Программирование методов разрешения сущностей и слияния данных при реализации ETL в среде Hadoop*

© Вовченко А.Е. © Калиниченко Л.А. © Ковалев Д.Ю.
Институт Проблем Информатики РАН (ИПИ РАН)
Москва
alexey.vovchenko@gmail.com leonidk@synth.ipi.ac.ru dm.kovalev@gmail.com

Аннотация

В статье обсуждаются вопросы разрешения сущностей (Entity Resolution) и слияния данных (Data Fusion) в контексте интеграции больших данных в среде Hadoop. Проблема разрешения сущностей ориентирована на решение таких задач как выявление дубликатов (Duplicate Detection), удаление дубликатов (Deduplication), связывание записей (Record Linkage), идентификация объектов (Object Identification), сопоставление связей (Reference Matching) и др. Проблема слияния данных является заключительным этапом интеграции данных. В работе дан краткий обзор методов разрешения сущностей и методов слияния данных. Затем в работе рассматриваются вопросы адаптации таких методов к их применению в ETL процессе при интеграции больших данных в Hadoop. Рассмотрены способы программирования методов разрешения сущностей и слияния данных как части ETL процесса.

1 Введение

В различных областях науки наблюдается экспоненциальный рост объема получаемых экспериментальных (наблюдательных) данных. Например, в астрономии текущий и ожидаемый темп роста данных от наземных и космических телескопов удваивается в течение периода от шести месяцев до одного года. Сложность использования таких данных увеличивается еще и вследствие их естественной разнородности. Разнообразие (информационная несогласованность) получаемой информации вызывается, в частности, не только большим числом организаций, производящих наблюдения, и их независимостью, но и разнообразием объектов наблюдения, непрерывным и быстрым совершенствованием техники наблюдений, вызывающим адекватные изменения структуры и содержания накапливаемой информации. Это приводит к необходимости использования неоднородной, распределенной информации, накопленной в течение значительного периода наблюдений технологически различными инструментами.

Для анализа больших объемов накапливаемых данных используются современные распределенные инфраструктуры обработки массивных данных (например, Hadoop [41, 45]). Основной особенностью подобных инфраструктур является почти линейная горизонтальная масштабируемость (производительность системы растет линейно относительно числа узлов кластера).

Главным достоинством подобных инфраструктур является возможность анализировать и обрабатывать разно-структурированные данные, например, реляционные, XML, JSON, тексты и другие. При этом возникает проблема интеграции информации, извлекаемой из разно-структурированных данных.

Традиционно процесс интеграции данных можно представить состоящим из следующих этапов:

- сопоставление схем (Schema Matching),
- интеграция схем (Schema Integration),
- трансформация данных (Data Transformation),
- разрешение сущностей (Entity Resolution [17, 22, 34]),
- слияние данных (Data Fusion [10]).

В разделах 2 и 3 дан краткий обзор традиционных методов разрешения сущностей и методов слияния данных. В разделе 4 показано как можно адаптировать стандартные методы разрешения сущностей при интеграции массивных данных в среде Hadoop. Наконец, в разделе 5 показаны способы программирования методов разрешения сущностей и слияния данных как части ETL процесса в Hadoop.

* *Работа выполнена при поддержке РФФИ (гранты 13-07-00579, 14-07-00548) и Президиума РАН (Программа фундаментальных исследований Президиума РАН № 16 «Фундаментальные проблемы системного программирования»).*

Труды 16-й Всероссийской научной конференции «Электронные библиотеки: перспективные методы и технологии, электронные коллекции» — RCDL-2014, Дубна, Россия, 13–16 октября 2014 г.

2 Краткий обзор методов разрешения сущностей

В общем случае под термином разрешения сущностей (entity resolution [17, 22, 25–26, 31–32, 34]) понимается извлечение информации об одной и той же сущности реального мира из разнообразных структурированных коллекций данных, приведение извлеченных данных к унифицированному представлению. При этом применяются методы извлечения, сопоставления (matching), группирования, связывания (linking), устранения дублирования (deduplication) различных представлений информации.

В общем случае процесс разрешения сущностей включает следующие этапы [26]:

- Подготовку данных (Data preparation);
- Выбор методов сопоставления данных (Match Feature);
- Определение методов разрешения пар сущностей (Pairwise ER);
- Определение ограничений (ER Constraints);
- Реализацию алгоритма.

Важным этапом для успешного разрешения сущностей является подготовка данных, которая включает нормализацию схем и нормализацию данных. Нормализация схем включает, например:

- сопоставление атрибутов схем (например, «контактный телефон» и «мобильный телефон»);
- слияние атрибутов (например, «полный адрес» получается из атрибутов «город», «индекс» «улица», …);
- слияние множественных значений и списков (например, «контактные телефоны» и «основной номер телефона», «дополнительный номер телефона») и др.

Нормализация данных может включать приведение к строчному или заглавному регистру; удаление разделителей; поиск и исправления опечаток; поиск сокращений и аббревиатур и замена их на полные стандартные формы; использование словарей для нормализации строк, и много другое.

