Санкт-Петербургский государственный университет

Математико-механический факультет

Кафедра системного программирования

Архитектурные аспекты решения задачи фильтрации документов на потоке запросов

Курсовая работа студента 445 группы

Федотовского Павла Валерьевича

Научные руководители:

Чернышев Г.А.,

ассистент кафедры информатики СПбГУ

Смирнов К.К.,

Инженер ЗАО “Ланит-Терком”

Санкт-Петербург

2013

Оглавление

Введение

При разработке высокопроизводительных приложений встает задача о распределении заданий между исполнительными узлами системы. При этом крайне желательно избежать ситуации, при которой часть узлов оказывается перегруженной заданиями, а другая часть простаивает. Решение этой задачи включает в себя выбор архитектуры приложения, принципы разделения и распределения задач между вычислительными узлами, выбор фреймворка и многое другое. Также нужно учитывать специфику той конкретной области, в рамках которой разрабатывается приложение.

Задача нечеткого поиска формулируется как нахождение всех строк из данного набора, близких к заданной строке по заданной метрике. В качестве метрики может выступать расстояние Левенштейна, расстояние Хэмминга или точное совпадение.

Нечеткий поиск является важной подзадачей во многих современных системах. Он применяется в таких областях как информационный поиск, биоинформатика, проверка орфографии, и т. д. В информационном поиске он полезен при исправлении опечаток в поисковых запросах. То есть, если пользователь ошибся в написании запроса, то поисковая система предложит правильный вариант. Или пользователь может не знать правильного написания термина и ввести его с ошибками. Это возможно благодаря тому, что поисковая система “знает”, что имелось в виду. Она будет искать правильное написание слов, ориентируясь на словарь. Если расстояние до слова из словаря невелико, то можно предполагать, что пользователь ошибся и предложить ему правильный вариант. Такие алгоритмы применяются в таких поисковых системах как Google, Yandex и др. В биоинформатике нечеткий поиск применяется для различных операций с последовательностями ДНК [9].

На данный момент существует множество подходов к решению задачи нечеткого поиска. Разработаны метрики для сравнения строк, алгоритмы, способные быстро вычислять метрику или проверять соответствие двух слов друг другу по заданной метрике и допустимому отклонению. Среди них можно отметить bitap [1], trie [4], различные метрические деревья – M-tree [3], BK-tree [5]

Несмотря на то, что предметная область, связанная с нечетким поиском, достаточно стара, она до сих пор актуальна. Активно публикуются статьи, проводятся различные соревнования прототипов систем, включающих нечеткий поиск. Например,

EDBT 2013 String Similarity Search/Join Competition[1], ACM SIGMOD Programming Contest 2013[2]. В них ставится определенная задача и дается время на изучение алгоритмов и реализацию (порядка нескольких месяцев).

Данная работа выполнялась в рамках соревнования ACM SIGMOD Programming Contest 2013 [3] . Полный состав команды ‘RotaFortunae’ (колесо фортуны, лат.): Федотовский П.В., Ерохин Г.А., Чередник К.Е.

[1] http://www2.informatik.hu-berlin.de/~wandelt/searchjoincompetition2013 [Дата просмотра 26.05.13]
[2] http://sigmod.kaust.edu.sa/index.php [Дата просмотра 26.05.13]
[3] http://sigmod.kaust.edu.sa/task-details.php [Дата просмотра 26.05.13]

Общий обзор системы

На вход системе поступает поток документов и запросов. Цель – для каждого документа найти все запросы, которые ему удовлетворяют. В качестве примера можно рассмотреть твиттер. Документы в данном случае это твиты, которые публикуются пользователями. Запросы – это хэштеги, на которые пользователь хочет подписаться. Хэштег – это обычное слово, если оно встречается в твите, то все пользователи, которые на него были подписаны, увидят этот твит в своей новостной ленте.

Другим примером могут быть новостные рассылки, пользователи указывают ключевые слова и, если новый документ их содержит, то его увидят все заинтересованные пользователи.

Более формально, нужно определить понятия документа и запроса. Документ – это множество слов, ограничения на их количество нет. Однако в задаче контеста длина документов была ограничена сверху – максимальное количество слов было около миллиона. Запрос – небольшой набор слов плюс используемая метрика и допустимое отклонение.

