Multi-threading the YAGO Extraction

Weijia Shao
Computer Science
Universität des Saarlandes

A thesis submitted for the degree of Bachelor in Computer Science

Nov. 2010
1. Reviewer: Prof. Gerhard Weikum

2. Reviewer: Dr Martin Theobald
Acknowledgements

First I would like to thanks Prof. Gerhard Weikum give me the opportunity to write my Bachelor thesis in the Databases and Information Systems Departments of Max Planck Institute informatik. Secondly i would like to appreciate my advisor PhD Edwin Lewis-Kelham who helped me a lot with any problems during the process of my thesis. I also have to thanks all the people who work for the YAGO project. Last but not least I have to thank all of my friends and my family, I could not feel up to the task without their continuous helps and supports.
Abstract

In this thesis, the author presents a multi threaded framework for YAGO extraction. YAGO has been automatically derived from Wikipedia and WordNet. Because Wikipedia is a frequently updating resource it becomes vital to improve the performance of the YAGO extraction. Since there are many different extraction techniques, the extensibility of the framework is also very important. Therefore extraction has been broken into independent parts so that each processing element can execute its part of the extraction simultaneously. Consequently, the performance of the YAGO extraction has been considerably improved, while still keeping a flexible framework.
Contents

List of Figures vii

List of Tables ix

1 Introduction 1

2 Yago 3
  2.1 Yago Model 3
     2.1.1 Reification Graphs 4
     2.1.2 Query language 4
  2.2 Information Extraction 6
     2.2.1 Infobox Heuristics 6
     2.2.2 Type Heuristics 8
     2.2.3 Word Heuristics 9
     2.2.4 Category Heuristics 9
  2.3 Summary 10

3 Multi-threading Framework 11
  3.1 Structure 12
  3.2 Fact Data type 16
  3.3 Configuration 17
  3.4 Summary 19

4 Extensibility 21
  4.1 Adding a new source 21
  4.2 Adding a new extraction 22
  4.3 Summary 24
CONTENTS

5 Performance
5.1 Summary .......................................................... 28

6 Conclusion ..............................................................
6.1 Future Work ....................................................... 32

vi
# List of Figures

2.1 Infobox of Wikipedia ........................................... 7  
2.2 Category System of Wikipedia ................................. 8  
3.1 Multithreaded Extraction ..................................... 12  
3.2 WikiSourceReader ............................................. 13  
3.3 WikiSource .................................................... 14  
3.4 ExtractThread .................................................. 15  
3.5 Transplant Article Extractor ................................. 17  
4.1 Multithreaded Extraction With HardExtractor ............... 22  
4.2 WikiSource with Gender Extractor ............................ 23
List of Tables

5.1 Performance of the extraction ........................................ 26
5.2 Satisfied Relations ....................................................... 27
5.3 Mistaken Relations ........................................................ 29
LIST OF TABLES
1

Introduction

YAGO is a huge semantic knowledge base with high coverage and precision (? ). It has been automatically derived from Wikipedia and WordNet. It comprises entities and relations, and currently contains more than 1.7 million entities and 15 million facts (? ). The construction of YAGO ontology takes place in two stages (? ? ): First, different heuristics are applied to Wikipedia to extract candidate entities and its associated facts. Then, quality control techniques are applied. Each of these parts work independently, thus it is possible for us to implement a multi threaded framework for these independent parts of YAGO. Secondly, YAGO is implemented in JAVA, and therefore by taking full advantage of dynamic dispatch and subtype polymorphism of the JAVA language the software flexibility can be improved. This framework makes it possible to update YAGO as well as free the developer from troubles caused by adding new techniques to the old construction.