Сопоставление сущностей может осуществляться разнообразными способами оценки сходства (similarity) сущностей. Мера сходства может быть как булева, так и вещественная. Применяют следующие методы оценки сходства:

- эквивалентность булевых предикатов;
- вычисление функции сходства значений (Levenstein [52], Smith-Waterman [52]);
- вычисление функции сходства множеств (Jaccard [52], Dice [52]);
- вычисление функции сходства векторов (Cosine similarity [49], TFIDF [50]);
- сходство на основе выравнивания (Jaro – Winkler [52], Soft – TFIDF [8], Monge – Elkan [51]);
- сходство фонетических данных: Soundex [52];
- сходство, основанное на переводе (может использоваться для нормализации аббревиатур);
- сходство, основанное на знаниях о предметной области, и др. [52].

Рассматривают также сходство отношений. Меры, используемые для отношений, обычно основаны на сходстве множеств, и предполагают использование аналогичных функций:

- Common Neighbors,
- Jaccard's Coefficient,
- Adar Coefficient [1].

При сравнении пар сущностей они рассматриваются как вектора, для которых нужно вычислить их сходство. Традиционным подходом является подсчет сходства некоторым методом для каждого из атрибутов независимо. А затем реализуется подсчет взвешенной суммы. Например:

```
0.5*1st – author – match – score +
0.2*venue – match – score +
0.3*paper – match – score
```

Недостатком этого подхода является сложность выбора весов для каждого из атрибутов и сложность выбора порога сходства сущностей. Другим подходом является задание правил для каждого атрибута независимо. Например:

```
(1st – author – match – score > 0.7 AND
venue – match – score > 0.8)
OR (paper – match – score > 0.9 AND
venue – match – score > 0.9)
```

Недостатком этого подхода является сложность формулирования подобных правил вручную. Применяются также методы, основанные на модели Fellegi & Sunter [24].

Для сопоставления пар сущностей применяют также специальные методы машинного обучения, которые позволяют автоматизировать процесс формулирования критериев для сопоставления сущностей:

- Decision trees [18],
- Support vector machines [9, 16],
- Ensembles of classifiers [15],
- Conditional Random Fields (CRF) [27].

Недостатком этих подходов является: несбалансированность результирующих классифицированных множеств (так, в результате образуется значительно больше несхожих объектов, чем схожих), а также высока вероятность того, что объект не будет причислен ни к какому классу (схожих, несхожих). Но оба эти недостатка могут решаться путем тонкой настройки алгоритмов. Ключевой проблемой при использовании методов машинного обучения при сравнении пар сущностей является выбор обучающего множества.

Выделяют следующие методы, не требующие построения обучающей выборки для классификации сущностей:

- Обучение без учителя или с частичным привлечением учителя [29, 36, 42];

- Методы с активным обучением
  – Ансамбли классификаторов [38, 39];
  – Доказуемая оптимизация точности/полноты (precision/recall) [3, 4];
  – Краудсорсинг [33, 40].

Таким образом, при выборе методов сопоставления сущностей выделяют: множество алгоритмов сходства, методы, основанные на машинном обучении, и методы, основанные на активном обучении и краудсорсинге. Последняя группа методов сейчас считается наиболее перспективной, но требует проведения дополнительных исследований.

Примеры правил, используемых для установления сходства сущностей:

- Транзитивность: если M1 и M2 схожи, и M2 и M3 схожи, тогда и M1 и M2 схожи;
- Эксклюзивность: если M1 и M2 схожи, тогда M3 не может быть схож с M2;
- Функциональные зависимости: если M1 и M2 схожи, тогда M3 и M4 должны быть схожи.

Транзитивность часто используется для методов удаления дубликатов (Deduplication), а эксклюзивность используется в методах установления связей (Record Linkage).

В заключение можно отметить, что разрешение сущностей является быстро развиваемой областью. Исследуются новые меры сходства [52], ведутся работы по применению перспективных методов машинного обучения [3, 4, 33, 38–40]. Развивается применение функциональных зависимостей при очистке данных (data cleaning) [2, 13, 23]. Ведутся работы по построению сущностей с наиболее представительными данными (включающими данные из разнообразных дубликатов – Canonicalization [5]). Также ведутся работы по методам, когда решения по сходству двух сущностей принимается на основе анализа совокупности сущностей, применения вероятностных логик сходства, латентной модели Дирихле [6, 7, 14].

3 Краткий обзор методов слияния данных

Под слиянием данных (Data Fusion [10, 12, 21]) понимается образование интегрированного представления информации об одной же сущности реального мира, полученной из различных источников данных. Задачами процесса слияния данных является: слияние записей о сущностях, разрешение возможных конфликтов, обнаружение и удаления ошибочных данных. Методы слияния данных, кратко рассмотренные в данном разделе, исследованы в Потсдамском университете в диссертации [12]. Различные аспекты проблемы слияния данных представлены на рис. 1.

3.1 Типы конфликтов при слиянии данных

Различают два типа конфликтов: конфликты, вызванные неопределенными значениями и конфликты, вызванные противоречивыми значениями. Неопределенность означает, что в одном источнике данных содержатся неизвестные значения (null), а в другом известные. Проблема заключается в том, что семантика неопределенных значений (null) может сильно отличаться. Различают три варианта: неизвестные значения, несуществующие значения (например, атрибут «имя супруга» всегда будет null для неженатых), скрытые значения (такие данные, которые по каким-то причинам не позволено видеть). Противоречивость значений означает появление двух различных не нулевых (not null) значений. Возможны различные стратегии обработки подобных конфликтов.