Запрос считается подходящим документу, если все слова из запроса имеют соответствие в документе. То есть, в документе имеется слово, расстояние до которого от слова в запросе не превосходит данного отклонения по данной метрике.

Итак, общая постановка задачи следующая: по каждому документу определить все запросы (на выход подать их идентификаторы), которые ему подходят. Основной особенностью системы является наличие нечеткого поиска при поиске соответствия. Требовалось разработать разделяемую библиотеку, реализующую предоставленный орган изаторами API, описание которого приведено в Приложении 1.

.

Постановка задачи

В рамках проекта по реализации рассмотренной системы, передо мной были поставлены следующие задачи:

- обзор подходов и выбор общей архитектуры системы, описанной выше;
- реализация основных аспектов архитектуры.

Выбор архитектуры

Поскольку система, описанная выше, позволяет выполнять задачи параллельно, необходимо придумать эффективный алгоритм для распределения задач по потокам. В пользу многопоточности говорит еще и то, что целевая платформа (предоставленная организаторами) имеет 24 логических ядра. Существуют различные подходы к распараллеливанию: Round-Robin [7], Work Stealing [2], Common Queue [6], Thread Pool [8]. В каждом из них имеется главная нитка (master) и подчиненные нити, выполняющие задачи (slave). Главная нить оперирует непрерывно поступающим потоком задач, которые она должна распределять между подчиненными нитками. Опишем алгоритмы подробнее:

Round-Robin: Схема алгоритма представлена на рис. 1. Главная нить распределяет задачи по очереди между каждой подчиненной нитью. То есть, первая задача будет отдана первой нити, вторая - второй и так далее. Плюсом является отсутствие блокировок, главная нить не простаивает при распределении задач. Также этот алгоритм относительно просто в реализации. Недостатком этого алгоритма является неравномерное распределение задач. Возможен вариант, при котором одной из нитей достанутся затратные по времени задачи, и, когда нужно будет синхронизировать результаты работы, все потоки будут находиться в ожидании этой нити. Также некоторые нити могут быстро выполнить свои задачи и будут вынуждены ждать поступления очередной задачи.

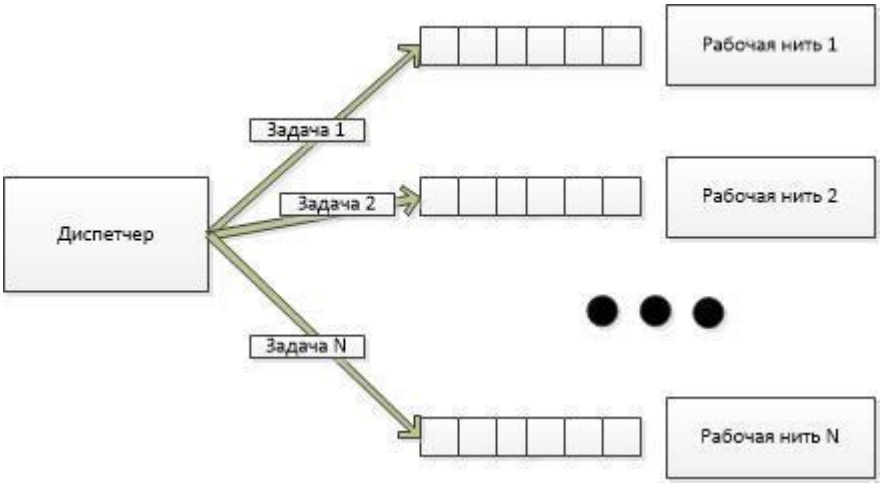

Рис. 1 Round-Robin.

Work-Stealing: Этот алгоритм является усовершенствованной версией предыдущего. Во избежание простоев, нить, которая закончила все свои задачи, имеет возможность “украсть” задачу у другой нити. Таким образом, обеспечивается более равномерное распределение задач. Однако, данный подход не так прост в реализации, как предыдущий, и требует синхронизации при “краже” задач.

Common Queue: Схема алгоритма представлена на рис. 2. Хранится общая очередь задач, главная нить помещает в нее новые задания по мере их поступления. Рабочие нити постепенно выбирают из нее задания. Таким образом нити не простаивают, однако требуется синхронизация доступа к общей очереди заданий. Данный алгоритм достаточно прост в реализации, что является плюсом.