The slowest part of YAGO is the Article extractor, which has to parse an ever growing Wikipedia file; currently the size is 27GB. When a single threaded program initiates an I/O operation, a system call is made, and does not return until the I/O operation has been completed. In the intervening period, the entire program cannot run. This advantage of a multi threaded extraction allows it to separate the parsing and I/O. With multiple cores, the threads of the program naturally lend themselves to truly concurrent execution. Thus most programs are written to perform I/O synchronously. On the other hand, parsing the string in the main memory can also work in parallel, because each wikipedia page can be parsed independently. By parsing in
1. INTRODUCTION

parallel we can improve the run time of the article extractor.

Since YAGO requires adjusting and improving on each extraction round, and the Wikipedia file structure changes over time, a flexible framework is required. This extensibility is a distinguishing feature that separate frameworks from libraries or normal user applications. Extensions can be the addition of new functionality or modification of existing functionality. With a framework with good extensibility we can minimize the impact to existing system functions when new heuristic techniques are added, or old heuristics are changed.

In this thesis we present a multi threaded framework for YAGO extraction, where

- the first aim is to improve the performance
- the second aim is the extensibility

The remaining chapters are outlined follows. In chapter 2 we introduce YAGO. Chapter 3 describes the multi threaded framework. In Chapter 4 we demonstrate the extensibility by adding new extraction techniques. Chapter 5 presents an evaluation between a single threaded and a multi threaded extraction. We conclude with a summary and future work in Chapter 6.
2

Yago

In this Chapter we introduce the YAGO model (\cite{YAGO1} \cite{YAGO2}). The data source of YAGO, and the heuristic techniques applied to construct YAGO \cite{YAGO3} \cite{YAGO4}. YAGO has its own data model which is simple and can express transitivity of the relations and the relations between facts. The entities are extracted from Wikipedia, concepts are extracted by linking Wikipedia category and WordNet, and the hierarchy of the concepts is taken from WordNet.

2.1 Yago Model

YAGO is a huge semantic knowledge base and is represented by facts. A fact is a triple (entity relation entity). If you open the page for Albert Einstein in Wikipedia, you can find that Albert Einstein won the Nobel Prize in Physics in the year 1921. Two facts are used to express this knowledge. The first fact is (AlbertEinstein HASWONPRIZE NobelPrize). In order to express the first fact happened in the year 1921, the fact is assigned with an identifier. Thus the second fact is (Fact1 ONDATE 1921). This representation is used to capture YAGO's knowledge.

Entities are abstract ontological objects which represent all objects in YAGO, including numbers, strings, facts, and relations. In the ideal case they are language-independent, but usually in different language people use certain word, to refer to a certain entity, for example the German word Nobelpreis and English word Nobel Prize refer to the same entity. In order to express the relationship between words and entities, the relation
2. YAGO

MY ANS is introduced. The fact (“Nobelpreis” MEANS NobelPrize) for example means that word Nobelpreis refers to entity NobelPrize. Quotes are used to distinguish words from other entities.

Entities with same property are grouped into classes. Each entity belongs to at least one class. We express this with the relation TYPE. And the taxonomic subsumption of the classes are expressed with the relation SUBCLASSOF. The relations are entities as well. They are grouped into class like TransitiveRelation expressed by the fact (relation1 TYPE TransitiveRelation).

2.1.1 Reification Graphs

YAGO can be expressed with a reification graph. the reification graph is defined over

- a set of nodes $N$.
- a set of edges $I$.
- a set of labels $L$.

The reification graph is an injective total function $G_{N,I,L} : I \rightarrow (N \cup I) \times L \times (N \cup I)$ A YAGO ontology over a finite set of common entities $C$, a finite set of relation names $R$ and a finite set of fact identifiers $I$ is a reification graph over the set of nodes $(I \cup C \cup R)$ and the set of labels $R$, i.e an injective total function $y : I \rightarrow (I \cup C \cup R) \times R \times (I \cup C \cup R)$

Thus the YAGO ontology can be expressed with the list of all elements of the injective function in following form:

$id_1 : arg_{11}$ $rel_1$ $arg_{21}$

$id_2 : arg_{12}$ $rel_2$ $arg_{22}$

2.1.2 Query language

A pattern for a reification graph $G_{N,I,L}$ over a set of variables $V$, where $V \cap (N \cup I \cup L) = \emptyset$, is a reification graph over the set of nodes $U \cup V$, the set of identifiers $I \cup V$ and the set of labels $L \cup V$. A matching of a pattern $P$ for a graph $G$ is a substitution
2.1 Yago Model

\[ \sigma : V \to N \cup I \cup L, \text{ such that } \sigma(P) \subset G. \sigma(P) \text{ is called a match.} \]

A filter pattern \( P \) for a reification graph \( G_{N,I,R} \) over a set of literals \( L \), a set of variables \( V \), where \( V \cap (N \cup I \cup R \cup L) = \emptyset \) and a set of filter relations \( F \), is a reification graph over the set of nodes \( N \cup V \cup L \), the set of identifiers \( I \cup V \) and the set of labels \( R \cup V \cup F \). A matching for a filter pattern is a matching \( \sigma \) for the pattern \( P \setminus \{(i, (a_1, r, a_2)) | r \in F\} \), such that \( \forall (i, (a_1, r, a_2)) \in P, r \in F : r(\sigma(a_1), \sigma(a_2)) = 1. \)

The query to the YAGO knowledge base can be transferred into patterns or filter patterns. And the answer can be transferred into a matching for that pattern. For example, the question ”when did Albert Einstein win the Nobel Prize” can be transferred into the pattern

\[
\begin{array}{|c|c|c|c|}
\hline
\text{id1?} & \text{AlbertEinstein} & \text{HASWONPRIZE} & \text{NobelPrize} \\
\text{id2?} & \text{id1?} & \text{ONDATA} & ?x \\
\hline
\end{array}
\]

And the answer is

\[
\begin{array}{|c|c|c|c|}
\hline
\text{id1:} & \text{AlbertEinstein} & \text{HASWONPRIZE} & \text{NobelPrize} \\
\text{id2:} & \text{id1} & \text{ONDATE} & 1921 \\
\text{id3:} & ?y & \text{AFTER} & 1900 \\
\hline
\end{array}
\]

The question like ”Who won the Nobel Prize after year 1900” can be transferred into a filter pattern

\[
\begin{array}{|c|c|c|c|}
\hline
\text{id1?:} & \text{?x} & \text{HASWONPRIZE} & \text{NobelPrize} \\
\text{id2?:} & \text{id1?} & \text{ONDATE} & ?y \\
\text{id3?:} & \text{?y} & \text{AFTER} & 1900 \\
\hline
\end{array}
\]

And the matching should contain

\[
\begin{array}{|c|c|c|c|}
\hline
\text{id1:} & \text{AlbertEinstein} & \text{HASWONPRIZE} & \text{NobelPrize} \\
\text{id2:} & \text{id1} & \text{ONDATE} & 1921 \\
\text{id3:} & \text{1921} & \text{AFTER} & 1900 \\
\hline
\end{array}
\]
2. YAGO

2.2 Information Extraction

The first data source for YAGO is the WordNet. WordNet is a large lexical database in English. Words which have same sense are grouped into synsets. Each synset expresses a distinct concept. Synsets are linked by conceptual-semantic and lexical relations. The second data source for YAGO is Wikipedia. Each Wikipedia article describes usually one topic. It has been written collaboratively by volunteers around the world, and currently contains 16 million articles, of which over 3.4 million are in English (?). Different heuristics are applied to Wikipedia to extract entities and facts.

The individuals for YAGO are extracted from Wikipedia. Each title of a Wikipedia page is a candidate to become an entity in YAGO. The facts with respect to the entity are taken from the article. Four different heuristics are applied to extract facts from an article, and are described as following.

2.2.1 Infobox Heuristics

Every Wikipedia page may contain an Infobox. Each row in the Infobox contains an attribute and a value. A YAGO relation will be manually designed for each of these attributes. Facts can then be generated from each row. The first entity is the title, the relation is the attribute, and the second entity is extracted from the value. Rows with multiple values generate multiple facts.