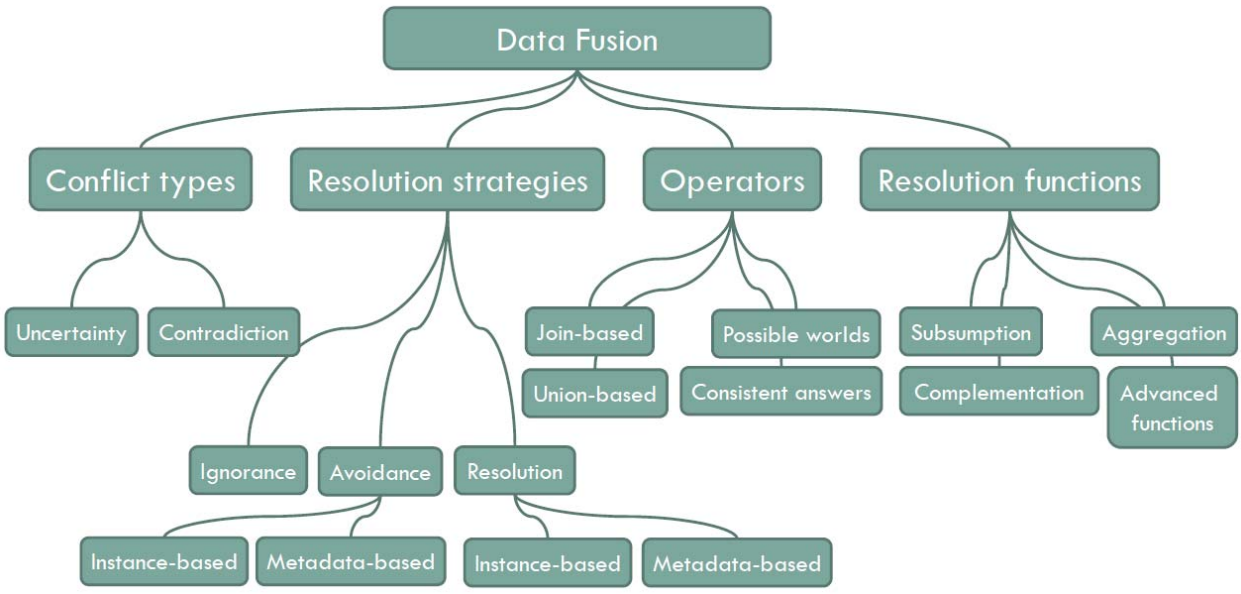

Рис. 1. Различные аспекты проблемы слияния данных

3.2 Стратегии разрешения конфликтов

Различают следующие виды подходов к разрешению конфликтов:

- игнорирование конфликтов;
- избегание конфликтов;
- разрешение конфликтов.

Стратегия игнорирования конфликтов предполагает извлечение всей доступной информации. Например, для строк это может быть обычная конкатенация строк, а пользователь уже сам решает, какие данные верны.

Стратегия избегания конфликтов предполагает выбор данных на основе самих данных (по некоторому алгоритму) или на основе метаданных. Примером функции на основе данных может служить функция coalesce (выбор первого не нулевого значения), или функция выбора самого длинного значения. Примером функций на основе метаданных может выступать выбор в зависимости от самого источника (например, известно, что один из источников наиболее достоверный). Другим примером является функция выбирающая значение из того источника, в котором большее число значений было выбрано для других атрибутов.

Стратегии разрешения конфликтов учитывают все значения, и выбирают из них «достоверное». Примером подобной функции могут выступать всевозможные функции голосования, функции выбора случайного значения, функции среднего значения, функции наиболее часто встречающегося значения и др.

3.3 Основные функции разрешения конфликтов

Вводится операция ***outer union*** [12], результатом которой является объединение двух отношений. Если схемы не совпадают, то результирующая схема является объединением двух исходных схем. Например, пусть даны два отношения A с набором атрибутов = {a, b, c, d}, и отношение B с набором атрибутов = {c, d, e, f}. Результирующая схема будет содержать набор атрибутов = {a, b, c, d, e, f}. В результирующие кортежи для недостающих атрибутов помещаются нулевые значения. Эта операция не является стандартной и отсутствует в большинстве реляционных СУБД. В реляционной алгебре подобная операция может быть представлена как:

```
(SELECT a, b, c, d, NULL as e, NULL as f
FROM A)
UNION
(SELECT NULL as a, NULL as b, c, d, e, f
FROM B)
```

Вводится функция tuple ***subsumption*** [12]. Говорят, что кортеж t1 поглощает другой кортеж t2 (поглощаемый кортеж), если у них:

- совпадают схемы;
- в t2 больше неизвестных (null) значений чем в t1;
- в t2 все известные значения совпадают со значения в t1.

Например, пусть дан кортеж t1 = (5, ‘text’, null, 7) и t2 (5, null, null, 7). Видно, что каждый атрибут в t2 либо совпадает с аналогичным атрибутом в t1, либо он null. Для этого примера кортеж t1 поглощает кортеж t2.