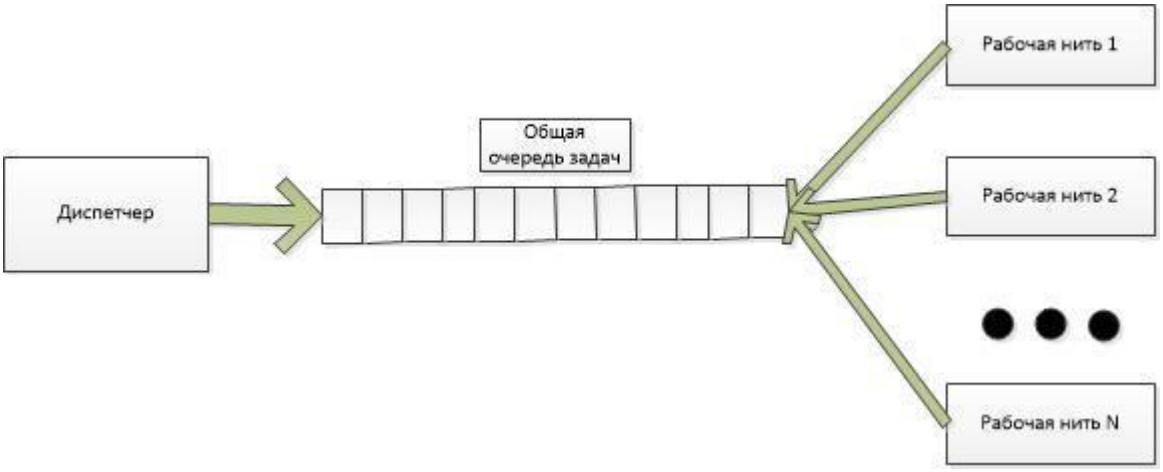

Рис. 2 Common Queue.

Thread Pool: Схема алгоритма представлена на рис. 3. В данном алгоритме хранится очередь из нитей, готовых к выполнению работы. При поступлении новой задачи главная нить берет первую свободную рабочую нить и отдает ей текущую задачу. Соответственно, после выполнения работы, рабочая нить добавляет себя в эту очередь. Этот подход также довольно просто в реализации. Минусом алгоритма является синхронизация доступа к очереди свободных ниток.

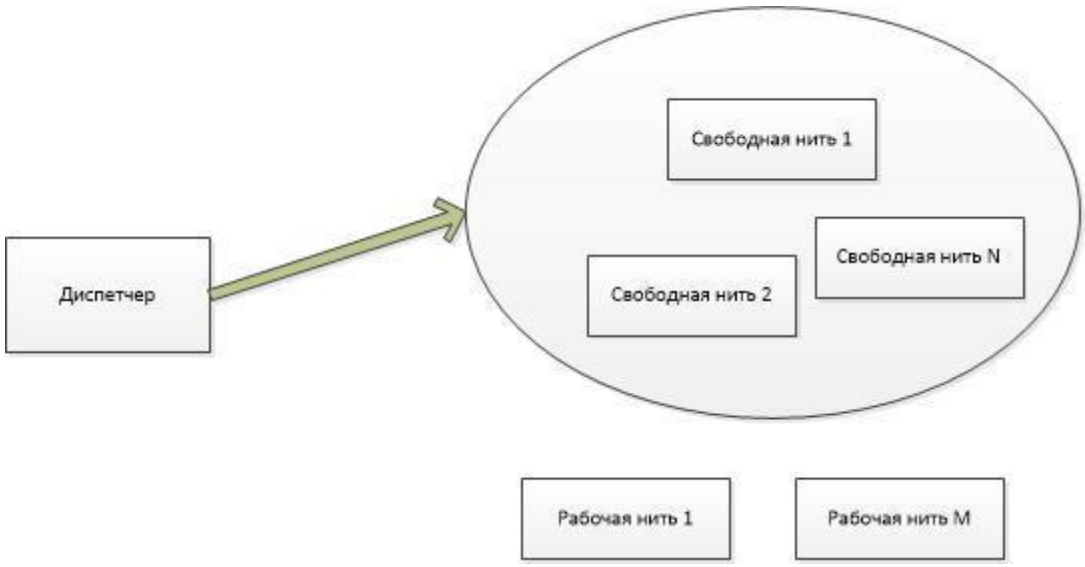

Рис. 3 Thread Pool.

