Integrating Open Government Data with Stratosphere for more Transparency

Arvid Heise and Felix Naumann

Hasso Plattner Institute, Potsdam, Germany
firstname.lastname@hpi.uni-potsdam.de

Abstract

Governments are increasingly publishing their data to enable organizations and citizens to browse and analyze the data. However, the heterogeneity of this *Open Government Data* hinders meaningful search, analysis, and integration and thus limits the desired transparency.

In this article, we present the newly developed data integration operators of the Stratosphere parallel data analysis framework to overcome the heterogeneity. With declaratively specified queries, we demonstrate the integration of well-known government data sources and other large open data sets at technical, structural, and semantic levels. Furthermore, we publish the integrated data on the Web in a form that enables users to discover relationships between persons, government agencies, funds, and companies. The evaluation shows that linking person entities of different data sets results in a good precision of 98.3% and a recall of 95.2%. Moreover, the integration of large data sets scales well on up to eight machines.

Keywords: Data integration, data cleansing, schema mapping, record linkage, data fusion, parallel query processing, map-reduce

1. Integrating Open Government Data

The wealth of freely available, structured information on the Web is constantly growing. Especially governments and public administrations are increasingly publishing their data to enable organizations and citizens to browse and analyze the data. Most prominently, data.gov for the US and data.gov.uk for the UK are contributing thousands of data sets over a vast set of domains, such as public spending, health care, traffic, agriculture, etc.

However, simply publishing the data on the Web does not guarantee more transparency. On the one hand, the large size of the data sets render manual inspection of the data futile. On the other hand, the technical, structural, and semantic heterogeneity of the data prevents meaningful automatic processing and analysis. Furthermore, complex data analysis tasks need additional background information, such as family or business relationships.

We base our research on the Stratosphere parallel data analysis framework[1], which we extend with data integration operators to help to declaratively specify the integration of large data sources. We demonstrate how the integration of governmental data and other popular data-providing projects, such as DBpedia and Freebase, make it possible to create a single data set with a wide and deep set of facts about entities. The integrated data sets facilitates complex queries over the relationships of the different types of entities in a straight-forward manner.

Apart from data consumers within government organizations we distinguish two types of users with a special interest in such data sets: On the one hand professionals including data journalists and employees of (non-government) organizations, and on the other hand interested citizens.

Data journalism analyzes and filters large data sets to create news stories [1]. The journalists investigate publicly available data, such as the Afghan War Diary released on whistle-blower platforms. As a prerequisite, they need simple access to interesting and relevant data sets. While individual data sets might already contain valuable material, integration of multiple data sets often yields new insights: Recognizing that a politician sponsors large earmarks to a particular industry is already interesting, but recognizing that the politician's spouse is a known lobbyist for that industry might be news.

[1] www.stratosphere.eu

Individual interested citizens might not be willing or able to formulate such questions, so we also provide the searchable and browsable GovWild portal[2]. However, in this article, we focus on the data journalists.

Contributions and article structure. We illustrate the heterogeneity of well-known data sets with a motivating complex query and highlight the integration challenges in Section 2. Our contributions in this article are the following:

- We develop data cleansing operators for the Stratosphere framework (Section 3). Moreover, we embed the operators into the declarative query language Jaql.
- In Section 4, we evaluate the operators and show that linking person entities of different data sets results in a good precision of 98.3% and a recall of 95.2%. Furthermore, the integration of large data sets scales well on up to eight machines.

We also present briefly the Web portal GovWild to access an already integrated data in Section 5, discuss related work in Section 6, and finally conclude in Section 7.

2. Investigating Governmental Money Flows

In this section we follow the imaginary data journalist Alice, who wants to investigate nepotism in the United States. She wants to analyze the official governmental data sets to find suspicious money flows and then manually inspect the records to find out whether they are actually cases of nepotism. The analyzing query could be: *For each member of congress, find all earmarks awarded to organizations or subsidiaries that have employed a relative of that member of congress.*

To find interesting money flows, she starts by looking for appropriate data sources that contain congress members and their relatives, funds awarded to legal entities (companies), and employment information of the legal entities. Next, she uses our Stratosphere system to integrate the data to an integrated data source with the schema shown in Figure 1. Finally, she formulates a query that extracts records that are potential cases of nepotism. Each of the steps is elaborated in the following paragraphs.

2.1. Finding and Integrating Data Sources

Alice primarily needs data sets containing records of money flows. In the federal sector, there are usually two publicly available data sources: First, agencies may engage in contracts with private companies.

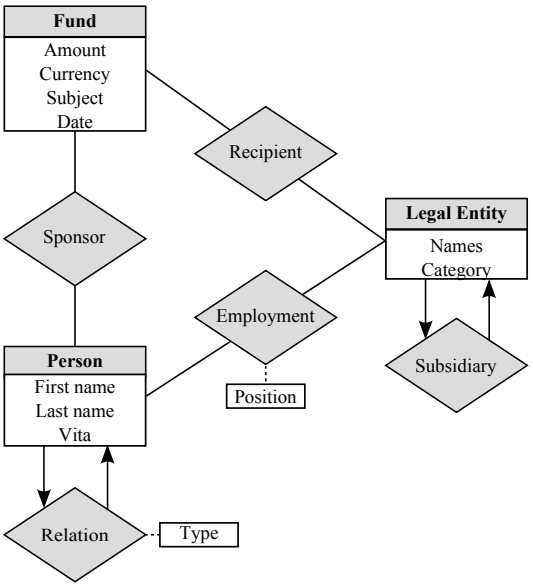


Figure 1: Excerpt of the integrated schema

As required by the Federal Funding Accountability and Transparency Act, these expenses have been listed on the USA spendings website[3] since 2007. Second, congress members may direct long-term funds to private companies, non-governmental organizations, or federal states. The presidential Office of Management and Budget publishes the so-called earmarks on its public website[4].

These two data sources offer reliable information about the sponsor and the receiver of the funds. To normalize names and title of the congress members, as well as to add some family relations, Alice decides to integrate the US Congress data source with the two former data sources. Furthermore, Alice still needs more details about legal entities, especially the subsidiary relationship between companies, and their key employees. Alice discovers that Freebase has two specific data sets that cover these relationships. She also includes data sets about politicians to find more relatives.

Table 1 summarizes the extractable entity and relationship types in the data sources. Obviously, the data of these sources overlap somewhat; for instance, a politician might be mentioned in Freebase and in the Congress data set. Recognizing such overlap increases the amount of information available for that politician.

To obtain an integrated data set, Alice will integrate the data sources technically, schematically, and semanti-

[2] www.govwild.org

[3] http://www.usaspending.gov/

[4] http://earmarks.omb.gov

Contained Entity	Spending	Earmarks	Congress	Freebase
Fund	X	X		
Person		X	X	X
Legal Entity	X	X	X	X
Public Employment		X	X	X
Industry Employment				X
Family Relation			X	X
Subsidiary				X

Table 1: Extractable entities in US-related data sources

cally following the classic approach of data integration. First, the data sets have to be downloaded and converted to the same format to eliminate the technical heterogeneity. Second, individual data sources often contain specific data errors that are addressed in source-specific data scrubbing operations. Additionally, the attributes of the data sources have to be mapped to the integrated schema to solve schematic heterogeneity and the values need to be normalized to a canonical form to remove semantic heterogeneity between data sources. Finally, the records of different sources representing the same real world entity are fused and links between the fused records are established to create the integrated data set. Alice uses Stratosphere to formulate the complex query shown in Figure 2.

Obviously, she needs a good understanding of the domain and the data sources and is able to formulate such a complex query. A typical way to explore the data sources is to download the different data sources and use tools such as Google Refine[5] to discover the relevant parts of the schemata and unclean attributes. In the following, we describe the different data sources with their unique characteristics in more detail.

