Санкт-Петербургский Государственный Университет

Математико-механический факультет

Кафедра системного программирования

Система кластеризации мульти-язычных данных большого объема

Дипломная работа студентки 545 группы

Нишневич Анастасии Юрьевны

Научный руководитель	..... / подпись /	к.х.н., Изъюров А.Л.
Рецензент	..... / подпись /	к.ф.-м.н.,Шалымов Д.С.
“Допустить к защите” заведующий кафедрой,	..... / подпись /	д.ф.-м.н., проф. Терехов А.Н.

Санкт-Петербург

2012

Saint-Petersburg State University
Mathematics and Mechanics Faculty
Software Engineering Department

Large Amount Multilanguage Clusterization System

Graduate paper by
Anastasia Nishnevich
545 group

Supervisor	.....	PhD A.L. Izyurov
Reviewer	.....	PhD D.S. Shalymov
"Approved by" Head of Department	.....	Professor A. N. Terekhov

Saint-Petersburg
2012

Оглавление

Глава 1

Введение

В современном мире информация имеет огромную ценность. Интернет является всемирным хранилищем информации. Однако, в сеть попадает огромное количество неполных и дублирующих друг друга данных.

Поисковые компании получают доход за счет данных, поэтому для них очень актуальна проблема качества информации. Один из способов повышения качества данных - кластеризация.

Кластеризация (или кластерный анализ) — это задача разбиения множества объектов на группы, называемые кластерами. Внутри каждой группы должны оказаться «похожие» объекты, а объекты разных группы должны быть как можно более отличны. Главное отличие кластеризации от классификации состоит в том, что перечень групп четко не задан и определяется в процессе работы алгоритма.

Одна из важных Областей данных - это данные о компаниях. Улучшение качества и объема такой информации ведет к повышению прибыли поисковых компаний.

Задача, поставленная перед автором этого диплома и сотрудником компании “Яндекс”, состоит в разработке системы кластеризации данных о компаниях. Система должна работать с данными на русском и турецких языках и предоставлять возможность удобного подключения новых языков. Процесс кластеризации должен запускаться ежедневно. Компании представляют из себя сущности с большим количеством атрибутов, таких как название, урл, адрес , телефон и т.д.

1.1 Существующие алгоритмы кластеризации

Алгоритмы иерархической кластеризации

Среди алгоритмов иерархической кластеризации ([1.], [2.] , [3.], [4.]) выделяются два основных типа: восходящие и нисходящие алгоритмы. В нисходящих алгоритмах в начале все объекты помещаются в один кластер, который затем разбивается на все более мелкие кластеры. В восходящих , в начале работы каждый объект помещают в отдельный кластер, а затем объединяют кластеры во все более крупные, пока все объекты выборки не будут содержаться в одном кластере. Таким образом строится система вложенных разбиений.

Стоит заметить, что система вложенных кластеров не всегда является необходимой.

Неиерархические алгоритмы

Задачу кластеризации можно рассматривать как построение оптимального разбиения объектов на группы. При этом оптимальность может быть определена на основе выбранного функционала качества(функционал среднего риска, функционал ошибки и т.д.).

Такие алгоритмы относятся к типу плоских алгоритмов. Самым распространенным алгоритмом этой категории является метод k-средних ([5.], [6.]).

Основная идея заключается в том, что на каждой итерации перевычисляется центр масс для каждого кластера, полученного на предыдущем шаге, затем векторы разбиваются на кластеры вновь в соответствии с тем, какой из новых центров оказался ближе по выбранной метрике.

Алгоритм завершается, когда на какой-то итерации не происходит изменения кластеров. Это происходит за конечное число итераций, так как количество возможных разбиений конечного множество конечно, а на каждом шаге суммарное квадратичное уклонение V уменьшается, поэтому зацикливание невозможно.

К недостаткам данного алгоритма можно отнести необходимость задавать количество кластеров для разбиения, начальные парметры и сходимость к локальному минимуму.

Нечеткие алгоритмы

Наиболее популярным алгоритмом нечеткой кластеризации является алгоритм c-средних ([7]). Он представляет собой модификацию метода k-средних.

Шаги работы алгоритма:

1. Выбрать начальное нечеткое разбиение *n* объектов на *k* кластеров путем выбора матрицы принадлежности *U* размера *n x k*.
2. Используя матрицу U, найти значение критерия нечеткой ошибки.
3. Перегруппировать объекты с целью уменьшения этого значения критерия нечеткой ошибки.
4. Возвращаться в п. 2 до тех пор, пока изменения матрицы *U* не станут незначительными.

Так же как k-means этот алгоритм предполагает заранее известное число кластеров.

Алгоритмы основанные на теории графов

Суть таких алгоритмов ([8], [9], [10]) заключается в том, что выборка объектов представляется в виде графа *G=(V, E)*, вершинам которого соответствуют объекты, а ребра имеют вес, равный «расстоянию» между объектами. Достоинством графовых алгоритмов кластеризации являются наглядность, относительная

простота реализации и возможность внесения различных усовершенствований, основанные на геометрических соображениях. Основными алгоритмам являются:

- алгоритм выделения связных компонент

  (В алгоритме выделения связных компонент задается входной параметр *R* и в графе удаляются все ребра, для которых «расстояния» меньше *R*. Соединенными остаются только наиболее близкие пары объектов. Смысл алгоритма заключается в том, чтобы подобрать такое значение *R*, лежащее в диапазон всех «расстояний», при котором граф «развалится» на несколько связных компонент. Полученные компоненты и есть кластеры)
- алгоритм построения минимального покрывающего (остовного) дерева

  (Алгоритм минимального покрывающего дерева сначала строит на графе минимальное покрывающее дерево, а затем последовательно удаляет ребра с наибольшим весом)
- алгоритм послойной кластеризации

  (Алгоритм послойной кластеризации основан на выделении связных компонент графа на некотором уровне расстояний между объектами (вершинами))

Кластеризация на основе опорных векторов

Алгоритм кластеризации на основе опорных векторов([11.], [12.]) состоит из двух этапов. На первом этапе происходит обучении SVM (Support Vector Machine) ([13.]) , а на втором происходит маркировка кластеров.