Вводится функция ***tuple complementation*** [12]**.** Говорят, что кортежи t1 и t2 дополняют друг друга если:

- у них совпадают схемы;
- они не совпадают;
- значения соответствующих атрибутов в t1 и t2 совпадают, либо одно их не определено, либо оба не определены;
- t1 и t2 имеют как минимум один атрибут, значения которого совпадают.

Например, пусть дан кортеж t1 = (5, ‘text’, null, null) и t2 (5, null, null, 7). Видно, что кортежи дополняют друг друга. Результатом операции дополнения для этих двух кортежей будет новый кортеж t = (5, ‘text’, null, 7).

3.4 Операторы слияния данных

Различают два основных подхода к слиянию данных. Это подходы основаны на операции объединения (union based) или на операции соединения (join based). Различают следующие основные операции.

Minimum Union [12] (union based). Операция представляет собой выполнение операции ***outer union***, а затем удаления из результата всех поглощаемых (subsumed [12]) кортежей. Пример операции представлен на рис. 2.

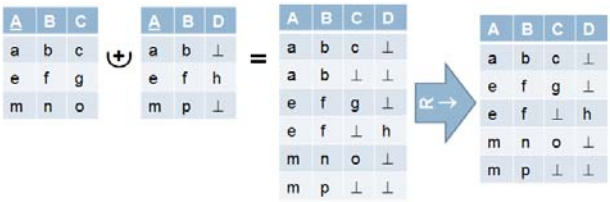

Рис. 2. Пример операции Minimum Union

Complemetation Union [12] (union based). Операция представляет собой выполнение операции outer union, а затем дополнения (complementation) всех возможных кортежей. Пример операции представлен на рис. 3.

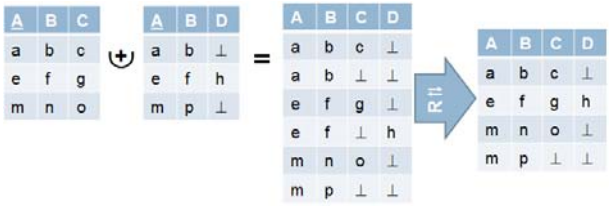

Рис. 3. Пример операции Complementation Union

Grouping and Aggregation [12] (union based). Операция предполагает выполнение outer union, а

затем группировки по общему атрибуту и применения функции агрегации к остальным атрибутам. Пример операции на языке SQL представлен ниже.

```
WITH OU AS (
  ( SELECT A, B, C, NULL AS D FROM U1 )
       UNION (ALL)
  ( SELECT A, B, NULL AS C, D FROM U2 )
),

SELECT A, MAX(B),  MIN(C), SUM(D)
FROM   OU
GROUP BY A
```

Full Disjunction [37] (join based). Операция представляет собой ***full outer join*** (стандартная реляционная операция), после чего применяется ***subsumption*** к результату. Пример представлен на рис. 4.

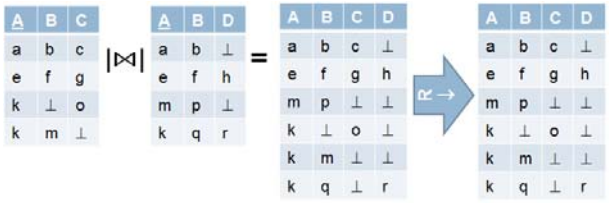

Рис. 4. Пример операции Full Disjunction

Match Join [12] (union+join based). В операции выбираются всевозможные комбинации значения атрибутов, после чего выполняется ***full outer join***. Фактически реализуется ***outer union*** двух коллекций. После чего определяется N – 1 вспомогательных отношений, где N – число атрибутов, каждое из которых содержит по два атрибута, один общий, и какой-то другой. После чего происходит ***full outer join*** N – 1-го отношения. Пример реализации операции на языке SQL представлен ниже.

```
WITH
  OU(A,B,C,D) AS (
  ( SELECT A, B, C, NULL AS D FROM U1 )
    UNION
  ( SELECT  A, B, NULL AS C, D FROM U2 )
),// ← Outer Union
  B_V (A,B) AS ( SELECT DISTINCT A, B
FROM OU ), // ← 1-е отношение (N = 4)
  C_V (A,C) AS ( SELECT DISTINCT A, C
FROM OU ), // ← 2-е отношение (N = 4)
  D_V (A,D) AS ( SELECT DISTINCT A, D
FROM OU ), // ← 3-е отношение (N = 4)
SELECT A, B, C, D
FROM B_V FULL OUTER JOIN C_V FULL OUTER
JOIN D_V // ← Full Outer Join
```

Merge (union+join based). Операция объединяет операции соединения и объединения. Для каждого общего атрибута формируются две версии значений, нулевые значения удаляются функцией COALESCE (выбор первого ненулевого значения). Пусть даны два отношения A с набором атрибутов {a, b, c} и B с набором атрибутов {a, b, d}. a – конфликтующий атрибут, b – атрибут с нулевыми значениями. Пример реализации на SQL представлен ниже, а результат операции представлен на рис. 5.

```
(SELECT A.a, COALESCE(A.b, B.b), A.c, B.d
FROM A LEFT OUTER JOIN B ON A.a = B.a)
UNION
(SELECT B.a, COALESCE(B.b, A.b), A.c, B.d
FROM A RIGHT OUTER JOIN B ON A.a = B.a)
```