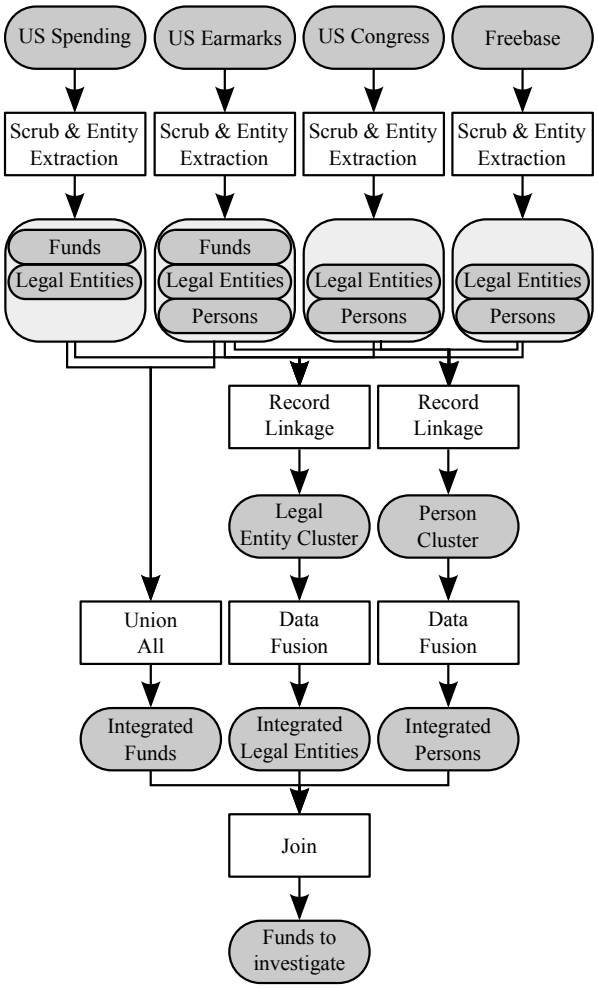


Figure 2: Complete data integration & query to find potential cases of nepotism

2.2. USA Spending

The website http://www.usaspending.gov/ provides the data sets of spendings in several formats starting from the year 2001, and it visualizes important trends for all federal states. It forms the largest data source for Alice's query consisting of approximately 1.7 million entries for the year 2009.

The data set is already the result of an integration of eight major data sources. Among them are federal procurement and award systems, contractor registries, and a commercially maintained company repository. Furthermore, the Federal Procurement and the Federal Assistance Award Data Systems are information integration projects themselves managing the data of over 60 and 30 agencies, respectively.

As the schema of USA Spending is a superset of all the attributes of the integrated data sources, the entries have the overwhelming number of 176 attributes. Although all of them are documented, Alice has to sometimes guess which attribute is suited best, e.g., whether the value of *obligated amount*, *base and exercised options value*, or *base and all options value* should be used as the value of *expense* in the global schema.

Another consequence is that very many of the attributes are empty or filled with default values. Additionally, Alice discovers data conflicts between different records, e.g., for the same company the *annual revenue* in several entries differs strongly. A formal schema definition is missing and fields such as *modnumber* have different data types. Finally, she finds obvious data errors, such as the field *annual revenue* with mentions over a

[5] http://code.google.com/p/google-refine/

trillion dollars for one company.

2.3. US Earmark

In comparison to USA Spending, the website of the US Earmarks lists the entries of the earmark database in tabular form only. The earmarks are hierarchically organized at the levels of the spending committee, agency, and bureau. For each year, Alice can browse these levels individually or download all entries as a CSV file.

Since 2008, the database is designed to accurately identify the congressional sponsor. Therefore, Alice can extract person entities and employment relations in addition to the information about the sponsoring agency, the contractor, and the earmark itself. The data sets of the years 2008 to 2010 contain approximately 50,000 earmarks each. Unfortunately, every year has a different schema with between 17 and 132 attributes. Hence, Alice needs to integrate every year independently.

When an earmark has more than one sponsor or recipient, the earmark entry is split into multiple records, which Alice needs to group to collect all information about the earmark. Furthermore, Alice wants to filter some earmarks of the most recent data set that are placeholders and do not contain a valid value for the monetary amount.

2.4. US Congress

The online US Congress directory[6] provides biographic information of all former and current congress members in about 95,000 entries. Unfortunately, the data set is not available for download but is only accessible through a search form. Alice would have to either ask the webmaster for the complete data set or request the help from a programmer to write a crawler (we actually crawled the website).

The result list of the search form contains the attributes name, date of birth and death, position in the congress, party, state, number of the congress, and an id. Unlike the earmarks and the spendings, the US Congress data set contains additional semi-structured values in the form of the biography attribute, which lists family relations, amongst others. An exemplary entry starts with

> CLINTON, Hillary Rodham, (wife of President William Jefferson Clinton), a Senator from New York...

As this entry shows, the name is complemented by the profession or position of the relative, if known. We found no misspellings or abbreviations in the positions, which indicates an automatic generation of the relationship texts. In order to parse such entries, a dictionary of positions is sufficient to distinguish the positions and the first names.

In cases where several persons share the same name, the directory, fortunately, adds the date of birth and death to the name, which allows effective record linkage

> CLINTON, George, (uncle of George Clinton [1771-1809], ...

2.5. Freebase

Given the distribution of entity type in Table 1, Alice decides to use Freebase[7] to complement the data sources. Freebase is a large, community-based open data set covering every topic of the contributors' interests. However, the comparably low quality of the data makes integration with the other data sources challenging.

For Alice only company and politician data subsets are relevant; about 1.4 million entries. These data sets contain facts about companies, board members, and subsidiary companies, as well as information about presidents, congress members, politicians, and persons in general.

Since 2008 Freebase evolved fast, which led to several problems. First, the schema in Freebase is very heterogeneous, because several concurrent type hierarchies exist. This originates from the restriction in Freebase that only the original editor of a type or base may change it. Second, in 2010 the ID system changed completely, rendering all previously extracted links invalid.

A major drawback of the relevant company and person data sets is the high number of missing values. Figure 3 visualizes the percentage of the non-null values for the 73,074 company entries. Only 20 of 32 attributes have more than 0.5% non-null entries and the number of present values per entity follows a Zipf distribution. As the other data sources contain only information about locations and the revenue of companies, Alice can solely depend on the company name to link the records in roughly 60% of the cases.

Similarly, the name of the person is the only attribute that is consequently set in all 1,276,278 person entries (Figure 4). The date of birth and gender is available for

[6] http://bioguide.congress.gov/

[7] http://www.freebase.com/

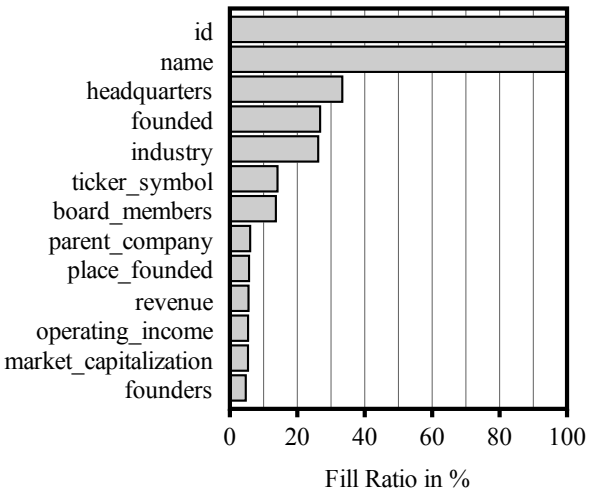


Figure 3: Ratio of filled attributes for all Freebase entities of type 'company'

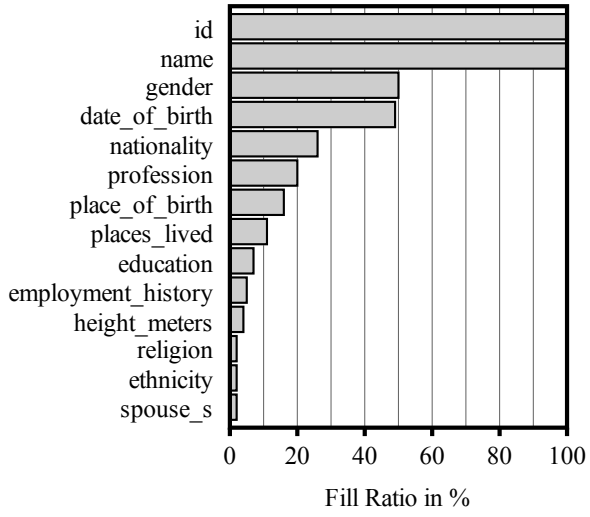


Figure 4: Ratio of filled attributes for all Freebase entities of type 'person'

only half of the entries. Especially the year of birth is crucial to distinguish two persons with similar names. The Freebase data sets should complement the information of the official data sources with additional relationships between persons and companies. However, only 1% to 2% of the records contain family relationships and only 6% have an employment history.