Кластеризация на основе нейронных сетей

Нейронные сети решают широкий круг задач, в том числе задачи кластеризации. Этот подход разрабатывался в работах [14.], [15.].

Кластеризацию с использованием нейронных сетей можно разделить на два случая:

1. Когда известно количество кластеров. Например распознавание букв русского алфавита. В этом случае применяют модель в котором каждому кластеру соответствует один выходной нейрон. Наиболее возбужденный и считается истинным. В нашем случае такие алгоритмы не применимы.
2. Когда конечное количество кластеров не известно. Мы рассмотрим их на примере сомоорганинезующейся карты Кохонена.

Самоорганизующаяся карта состоит из компонент, называемых узлами или нейронами. Их количество задаётся аналитиком. Каждый из узлов описывается двумя векторами. Первый — т. н. вектор веса m, имеющий такую же размерность, что и входные данные. Второй — вектор r, представляющий собой координаты узла на карте. Обычно узлы располагают в вершинах регулярной решётки с квадратными или шестиугольными ячейками.

Изначально известна размерность входных данных, по ней некоторым образом строится первоначальный вариант карты. В процессе обучения векторы веса узлов приближаются к входным данным. Для каждого наблюдения (семпла) выбирается наиболее похожий по вектору веса узел, и значение его вектора веса приближается к наблюдению. Также к наблюдению приближаются векторы веса нескольких узлов, расположенных рядом, таким образом если в множестве входных данных два наблюдения были схожи, на карте им будут соответствовать близкие узлы. Циклический процесс обучения, перебирающий входные данные, заканчивается по достижении картой допустимой (заранее заданной аналитиком) погрешности, или по совершении заданного количества итераций.

Для любой нейронной сети вычислительная сложность $m*C$ - где C - большая константа (параметр сети), а m – количество элементов в выборке.

Вычислительная сложность алгоритмов

Алгоритм кластеризации	**Вычислительная сложность**
Иерархический	O(n2)
k-средних	O(nkl), где k – число кластеров, l – число итераций
c-средних	O(nkl), где k – число кластеров, l – число итераций
Минимальное покрывающее дерево	O(n2 log n)
Послойная кластеризация	O(max(n, m)), где m < n(n-1)/2

Глава 2

Обзор существующих решений

Удачный алгоритм кластеризации данных о компаниях является важной частью бизнеса, прямо влияющей на прибыль, и владельцы поисковых систем не публикуют такие алгоритмы в открытой литературе.

Разработка системы кластеризации данных, описанная в этой дипломной работе, велась для компании "Яндекс", и в основу системы лег процесс обработки данных, существовавший в компании.

2.1 Имеющаяся система кластеризации

Описанная система кластеризации - часть программной системы для обработки данных о компаниях, сохраняющей полученные сведения в базе данных. В дальнейшем сведения из базы используются в поисковой системе Яндекса.

Хранение в базе данных сведений в том виде, как они поступили, может приводить к появлению дублей на выдаче поисковой системы и другим неприятным последствиям. Поэтому после поступления данных в базу и до передачи их в поисковую систему должна быть проведена кластеризация для устранения дублирующих сведений.

Наша цель - организовать регулярный процесс кластеризации. Стоит подчеркнуть, что время выполнения всего процесса должно быть минимальным.

На данный момент в системе находится около 3 млн компаний и их количество постоянно растет. Алгоритмы кластеризации имеющие квадратичную и более чем квадратичную сложность выполнения (иерархический, послойная кластеризация) нам не подходят, так как время их выполнения слишком большое. Такие алгоритмы как k-средних и c-средних предполагают заранее известно число кластеров, а значит тоже не могут быть использованы.

Вместо этого реализован двухфазный алгоритм кластеризации, разработанный в компании “Яндекс”.

- На первом шаге – для каждой компании из входной выборки выбираются «похожие» компании – «кандидаты на сравнение»
- На втором шаге все кандидаты сравниваются на «дубль\не дубль»(В сравнении учавствуют такие атрибуты компаний как название, урл, телефон, род занятий и поля, относящиеся к адресу. Для разных регионов адресные поля могут быть разными. В сисмете реализовано несколько алгоритмов сравнения:
    - с помощью “дерева решений”. В вершинах дерева находятся сравнения разных полей .
    - с помощью весов. Результат сравнения разных полей учавствует в конечном решении с разным весом .
- Кластера объединяются в сессии кластеризации с заголовком (выделенной компанией) для каждого кластера.

2.1.1 Архитектура системы

Система кластеризации использует несколько программных компонентов,взаимодействующих по HTTP-протоколу([16].) или путем файлообмена по torrent-протоколу([17].).