Был выбран последний алгоритм как относительно простой в реализации и обеспечивающий отсутствие простоев нитей. Для синхронизации доступа к очереди свободных ниток используется мьютекс, гарантирующий то, что доступ в каждый момент времени имеет только одна нить.

Выбор задач для потоков

При реализации системы, удовлетворяющей требованиям, описанным выше, встает вопрос о выборе задач для рабочих нитей. Для оптимального распределения нагрузки нужно выбрать задачи, которые будут выполняться параллельно. Это имеет значение, поскольку при неравномерной сложности задач может случиться так, что нити при синхронизации будут находиться в ожидании нити, которая выполняет затратную по времени задачу. Соответственно, задачи должны быть примерно эквивалентны по сложности.

Постановка задачи также включает в себя тот факт, что множество слов повторяется как в документах, так и в запросах. Поэтому разумно строить архитектуру с учетом этого факта.

Было выбрано два варианта, кратко которые можно описать как “нить/документ” и “нить/новое уникальное слово”. Рассмотрим каждый из них подробнее:

“Нить/документ”: В данном подходе каждой нити дается на обработку целый документ. Весь алгоритм состоит в следующем: при поступлении нового документа на проверку, он отдается рабочей нити, которая строит для него ответ. Для этого осуществляется проход по всем словам из всех запросов и для каждого из них проверяется соответствие в документе. То есть, для конкретного слова из запроса выполняется проход по всем словам из данного документа и определяется, имеется ли соответствие. Соответственно, если в конкретном запросе подходят все слова с учетом используемой метрики и расстояния, то его идентификатор добавляется в результирующее множество. Этот алгоритм прост в реализации, однако имеет ряд существенных недостатков. Во-первых, время проверки одного документа напрямую зависит от его длины. Поэтому высока вероятность простоя нитей при синхронизации из-за ожидания нитки, которая обрабатывает длинный документ. Во-вторых, никак не используется результат предыдущей проверки одних и тех же слов. В-третьих, затруднительно строить дополнительные структуры для ускорения проверок слов. Поэтому была выбран другой подход.

“Нить/новое уникальное слово”: В данном подходе используется то, что многие слова повторяются как в запросах, так и в документах. Для каждого уникального

слова из документа хранится список слов из запросов, которые ему соответствуют. Эта структура постоянно обновляется при поступлении новых запросов и документов. Запросы хранятся как списки идентификаторов уникальных слов. Рассмотрим реализацию основных функций:

- *StartQuery(), EndQuery():* При поступлении нового запроса нужно выявить в нем слова, которые еще не встречались ранее и для них найти слова из документов, известные на данный момент и обновить структуру. То есть, если находится слово из документа, подходящее под данное слово из запроса, в его список подходящих под него слов из запросов заносится данное слово из запроса. Аналогичные действия происходят при удалении.
- *MatchDocument():* Документ обрабатывается и выявляются слова, еще не встречавшиеся ранее. Для каждого такого слова находятся слова из запросов, ему соответствующие. После этого новое слово запоминается для использования результатов проверок в будущем. После обработки документа мы знаем все слова из запросов, которые подходят под данный конкретный документ. Остается лишь пройтись по всем запросам и занести в результат нужные.
- *GetNextAvailRes():* Выдается первый доступный результат, при необходимости происходит синхронизация для завершения текущий задач по проверке слов.

Задачи, которые отдаются на выполнение потокам в данном подходе почти эквивалентны по сложности, поэтому вероятность возникновения неравномерного распределения задач гораздо меньше, чем в первом алгоритме. Также существенно меньше количество вызовов дорогих функций для проверки расстояния между словами. Это достигается за счет запоминания результатов проверок и их переиспользования.

Были реализованы оба этих алгоритма, второй оказался в несколько раз быстрее на целевой платформе, предоставленной организаторами. Он был включен в финальную версию прототипа разрабатываемой системы, который на данный момент занимает второе место на публичных тестах[4].