Due to the described problems, data sets from Freebase should be carefully integrated to avoid worsening the quality of the resulting integrated data set. The integration of messy data sets usually takes much more time than clean data sets because different data flaws have to be treated independently. If the gained information is too small, the time invested into the integration does not pay off. In Alice's use case, the data is valuable to identify relationships between persons and legal entities and she decides to integrate the data sets nonetheless. As she manually verifies all results in the end, data errors in the integrated data set does not result in wrong conclusions.

After Alice explored the individual data sets, she formulates the query in Stratosphere.

3. Data Cleansing with Stratosphere

In this section, we present the extension of Stratosphere to cleanse data in a potentially large-scale data integration project. All data cleansing operators are first-class citizens and fully embedded in the optimization model. Therefore, Alice can embed the data cleansing operators into a complex query context to answer specific questions. For example, she might focus her analysis on specific persons, states, or companies.

First we describe the general architecture of Stratosphere and then we explain the technical, schematic, and semantic integration with the new operators.

3.1. Stratosphere

Stratosphere is a joint research project that explores how the elasticity of Clouds can be exploited to process analytical queries massively in parallel. It includes an declarative query language to specify queries consisting of basic operators as well as domain-specific high-level operators. Unlike most traditional DBMS, Stratosphere inherently supports text-based and semi-structured data and is thus well suited to perform data cleansing.

The parallelization of data cleansing with Stratosphere decreases the computation time and thus facilitates the following exemplary use cases:

Large-scale data sources: The Stratosphere system is built to process web-scale data in a massively parallel fashion. With data cleansing algorithms that scale well with the number of nodes, big data sets with tens or hundreds of gigabytes can still be processed in reasonable time.

Incremental tuning: Data integration is often considered a one off affair. However, in reality, the immense variety and number of parameters, settings, and mappings require many passes until the integrated data is in a state ready to be used further. A shorter round trip time allows more trial runs within a given time slot, and minimizes wait-time for the developer.

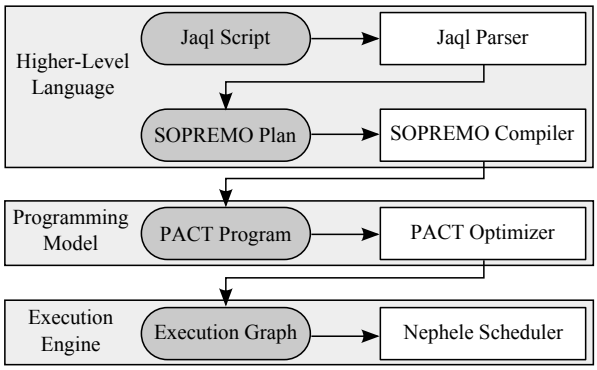


Figure 5: The Stratosphere stack

Fast changing data sources: In near real-time application areas, such as fraud detection in financial transactions [13], new data is constantly added and needs to be processed in a timely manner.

Virtual integration: When only a small part of the integrated data set is needed, partial or virtual integration may strongly speed up the query processing. Stratosphere reoptimizes queries and may early on reduce the amount of data to be cleansed (note that the rewriting rules of cleansing operators is still future work).

The design of Stratosphere is heavily influenced by Hadoop[8] and aims to overcome some shortcomings of Hadoop. It processes directed acyclic data flow graphs with semantically richer base operators in addition to the traditional map and reduce. A built-in optimizer adjusts the data flow graph to the actual data set for improved performance. The complete Stratosphere stack is shown in Figure 5.

Nephele constitutes the fault-tolerant, parallel execution engine of Stratosphere [24]. It interprets job graphs and distributes the tasks to the network nodes. Nephele is specifically designed to run on heterogenous environments. Furthermore, it fully exploits the elasticity of Clouds by automatically booking and releasing instances during runtime, whenever needed.

The parallel programming model PACT (PArallelization ConTracts) [2] builds upon the Nephele execution engine. It is a data-oriented generalization of Map/Reduce and consists of second-order functions that execute first-order functions with specific semantics. Aside from the well-known *map* and *reduce*, Stratosphere introduces three additional PACTs, which process two input streams. Similar to *map* and *reduce*, these functions are responsible for partitioning the input of the data sources and call the user-defined first-order function.

The *cross* PACT generates the cartesian product of the two input tuple sets and transfers the pairs independently to the wrapped user-defined function. For join-related operations, the *match* PACT guarantees that the user-defined function is called for every pair of entries with the same key. Lastly, *cogroup* groups all entries with the same key, but still allows to distinguish between the data sources.

While all PACTs could be efficiently implemented in Hadoop by exploiting the existing user-definable functions as shown by Dittrich et al. [10], the extended expressiveness of the PACTs in Stratosphere leads to more concise and comprehensible query plans [3]. Moreover, Stratosphere adaptively optimizes the query plans and benefits from the additional semantic information of the contracts [4]. Lastly, Stratosphere introduces a complex cost model with these operations, which allows better usage of computation units to minimize the effective monetary costs.

To help data analysts to formulate common analysis queries without delving into depth of the parallel programming model, high level query languages, such as Jaql [5], Pig, and Hive, have been evolving for Hadoop. As a consequence, Stratosphere offers an abstraction for declarative languages with the Stratosphere data and processing model (SOPREMO) for a wide support of different query languages. SOPREMO is an extensible framework for high-level operators including domain-specific complex operators such as the data cleansing operators and basic relational operators such as selection and projection.

As Stratosphere aims to facilitate a possibly large range of structured and semi-structured applications, SOPREMO uses Json as its data model and thus abstracts from the underlying key/value data model of PACT. Operators implemented in SOPREMO process one or more input streams of Json objects and apply transformation expressions to produce one more output streams. A stream is basically a list of objects where the actual order of items is not relevant and undefined in most cases. Only special operators such as Sort define a globally applicable order.

The primary query language of Stratosphere is an adoption of Jaql (Json Query Language) [5]. It provides transformation-based query operators that fit well into the data cleansing use case. The Jaql parser transforms the scripts into Abstract Syntax Trees and maps them to the SOPREMO operators to form a SOPREMO plan. The SOPREMO compiler then successively translates the plans into a PACT program. The SOPREMO layer,

[8] http://hadoop.apache.org

```
1  $dirty_earmarks = read hdfs('UsEarmark.json');
2  $nick_names = read hdfs('UsNickNames.json');
3
4  $scrubbed_earmarks = scrub $dirty_earmarks
       with {
5    // normalization with built-in expressions
6    amount: [type decimal, amount * 1000],
7    // normalization with user-defined functions
8    sponsorLastName: [required,
9      NormalizeName(sponsorLastName)],
10   sponsorFirstName: [required,
11     NormalizeName(sponsorFirstName),
12     replace sponsorFirstName with $nick_names
           default sponsorFirstName],
13 }
```

Listing 1: Scrubbing the US Earmarks Data Set

however, is out of scope of this article; we depict for each integration step the final PACT program generated from the respective query script.

The following subsections demonstrate how Alice specifies the different parts of her analytical query.

3.2. Data Scrubbing

Alice starts by scrubbing the individual data sources with the script in Listing 1. For each (nested) value, she specifies a scrub expression that is a list of constraints and transformations. The scrub operator checks if the values of each record meet the respective constraints and either corrects invalid values or filters the record. In contrast, the transformations map the current value to exactly one new value. The first scrub expression ensures that each *amount* is a number and normalizes the amount from thousand dollars to dollars. The next two expressions use a user-defined function to normalize the names that can be either formulated directly in Jaql or as a registered Java function. Finally, the last scrub expression additionally uses the replace operator to resolve nicknames to the canonical name, e.g., Bill to William.

The current set of validation rules allow to specify that the field must be present and non-null as well as to define the expected type, range, and set of possible values. We plan to extend them with set-based constraints such as uniqueness and automatic outlier detection depending on the standard deviation. A transformation may be an arbitrary Jaql expression that operates on the current record. Currently, there is a basic set of predefined functions for string formatting and array manipulation. Additionally, the replace operator may be used to lookup values in a reference table and replace them by standardized values, e.g., abbreviations or misspellings

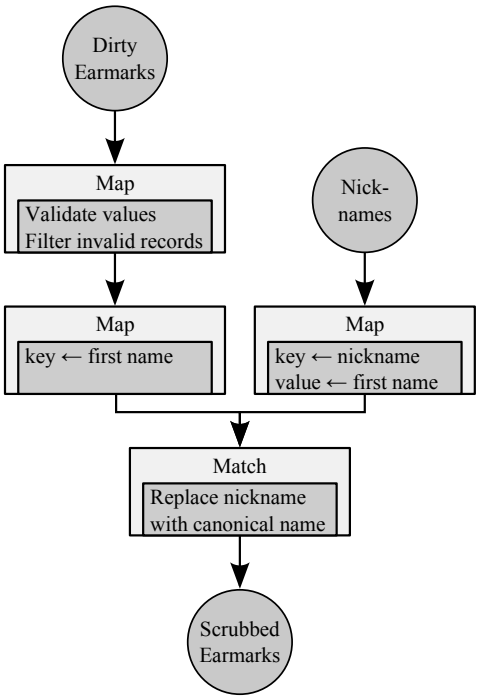


Figure 6: Scrub with embedded replace operator

of the states of the US are normalized. If the value is not in the dictionary, the default expression is evaluated.