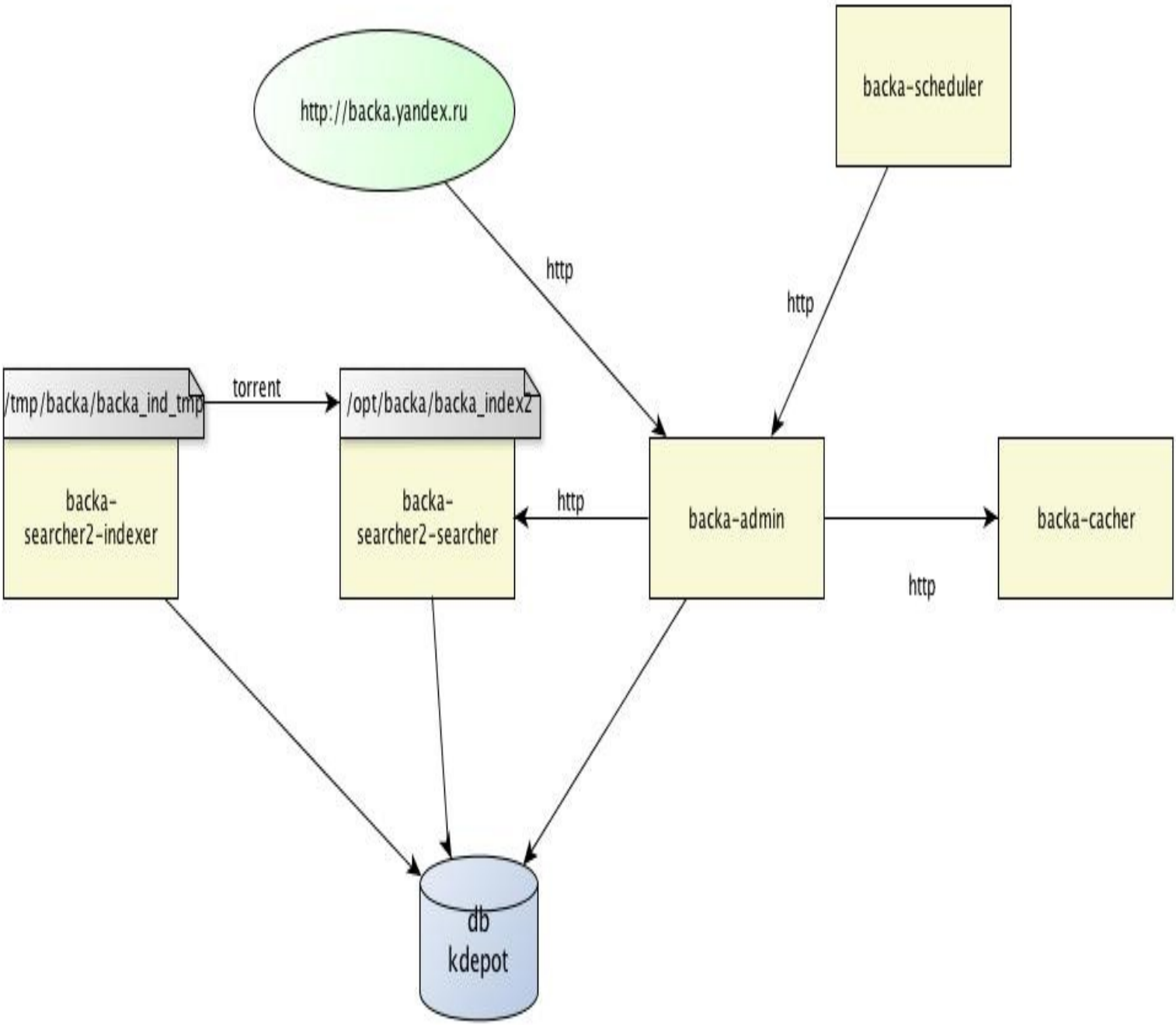

Рис. 1: Архитектура системы

Каждый компонент является отдельным java-приложением; различные компоненты могут располагаться на различных машинах. Компоненты построены с использованием Spring framework([18]). Точкой входа являются сервантлеты - классы, обрабатывающие HTTP-запрос, производящие разбор параметров и обращающиеся к классам, реализующим логику поиска кандидатов и кластеризации. Сервантлеты также преобразуют результат в HTTP-ответ.

Название	Как участвует в кластеризации	Как взаимодействует с другими модулями
backa-searcher2-indexer	• Раз в сутки строит полный индекс по данным о компаниях из базы • С помощью torrent'ов кладет инекс в модуль backa-searcher2-searcher • tmp/backa/backa_ind_tmp – папка, где хранится индекс	• Взаимодействует с базой по JDBC(http://www.oracle.com/technetwork/java/javase/jdbc/index.html) • Передает индекс в модуль backa-searcher2-searcher по torrent-протоколу • opt/backa/backa_index2 – папка, где хранится индекс
backa-searcher2-searcher	• Строит раз в несколько минут инкрементальный индекс • Отвечает на поисковые запросы	• Взаимодействует с базой по JDBC(http://www.oracle.com/technetwork/java/javase/jdbc/index.html) • Отвечает на запросы backa-admin по http-протоколу
backa-admin	• Получает данные от backa-cacher • Осуществляет полный процесс кластеризации по запросу от scheduler'a или из интерфейса • Строит поисковые запросы, для поиска кандидатов • Отправляет поисковые запросы в backa-searcher2-searcher • Сравнивает кандидатов • Записывает результаты кластеризации в бд	• Отправляет поисковые запросы backa-searcher2-searcher'y по http-протоколу • Взаимодействует с базой по JDBC • Отвечает backa-scheduler'y по http-протоколу • С backa-cacher по http-протоколу
backa-cacher	• Хранит часто используемые данные, например рубрики, для того чтобы не надо было постоянно обращаться в базу	• По http-запросу отдает данные в backa-admin • Взаимодействует с базой по JDBC
backa-scheduler	• Раз в сутки запускает процесс кластеризации	• По http-запросу запускает кластеризацию в backa-admin
Backa.yandex.ru	• Внешний интерфейс справочника организаций • позволяет запускать процесс кластеризации	• По http-запросу запускает кластеризацию в backa-admin
Db Kdepot	• База данных Oracle([19].) • хранит данные о компаниях • хранит результаты кластеризации • хранит информацию о запущенных и отработавших процессах кластеризации	• Модули взаимодействует с базой по JDBC

Таблица 1: Описание модулей

2.1.1.1 Хранение компаний

Структура сведений о компании не является жесткой; состав атрибутов компании меняется со временем, поэтому есть необходимость хранить данные не в обычной реляционной базе, а в базе с нестрогой структурой.

