HUMBOLDT-UNIVERSITÄT ZU BERLIN

Ontology construction on a cloud computing platform

Exposé for a Bachelor's thesis in Computer science -

Knowledge management in bioinformatics

Tobias Heintz

1 Motivation

1.1 Introduction

PhenomicDB is a database of natural language descriptions of phenotypes from various species [Kahraman2005]. As such it contains a multitude of phenotype concepts, for example "optic nerve degeneration" or "decreased blood uric acid level".[1] Since these concepts are often interrelated, it is desirable to obtain an overview of the relationships among them. In order to achieve this, attempts are being made to extract concepts and their relationships from the texts and store them in suitable formats that are both comprehensive and accessible, see for example [Maedche2000] or [Cimian-02005]. These formats are required to be easily machine readable - unlike the natural language texts comprising PhenomicDB - and must be able to accurately represent the hierarchical nature of the data.

Ontologies are hierarchical structures that express the relationships among its constituents and are very useful to organize data such as phenotype concepts. Various ontologies for phenotypes exist, for example the *Mammalian Phenotype Ontology (MPO)* [Smith2004] or the *Human Phenotype Ontology (HPO)* [Robinson2008]. These ontologies have been created by hand from selected data sources.

1 Examples are taken from MPO, see [Smith2004].

However, no ontology exists that encompasses all phenotype concepts found in PhenomicDB, which is partially due to the large number of texts contained. Hence, one of the central problems in creating an ontology like the one outlined, is to develop an automated process for extraction of phenotype concepts from texts.

1.2 Existing work

Christoph Böhm's 2008 diploma thesis titled "Ontology construction from Phenotype Data" [Böhm2008] solves the problem of generating an ontology from the textual data in PhenomicDB. It splits the task up into four separate phases:

1. Concept definition
2. Concept discovery
3. Evidence discovery
4. Ontology construction

Concept definition is the process of determining which terms from the texts in PhenomicDB constitute phenotype concepts. This is achieved by comparing term frequencies in PhenomicDB to those found in other document collections, where significantly different distributions mark phenotype concept candidates.

Concept discovery locates the previously defined concept strings in the documents by employing a fuzzy search over the entire corpus.

Evidence discovery uses the positions of the concepts to determine relationships among them. Three algorithms are used: subsumption, which classifies concepts based on the frequencies of occurring together, Hearst patterns, which exploit certain linguistic features of the texts and GO score, which uses links of phenotype concepts to their associated genes via the *Gene Ontology (GO)* and tries to derive a classification from that.

Finally, ontology construction uses the classification data generated in the previous step to construct an ontology which contains all the discovered concepts and the relationships among them.

Each of these phases is implemented as a Java program and evaluated with respect to accuracy of the results. The application allows for fine-grained tuning using a number of parameters such as window size for fuzzy searching or whether or not to apply stemming. The thesis also explores these

parameters and attempts to identify optimal settings.

The existing solution is single-core oriented and makes heavy use of relational database systems such as *MySQL*.[2] Due to the large amount of input data and intricate processing steps, a full execution of all phases takes several days. The author did not explore the possibilities of run-time optimization, for example parallel execution. This issue will be investigated in this thesis.

2 Aim

The aim of this thesis is to increase the performance and efficiency of the existing solution to the ontology construction task. This is achieved by porting the code from a single-core architecture to the cloud computing platform Stratosphere, where the code can be executed in parallel on multiple machines. Due to time constraints, only the first two phases of Böhm's solution will actually be implemented in this thesis.

The aim is furthermore to implement the solution in such a manner that an optimizer as introduced in [Hueske2012] will be able to optimize the solution through reordering of the application structure. This step requires careful design of the application components and especially the data model, as will be discussed in Section 3.3.

3 Approach

3.1 Stratosphere

Stratosphere is a joint project between Technische Universität Berlin, Humboldt Universität zu Berlin and the Hasso-Plattner-Institut in Potsdam. It is a cloud computing platform whose programming paradigm is a generalization of the *MapReduce* approach [Dean2008].

Stratosphere consists of an execution layer called *Nephele* [Battre2010], which is responsible for coordinating processing instances and dividing up data between them. It consists furthermore of a programming model involving so called *Parallelization Contracts (PACTs)*. This programming model is a generalization of the *MapReduce* approach, since it offers more flexibility regarding the choice of second level functions. Whereas *MapReduce* offers precisely two functions, *map* and *reduce*, Stratosphere has five different *PACTs* that can be used to describe the relationship of data processing steps.