The implementation of the scrubbing operator is basically a *map* that filters invalid and uncorrectable records. For each scrub expression it locates the respective value, and applies the constraints and transformation successively. After all constraints have been met and all transformations have been applied, the record with the updated values is emitted. However, the nested replace operator cannot be executed in the map as the auxillary nickname list is not available inside the map.

Therefore, the replace operator is pushed outside the scrub operator and the partial query is rewritten to

```
replace $earmark.sponsorFirstName in
    $scrubbed_earmarks with $nick_names
    default $earmark.sponsorFirstName
```

The replace operator uses two *maps* to create the lookup dictionary and extract the values to replace. It then partitions the corresponding records and lookup values together with a *match*. The first-order function replaces the value with the lookup value or evaluates the default expression if no lookup value has been found.

After Alice specified the basic rules of scrubbing operations for the US Earmark data set, she continues to extract the different entities. If she encounters more data errors in the different sources, she has always the possi-

```
1  extract from $earmark in $scrubbed_earmarks
       into {
2    $funds = group by $earmark.earmarkId {
3      id: generateId('earmark'),
4      amount: sum($earmark[*].amount),
5      currency: 'USD',
6      date: {
7        year: $earmark[0].enactedYear
8      },
9      subject: $earmark[0].shortDescription, ...
10   }
11   $recipients = group by $earmark.recipient {
12     id: generateId('earmark_person'),
13     names: [$earmark.recipient],
14     receivedFunds: transform $earmark into {
15       id: $funds[$earmark.earmarkId].id,
16       amount: $earmark.amount
17     },
18     category: $earmark[0].recipientType, ...
19   }
20 };
21
22 write $funds to hdfs('Earmark_Funds.json');
```

Listing 2: Simultaneous extraction of entities

bility to refine existing or define additional scrub rules.

3.3. Entity Extraction

To formulate her query, Alice wants to extract the funds, persons, and legal entities as well as their respective relationships from the designated data sets (see Table 1).

The design of the extract operator is optimized to maintain the crucial information of relationships within a data set. Therefore, the operator allows the simultaneous specification of the mapping rules for all dependent entities types as shown in Listing 2. In the script, all three entity types fund, person, legal entity are logically extracted at the same time. Note that legal entities appear in two roles: as the recipient of a fund and as the public employer of the sponsor.

As briefly described in Section 2.3, one earmark is split into multiple records for each sponsor and recipient. The extraction of an earmark thus groups all relevant records by the *earmarkId* and chooses or aggregates the respective values (Line 2–10). For example, the total amount spent is the sum of the amounts of the individual records.

The other entities are extracted similarly with one noticeable exception. Within the extraction operator, the result sets remain indexed by their grouping key and other extraction definition may directly access these indices. Line 14 demonstrates how the reified relationship *received_funds* links to the *id* of the corresponding fund.

```
1  $sponsors = read hdfs('EarmarkSponsors.json');
2  $members = read hdfs('CongressMembers.json');
3
4  $linked_persons = cluster records
5      $sponsor in $sponsors, $member in $members
6    where jaccard(lastName) >= 0.9 &&
7      [ 5 * jaroWinkler(firstName),
8        5 * jaroWinkler(lastName), ...] >= 0.8
9    partition on
10     [ removeVowels(lastName)[0:3],
11       state + firstName[0:2] ]
12   into { $sponsor, $member };
```

Listing 3: Record linkage of persons

The current set of recipient records is transformed into an array of two elements. While the value of the amount is directly copied from the record, the fund *id* is resolved with the index notation on the fund result set.

The implementation of the basic extraction definitions is essentially a *map* and *reduce* aggregation job. Additionally, all index notations (see Line 15) are resolved in three steps. First, the assignment of the index notation to the target field is substituted by the key value *earmarkId*. Second, an auxiliary lookup table *$earmarkId_to_fundId* is built transparently. Finally, a strict replace operator without default expression is added after the main extraction process.

```
replace $recipients.received_funds[*].id
    with $earmarkId_to_fundId
```

At the end of the scrubbing process, all types of entities are separately accessible for each data source. The respective data sets of the same type are then merged in the following record linkage step.

3.4. Record Linkage

An important step of the data integration process identifies the real world entities across the different data sources. Only these connection effectively allow Alice's complex query over several data sets. For governmental money flows, persons and legal entities appear in several data sources and need to be linked, while there should be no fund in more than one data source.

The Stratosphere query language introduces two new operators to ease the formulation of the conditions and parameters of a record linkage task. First, the *link record* operator performs basic entity resolution and returns the links between two records. Second, the *cluster record* operator additionally groups all transitively connected records and adds all unconnected records. Listing 3 demonstrates how Alice clusters two data sources with person entities.

In this query, the similarity is declared as two conditions (Lines 6–8). First, a strict condition on the *last name* allows only minimal errors in the respective values. The second condition combines several similarities on attributes with possibly different weights. In this case, both the *first* and the *last name* have a weight of five. The wrapping array represents the combination of the individual similarity values, which normalizes the similarity to $[0; 1]$ by definition. The result is finally compared to the more relaxed threshold of 0.8.

When integrating more than two data sources, rules for specific combinations can be created by referencing the bound variable. For example, the following rule compensates incorrect splits in the names of US congress members during the crawl.

```
jaroWinkler(firstName + middleName,
  $member.firstName + $member.middleName) > 0.9
```

With the *into* statement, the user specifies which values of the correspondences have to be selected to identify the entities in the fusion phase. In this case, the complete records for both persons are emitted. As this is the default configuration, it could be omitted. If needed, the similarities of all evaluated conditions can be included in the resulting correspondence record with *similarities*. It is planned to provide further information about the match to ease debugging.

Optionally, a data partitioning method can be specified on one or more attributes. The naïve approach of comparing every two entities of the given data sources is not feasible for Alice's query, because it would result in more than two billion comparisons for the persons in US Congress and US Earmarks alone. The script specifies to use multi-pass blocking [12] (Lines 9–11). For the first pass, persons are grouped by their first three consonants of the last name. In the second pass, all persons are regrouped using the state and the first two letters of their first name to match persons with either severe errors in their last name or a name change after marriage.

The cluster records operator calculates the transitive closure of the pairs of corresponding entities to form clusters of entities. The underlying algorithm corresponds to graph component enumeration in an undirected graph, where entities are vertices and correspondences become edges. Cohen presented an iterative approach of finding components for Hadoop [8]. Alexandrov et al. showed that the most compute-intensive part of enumerating the triangles in a graph can be well optimized in Stratosphere [3].

Note that the actual numbers for thresholds and weights depend strongly on the domain and the query. For example, when establishing links between Linked Open Data entities, false positives would result in a major loss of data quality as consuming application might draw wrong conclusions from the false connections. Nevertheless, since Alice reviews all results from the complete query in any case, she could choose to use even smaller thresholds and allow more false positives to occur in exchange for less false negatives.

The translation of the link operators to PACT programs are defined by the applied partition algorithm implementations. Blocking results in a *map* call to extract the blocking key and a subsequent *match* to compare tuples with the same key from different sources as depicted in Figure 7a.

In the case of a sorted neighborhood method [16], an adoption of the RepSN Hadoop algorithm by Kolb et al. [18] is applied (Figure 7b). It basically partitions the data depending on the sorting key and creates overlapping partitions by prepending a prefix to the key. In the shuffle phase, the partitioner uses the prefixes to partition the entries, while the sorting phase sorts the entries within a partition in the same run. To handle two input data sources, we replaced the *reduce* of the original algorithm RepSN with a *cogroup* and consequently maintain separate window buffers for the data sources inside the user-defined functions of the *cogroup*.