“Key-value” хранилище ([20.]) позволяет хранить данные без строгой схемы. Сущность представляет из себя набор аттрибутов и их значений. Существует масса хранилищ, поддерживающих такую систему(Например: Apache Cassandra([21.]), Dynamo([22.]), Project Voldemort([23.])). Однако, на этих же данных работает большое количество аналитических запросов, поэтому используется key-value хранилище данных реализованное на СУБД Оракл и обозначенное на схеме как "db kdepot".

Таблица, позволяющая зхранить данные таким образом описана в запросе 1.

```
CREATE TABLE tr_entity_attr (
    attr_id                         NUMBER(*,0)          PRIMARY KEY
  , entity_id                       NUMBER(*,0)          NOT NULL
  , attr_name                       VARCHAR2(125 CHAR)   NOT NULL
  , attr_npp                        NUMBER(*,0)          NOT NULL
  , attr_as_str                     VARCHAR2(950 CHAR)   NOT NULL
  , CONSTRAINT fk_tr_ea_e_01
      FOREIGN KEY(entity_id)
      REFERENCES tr_entity(entity_id)
);
```

Запрос 1: таблица с данными о компаниях

2.1.1.2 Поиск кандидатов на сравнение

Поиск кандидатов - один из самых сложных процессов в системе. Он должен быть максимально быстрым, поддерживать поиск по логическим условиям на атрибуты компаний, включая комбинацию условий по "И" и "ИЛИ". Поиск непосредственно в реляционной базе при существующих
объемах данных не дает необходимой производительности, кроме того, он повышает нагрузку на хранилище данных, что мешает другим процессам, взаимодействующим с хранилищем (процессы сохранения новых данных).

Для того, чтобы обеспечить необходимую производительность и снизить нагрузку на базу, используется индекс и поиск кандидатов осуществляется по этому индексу.

Для построение индекса был использован lucene([24.]). С ним легко работать с помощью java (язык разработки всей системы) и он предоставляет возможность поиска по сложным запросам с комбинацией условий по “И” и “ИЛИ” .

Для работы с индексом предназначены два модуля:

- searcher2-indexer
- searcher2-searcher

Внутри indexer’a происходит полное построение индекса и отправка этого индекса searcher’y.

Индекс необходимо поддерживать в актуальном состояние. Для этого строятся два индекса - основной и инкрементальный. Основной индекс строится раз в сутки и отражает состояние базы на начало суток. Инкрементальный индекс строится с большей частотой (раз в несколько минут) на основе анализа изменений в базе, произошедших с момента построения основного индекса Поскольку количество изменений намного меньше общего количества сущностей в базе, время построения инкрементального индекса намного меньше времени построения полного
индекса.

Компонент поиска searcher2-searcher при поиске использует оба индекса - основной и инкрементальный и список ид компаний, которые встречаются в обоих индексах, тем самым обеспечивается актуальность информации.

Процесс поиска по индексам описан на рис. 2.

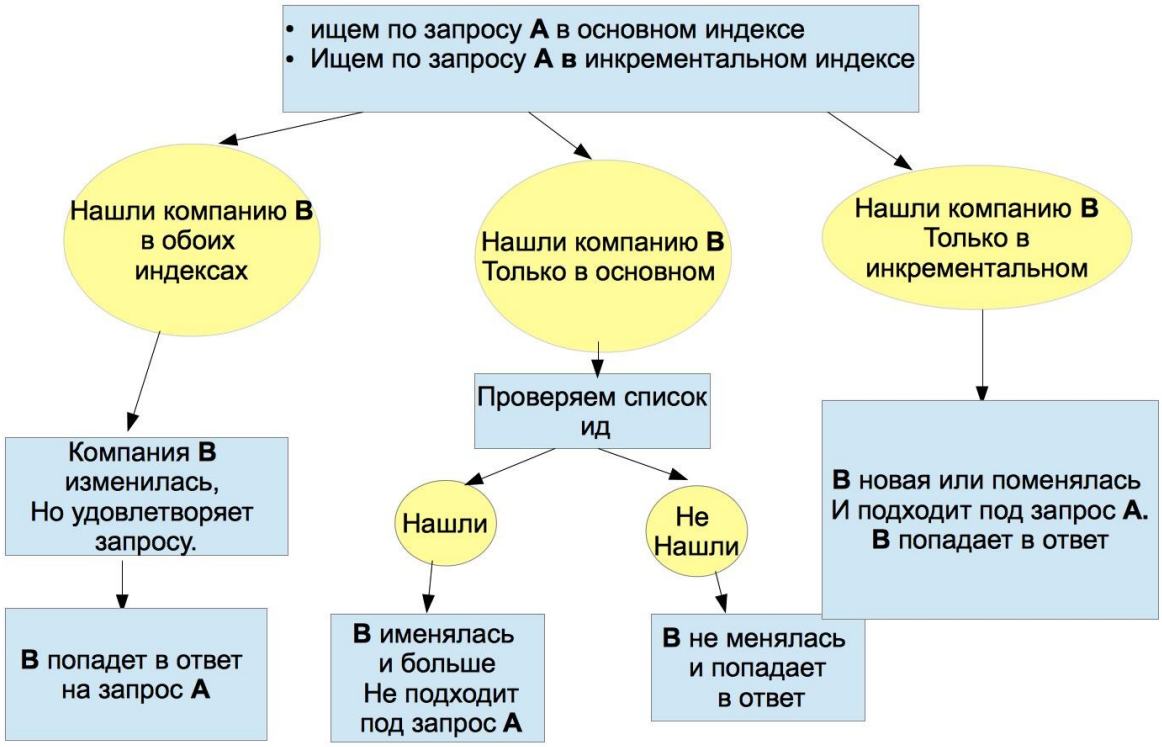

Рис. 2:Описание поиска в индексах

2.1.1.3 Соединение двух фаз кластеризации

Кластеризация может инициироваться пользователем системы через Web-интерфейс модуля backa-admin или по расписанию модулем backa-sheduler, который обращается к модулю backa-admin.