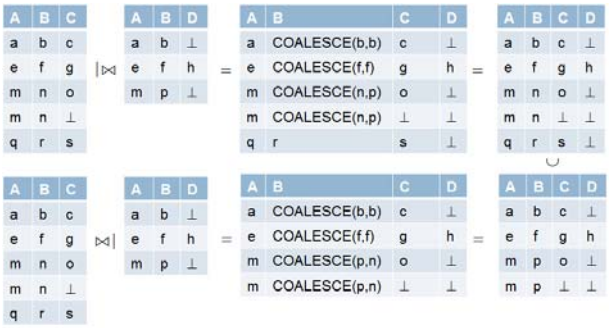

Рис. 5. Пример операции Merge

4 Разрешение сущностей для больших данных

Для манипулирования большими разно-структурированными данными служат Hadoop инфраструктуры [41, 45], предоставляющие масштабируемое хранилище и обеспечивающие высокую скорость обработки больших данных за счет распределенной их обработки. Таким образом, для применения методов нужна *адаптация* алгоритмов для их распределенного выполнения на различных узлах Hadoop кластера.

В среде Hadoop реализована парадигма распределенного программирования для анализа данных Map-Reduce [20, 46], по имения основных функций. В начале на всех узлах кластера обрабатываются блоки данных независимо друг от друга (Map). После чего данные группируются по заранее выбранным для алгоритма ключам и поступают на выполнение на один или более узел в зависимости от алгоритма (Reduce).

Таким образом для реализации любого алгоритма в Hadoop инфраструктуре требуется его адаптация к виду Map-Reduce. Другим вариантом является реализация алгоритма на одном из языков высокого уровня, таких как: Pig [47], Hive [48], Jaql [43, 44]. Все эти языки автоматически переписывают программы, реализованные на них, в Map-Reduce приложения для выполнения на Hadoop кластере.

В случае больших данных и распределенных инфраструктур традиционные подходы требуют доработок. Различают два основных метода разрешения сущностей над большими данными: разбиение данных на блоки (blocking [19, 35]) и распределенный метод разрешения сущностей.

Суть разбиения на блоки заключается в следующем. Пусть у нас представлены 1000 компаний в 1000 городах. И нам нужно сравнить компании. Алгоритм полного попарного сравнения

потребует 10^{12} сравнений. При этом, если предположить, что компании из разных городов не могут совпадать, то потребуется 10^9 сравнений. Ключевой проблемой данного подхода является выбор критерия, по которому разбивать данные. Различают два основных метода: основанный на хэш функции [26], и основанный на сходстве соседей [26]. Метод, основанный на хэш функции, предполагает разбиение на блоки по хэш ключу. Основной проблемой алгоритма является выбор хэш функции. Метод, основанный на сходстве соседей, предполагает, что совпадать могут только объекты, схожие по некоторой мере. Все объекты сортируются по какому-то признаку (ключу – простому или составному, уникальность ключа не требуется). После этого выбирается размер окна. И объекты сравниваются только внутри окна. Проблемой данного метода является выбор ключа сортировки.

Распределенный метод разрешения сущностей предполагает реализацию традиционных алгоритмов этого семейства в виде Map-Reduce приложения, что требует зачастую полного пересмотра исходного алгоритма. Другой вариант – реализация алгоритма разрешения сущностей на специализированных языках, чему будет посвящен следующий раздел. Третий вариант – использование специализированных инструментов, направленных на распределенное выполнение методов разрешения сущностей над Hadoop [30].

5 Программирование операций разрешения сущностей и слияния данных на языке HIL

Язык HIL (Highlevel Integration Language) [28], новый специализированный язык, разработанный IBM, ориентированный на разрешение и интеграцию сущностей в Hadoop инфраструктуре.

HIL компилируется в язык Jaql [43, 44], который в свою очередь автоматически переписывается в Map-Reduce, если этого требует алгоритм.

5.1 Реализация методов разрешения сущностей

Пусть даны структуры данных, включающие три атрибута: id, value, name. Тогда простейшее правило разрешения сущностей на языке HIL будет выглядеть следующим образом:

```
declare Duplicated: ?;
declare Generated: ?;
declare Deduplicated: ?;

create link Deduplicated as
select
[gen: [id: g.id, name: g.name, value:
g.value],
dup: [id: d.id, name: d.name, value:
d.value]]
from Generated g, Duplicated d
match using
  rule_id: g.id = d.id,
  rule_name: g.name = d.name,
  rule_value: g.value = d.value;
```

В этом примере используется простое сопоставление сущностей, по совпадению значений. Если требуется ввести какую-то функцию меры для значений, это можно реализовать внешней функцией Jaql:

```
@jaql{
compareValue =
javaudf("org.ipiran.similarity.ValueSimil
arity");
}
```

После этого такую функцию можно вызывать из языка HIL:

```
declare compareValue: function ? to ?;
declare Duplicated: ?;
declare Generated: ?;
declare Deduplicated: ?;

create link Deduplicated as
select
[gen: [id: g.id, name: g.name, value:
g.value],
dup: [id: d.id, name: d.name, value:
d.value]]
from Generated g, Duplicated d
match using
rule_id:
  compareValue(g.id, d.id) > 0.7,
rule_name:
  compareValue(g.name, d.name) > 0.7,
rule_value:
  compareValue(g.value, d.value) > 0.7;
```