If no blocking method is provided, the resulting plan may vary strongly, as the record linkage operator tries to exploit properties of the similarity function as good as possible. Figure 7 illustrates an alternative plan for the link operation, which uses the first strict condition. The combination of the set-similarity function Jaccard with a relatively high threshold, allows an efficient execution of the algorithm introduced by Vernica et al. [22]. The main idea is that two strings of length ten need to have eight of nine 2-grams in common to achieve a Jaccard similarity of 0.9. Hence, the algorithm first counts the frequencies of the tokens in a preprocessing step. The algorithm then uses only the least frequent N-gram as a join criterion to drastically reduce the amount of pairs that need to be compared with the given conditions.

The corresponding PACT program consists of two parts (see Figure 7). First, in a preprocessing step, the join attribute of the smaller data source is tokenized and the frequency is counted. Afterwards, the tokens are sorted by frequency and globally aggregated in one sorted list. In the second part, the *cross* contracts processes each earmark sponsor or congress member using the list of sorted tokens. It also tokenizes the join attribute and emits the least frequent tokens. If too few tokens of a congress member are found in the list, the entry is not emitted, because it is impossible to pass the threshold.

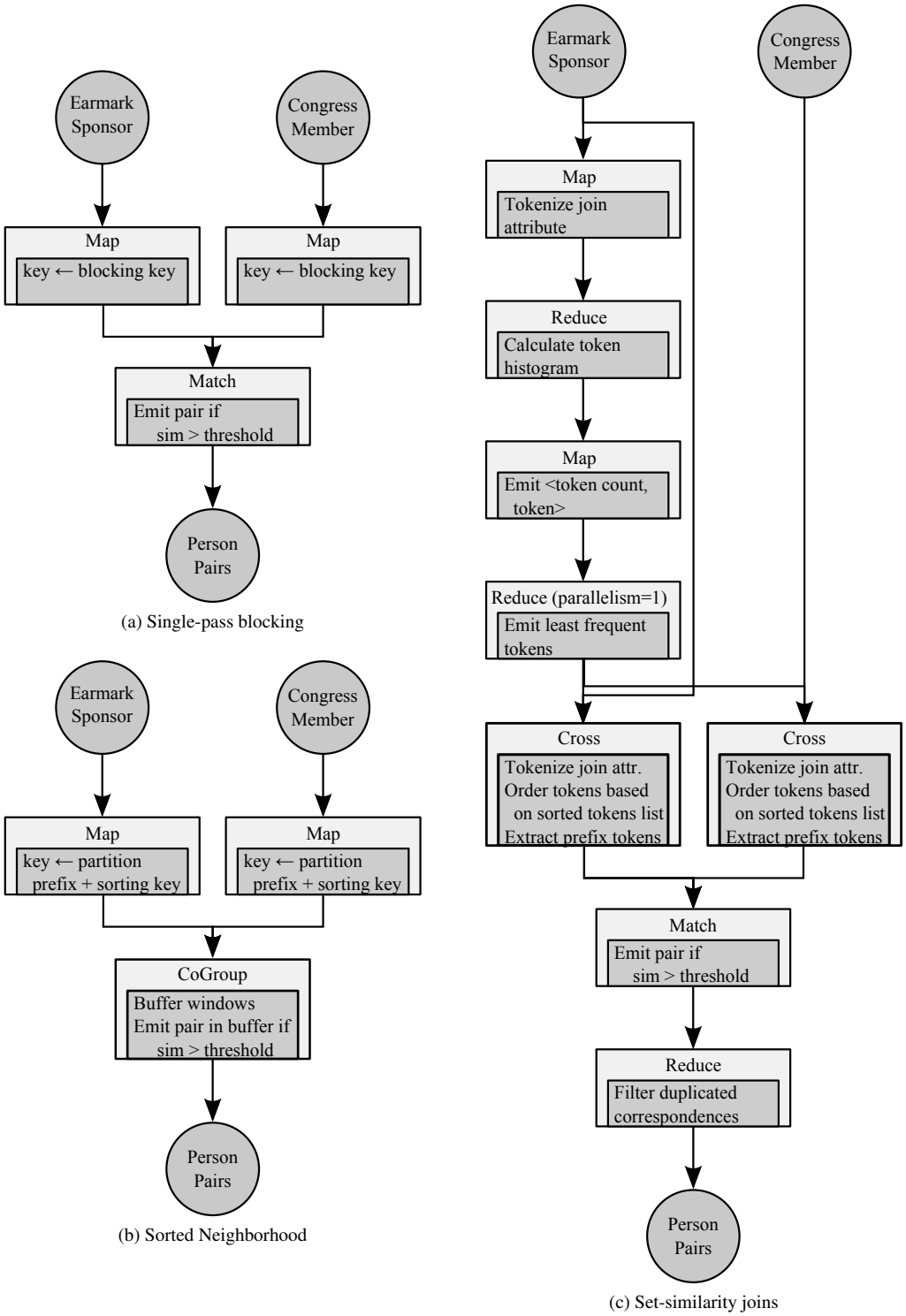


Figure 7: PACT programs for three partition algorithms of the link records operator

```
1  $fused_persons = fuse $cluster = [$sponsor,
       $member] in $linked_persons
2    with weights {
3      $member: 0.99,
4      $sponsor: 0.99,
5      $sponsor.addresses: 0.7, ...
6    }
7    into {
8      id: generateId('person'),
9      lastName: [vote(abbr), longest],
10     firstName: [vote(abbr), first],
11     addresses: MergeAddresses, ...
12     originalRecords: $cluster[*].id,
13   }
14   update $cluster.relative[*].id in $cluster
         with id;
15
16 write $fusedPersons to hdfs('Persons.json');
```

Listing 4: Data fusion of persons

Alice's query offers an inherent parallelization as the record linkage can be applied to person and legal entities independently. Further parallelization is realized through multiple passes and the implementation algorithm of the record linkage strategy.

3.5. Data Fusion

In general, links between entities of several data sources can be used in two ways. First, they may be published to connected the entities of the different data sources, e.g., finding new links between Linked Open Data sources. Second, the linked records can be ultimately merged into one entry with data fusion. The former approach would leave the interpretation of the value and the conflict resolution to the application developer that wants to use the data, while the latter approach usually looses data that does not fit in the applied conflict resolution strategies.

As Alice is not interested in the relationships between data sets but between entities, she decides to merge the relevant attributes with the fusion script shown in Figure 4.

The *into* clause specifies the conflict resolution rules for the attributes. The resolution functions are applied as long as there is more than one value. For example, the last name is determined by the value with the highest weight or the longest of two equivalently high weighted value candidates.

Additionally, the *update* clause allows to define foreign key relationships that should be updated with the merged id. The clause is optional and is syntactic sugar for a replace operator that replaces the old value with the new value in a temporary set created in an additional step.

```
replace $fused_persons.relative[*].id
    with $oldId_to_newId
```

At the end of the script, weights for the different data sources and fields are recursively defined. The weights of nested structures accumulate by multiplication, e.g., the actual weight of *sponsors.addresses* is $0.99 * 0.7$. These weights are passed to the conflict resolution functions as an contextual parameter. The resolution function may choose to honor or to ignore the meta information depending on the semantics. For example, the *first* function does not use the weights, while *vote* uses the Dempster-Shafer theory of confidence [20] to determine the suitable candidate as described in the following.

With the Dempster-Shafer theory of confidence, we address the question which value v_i of a set of conflicting values we believe most. We start with some initial beliefs in the respective source and translate it into a belief function. Then we combine the respective belief functions of alternative values. Finally, we choose the value with the highest belief.

Given a set of possible values $V = \bigcup\{v_i\}$, we first assign the basic belief masses $m_v : 2^V \to [0; 1]$ to all occurred value combinations with the following properties $m_v(\varnothing) = 0$ and $\sum_{V \in 2^V} m_v(V) = 1$.

For simplicity, we assume non-array values in this article resulting in $m_v(V) = \begin{cases} w & \text{if } |V| = 1 \\ 0 & \text{else} \end{cases}$ with w being the manually assigned weight to the source and attribute.

Next, we combine the different masses with Dempster's rule of combination

$$m_1 \oplus m_2(\{v\}) = \frac{\sum_{V_i \to v} m_1(\{v\}) m_2(V_i)}{1 - \sum_{V_j \nleftrightarrow v} m_1(\{v\}) m_2(V_j)} \qquad (1)$$

The nominator collects the evidence of all $V_i \subseteq V$ that support the specific value v denoted with $\to$. In the example script, Alice uses the abbr function that supports *'John'* with the value *'J.'* but not vice versa. The denominator normalizes small mass sums for dissimilar values.