Процесс начинается с первой фазы, для чего создается класс FirstPhase, который осуществляет поиск кандидатов с помощью индекса и передает результат - список групп кандидатов - во вторую фазу.

Во второй фазе осуществляется по-парное сравнение кандидатов внутри группы. Сам процесс сравнения описан в классе PairReducer.

При сравнении кандидатов учитываются следующие атрибуты компаний: название компании, телефон, урл, адрес. Стоит отметить, что для разных стран структура адреса может отличаться, но в него всегда входят страна, город, улица и другие единицы административного деления, принятые в регионе, поэтому

необходима гибкая система для работы с атрибутами. В процессе интернационализации системы, описанной в дипломе, все классы в коде, работающие с атрибутами, стали легко настраиватся под необходимые названия атрибутов.

Сравнение компаний имеет сложную структуру, так как его надо проводить по нескольким полям. Для каждого языка было реализовано несколько алгоритмов сравнения:

- с помощью весов (Сравнению рзных полей присвоен разный вес. Если сумма всех весов привышает пороговое значение, то кандидаты считаются дублями). В примере 1 описана функция сравнения с весами.

```
public double distance(final Aliaser<String, String> aliaser, @NotNull final
QEntity o1, @NotNull final QEntity o2, final HistoryCollector
historyCollector) {

        int result = 0;

        if (NAME_CONTAINS_EXCEP.distance(o1, o2) < TRESHOLD) return
TOTALLY_DIFFERENT;

       if (getNameWithRemovedRubricMetricFunction(rubricProvider,
rubricatorCache).distance(o1, o2) > TRESHOLD) {

            result++;

            if (TR_URL.distance(o1, o2) > TRESHOLD) result++;

        } else {

            if (URL.distance(o1, o2) > TRESHOLD) result++;

        }

        if (PHONE.distance(o1, o2) > TRESHOLD) {

            result++;

        }

        if (RUBRIC.distance(o1, o2) > TRESHOLD) {

            result++;

        }

        if (getAddressDecisionTree().resolve(o1, o2)) result += 2;

        return (result >= 5) ? 0.0 : TOTALLY_DIFFERENT;

    }
```

Пример 1: Функция сравнения с весом

У адреса вес больше, чем у остальных полей, так как в ходе статистических исследований на данных стало ясно, что компании с одинаковыми адресами чаще бывают дублями.

- с помощью дерева “принятия решений” (Desicion Tree) ([25.])(Строится дерево, в вершинах которого находятся операции сравнения. В зависимости от расстояния между полями, происходит переход по правому или левому ребру. В листьях дерева – результаты сравнения.) В примере 2 описана функция сравнения с деровом принятия решений.

```
private DecisionTree<QEntity, Boolean> getDecisionTree(final RubricProvider
rubricProvider, final RubricatorCache rubricatorCache,
                                                       final
QEntityFeatureService entityFeatureService, final FeatureValueService
featureValueService) {
        return node(SYSTEM_ATTR, TRESHOLD, NO,
                node(getNameWithRemovedRubricMetricFunction(rubricProvider,
rubricatorCache), TRESHOLD, NO,
                        node(PHONE, TRESHOLD, node(RUBRIC, TRESHOLD, NO,
                                node(LOCALITY_TR, TRESHOLD, NO,
                                        node(SIDESTREET_TR, TRESHOLD, NO,
                                                node(STREET_TR, TRESHOLD, NO,
                                                        node(BUILDING_NUM,
TRESHOLD,
                                                                node(SUPPLIER
_COORDINATE, TRESHOLD, NO, YES),
                                                                YES)
                                                )

                                        )
                                )
                        ),
                                node(RUBRIC, TRESHOLD, NO,
                                        node(LOCALITY_TR, TRESHOLD, NO,
                                                node(SIDESTREET_TR, TRESHOLD,
NO,
                                                        node(STREET_TR,
TRESHOLD, NO,
                                                                node(SUPPLIER
_COORDINATE, TRESHOLD, NO, YES),
                                                                YES)


                                                )
                                        )
                                )
                        )
                )
        );
    }
```

Пример 2: Функция сравнения с деревом принятия решений

Для подсчета расстояния между полями используются n-граммы([26.]). Алгоритм сравнения заключается в следующем:

- из строк выкидываются слова, длина которых не превосходит 2

- строки вместе с пробелами и другими знаками препинания разбиваются на 2 - граммы
- подсчитывается количество общих 2-грамм и делится на сумму для двух строк
- тоже проделывается для 3-грамм
- и 4-ех грамм
- полученные значения умножаются на коэффициенты и складываются
- результат сравнивается с пороговым значением

Для определения весов и пороговых значений в дереве было использовано машинное обучение с учителем на обучающей выборке выборке.

Для некоторых компаний помимо адреса была указана точная гео-координата. Наличие гео-координаты позволило в разных случаях применять разные подходы к сравнению.