Figure 2.1 shows the Infobox of Wikipedia page for Albert Einstein. The relation BIRTHDATE has been designed for attribute ”Born”. As such a fact

| id1: AlbertEinstein | BIRTHDATE | 1879-03-14 |

is extracted from the first row.

Some attributes are mapped to the inverse of a relation. For example, the attribute ”official name” has as its value the official name of the article entity. But it is mapped to the fact

| id1: | official name | MEANS | entity |

Some of the meaning of the attribute depends on the type of the infobox. The length for example is an extent space of a car or a duration of a song.
2.2 Information Extraction

Figure 2.1: Infobox of Wikipedia - Albert Einstein (1879–1955)

- Born: Ulm, Kingdom of Württemberg, German Empire
- Died: Princeton, New Jersey, USA
- Resting place: Grounds of the Institute for Advanced Study, Princeton, New Jersey
- Residence: Germany, Italy, Switzerland, USA
- Ethnicity: Jewish
When an Infobox has been found in a Wikipedia page, all the attributes have to be parsed. Numbers dates and quantities are parsed with regular expression. The units are normalized to ISO units. The non-literal value which are found, are expected to be a Wikipedia link. If the parser fails, the attribute is ignored. The type of the article entity is generated as a candidate fact from the type of the Infobox. e.g.  

| id1: | Albert Einstein | TYPE  | Person |

### 2.2.2 Type Heuristics

The Wikipedia category system can be used to group each individual into a class. Figure 2.2 shows the Categories of the page for Albert Einstein in wikipedia. The conceptual categories like ”American physicists” indeed identify a class for the entity of the page. Some categories yield relational information, e.g. ”1879 births”. Some of them only serve for administrative purposes. The class of an individual can only be taken from the conceptual categories. In order to distinguish the conceptual categories from the others, a Noun Group Parser is used. A category name is broken into pre and post-modifier. The category is most likely a conceptual category if the category name is plural. The Pling-Stemmer is used to identify the plural words.

Although there is a hierarchy of wikipedia categories, they are not used for extracting class hierarchy of YAGO. Here, WordNet is used to establish the hierarchy of classes, because WordNet offers an ontologically well-defined taxonomy of synsets. Each synset of WordNet becomes a class of YAGO. Some words however are both a wikipedia title and a WordNet synset. The name of some individuals can be same as a common noun, like Time exposure. And some titles of Wikipedia are simply common noun, e.g. Physicist. In case of conflict the Wikipedia individuals are discard. The SUBCLASSOF
hierarchy is taken from the hyponymy hierarchy of the WordNet. Wikipedia and WordNet are connected by finding the head in WordNet synsets. The "head compound", "the pre-modifier" and the "post-modifier" are determined. The "head compound" is stemmed to its singular form, if there is a WordNet synset for the concatenation of "pre-modifier" and "head compound". The wikipedia class becomes a subclass of the WordNet class. WordNet stores with each word the frequencies with which it refers to the possible synsets. If there is no WordNet synset for the concatenation of pre-modifier and the head compound, the head compound is mapped to the most frequent synset.

The complete hierarchy of classes is obtained in this way. There are around two dozen prominent cases in which the disambiguation of the Wikipedia category names failed. But they are corrected manually.

2.2.3 Word Heuristics

Some information on word meaning can also be extracted from WordNet and Wikipedia. The words "urban center" and "metropolis" for example both belong to the synset city. After introducing a concept for each synset. A means relation is established between each word of synset and the corresponding class.

| id1:   | metropolis | MEANS | CITY |

Means facts can also be extracted from Wikipedia redirect pages. The redirect pages serve to redirect users to the correct Wikipedia article. For example, if the user typed "Albert einstein" then there is a link in the redirect page that links to "Albert Einstein".

Beside the MEANS facts, the GIVENAMEOF or FAMILIYNAMEOF can also be extracted by word heuristics. This can simply be done after identifying if the type of an entity is a person.