[4] http://sigmod.kaust.edu.sa/leaderboard.php [Дата просмотра 26.05.13]

Результаты

В данной работе были изучены различные подходы к выбору архитектуры системы фильтрации документов на потоке запросов. Были выбраны наиболее подходящие, и они были реализованы в рамках соревнования ACM SIGMOD 2013 Programming Contest[5]. Был разработан прототип системы, реализующий поставленную в данном соревнованию задачу. Команда RotaFortunae вошла в число 5-ти команд-финалистов [6] и будет представлять свой алгоритм на конференции ACM SIGMOD 2013[7].

[5] http://sigmod.kaust.edu.sa/index.php
[6] http://sigmod.kaust.edu.sa/finalists.php [Дата просмотра 26.05.2013]
[7] http://www.sigmod.org/2013/ [Дата просмотра 26.05.2013]

Литература

1 Bálint Dömölki, An algorithm for syntactical analysis, Computational Linguistics 3, Hungarian Academy of Science pp. 29–46, 1964.

2 Blumofe R., Leiserson C. Scheduling Multithreaded Computations by Work Stealing // Proceedings of the 35th Annual IEEE Conference on Foundations of Computer Science (FOCS'94). Santa Fe. 1994.

3 Ciaccia, Paolo; Patella, Marco; Zezula, Pavel. "M-tree An Efficient Access Method for Similarity Search in Metric Spaces". *Proceedings of the 23rd VLDB Conference Athens, Greece, 1997*. IBM Almaden Research Center: Very Large Databases Endowment Inc. pp. 426–435. p426. Retrieved 2010-09-07.

4 Gonzalo Navarro. A guided tour to approximate string matching. *ACM Comput. Surv.* 33, 1 (March 2001), pp. 31-88.

5 W. A. Burkhard and R. M. Keller. Some approaches to best-match file searching. *Commun. ACM* 16, 4 (April 1973), pp. 230-236.

6 Message Queue // Википедия. [2004-2013]. Дата обновления: 12.05.2013. URL: https://en.wikipedia.org/wiki/Message_queue (дата обращения: 26.05.13)

7 Round-robin scheduling // Википедия. [2011-2013]. Дата обновления: 23.03.2013. URL: https://en.wikipedia.org/wiki/Round-robin_scheduling (дата обращения: 26.05.13)

8 Thread pool pattern // Википедия. [2004-2013]. Дата обновления: 11.03.2013. URL: https://en.wikipedia.org/wiki/Thread_pool_pattern (дата обращения: 26.05.13)

9 Гасфилд Д. Строки, деревья и последовательности в алгоритмах: Информатика и вычислительная биология. СПб.: Невский Диалект; БХВ-Петербург, 2003. 654 с.

Приложение 1. Описание API

Формальная постановка задачи следующая: реализовать систему, которая реализует следующие 4 функции:

- *StartQuery();*
- *EndQuery();*
- *MatchDocument();*
- *GetNextAvailRes().*

Рассмотрим каждую из них подробнее:

StartQuery() - добавить запрос к множеству активных запросов. Запрос представляет собой набор строк длиной от 4 до 31 символа, используемую метрику и максимально допустимое отклонение в данной метрике. Количество слов в запросе не превышает пяти.

EndQuery() - удалить запрос из множества активных запросов.

MatchDocument() - добавить документ на проверку, где документ представляет собой набор слов длиной от 4 до 31 символа, а общее количество символов в документе (учитывая разделительные пробелы) не превышает миллиона. Проверка же представляет собой построение списка запросов из числа активных на момент добавления документа, для которых каждое слово имеет близкое к нему (в соответствующей метрике и с соответствующим пороговым значением) среди слов данного документа.

GetNextAvailRes() - получить результат (если таковой имеется) для какого-либо документа, данного системе. Эта функция позволяет проверять документы параллельно, так как не требуется выдавать ответ сразу по окончанию проверки текущего документа. При вызове этой функции нужна синхронизировать все выполняющиеся в данный момент задачи и вернуть первый готовый ответ.

Система должна поддерживать следующие метрики.

- 0-1 метрика [9]: то есть точное сравнение двух строк.
- Метрика Хэмминга [9]: определяется только для строк одинаковой длины. Она равна количеству несовпадающих символов в соответствующих позициях.

- Метрика Левенштейна (редакционное расстояние) [9]: минимальное количество операций вставки, удаления и замены символов, необходимое для преобразования одной строки в другую.