- стандартный(адрес обрабатывается как строчка)
- по гео-координате

При обоих этих подходах остальные поля сравниваются как обычно.

2.1.1.4 Сохранение результатов кластеризации

Кластеры хранятся в таблице, описанной на Запросе 2.

```
create table tr_du_cluster (
        session_id                number(22,0)
      , cluster_id                number(22,0)
      , company_id                number(22,0)
      , cluster_size              number(10,0)
      ,IS_BAD                     NUMBER(1,0) DEFAULT 0
      ,IS_MERGE                   NUMBER(1,0) DEFAULT 1
      ,IS_TASK                    NUMBER(1,0) DEFAULT 0
      , CONSTRAINT pk_tr_cluster
       PRIMARY KEY (session_id, company_id)
      , CONSTRAINT fk_tr_du_cluster_session_id
      FOREIGN KEY (session_id)
       REFERENCES tr_du_session(id)
);
```

Запрос 2: Таблица с данными о кластерах

Если две пары дублей пересекаются, то они объединяются в один кластер.

Процесс кластеризации запускается регулярно и ищет дубли для новых и измененных компаний. Такой процесс называется сессией кластеризации.

Информация о сессиях хранится в таблице, описанной в Запросе 3.

```
create table tr_du_session (
      id                        number(22,0)                      primary key
       , start_time                date
      , finish_time               date
      , rule_set_id               number(22,0)
      , bulker_session_id         number(22,0)
      , status                    varchar2(256 char)
      , descr                     varchar2(512 char)
      , region_id                 number(22,0)
      , process_id                number
);
```

Запрос 3: Таблица с информацией о сессиях кластеризации

2.1.2 Проблемы имеющейся системы

В описанной выше системе поддерживаются данные о компаниях из России. Данные из других стран могут иметь другую структуру. Процессы обработки этих данных могут отличаться.

Одна из целей описанной в данном дипломе разработки - переделка системы таким образом, чтобы подключение нового механизма обработки было максимально простым.

Тут стоит подчеркнуть проблемы существующей системы:

- Хранение региональных данных в базе
    - При добавлении нового региона и данных на его языке в базу добавляется большое количество компаний. Компании от разных регионов нет смысла хранить в одной таблице, так как это существенно увеличивает размеры таблиц и затрудняет работу с ними. Однако, и обрабатывать несколько регионов вместе не планируется. Таким образом все классы, работающие с базой должны легко настраиваться для работы с тем или иным региональным хранилищем.
- Индексы на разных языках
    - Тут ситуация такая же как с хранением в базе.
- Настройка первой фазы для работы с конкретным регионом
    - В течение первой фазы происходит несколько процессов, завязанных на конкретные данные. Это выборка сущностей из базы для дальнейшей работы и построение запроса к индексу. Для подключения новых языков нужно реализовать возможность легкой настройки этих процессов.
- Сохранение результатов кластеризации для региона
    - Как и при хранении компаний, результаты кластеризации для разных стран не имеет смысла хранить в одной таблице. А значит и классы, работающие с результатами кластеризации должны легко настраиваться для работы с конкретным языком.

Глава 3

Описание системы

Разрабатываемая система представляет из себя мультиязычную систему кластеризации. Цель - находить дубли среди сущностей. Сущности представляют из себя компании. У компании есть адрес, название, урл, телефон, рубрика(род деятельности), иногда координаты. В основу системы легла существующая система компании Яндекс.

Перед системой ставились следующие требования:

- легкое подключение разнообразных языков
- поддержка турецкого языка
- тестирование разработанных компонент

3.1 Поддержка легкого подключения новых языков

Одна из важных задач - легкое подключение данных на разных языках. Существующая система работает с русским языком. Выше описаны проблемы системы, которые необходимо решить для дальнейшей удобной работы с компаниями из разных стран.

3.1.1 Хранение региональных данных в базе

Для работы с компаниями в базе существует огромное количество таблиц. Для хранения атрибутов компаний в базе создана эмуляция “key-value” хранилища.

То есть для каждого атрибута в специальной таблице entity_attr существует строка с названием и значением этого атрибута. Несложно заметить что такая структура требует большого количества памяти. Для данных только по одной компании в таблице атрибутов может быть около 20 строк.

Только для русского языка в базе находится около 3 млн компаний. При работе с этими компаниями происходит неоднократное обращение к их атрибутам,

то есть к таблице entity_attr. Огромный размер этой таблицы делает работу с ней узким местом в любом процессе.

При добавлении нового языка встает вопрос о хранении данных для него в базе. Стоит отметить , что данные на разных языках не участвуют одновременно в одних процессах. А значит нет необходимости хранить их вместе.

Исходя из этого было принято решение о создании отдельных хранилищ для каждого нового языка. Названия таблиц в разных хранилищах отличаются префиксом, обозначающим соответствующий язык.

Такая структура базы позволит успешно шардировать ее.

Так же различие в префиксе таблиц позволит легко настраивать java-классы для работы с базой.

3.1.2 Индексы для разных языков

Проблема с индексами в целом очень похожа на проблему с хранением данных в базе. Создание одного индекса для всех языков не имеет смысла, так как нет необходимости в общим поиске для разных языков.

Будем строить разные индексы для разных языков, при этом сохранять их в папки, названия которых отличаются префиксами. Эти префиксы дадут возможность точно идентифицировать язык индекса и легко настраивать java-класса searcher’a и indexer’a для работы с тем или иным языком.

3.1.3 Настройка первой фазы

В течение процесса поиска кандидатов для дальнейшего сравнения происходит

- выгрузка из базы компаний

  (Компании выгружаются партиями. Потом для каждой компании строится запрос к индексу и , если кандидаты в индексе найдены, они отправляются во вторую фазу. Так как для разных языков используются разные хранилища, то нужно иметь возможность настройки выгрузки из базы.)