2.2.4 Category Heuristics

Relational Wikipedia categories are also very useful, because not every article has an Infobox, but most articles have categories. Facts are also taken from relational Wikipedia
2. YAGO

categories. This is done by detecting regular expressions in the categories. For example the regular expression \( d\{1,4\}(\_BC)? \_births \) births indicates the BORNODATE relation.
The language categories indicate the name of the article entity in other language.

2.3 Summary

The YAGO ontology consists of a list of facts, where the "low level" facts are extracted from Wikipedia and the "high level" are taken from WordNet by applying four different extraction techniques. The queries, the answers to a query, and YAGO itself can be described by a list of facts.
In this chapter we introduce our multi threaded framework for YAGO extraction. As shown in chapter 2, different heuristics are applied to Wikipedia and WordNet in order to construct YAGO. Note that none of the facts are extracted from more than one wikipedia page. Therefore we can split the Wikipedia source file into page or pages, where parsing though a page or pages can be done in parallel. We may need to link Wikipedia and WordNet, but WordNet is small enough to be kept in the main memory. With multiple cores the parsing can be truly concurrently executed, and performance can be improved.

By definition of the YAGO ontology, we know that YAGO consists of tuples. Each tuple has an ID, two entities and a relation. No matter what kind of extraction technique we applied to the data source, we always get a list of tuples. And no matter how or where we store YAGO, we store a list of tuple. Based on this feature, our framework provide an interface ”Extractable”, which can be implemented by every components that extract facts for YAGO from any kind of data source. Another interface ”Technique” is designed for the components that extract information from Wikipedia. For storing YAGO facts, we provide an interface ”Writer”. Different implementation of the ”Writer” allows YAGO to be stored into different storage mediums. The details of this new structure are described in the next section. This is followed by a new Fact Data type and an explanation of the configuration File.
3. MULTI-THREADING FRAMEWORK

3.1 Structure

The main part of the framework is the "ThreadManager". As shown in Figure 3.1 it reads the configuration file, and generates all its components. The "ReadThread" reads the data source, generates a lists of "Extractable", and puts them into the "SourcePool". Each "ExtractThread" takes an "Extractable" out of the "SourcePool" generates facts and stores them. After all the threads have been finished, the "ThreadManager" terminates the program. The details of the configuration file will introduced in section 3.3.

![Figure 3.1: Multithreaded Extraction - Structure](image)

The "ReadThread" consists of some "SourceReaders". It runs each source reader one after the other, and produces a list of instances of "Extractable" and puts them into the "SourcePool". The Source reader usually reads data from storage, and puts them into main memory. Since the size of the main memory is fixed, it makes no sense to let the readers work in parallel. However our "WikiSourceReader" shown in Figure 3.2 for example, consists of two threads. One thread (ReadThread) reads the Wikipedia source file from hard disk into the main memory. An other thread (PackThread) splits the part of the source file in main memory into pages. And packs them into a "Extractable"
3.1 Structure

We separate the I/O from the parsing in this way. Algorithm 1 and Algorithm 2 show that how the ReadThread and PackThread work.

```
while not finished do
    item= file.read();
    if item is null then
        set finished;
    end
    empty.acquire();
    buffer.offer(item);
    fill.release();
end
```

**Algorithm 1: ReadThread**

The ReadThread reads a piece of the source file, waits until this piece can be put into the buffer, and terminates until nothing can be read. The PackThread tries to remove piece file from the buffer, until nothing can be read and the buffer is empty.

An "Extractable" is a chunk of data source. By calling the method "extract" facts are produced. The "WikiSource" shown in Figure 3.3 is an example. WikiSource contains some wikipedia pages. When each of the techniques are applied to each page, and

![Image of WikiSourceReader - Structure](image-url)
3. MULTI-THREADING FRAMEWORK

![Algorithm 2: PackThread](algorithm2.png)

```
while not finished do
    fill.acquire();
    item = buffer.poll();
    empty.release();
    pack(item);
end
```

the results are added to the list of facts.