2 http://www.mysql.com/

Several of these *PACTs* were designed to operate on multiple inputs contrary to the single input of the *map* and *reduce* functions [Battre2010, p. 5].

A *PACT* program consists of several distinct steps that are connected to one another through their inputs and outputs. They are data flow oriented programs. The data is structured as records that have multiple fields.

3.2 Port

Porting a sequential algorithm to a parallel platform involves multiple steps:

1. Analysis
2. Decomposition
3. Evaluation

First the algorithm is analyzed for its portability. Certain problems are inherently serial and thus impossible to execute in parallel, for example any calculation where step *n* depends on the result of step *n-1*. More general, the more dependencies exist in an algorithm, the more difficult it becomes to parallelize. In fact, phase 4 of Böhm's solution is impossible to properly parallelize, since the algorithm used modifies its own input with every iteration.

The input data is also examined for its properties in order to ensure reasonable partitioning. Ideally, the data consists of a set of similar items that are not interdependent, which means it is well suited for processing in a parallel environment.

The algorithm is then decomposed into smaller steps that each perform one distinct operation.[3] Communication paths are developed to allow the flow of data between algorithm components. In Stratosphere this is as simple as connecting the output of one component to the input of the next one.

While constructing the algorithm, a trade-off must be made between granularity and performance. Higher granularity means higher parallelism but also more communication overhead, which may effectively reduce the overall performance. The actual data is partitioned as fine-grained as possible according to the algorithm specifications.

3 In [Foster1995] the author calls this step "Partitioning".

3.3 Design

Each of the phases of the original solution is ported to a *PACT* program as illustrated in Figure 1.

The phases can be executed separately with intermediate output stored on a file system, or chained together to produce one large *PACT* program work-flow.

Code organized in *PACTs* however is inherently strictly encapsulated. Each component of the *PACT* program is in itself independent from the others, which allows for interesting optimization strategies, such as reordering of the entire program flow. In [Hueske2012], the authors formally introduce this strategy and implement an optimizer based on static code analysis, which can perform such optimizations. They also outline the requirements for achieving optimizations through reordering, which include devising a global data model, which must be structurally immutable. Hence the conception of the data model is another essential part of the design of the solution.

3.4 Evaluation

In a final step, the solution will be evaluated in varying settings, such as local multi-core execution and execution on a cluster, with respect to performance compared to single-threaded execution. The influence of certain parameters, like degree of parallelism or the size of input chunks, will also be evaluated.

Additionally, the solution will be tested for bottlenecks, i.e. components that take a very long time to execute, using simple timing techniques. Finally, the effect of applying *PACT* reorderings will likewise be investigated.

4 Literature

- Battre2010: D. Battré, S. Ewen, F. Hueske, O. Kao, V. Markl, and D. Warneke, "Nephele/PACTs: a programming model and execution framework for web-scale analytical processing," in *Proceedings of the 1st ACM symposium on Cloud computing*, New York, NY, USA, 2010, pp. 119–130.
- Böhm2008: C. Böhm, "Ontology Construction from Phenotype Data," Knowledge Management in Bioinformatics, Humboldt-Universität zu Berlin, 2008.
- Cimiano2005: P. Cimiano, A. Pivk, L. Schmidt-Thieme, and S. Staab, "Learning taxonomic relations from heterogeneous sources of evidence," *Ontology Learning from Text: Methods, evaluation and applications*, vol. 123, pp. 59–73, 2005.
- Dean2008: J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large

clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008.

- Foster1995: I. T. Foster, "Designing and building parallel programs: concepts and tools for parallel software engineering," Reading, Mass. [u.a.]: *Addison-Wesley*, 1995.
- Hueske2012: F. Hueske, M. Peters, M. J. Sax, A. Rheinländer, R. Bergmann, A. Krettek and K. Tzoumas, "Opening the Black Boxes in Data Flow Optimization," *38th International Conference on Very Large Data Bases (VLDBs),* Istanbul, Turkey, 2012.
- Kahraman2005: A. Kahraman, A. Avramov, L. G. Nashev, D. Popov, R. Ternes, H.-D. Pohlenz, and B. Weiss, "PhenomicDB: A Multi-Species Genotype/Phenotype Database for Comparative Phenomics," *Bioinformatics*, vol. 21, no. 3, pp. 418–420, Feb. 2005.
- Maedche2000: A. Maedche and S. Staab, "Discovering conceptual relations from text," in *Ecai*, Amsterdam, 2000, pp. 321–325.
- Robinson2008: P. N. Robinson, S. Köhler, S. Bauer, D. Seelow, D. Horn, and S. Mundlos, "The Human Phenotype Ontology: A Tool for Annotating and Analyzing Human Hereditary Disease," *The American Journal of Human Genetics*, vol. 83, no. 5, pp. 610–615, Nov. 2008.
- Smith2004: C. L. Smith, C.-A. W. Goldsmith, and J. T. Eppig, "The Mammalian Phenotype Ontology as a tool for annotating, analyzing and comparing phenotypic information," *Genome Biol*, vol. 6, no. 1, p. R7, 2005.

