslant 2 \cdot n_s \cdot n_{url}$, где n_{url} — число записей в crawldb.

CrawlDbExtendedStatReducer

CrawlDbExtendedStatReducer — класс имплементирующий стандартный интерфейс Hadoop — Reducer, который служит для создания собственных этапов reduce.

На этапе reduce суммируются все показатели для префикса и если $n_{unfetched} + n_{fetched} < n_s$ префикс отбрасывается. Для совместимости со стандартным CrawlDbReader'ом и уменьшения нагрузки на базу данных, префиксы не сразу добавляются в базу данных, а выводятся в файл.

Добавление в базу данных

После отработки MapReduce работы, её результат считывается из файла, после чего все записи добавляются в базу данных одним запросом. Так же в базу передается время в которое статистика была создана.

Получать статистику можно достаточно редко — например один раз в 5-20 циклов, таким образом, время на сбор статистики не оказывает существенного влияния на эффективность.

Результаты

Для crawldb с 10 000 000 url время получения статистики составляет порядка 15% от общего времени работы цикла с 20 000 ссылок. Таким образом, при создании статистики каждый двадцатый цикл, потеря времени составляет порядка 0.75%, которой можно пренебречь.

Генерация фильтров

При создании алгоритма были сделаны следующие предположения:

- все "полезные" можно ограничить некотором числом разделов, не зависящем от числа документов. (под разделом понимается некий префикс url)
- архив документов находится в определенном разделе (под архивом понимаются документы с ссылками на "полезные" документы)
- архив достижим из корневого документа
- документов архива меньше чем "полезных"

Алгоритм

Сам алгоритм, пользуясь некоторой эвристикой, признает некоторые префиксы ненужными, из которых потом создаются фильтры. Простейший алгоритм выбирающий префиксы согласно сделанным предположениям выглядит так:

1. Выбирается вся статистика для конкретного домена.
2. Домены для которых $u_d < n_d$ далее не рассматриваются.
   - u_d — число "полезных" ссылок во всем домене
   - n_d — пороговое значение, введенное для того, что бы не начать создавать фильтры до того, как будут получены ссылки на архив.
3. По префиксам для которых $u_p = 0$ и $s_p > u_d$ создаются исключающие правила.
   - u_p — число "полезных" ссылок с данным префиксом
   - s_p — общее число известных ссылок с данным префиксом

   Сам по себе префикс уже представляет из себя префиксный фильтр.

Реализация

Данная функциональность была реализована в виде отдельного приложения запускающегося сразу после создания статистики. Поскольку пользовательские фильтры были организованы на базе RegexURLFilter, для простоты управления автоматически созданные фильтры были тоже реализованы регулярными выражениями.

Была предусмотрена возможность отключения конкретного фильтра таким образом, что бы он не создавался заново. Для этого фильтр не удалялся из базы данных, а просто становился не активен.

Результат

Качество результатов работы данного алгоритма достаточно сложно оценить, однако на тестовых данных автоматические фильтры достаточно успешно находили бесполезные разделы. Предполагается что существенное увеличение производительности будет получено на позднем этапе сборки - когда большая часть “полезных” документов уже будет скачена.

Заключение

В рамках данной работы было выполнено прикладное исследование системы интернет поиска nutch. В качестве изначальной цели работы была поставлена оптимизация работы nutch, с учетом дополнительной информации о содержании ссылок.

В ходе работы была изучена архитектура nutch. На основе этой информации были найдены основные направления для оптимизации работы:

- изменение ранжирования
- автоматическое создание фильтров url

Были написаны необходимые расширения nutch и дополнительное приложение для генерации фильтров на языке Java. В ходе разработки были использованы следующие технологии: hadoop, spring, ibatis, nutch plugin.

Были произведены эксперименты и получены количественные данные, характеризующие эффективность нового ранжирования. Было показано что даже простое изменение ранжирования с учетом дополнительной информации может на порядок увеличить производительность.

Было показано, что разработанные алгоритмы в совокупности позволяют достичь высокой эффективности как на ранних стадиях сборки (ранжирование), так и на поздних (генерация фильтров).

Литература

[1] Пименов Александр, Hadoop Nutch и Lucene v3, 2009.

[2] D Cutting, Nutch: an Open-Source Platform for Web Search

[3] E Hatcher, O Gospodnetic, Lucene in action

[4] R Khare, D Cutting, K Sitaker, A Rifkin, Nutch: A flexible and scalable open-source web search engine

[5] T White, Hadoop: The Definitive Guide