Можно также ввести меру для сравнения не отдельных значений, а для сравнения объектов целиком. Пусть описана функция **compareObject**, которая принимает на вход объекты, тогда правило на языке HIL изменится, т.к. в этом случае используется другой вид правил:

```
insert into Deduplicated
select
[gen: [id: g.id, name: g.name, value:
g.value],
dup: [id: d.id, name: d.name, value:
d.value],
value: compareObject(g,d)]
from Generated g, Duplicated d
where compareObject(g, d) > 0.7;
```

Во всех этих случаях происходит сравнение всех объектов со всеми, сложность подобного сравнения $O(n^2)$. Несмотря на то, что сравнения будут выполняться независимо и распределены на всех узлах кластера (т.к. HIL переписывается в Jaql, а тот в свою очередь в Map-Reduce), время их выполнения может быть достаточно большим. Для уменьшения количества сравнений, как было описано в четвертом разделе, можно разбивать данные на блоки.

Пусть имеется функция **calcHash**, которая вычисляет hash для объектов. В результате функция может выдавать столько уникальных значений, на сколько блоков нам надо разбить данные. Тогда объединив правила, рассмотренные выше, выбрав в начале те объекты что совпадают по хэш функции, а далее вычислив общую меру, можно получить результат за более короткое время:

```
declare calcHash: function ? to ?;
insert into GeneratedHash
select [$.*, hash: calcHash($.*)]
from Generated;
insert into DuplicatedHash
select [$.*,hash: calcHash($.*)]
from Duplicated;

create link Deduplicated as
select [
  gen: [id: g.id, name: g.name, value:
g.value],
  dup: [id: d.id, name: d.name, value:
d.value]]
from GeneratedHash g, DuplicatedHash d
match using
  rule_id: g.hash = d.hash;
insert into Measured
select [gen: dd.gen, dup: dd.dup, value:
compareObject(dd.gen, dd.dup)]
from Deduplicated dd
where compareObject(dd.gen, dd.dup) > 0.8
```

5.2 Реализация методов слияния данных

На данном этапе будем считать, что этап разрешения сущностей уже пройден и нам дана некоторая коллекция Deduplicated где уже установлены соответствия одним из выше перечисленных способов. Например, пусть у нас есть две коллекции A (id, a, b, c) и B (id, a, b, d). Атрибуты a, b, c, d могут содержать NULL значения, атрибуты id совпадают. Ниже дан пример подобных данных для коллекции A в формате JSON:

```
[{"a":null,"b":null,"c":"wmqhxfgmac","id"
:919132322},
{"a":null,"b":null,"c":"wmqhxfgmac","id":
919132322}]
```

Тогда коллекция разрешенных сущностей может быть получена следующим образом:

```
create link Deduplicated as
select
[gen: [id: a.id, a:a.a, b:a.b, c:a.c],
dup:  [id: b.id, a:b.a, b:b.b, d:b.d]]
from A a, B b
match using
  rule1: a.id = b.id;
```

Рассмотрим теперь реализацию Minimum Union и Fusion оператор [11] на языке HIL.

Как было описано в третьем разделе, **Minimum Union** - это последовательное применение операций outer union и subsumption [12]. Outer Union фактически реализуется с помощью индекса **FusionIndex**. Использование индекса оправдано, т.к. существует несколько записей, описывающих одну сущность. Ключом является атрибут id. Ниже представлена реализация операции Outer Union.

```
insert into FusionIndex![id: f.gen.id]
select [a: f.gen.a, b: f.gen.b, c:
f.gen.c] from Deduplicated f;

insert into FusionIndex![id: f.dup.id]
select [a: f.dup.a, b: f.dup.b, d:
f.dup.d] from Deduplicated f;
```

Далее для реализации subsumption требуется удалить все ненужные кортежи. Это делается на языке Jaql. Для этого нужна функция, которая бы определяла, поглощается ли один кортеж другим. К сожалению, в языке Jaql нет возможностей написания общих (generic) методов, универсальных для всех коллекций, поэтому функцию сравнения можно реализовать на java и подключить к языку Jaql как демонстрировалось в разделе 5.1 на примере функций вычисления меры. Либо же можно реализовать функцию для сравнения конкретных коллекций на языке Jaql, как показано ниже:

```
is subsumed = fn(i,j) ((
isnull(j.a) or (i.a == j.a) ) and (
isnull(j.b) or (i.b == j.b) ) and (
isnull(j.c) or (i.c == j.c) ) and (
isnull(j.d) or (i.d == j.d) ) and (
i != j)) ;
```

Функция **is_subsumed(i,j)** проверяет, поглощает ли один кортеж другой кортеж при помощи попарного сравнения атрибутов или проверки на null.

```
removeSubsumed = fn (a) ( b = a,
subs = for (i0 in b) [a->filter
is_subsumed(i0,$)], s = subs -> expand,
a -> filter not $ in s);
```

Функция **removeSubsumed** удаляет все поглощенные записи из кортежа. Здесь реализован наивный алгоритм, который попарно для каждого кортежа находит все поглощенные им, и удаляет их.

```
minUnion = fn(id,a) ( {id:id, minunion :
removeSubsumed(a)});
```

Функция **minUnion** нужна для построения результирующих кортежей при реализации Minimum Union.