Each extractor thread, shown in Figure 3.4, takes the instances of "Extractable"

![Figure 3.3: WikiSource - Structure](figure3.3.png)

from the source pool, extracts facts from the packed source by calling its "extract" method. Algorithm 1 and Algorithm 2 cannot be used in this case, because buffer may be written or read by multiple thread at the same time, but this is simply solved by introducing a concurrent structure.

Each extractor thread is initiated with an instance of SourceWriter which is an interface for storing the facts. The write method of the Writer is called in order to store the extracted facts. Currently we implement an OracleWriter, a FileWriter and a ConsoleWriter.
3.1 Structure

Figure 3.4: ExtractThread - Structure
The OracleWriter writes the facts directly into an Oracle database. The results are written into four tables. The table "Facts" store all checked facts. The table "Checked Entity" "Concept" store the to be checked facts, entities and concepts for quality control. By writing the facts into a relational database, we can implement the query and matching in a simply way. Secondly, the quality control has to compare the facts that extracted from the Wikipedia. For example, to check synonyms, where each element in the "entity" set and each element in "to be checked facts" set must be compared. Size of both sets is more than one million. This can be done by applying join operation in the database.

The old YAGO writes the results into the file system. Therefore we implement the FileWriter which also writes the results into the file system in order to compare the multi threaded YAGO with the old one.

The ConsoleWriter doesn’t store the facts. It outputs the results in the console. This is useful for testing and debugging the system.

As shown in chapter 2, four different heuristics are applied to wikipedia and WordNet. In the single threaded YAGO extraction this is done by a program called "article extractor". We transplant this "article extractor into our new framework as following

- reading the wikipedia file is done by the "WikisourceReader".
- WordNet and the heuristics are embeded into the "Wikitechnique"
- the "ExtractThread" applies the heuristics to the Wikipedia and WordNet.
- the "ExtractThread" writes the extracted facts with the associated "Writer".

This new structure is shown in Figure 3.5.

### 3.2 Fact Data type

The interface "Fact" represents a fact in YAGO. It contains not only the information on the entities and the relation of a fact, but also the heuristic that is used to extract the fact and the URL from which the fact is taken. We also implement this interface for YAGO quality control such that it can distinguish if it has been checked. The YAGO quality control need to write information about the entity and its concept. These two
structures also implement "Fact" interface. For example an entity is expressed with a fact

<table>
<thead>
<tr>
<th>id1:</th>
<th>entity name</th>
<th>TYPE</th>
<th>YAGO entity</th>
</tr>
</thead>
</table>

and the concept is expressed with the fact

<table>
<thead>
<tr>
<th>id2:</th>
<th>entity name</th>
<th>TYPE</th>
<th>YAGO concept</th>
</tr>
</thead>
</table>

### 3.3 Configuration

Each of the components of our framework is generated according to the configuration file shown as follows.

Path = conf\extractor.ini
WikiSource = E:\enwiki-100000-test.xml #path of wikipedia file
wordNet = E:\\wordNet # path of wordNet
yagoDefinitionsFolder = E:\\YagoDefinition #path of definition files for
3. MULTI-THREADING FRAMEWORK

The path of necessary files must be given in this part.

WikiBufferSize = 20480
WikiBatchSize = 50
NumThread = 3
BatchSize = 50
ListSize = 50
FetchNum = 10

- These are the parameters for the Framework.

- WikiBufferSize is the number of chars should be read from the Wikipedia file each time.

- WikiBatchSize is the number pieces of Wikipedia files can be in the main memory

- Number of the extractor threads is defined by NumThread

- BatchSize is the size of source pool.