As an example, consider having the three possible values 'J.', 'John', and 'Bill' constituting V. The initial belief mass functions is $m_1(J.) = 0.7$, $m_1(V) = 0.3$, $m_2(John) = 0.8$, $m_2(V) = 0.2$, $m_3(Bill) = 0.9$, and $m_3(V) = 0.1$. The combination of the first two masses yields $m_{12}(J.) = (0.7 * 0.2)/1 = 0.14$ because 'J.' is only right if we do not believe 'John'. Further $m_{12}(John) = (0.7 * 0.8 + 0.3 * 0.8)/1 = m_2(John)$ since

```
join
  $sponsor in $persons,
  $relative in $persons,
  $fund in $funds,
  $recipient in $legal_entity,
  $subsidiary in $legal_entity
  where
  ($relative.id in $sponsor.relatives[*].id
  or $relative.id = $sponsor.id) and
   $fund.id in $sponsor.enacted_funds[*].id
  ($subsidiary.id in $recipient.subsidiaries
      [*].id
  or $subsidiary.id = $recipient.id)
  into {
  ...
}
```

Listing 5: Alice's query for potential cases of nepotism

'J.' supports our belief in 'John'. Finally, the last combination m_{123} results in the following denominator $d = 1-(0.8*0.9+0.14*0.9) = 0.154$ and masses $m_{123}(J.) = (0.14*0.1)/d = 0.09$, $m_{123}(John) = (0.8*0.1)/d = 0.52$, and $m_{123}(Bill) = (0.9*0.2*0.3)/d = 0.35$. Although we initially had more belief in 'Bill', the two evidences for 'John' outweighs it.

The implementing PACT program is a straightforward *map* that applies all the resolution strategies to the occurring conflicts. If updates are specified, it transparently retains all original values for the given field, and uses a *map* for each update to create a lookup list from old values to new values. Subsequently, a replace operator is executed for each update.

3.6. Analysis of the Integrated Data Set

Alice finally specified all the integration operations and can now formulate her query on the integrated data set. Note that most of the integration rules as well as the analytical query evolve with time and that Alice would probably start to query the data set at an earlier point in time.

Nevertheless, Listing 5 shows the Jaql script that executes Alice's initial query to find nepotism:

For each member of congress, find all earmarks awarded to organizations or subsidiaries that have employed a relative of that member of congress.

All conjunctions are translated into subsequent *matches* on the join key, while disjunction result in unions of parallel joins where a union is implemented with *reduce*.

Alice receives 263 potential cases that she can thoroughly investigate. Most of these funds have been enacted to counties for which a relative of the sponsor works. Whether this is an actual case of nepotism, or a regular spending can only be decided by an expert with profound domain knowledge such as Alice.

Operator	Syntax
Data Scrubbing	**scrub** input **with** rules
Entity Extraction	**extract from** input **into** entities
Record Linkage	**link**/**cluster records** input **where** similarity_condition **partition on** partition_key **into** target_expr
Data Fusion	**fuse** input **with weights** weight_map **into** result
Dictionary Lookup	**replace** input **with** dictionary **default** default_expr

Table 2: Summary of data cleansing operators

This section demonstrated how the data integration can be formulated in a query with Stratosphere. We introduced five new operators in Jaql summarized in Table 2 that allow a concise and declarative specification of the data integration steps. All operators are translated to PACTs and thus integrated into the Stratosphere stack.

The Stratosphere project is at an early stage of development. The latest, stable version is publicly available as open source at http://stratosphere.eu. Some of the presented functions are experimental at the time of writing and will be merged in the next stable release. Interested developers are welcome to evaluate their PACT implementations for the operators.

4. Evaluation

We evaluated the Stratosphere platform on parts of Alice's use case to assess the effectiveness and efficiency. All experiments ran on a cluster of eight machines (Intel Core Duo 2x2.66GHz) and 2 GB RAM using Stratosphere 0.1.1 and java 1.6.0_16 with 1.5 GB RAM per JVM task manager.

For the experiments, we closely observed the scalability of the two integration scripts that integrate all three US sources. The first script integrates two comparably small data sets, namely the earmarks and the congress members. It scrubs the data sets, resolves persons and legal entities in both data sets, and then fuses the entity clusters. The second script similarly integrates the spendings into the previously integrated data set. We omit the results from the integration of Freebase, as the integration of the small politician data sets

behaved like the first script and the integration of the larger company data set had a similar performance to the second script. Table 3 lists the input format and sizes of the most recent data sets of 2010 and the number of resulting entities per type.

Source	Format	Size (in MB)	Resulting Entities (in k) Person	Legal	Fund
US-Earmark	CSV	54.9	2.3	15.6	9.8
US-Congress	HTML	22.2	33.0	0.4	0.0
First script	Json	13.9	2.3	15.8	9.8
US-Spending	CSV	2,508.3	0.0	1,662.8	3,158.1
Second script	Json	1,261.2	2.3	1,667.1	3,167.9

Table 3: Sizes of the data sets in the evaluation

4.1. Effectiveness of Record Linkage

To measure the quality of the record linkage for both linkable entity types, i.e., person and legal entity, we manually inspected a sample of 1000 linked pairs to calculate the *precision*:

$$precision = \frac{true\ positives}{true\ positives + false\ positives}$$

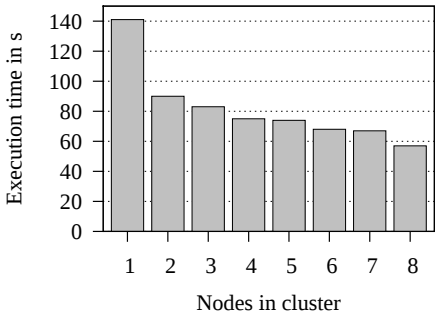
We evaluated the precision on the results of the first script for persons and on the results of the second script for legal entities. As all earmarks have to be sponsored by at least one congress member, we could additionally assess the *recall* for the record linkage between earmark sponsors and congress members:

$$recall = \frac{true\ positives}{true\ positives + false\ negatives}$$

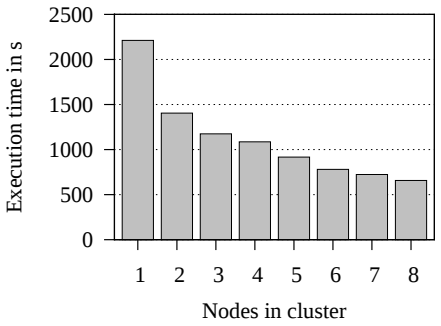
However, we did not evaluate the recall for legal entities as this would have required a large amount of manual inspections of negatives.

The record linkage of persons achieved good results with a precision of 98.3% and a recall of 95.2% as it uses the first name, last name, and temporal information (birth year and congress membership). In comparison, the precision in resolving legal entities is lower with 85.6% because it is mainly based on the name of the company.

Finally, we compared our distributed implementations of the record linkage algorithms with single-threaded implementation in the duplicate detection toolkit[11]. The single-threaded implementations conducted the same comparisons as our distributed implementations. That means that our algorithms work correctly in respect to the single-threaded implementations.



(a) Execution time of first script (see Table 3)



(b) Execution time of second script (see Table 3)

Figure 8: Execution times

Therefore, further improvements in the quality thus can only be achieved by applying a better similarity measure or more suitable record linkage algorithm.

4.2. Efficiency

We ran the scripts three times on up to eight nodes separately and averaged the execution times in Figure 8. The execution time of the first script strongly decreased for two nodes in comparison to the non-distributed execution at one node because the record linkage of the two entity types could be easily distributed between the two nodes.

However, further nodes did not substantially decrease the time for two reasons. First, the individual record linkages as the most compute-intensive part did not benefit much from the parallelization because the distributed candidate comparison did not amortize the additional data shipping due to the comparably small number of candidate pairs. Second, the current implementation of the input parsers and output writers cannot be executed in parallel, which adds a linear overhead to the parallel algorithms.