Теперь операцию Minumum Union можно описать следующим образом на языке HIL:

```
insert into MinimumUnion
select minUnion(i.dup.id,
            FusionIndex![id : i.dup.id])
from Deduplicated i;
```

Для каждого id достаются все соответствующие записи и удаляются те, которые ими поглощаются.

Data Fusion оператор [11] представляет собой особый вид функции, использующий группировку для преодоления конфликтов. Основная идея заключается в группировке различных представлений одной и той же сущности по общему атрибуту, а затем в применении функций разрешения конфликтов для всех остальных атрибутов, сливая данные в одну сущность. Различают два вида стратегии для функций разрешения конфликтов:

- deciding-стратегия – заключается в выборе какого-то одного значения каким-то способом (минимум, максимум, случайное значение);
- mediating-стратегия – заключается в агрегации всех значений (среднее значение, сумма).

Пусть имеются две коллекции A (id, name, age) и B (id, name, info), пример которых дан ниже:

```
A
[{"id":760046903,"name":null,"age":null},
{"id":15009544,"name":
"zvqcsxkzxk","age":938781652}]

B
[{"id":15009544,"name":null,"info":null},
{"id":760046903,"name":"pjltaghyug","info
":null}]
```

Пусть для них пройден этап разрешения сущностей и построена коллекция Deduplicated как описано выше в этом разделе. Пусть также для этих данных построен индекс FusionIndex, как показано выше для операции Minimum Union. Тогда Data Fusion Оператор на языке HIL может быть описан следующим образом:

```
@jaql{
average = fn($a) avg($a[*].age);
any = fn($a) any($a[*].name);
concat = fn ($a) strJoin($a[*].info,"_");
}

insert into Fused
select [
id : i.dup.id,
age:
    average(FusionIndex![id : i.dup.id]),
name:
    any(FusionIndex![id : i.dup.id]),
info:
    concat(FusionIndex![id: i.dup.id])]
from Deduplicated i;
```

Функции вычисления среднего, выбора случайного не-null значения, а также конкатенации реализованы на Jaql. Данное правило образует коллекцию **Fused**, причем для атрибута *age* будет подсчитано среднее значение, для имени *name* выбрано любое ненулевое значение, а для атрибута *info* будет получена конкатенация всех доступных значений. Таким образом, в данном примере показана реализация обеих стратегий для функций разрешения конфликтов в Data Fusion операторе.

6 Заключение

Рассмотренные методы и операции извлечения и интеграции информации о сущностях реального мира позволяют программировать интеграционные потоки вида ETL, образующие интегрированные структурированные данные, которые могут быть использованы в приложениях для дальнейшего анализа и обработки. В статье рассмотрены методы разрешения сущностей и слияния данных. В статье показаны способы программирования методов и операций извлечения и интеграции информации о сущностях реального мира, включая методы слияния данных на декларативном языке HIL.

7 Литература

[1] LA Adamic, E Adar. Friends and neighbors on the Web. Social networks 25 (3), 211–230, 932, 2003.

[2] Rohit Ananthakrishna et Al., Eliminating fuzzy Duplicates in data warehouses, VLDB 2002.

[3] A. Arasu et al., On active learning of record matching packages, SIGMOD 2010.

[4] K. Bellare et al., Active sampling for entity matching, KDD 2012.

[5] O. Benjelloun et al., Swoosh: A generic approach to Entity Resolution, VLDBJ. 18(1), 2009.

[6] I. Bhattacharya & L. Getoor, Collective Entity Resolution in Relational Data, TKDD 2007.

[7] I. Bhattacharya & L. Getoor, A Latent Dirichlet Model for Unsupervised Entity Resolution, SDM 2007.

[8] M. Bilenko, R. Mooney, W. Cohen, P. Ravikumar, and S. Fienberg. Adaptive name matching in information integration. IEEE Intelligent Systems, 18(5):16–23, Sep./Oct. 2003.

[9] M. Bilenko & R. Mooney, Adaptve Duplicate Detecton Using Learnable String Similarity Measures, KDD 2003.

[10] J. Bleiholder, F. Naumann. Data Fusion. ACM Computing Survey 2009.

[11] J. Bleiholder, F. Naumann. F. Declarative data fusion – syntax, semantics, and implementation. In Proceedings of the East European Conference on Advances in Databases and Information Systems (ADBIS), p. 58–73, 2005.

[12] J. Bleiholder. Data Fusion and Conflict Resolution in Integrated Information Systems. Dissertation, Hasso-Plattner-Institut, 2010.

[13] P. Bohannon et al., Conditional Functional Dependencies for Data Cleaning, ICDE 2007.

[14] M. Broecheler & L. Getoor, Probabilistic Similarity Logic, UAI 2010.

[15] Z. Chen et al., Exploiting context analysis for combining multiple entity resolution systems, SIGMOD 2009.

[16] P. Christen, Automatic record linkage using seeded nearest neighbour and support vector machine classificaton., KDD 2008.

[17] P. Christen. Data Matching – Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection. Data-Centric Systems and Applications, 2012.

[18] M. Cochinwala et al., “Efficient data reconciliaton”, Information Sciences, 2001.