- The Size of the List of Extratables that is taken by a extractor threads each time is given by ListSize

- The number of pages that contained in a extractable is defined by FetchNum

NumReader = 1
1: mpi.annotator.reader.WikiSourceReader
NumTechForWiki = 2
WikiTech1 = mpi.annotator.technique.WikiTech
WikiTech2 = mpi.annotator.technique.GenderTech
Writer = mpi.annotator.writer.VisualWriter

The framework generates the components according to this part of the file.

output = d:\output #If you use FileWriter, this is required for the path of the output files.
#Database Setting if you use OracleWriter
type = Oracle
hostname = infao5503.ag5.mpi-sb.mpg.de
port = 1521
serviceName2= orcl
datatype = orcltwo
type = orcltwo
username = wyago
password = wyago

#maxConnection should be at least the number of threads
maxConnection = 5

If the FileWriter is used, the files will be written into the path defined by output. If
facts are written into database, the parameters also have to be given in this part.

3.4 Summary

As shown in this chapter, the multi threaded YAGO consists of ReadThread and Ex-
tractorThread. Each of these two threads can be split further into smaller components
which are initiated by the ThreadManager according to the configuration file. The
extractors work in parallel, and by separating I/O from parsing, we expect that perfor-
mance of extracting information from wikipedia file will be improved. In next chapter
we will replace some of these components, and show that the frame work still works.
In chapter 5 we will demonstrate the performance of this structure.
3. MULTI-THREADING FRAMEWORK
Extensibility

In this chapter we demonstrate the extensibility of our new framework. As shown in Section 3.1 we split the YAGO article extractor into two parts called WikiSourceReader and WikiSource. We can associate them with one of the OracleWriter, FileWriter or ConsoleWriter. We will first extend it by first adding a new data source, then by adding a new extraction method.

4.1 Adding a new source

YAGO allows manually defined facts. We represent a HardSourceReader, which implements the interface SourceReader. The extended extraction is shown in figure 4.1. It reads the comma-separated values (csv) file from the hard disk. The csv file is a simple text format for a database table. It consists of rows of values. Based on the same idea that we deal with the Wikipedia file, we split the csv file into pieces, each piece contains several rows, and is then packed into a instance of ”HardLines” which implements the interface ”Extractable”. After adding the HardReader into the configuration file, the read thread runs the HardSourceReader. The instances of ”Extractable” are produced and added into the source pool. Then they are taken by extractor thread and extracted. The results are stored by the chosen Writer.
4.2 Adding a new extraction

In this section we present the Gender extractor, which finds all people from Wikipedia, and detects their gender. By applying the Infobox heuristics to a Wikipedia page, we can find if an entity is a person. But there are a lot of pages without Infobox. For example, the page for Queen Alliquippa, who was a leader of the Seneca tribe of American Indians. As we want to find out the gender of a person, we need to add a new extraction technique to Wikipedia.

In order to detect the gender of an entity, we make use of the category system of Wikipedia, as almost each wikipedia page has categories. The idea is as follows described:

- If the entity is in the category Birth Death with some date, it is with high probability a person.

- If it is a person, by searching remaining category for key words "woman", "female", or a title in the category like "Queen", we can deduce that the gender is female.

- If the gender of a person is not female then it is male.
Given a wikipedia page, the gender extractor applies first the category heuristic to the page. If the relation BIRTHONDATE or DEATHONDATE is detected, the gender extractor tries to find out if the page is included in some category containing the words "woman", "female" or some title for women, and return the fact:

| id1: id1 | entity | HASGENDER | female |

otherwise the fact:

| id1: id1 | entity | HASGENDER | male |

As we have provided an interface "Technique" for different extractors with the same data source, we can add new extraction method to the list of instances of extractable. By calling the method "extract", the new facts can be extracted and added into the list of existing facts. The extended WikiSource is shown in figure 4.2.

![Figure 4.2: WikiSource with Gender Extractor - Structure](image-url)
4. EXTENSIBILITY

4.3 Summary

Each part of our framework is extendable without changing any existing source code. This can free the developers from dealing with the trouble caused by adding new technique to the old system.
5

Performance

In this chapter we compare the multi threaded YAGO extraction with the old single threaded YAGO extraction. The largest part of YAGO is reading and parsing the wikipedia file. This is implemented with the article extractor in the single threaded program. In this part we focus on the run time and evaluation of the extracted facts. The test environment used a 16 2.4GHz CPU machine and allocated 10 GB of memory.