- построение запроса к индексу

  (Как уже упоминалось выше, для поиска кандидатов в индексе, по сущности компании строится запрос. Индекс lucene имеет java api, что дает возможность передачи ему запроса из кода по http-протоколу. Этот запрос меняется в зависимости от специфики страны, с данными которой мы работаем. Поэтому необходимо иметь возможность настраивать процесс построения запроса. В Примере 3 описан запрос к турецкому индексу)

```
 public static SourceDataQueryBuilder getDefaultBuilder(final String prefix)
{
        return
NAME_BUILDER.and(ADDRESS_BUILDER).or(PHONES_BUILDER).or(URL_BUILDER).and(DIST
RICT_BUILDER).and(getPrefixBuilder(prefix));
    }
```

Пример 3: Запрос к турецкому индексу

Часть запроса, относящаяся к каждому полю, строится отдельно. Построение части запроса для имени описано в Примере 4.

```
public final static SourceDataQueryBuilder NAME_BUILDER = new
SourceDataQueryBuilder() {
        @Override
        public String build(final SourceData data) {
            return buildQuery(data.getName(), "name-tr", "., ");
        }

        private String buildQuery(final @NotNull String name, final @NotNull
String field, final @NotNull String delims) {
            final StringBuilder sb = new StringBuilder();
            final StringTokenizer tk = new StringTokenizer(name, delims,
false);
            final Set<String> items = new TreeSet<String>();
            while (tk.hasMoreTokens()) {
                final String token = tk.nextToken();
                items.add(token);
            }
            for (final String item : items) {
                if (sb.length() > 0) {
                    sb.append(" AND ");
                }
                sb.append("\"").append(item).append("\"");
            }
            if (sb.length() == 0) {
                return "";
            }
            return field + "(" + sb.toString() + ")";
        }
    };
```

Пример 4: Построение запроса для имени компании

За выгрузку данных из базы и построение запросов отвечают отдельные классы. Будем передавать экземпляры этих классов внутрь первой фазы.

Для поддержки разных языков создаются имплементации базовых классов выгрузки и построения запроса.

3.1.4 Классы для работы с базой

Во время процесса кластеризации происходит много работы с базой. Все java классы, отвечающие за это должны легко настраиваться под конкретное хранилище, используется технология Spring. Она позволяет настраивать java классы в специальных конфигурационных файлах. На Примере 5 описаны несколько бинов для работы с базой.

```
<bean name="auditTrService"
class="ru.yandex.common.kdepotng.ext.audit.AuditServiceImpl">
        <property name="kdepot" ref="trKdepot"/>
        <property name="schemaPrefix" value="tr_"/>
    </bean>

    <bean name="auditTrSearchHelper"
class="ru.yandex.common.kdepotng.ext.audit.AuditSearchHelper">
        <property name="kdepot" ref="trKdepot"/>
        <property name="auditService" ref="auditTrService"/>
    </bean>


    <bean id="trKdepot"
class="ru.yandex.common.kdepotng.impl.DefaultService">
        <property name="schemaPrefix" value="tr_"/>
    </bean>
```

Пример 5: Бины для работы с базой

Так как результаты сессий кластеризации для разных регионов не используются одновременно, то данные о сессиях будем хранить в префиксных хранилищах. Это позволит вынести абсолютно всю работу с базой в разные хранилища для разных стран и легко настраивать все java-классы для работы с базой.

3.2 Подключение турецкого языка

В связи с недавним выходом компании Яндекс на турецкий рынок появилась необходимость обработки турецких данных о компаниях. Стоит отметить, что ранее поисковые системы не занимались специальной обработкой данных на турецком языке. Однако, для этой страны в ходе данной работы было выявлено много особенностей.

Серьезной сложностью был сам язык. Турецкий язык достаточно сложный и немногие им владеют. При работе с данными надо постоянно оценивать результат. И большинство выводов пришлось делать из статистической оценки данных, так как получить человека, который бы консультировал по турецкому языку не удалось.

Разработаны классы для создания турецкого индекса и работы с ним, а так же вся вторая фаза содержащая специфичную для языка обработку.

3.2.1 Особенности турецкого адреса

Сама структура турецкого адреса отличается от принятой в России. Встречаются другие единицы административного деления. Только 70% адресов уточнены до номера дома. Координаты полученные от поставщиков оказались в большинстве случаев недостоверными.

Стоит отметить, что при работе с русскими организациями использовался внутренний инструмент компании, который по адресу определял координату и , таким образом, давал возможность проводить сравнение адреса по координатам. Это помогало в случаях, когда в адресе присутствовала лишняя информация, которая мешала сравнению. Например, этаж или номер офиса.

По причине того, что Турция была подключена недавно, внутренний сервис, определяющий координаты давал недостаточную точность или вовсе ошибался. Таким образом, для турецких данных пришлось отказаться от сравнения по координатам.

3.2.2 Особенности турецкого языка

В русском языке для сравнения строк использовалось n-граммы. Алгоритм сравнения заключался в следующем:

- из строк выкидывались слова, длина которых не превосходила 2
- строки вместе с пробелами и другими знаками препинания разбивались на 2 - граммы
- подсчитывалось количество общих 2-граммы и делилось на сумму для двух строк
- тоже проделывалось для 3-грамм
- и 4-ех грамм
- полученные значения умножались на коэффициенты и складывались
- результат сравнивался с пороговым значением

Коэффициенты и пороговое значение было вычислено с помощью машинного обучения. Этому способствовало наличие тестовой выборки для русских данных.