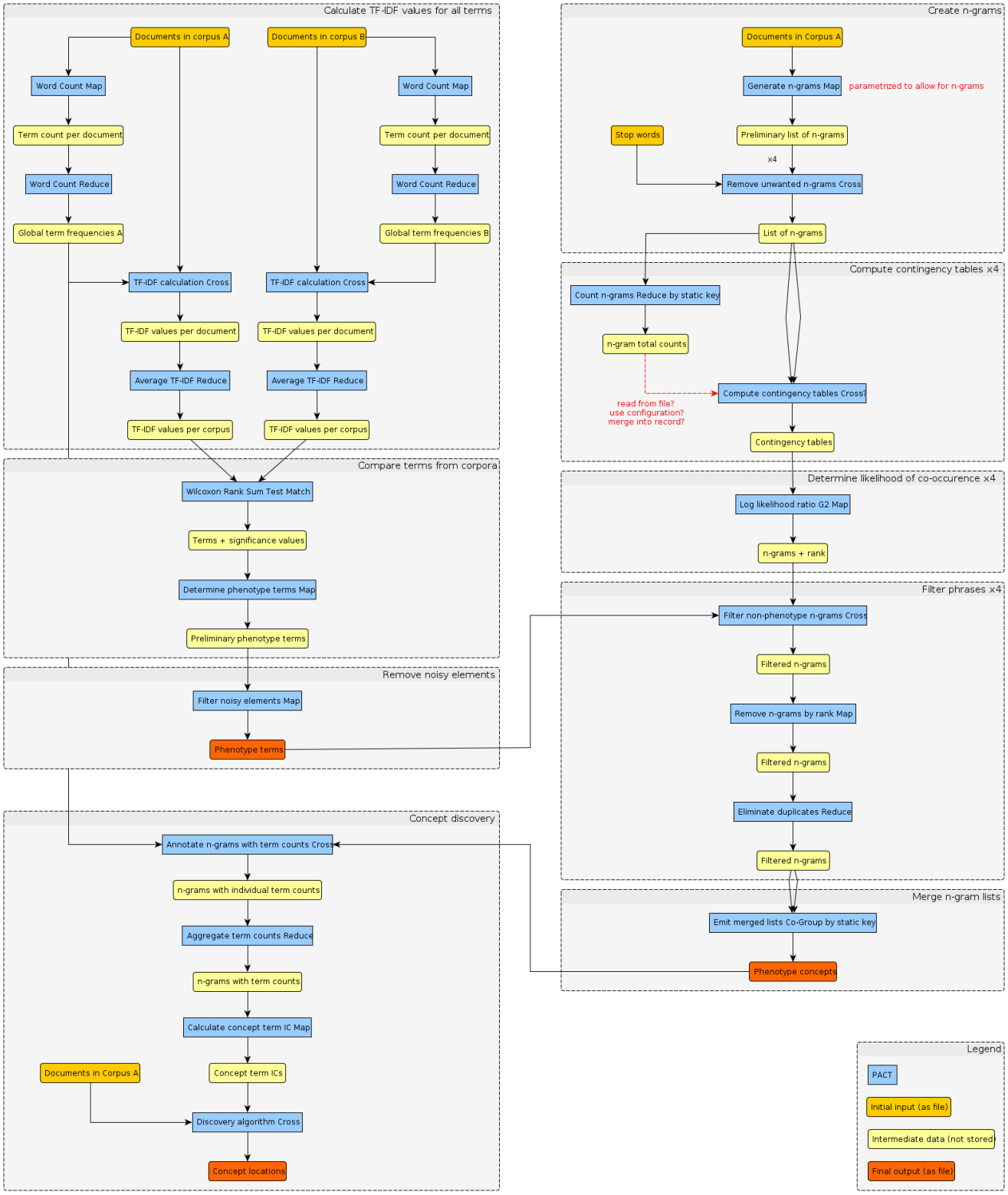

Figure 1: PACT overview