The integration of the large data set in the second script ran up to twenty times longer. The execution

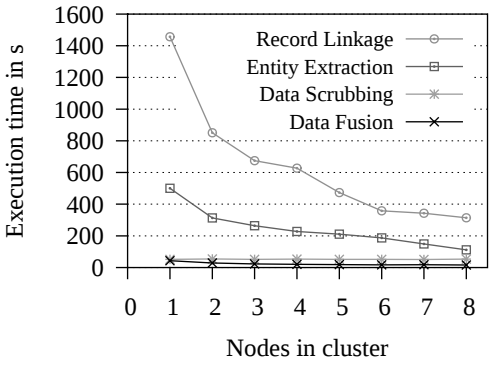


Figure 9: Execution time of the operators in the second script

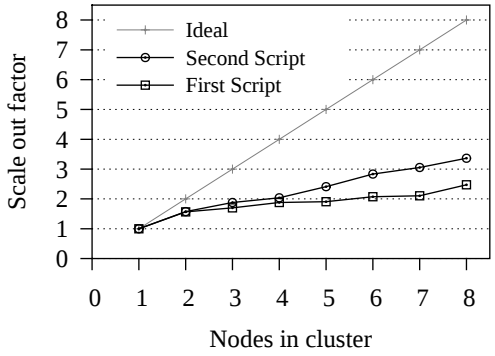


Figure 10: Scale out for both integration scripts

time steadily decreased when adding new nodes as the amount of comparisons in the record linkage is large enough to amortize the shipping costs.

We also determined how well the individual operators performed for the large integration script. Because Stratosphere uses a consumer-producer pattern, we cannot directly measure the execution time of one operator. Instead, we stopped the integration script three times before and three times after one operator ran and calculated the difference of the mean times. As illustrated in Figure 9, the record linkage operator experiences a noticeable speed up while the entity extraction operator only gains little time during the scale out. The speed up of latter operator is limited by the *group by* aggregations. Data fusion and data scrubbing perform in near-constant time because they read or write the data into the sequential input and output.

Figure 10 visualizes the scale out factors that were calculated by dividing the execution time on one node by the execution time of all cluster configurations. The integration of the large script leaves more potential for scaling out. However, the scale out properties of both scripts are still far from the ideal line. Some algorithms, especially the transitive closure are currently basic implementation that do not scale well. In the future, we will work on better algorithms that are transparently hidden behind the presented operators and available for the query optimizer to generate more efficient plans.

To put these measurements into perspective, the process to create the data for the original GovWILD portal can serve as a baseline (see Section 5). The same set of steps are performed, albeit on all sources of Table 4 and not just the three US sources. There, a mixed workflow of Jaql (on Hadoop) and Java programs have an overall runtime of approx. 5h. Thus, development cycles to improve the workflow are tremendously shortened with our approach.

5. GovWild Portal

Besides professional data analysts with profound domain knowledge, we also want to increases the transparency for interested citizens. While former is probably able to formulate a complex query, most citizen are most likely overwhelmed by the cheer number of possibilities and options. Therefore, we maintain the GovWild portal to increase the transparency of governmental money transactions by providing a materialized integration of major governmental data sources [6]. It builds upon IBM's information integration project Midas [19] and currently integrates US American, European, and German data sources related to money flows as listed in Table 4.

Source	Format	#Tuples	#Attributes
US-Spending (2009)	XML	1,724,654	141
US-Earmark (2008)	CSV	43,524	37
US-Congress	HTML	95,253	8
EU-Financial	HTML	60,733	14
EU-Parliament	HTML	3,133	18
DE-Party Donations	HTML	1,102	5
DE-Agricultural	HTML	103,652	8
DE-Bundestag	HTML	629	10
Freebase	TSV	1,389,602	54

Table 4: Data sources integrated into GovWild

The GovWild portal http://govwild.org provides the integrated data set as a download as well as a Web interface to explore it. The portal is based on the Information Workbench by fluid Operations AG, which helps users

interact with semantic Web data [15]. The user can investigate the large amount of data in several ways. He can search for politicians, examine their spending statistics, and browse through their funds. He may look for a company or legal entity, view the received funds, and work their way back to the sponsor. Additionally, funds are directly accessible and can be sorted by specific criteria such as the amount of money spent. Moreover, featured queries help users to find interesting connection and may be an entry point to learn formulating simple queries. Finally, arbitrary SPARQL queries can be processed.

The Information Workbench inherently handles RDF triples only. A customized transformation script from fluid Operations triplifies the Json files to RDF triples in the N-Triples format. In order to capture all attributes of GovWild in the RDF triples, we have created a custom ontology. We map persons to a subclass of foaf:Person, all legal entities to a subclass of foaf:Agent, and funds to a customly defined class.

Since Freebase represents a major hub of the Linking Open Data cloud and also constitutes a major data source for GovWild, the RDF data set of GovWild is connected to the Linking Open Data cloud. The portal offers the OWL ontology, the integrated data in Json and RDF for download.

6. Related Work

In this section, we review related work for the integration of Open Government Data and for scalable data integration frameworks similar to Stratosphere.

6.1. Government Data Portals

Generally, the integration of Open Government Data is an interesting intersection of the Semantic Web and data integration research areas. Semantic Web researcher usually aim to include the data sources in the Linking Open Data Cloud (LODC). The heterogeneity of the data sources remains unchanged, but interlinkage of the RDF subjects allows to derive complex relationships over several intermediate entities.

The model project of enriching publicly accessible Open Government Data with semantic information is data.gov.uk [21]. The portal serves as a central contact point for data publishers and consumers for official non-personal data in the UK. It helps governmental agencies to prepare their data sets for publication on the Comprehensive Knowledge Archive Network and manages meta data for all data sets such as descriptions and statistics. Consumers can browse through the hierarchically organized data sets. The rapid development of the portal stems from the broad support of the British government and the involvement of the main advisors Sir Tim Berners-Lee and Professor Nigel Shadbolt. It provides access to over 5,600 data sets at the time of writing, most of them need to be integrated into the LODC.

A similar repository has been launched for the US on data.gov. It currently lists over 300,000 data sets. However, the integration in the Semantic Web is still in the beginning. The collaborating Tetherless World Constellation at the Rensselaer Polytechnic Institute researches two ways of interlinking the entities [9] with the LODC. On the one hand, they allow portal visitors to manually link entities. On the other hand, they explore machine learning techniques to find these connections automatically. Overall they found about 5,800 links to DBpedia, GeoNames, and GovTrack.

Both portals follow the same approach. The data ownership remains at the agencies and thus the sources are highly heterogeneous. The interlinkage leverages users or applications to formulate complex queries, but they must also resolve occurring conflicts on their own. In comparison, the GovWild portal integrates fewer data sources, but additionally aligns schemata and fuses data. The data is partly integrated into the LODC as it maintains the links to the original Freebase records.

A major disadvantage of officially funded portals became obvious in the bill for the fiscal year 2011 as the US government cut the budget for the electronic government fund to $8 million from $34 millon (SEC. 1552. in the bill[9]). The future of the affected data.gov remains uncertain, while most agencies will still publish their data (independently).

6.2. Scalable Data Integration

Since the amount of data is steadily increasing, but the computation power per unit is only slowly raising, it becomes important to perform data integration in parallel. Beyer et al. demonstrated that it is feasible to achieve that goal with Hadoop and Jaql [5]. However, the development is rather time-consuming and error-prone, since data cleansing operations are not inherently supported. In the following, we shall present parallelizable frameworks that include data cleansing by design.

The key for massive parallelization is a declarative description of the cleansing tasks to facilitate optimization and distribution of the tasks. Galhardas et al. created custom SQL operations in AJAX to formulate such an declarative description and translate it to SQL

[9] http://rules.house.gov/Media/file/XML_112_1/WD/FINAL2011.XML

for execution [14]. While the proposed approach has the advantage that the query optimization techniques of DBMS could be exploited, it relied heavily on user-defined function for the data transformation tasks. They also needed to modify the DBMS itself to support the full range of their functions.

Similarly, Herschel and Manolescu extended XQuery to perform data cleansing with XClean [17]. In direct comparison with AJAX, they generated pure XQuery queries and thus did not need to modify the query processing engine. Additionally, the integrated data transformation were expressive enough to not depend on the user-defined functions, even though they also supported external function definitions.

Tasks specified in both AJAX and XClean can be executed in parallel if the query processing engine supports parallelization. However, the languages SQL and XQuery are not designed for massive parallelization and it is highly unlikely that the processing engines will fully exploit the complete computation power of larger clusters in the near future.

In comparison, Silk is inherently developed to run on a Hadoop cluster [23]. It interprets declarative job descriptions formulated in the Silk Link Specification Language to automatically find similar entries across multiple RDF data sets. Nonetheless, Silk is currently not embedded in a larger query system and provides no means to fuse the data leaving the conflict resolution to processing applications.