При работе с турецкими данными первоначально использовались те же алгоритмы. Однако, они не показали хорошего результата.

Выяснилось, что в турецком языке одни и те же названия могут встречаться как с пробелами или дефисами так и без них. Поэтому, при первоначальной обработке, из турецкого адреса выбрасываются все пробелы и дефисы.

Оказалось, что слова из одной и двух букв могут иметь важное значение и их не стоит выкидывать.

Так же были выявлены особенности, связанные с частой заменой одних букв другими.

Для турецкого языка отсутствовала тестовая выборка, поэтому коэффициенты использовались те же, что в русских данных. А пороговое значение подбиралось с помощью тестов и оценки их результата.

3.2.3 Определение “спам-слов” и слов с большим весом

Спам слова - это слова, наличие которых в названии или адресе не должно влиять на результат сравнения. Например, для русского языка “больница №2” и “городская больница №2” это одно и то же, а значит слово “городская” можно отнести к “спам-словам”.

Для определения значимости слов была посчитана статистика по встречаниям того или иного слова в конкретном поле и на основе этой статистики были выявлены “спам-слова”.

В турецких названиях часто встречаются названия вида “Профессия Фамилия”. При этом фамилии могут быть очень похожими. Поэтому наличие в названии профессии повышает пороговое значение.

3.3 Тестирование разработанных компонент

3.3.1 Тестирование турецкой кластеризации

К сожалению, получить для турецких данных отсутствовала тестовая выборка. Поэтому оценку качества кластеризации пришлось делать руками.

Для оценки качества турецкой кластеризации необходимо сначала найти точный дубли. Поиск дублей осуществлялся следующим путем:

- применялась кластеризация с упрощенными правилами. То есть в первоначальном запросе к поисковому индексу учавствовало меньше полей. И в последующем попарном сравнении правила сравнения были мягче.
- данные о членах получившегося кластера в определенном порядке записываются в файл
- данные в файле сортируются по какому-то из полей
- результат просматривается глазами

В результате такой оценки полученный процесс находил все дубли, которые удавалось увидеть глазами. Из почти миллиона турецких компаний около 2 тысяч имели дубли.

3.3.2 Тестирование работы всей системы

Разработка большой системы не может обходится без контроля качества и тестирования. На протяжении всей разработки широко применялось модульное тестирование (Unit Testing). Его применение способствовало написанию хорошо структурированного кода, уменьшению числа зависимостей и связей в системе. Также осуществление модульного тестирования упрощает рефакторинг, который необходим в крупных системах. В качестве фреймворка для модульного тестирование применялся JUnit.

Присутствие экспертов в команде также положительно влияло на качество продукта. При принятии того или иного решения учитывалось мнение экспертов.

Глава 4

Заключение

В ходе дипломной работы были рассмотрены различные алгоритмы кластеризации, поиска по данным, их хранения и работы с ними, проведен анализ их особенностей.

Переделана существующая система кластеризации, таким образом чтобы подключение новых языков стало удобным и простым.

Отдельное хранение данных позволяет шардировать базу данных. За счет технологии Spring одни и те же классы можно использовать для работы с разными языками. Разделение процесса на две фазы позволило существенно ускорить работу и вынести всю специфичную для языка обработку во вторую фазу.

Полученная система применена для подключения в процесс кластеризации турецкого языка. На текущий момент разработанная система является частью сервиса для обработки данных о компаниях. Разработанный процес имеет высокие показатели надежности и скорости работы.

На данный момент разработанная система является частью Справочника Организация компании Яндекс. Данные из этого справочника попадают в поиск, на карты и т.д. Процесс кластеризации запускается ежедневно для данных на русском и турецком языках.

Литература

1. Everitt et al., 2001

2. Hansen and Jaumard, 1997

3. Jain et al., 1999

4. Jain and Dubes, 1988

5. Steinhaus H. ,1956

6. Sur la division des corps materiels en parties. Bull. Acad. Polon. Sci., C1. III vol IV: 801—804. , Lloyd S. ,1957

7. Bezdek, 1981; Höppner et al., 1999

8. Delattre and Hansen ,1980

9. Hansen and Jaumard ,1997

10. Jenssen et al. ,2003

11. Ben-Hur et al., 2001

12. Lee and Lee, 2005

13. N. Cristianini and J. Shawe-Taylor, 2000

14. Fukushima , 1975

15. Rumelhart and Zipser (1985)

16. Fielding, Roy T., Gettys, James, Mogul, Jeffrey C., Nielsen, Henrik Frystyk, Masinter, Larry, Leach, Paul J.; Berners-Lee ,June 1999. "RFC 2616: Hypertext Transfer Protocol – HTTP/1.1".

Литература

17. Schulze, Hendrik; Klaus Mochalski (2009). "Internet Study 2008/2009". Leipzig, Germany: ipoque. Retrieved 3 Oct 2011. "Peer-to-peer file sharing (P2P) still generates by far the most traffic in all monitored regions – ranging from 43 percent in Northern Africa to 70 percent in Eastern Europe."

18. http://www.springsource.org/

19. http://www.oracle.com/us/products/database/index.html

20. Marc Seeger ,21 September 2009. "Key-Value Stores: a practical overview"

21. http://www.facebook.com/note.php?note_id=24413138919&id=9445547199&index=9

22.http://www.allthingsdistributed.com/files/amazon-dynamo-sosp2007.pdf

23.http://project-voldemort.com/

24. http://lucene.apache.org/core/

25. S.Murthy. Automatic construction of decision trees from data: A Multi-disciplinary survey.1997.

26. William W. Cohen, Pradeep Ravikumar, Stephen E. Fienberg. A Comparison of String Metrics for Matching Names and Records. 2003