[19] A. Das Sarma et al., “An Automatic Blocking Mechanism for Large-Scale De-duplication Tasks”, CIKM 2012.

[20] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. OSDI'04: Sixth Symposium on Operating System Design and Implementation, 2004.

[21] Xin Luna Dong, Felix Naumann. Data Fusion – Resolving data conflicts in Integration. VLDB 2009.

[22] Wenfei Fan, Floris Geerts. Foundations of Data Quality Management. Synthesis Lectures on Data Management № 29, 2012.

[23] Wenfei Fan, Dependencies revisited for improving data quality, PODS 2008.

[24] Fellegi, Ivan; Sunter, Alan. A Theory for Record Linkage. Journal of the American Statistical Association 64 (328): pp. 1183–1210. 1969.

[25] Venkatesh Ganti, Anish Das Sarma. Data Cleaning, A Practical Perspective. Synthesis Lectures on Data Management № 36, 2013.

[26] Lise Getoor, Ashwin Machanavajjhala. Entity Resolution for Big Data. 19th ACM SIGKDD Conference on Knowledge Discovery and Data Mining. Chicago: ACM SIGKDD, 2013.

[27] R. Gupta & S. Sarawagi, Answering Table Augmentaton Queries from Unstructured Lists on the Web, PVLDB 2(1), 2009.

[28] Mauricio Hernández, Georgia Koutrika, Rajasekar Krishnamurthy, Lucian Popa, Ryan Wisnesky. HIL: a high-level scripting language for entity integration. EDBT'13 Proceedings of the 16th International Conference on Extending Database Technology. P. 549–560, 2013.

[29] T. Herzog et al., Data Quality and Record Linkage Techniques, Springer, 2007.

[30] Kolb, L.; Thor, A.; Rahm, E. Dedoop: Efficient Deduplication with Hadoop Proc. 38th Intl. Conference on Very Large Databases (VLDB) / Proc. of the VLDB Endowment 5(12), 2012.

[31] Hanna Köpcke, Andreas Thor, Erhard Rahm. Evaluation of entity resolution approaches on real-world match problems. Proceedings of the VLDB Endowment, Volume 3, Issue 1–2, September 2010, P. 484–493.

[32] Hanna Köpcke, Erhard Rahm. Frameworks for entity matching: A comparison. Data & Knowledge Engineering, Volume 69, Issue 2, February 2010. P. 197–210.

[33] A. Marcus et al. Human-powered Sorts and Joins. PVLDB 5(1), 2011.

[34] Felix Naumann, Melanie Herschel. An Introduction to Duplicate Detection. Synthesis Lectures on Data Management. № 3, 2010.

[35] G. Papadias et al., Beyond 100 million entities: large-scale blocking-based resoluton for heterogenous data, WSDM 2012.

[36] P. Ravikumar & W. Cohen, A Hierarchical Graphical Model for Record Linkage, UAI 2004.

[37] A. Rajaraman and J. D. Ullman. Integrating information by outerjoins and full disjunctions. PODS1996.

[38] S. Sarawagi et al., Interactive Deduplication using Active Learning, KDD 2000.

[39] S. Tejada et al., Learning Object Identification Rules for Information Integration, IS 2001.

[40] J. Wang et al., CrowdER: Crowdsourcing Entity Resolution, PVLDB 5(11), 2012.

[41] Tom White. Hadoop: The Definitive Guide. O'Reilly Media; Third Edition. 2012.

[42] W. Winkler, Overview of Record Linkage and Current Research Directions, Research Report Series, US Census, 2006.

[43] Jaql Overview: Jaql, a query language for JavaScript Object Notation (JSON), 2011. https://code.google.com/p/jaql/wiki/JaqlOverview

[44] IBM InfoSphere BigInsights Version 3.0, Jaql reference. 2014. http://www-01.ibm.com/support/knowledgecenter/SSPT3X_3.0.0/com.ibm.swg.im.infosphere.biginsights.jaql.doc/doc/c0057749.html

[45] Apache Hadoop 2.4.1, 2014. http://hadoop.apache.org/

[46] MapReduce Tutorial, 2013. http://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html

[47] Apache Pig Project, 2014 http://pig.apache.org/

[48] The Apache Hive data warehouse, 2014. http://hive.apache.org/

[49] Cosine similarity. http://en.wikipedia.org/wiki/Cosine_similarity

[50] Term frequency–inverse document frequency. http://en.wikipedia.org/wiki/Tf%E2%80%93idf

[51] Monge-Elkan Distance Function. http://www.gabormelli.com/RKB/Monge-Elkan_Distance_Function

[52] String metric. http://en.wikipedia.org/wiki/String_metric

Programming of the Entity Resolution and Data Fusion Methods while Implementing ETL in the Hadoop Environment

A. Vovchenko, L. Kalinichenko, D. Kovalev

The paper is devoted to the problem of Entity Resolution and Data Fusion implementation in the context of big data integration. Entity resolution cares of Duplicate Detection, Deduplication, Record Linkage, Object Identification, Reference Matching, and other ETL-related tasks. Data fusion is the final step in the data integration process. This paper gives a short overview of methods for entity resolution and data fusion techniques. Then the paper presents the techniques for programming of the entity resolution and data fusion methods for implementing of the ETL process in the Hadoop environment.