Lastly, there are three relevant projects at the UC Irvine that together perform parallel data cleansing. The FLAMINGO Project focuses on similarity search and data integration queries through sophisticated index techniques. The second project is ASTERIX[10] which provides a data and programming model for scalable analysis of large amounts of semi-structured data. The ASTERIX Query Language allows to declaratively formulate similarity searches and fuzzy joins in a similar fashion as XQuery. Since both projects are closely connected, it is to expected that the parallel data cleansing algorithms of FLAMINGO are ported to ASTERIX [22]. Ultimately, the queries are executed on Hyracks, the parallel execution runtime [7]. The programs in Hyracks form an acyclic directed graph very similar to Stratosphere. However, the operations on the lowest level are rather database-oriented and might be limited to the domain of structured and semi-structured data. Additionally, there are currently no data cleansing operators in the query language.

[10] http://asterix.ics.uci.edu/

7. Conclusion and Future Work

In this article, we described the challenges that arise in the integration of Open Government Data. We motivated the integration process with the data journalist Alice who investigates nepotism. She identified four major data sources that yield the necessary entities and relationships. However, since the data sets are maintained independently and even official data sources contain errors and incomplete data, the integration is challenging. For example, often only one or two attributes can be effectively used for record linkage as many values are missing. Consequently, the chances increase to either assign incorrect link or miss correspondences.

We demonstrated how Alice formulates a complex query to integrate the data sources in Stratosphere. Stratosphere interprets Jaql queries enriched with operators specific to data integration tasks. We explained the translation of the queries to the basic parallel programming model that is a generalization of Map/Reduce. Furthermore, the evaluation showed that most implementations scale well for large data sets and the outlined settings result in a good precision and recall.

Nevertheless, Stratosphere requires good domain knowledge and the user must be able to formulate a non-trivial query. To give interested persons the opportunity to view the data without these skills, we publish the result of the integration of several sources on the GovWild portal http://govwild.org for online browsing or download.

For the future, all of the presented new integration operators need to be fully included into the cost model of Stratosphere. The greatest challenge is to estimate how many clusters the record linkage will produce. The number of clusters determine the runtime of the fusion and link operations. Moreover, the maximum size of the clusters also greatly influences the conflict resolution functions of the fusion phase. Random sampling of unclean data is not sufficient, as it selects a duplicate-free subset with a high probability. Thus, more sophisticated focused sampling methods need to be evaluated.

8. Acknowledgements

We thank Markus Freitag, Claudia Lehmann, and Andrina Mascher for their great contributions to GovWILD. This research was supported by the German Research Society (DFG grant no. FOR 1306) and by an IBM Innovations Award.

References

[1] How to be a data journalist, http://www.guardian.co.uk/news/datablog/2010/oct/01/data-journalism-how-to-guide, 2010. Accessed 08/10/2011.

[2] A. Alexandrov, D. Battré, S. Ewen, M. Heimel, F. Hueske, O. Kao, V. Markl, E. Nijkamp, D. Warneke, Massively parallel data analysis with PACTs on nephele, Proceedings of the VLDB Endowment (PVLDB) 3 (2010) 1625–1628.

[3] A. Alexandrov, S. Ewen, M. Heimel, F. Hueske, O. Kao, V. Markl, E. Nijkamp, D. Warneke, MapReduce and PACT - comparing data parallel programming models, in: Proceedings of the Conference Datenbanksysteme in Büro, Technik und Wissenschaft (BTW), GI, Bonn, Germany, 2011, pp. 25–44.

[4] D. Battré, S. Ewen, F. Hueske, O. Kao, V. Markl, D. Warneke, Nephele/PACTs: A programming model and execution framework for web-scale analytical processing, in: Symposium on Cloud Computing (SoCC), ACM, 2010, pp. 119–130.

[5] K.S. Beyer, V. Ercegovac, R. Krishnamurthy, S. Raghavan, J. Rao, F. Reiss, E.J. Shekita, D.E. Simmen, S. Tata, S. Vaithyanathan, H. Zhu, Towards a scalable enterprise content analytics platform, IEEE Data Engineering Bulletin 32 (2009) 28–35.

[6] C. Böhm, F. Naumann, M. Freitag, S. George, N. Höfler, M. Köppelmann, C. Lehmann, A. Mascher, T. Schmidt, Linking open government data: What journalists wish they had known, in: Proceedings of the International Conference on Semantic Systems (I-SEMANTICS), ACM, 2010, pp. 34:1–34:4.

[7] V. Borkar, M.J. Carey, R. Grover, N. Onose, R. Vernica, Hyracks: A Flexible and Extensible Foundation for Data-Intensive Computing, in: Proceedings of the International Conference on Data Engineering (ICDE), 2011, pp. 1151–1162.

[8] J. Cohen, Graph twiddling in a MapReduce world, Computing in Science and Engineering 11 (2009) 29–41.

[9] L. Ding, T. Lebo, J.S. Erickson, D. DiFranzo, A. Graves, G.T. Williams, X. Li, J. Michaelis, J. Zheng, Z. Shangguan, J. Flores, D.L.M. J, J.A. Hendler, TWC LOGD: A portal for linked open government data ecosystems, Web Semantics: Science, Services and Agents on the World Wide Web 9 (2011) 253–268.

[10] J. Dittrich, J.A. Quiané-Ruiz, A. Jindal, Y. Kargin, V. Setty, J. Schad, Hadoop++: Making a Yellow Elephant Run Like a Cheetah (Without It Even Noticing), Proceedings of the VLDB Endowment (PVLDB) 3 (2010) 518–529.

[11] U. Draisbach, F. Naumann, DuDe: The duplicate detection toolkit, in: Proceedings of the International Workshop on Quality in Databases (QDB), Singapore.

[12] A.K. Elmagarmid, P.G. Ipeirotis, V.S. Verykios, Duplicate record detection: A survey, IEEE Transactions on Knowledge and Data Engineering (TKDE) (2007) 1–16.

[13] W. Fan, J. Li, S. Ma, N. Tang, W. Yu, Interaction between record matching and data repairing, in: Proceedings of the International Conference on Management of Data (COMAD), SIGMOD '11, ACM, New York, NY, USA, 2011, pp. 469–480.

[14] H. Galhardas, D. Florescu, D. Shasha, E. Simon, AJAX: An extensible data cleaning tool, in: Proceedings of the ACM International Conference on Management of Data (SIGMOD), ACM, 2000, p. 590.

[15] P. Haase, A. Eberhart, S. Godelet, T. Mathäß, T. Tran, G. Ladwig, A. Wagner, The Information Workbench. Interacting with the Web of Data, Technical Report, fluid Operations AG, 2009.

[16] M.A. Hernández, S.J. Stolfo, The merge/purge problem for large databases, in: Proceedings of the ACM International Conference on Management of Data (SIGMOD), ACM, 1995, pp. 127–138.

[17] M. Herschel, I. Manolescu, Declarative XML Data Cleaning with XClean., in: J. Krogstie, A.L. Opdahl, G. Sindre (Eds.), Proceedings of the Conference on Advanced Information Systems Engineering (CAiSE), volume 4495 of *Lecture Notes in Computer Science*, Springer, 2007, pp. 96–110.

[18] L. Kolb, A. Thor, E. Rahm, Parallel sorted neighborhood blocking with mapreduce, in: Proceedings of the Conference Datenbanksysteme in Büro, Technik und Wissenschaft (BTW), pp. 45–64.

[19] A. Sala, C. Lin, H. Ho, Midas for government: Integration of government spending data on hadoop, in: Proceedings of the International Workshop on New Trends in Information Integration (NTII), ACM, 2010, pp. 163–166.

[20] G. Shafer, A Mathematical Theory of Evidence, Princeton University Press, Princeton, 1976.

[21] J. Sheridan, J. Tennison, Linking uk government data, in: Proceedings of the Linked Data on the Web Workshop (LDOW).

[22] R. Vernica, M.J. Carey, C. Li, Efficient Parallel Set-Similarity Joins Using MapReduce, in: A.K. Elmagarmid, D. Agrawal (Eds.), Proceedings of the ACM International Conference on Management of Data (SIGMOD), ACM, 2010, pp. 495–506.

[23] J. Volz, C. Bizer, M. Gaedke, G. Kobilarov, Discovering and maintaining links on the web of data, in: A. Bernstein, D.R. Karger, T. Heath, L. Feigenbaum, D. Maynard, E. Motta, K. Thirunarayan (Eds.), International Semantic Web Conference (ISWC), volume 5823, Springer Berlin Heidelberg, 2009, pp. 650–665.

[24] D. Warneke, O. Kao, Nephele: Efficient parallel data processing in the cloud, in: Workshop on Many-Task Computing on Grids and Supercomputers (SC-MTAGS), ACM, 2009.