Table 5.1 shows the comparison of the performance. The first row shows the performance of the single threaded article extractor of the old YAGO. The run time is more than five hours. The second row shows the performance when the multi threaded extraction only uses one extractor thread and a file writer. After separating the I/O from parsing the run time has been decreased by more than four hours. The performance when the multi threaded extraction uses multi extractor thread is shown by the rest of the rows.

The extraction time is nearly halved every time the number of the threads is doubled. The bottleneck of this extractor is reading the wikipedia file into the main memory and splitting them into pages, which is about 12 minutes. The system can not be faster than that.

The table 5.2 and 5.3 show the comparison of the extracted facts. The first and second column show the number of facts that are extracted from new and old YAGO respectively. The third column shows the number of facts of new YAGO that match a old one. By putting the facts of old YAGO into a hash set, we get the size of the old YAGO facts without duplication, which may smaller than the second
5. PERFORMANCE

<table>
<thead>
<tr>
<th>Run Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Old:</td>
</tr>
<tr>
<td>≥ 5 hours 20 minutes</td>
</tr>
<tr>
<td>1:</td>
</tr>
<tr>
<td>1 hours – 1 hours 20 minutes</td>
</tr>
<tr>
<td>2:</td>
</tr>
<tr>
<td>40 minutes – 45 minutes</td>
</tr>
<tr>
<td>4:</td>
</tr>
<tr>
<td>22 minutes – 25 minutes</td>
</tr>
<tr>
<td>8:</td>
</tr>
<tr>
<td>13 minutes – 15 minutes</td>
</tr>
<tr>
<td>16:</td>
</tr>
<tr>
<td>12 minutes – 15 minutes</td>
</tr>
</tbody>
</table>

Table 5.1: Performance of the extraction

<table>
<thead>
<tr>
<th>relation name</th>
<th>new number</th>
<th>old number</th>
<th>matches</th>
<th>hash set</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>7363353</td>
<td>6897381</td>
<td>6835643</td>
<td>6897380</td>
</tr>
<tr>
<td>hasMotto</td>
<td>11255</td>
<td>11244</td>
<td>11254</td>
<td>11244</td>
</tr>
<tr>
<td>establishedOnDate</td>
<td>180945</td>
<td>167331</td>
<td>180455</td>
<td>167331</td>
</tr>
<tr>
<td>isCalled</td>
<td>5852454</td>
<td>3526609</td>
<td>5850662</td>
<td>3526547</td>
</tr>
<tr>
<td>describes</td>
<td>2641227</td>
<td>2641311</td>
<td>2641195</td>
<td>2641146</td>
</tr>
<tr>
<td>bornOnDate</td>
<td>696037</td>
<td>651133</td>
<td>695212</td>
<td>651133</td>
</tr>
<tr>
<td>diedOnDate</td>
<td>319989</td>
<td>297271</td>
<td>318419</td>
<td>297271</td>
</tr>
<tr>
<td>familyNameOf</td>
<td>624225</td>
<td>752161</td>
<td>623161</td>
<td>752161</td>
</tr>
<tr>
<td>givenNameOf</td>
<td>622005</td>
<td>751095</td>
<td>622001</td>
<td>751095</td>
</tr>
<tr>
<td>hasWebsite</td>
<td>248975</td>
<td>213457</td>
<td>248966</td>
<td>213457</td>
</tr>
<tr>
<td>hasDuration</td>
<td>45832</td>
<td>45786</td>
<td>45832</td>
<td>45786</td>
</tr>
<tr>
<td>hasValue</td>
<td>150251</td>
<td>150173</td>
<td>150249</td>
<td>31210</td>
</tr>
<tr>
<td>writtenInYear</td>
<td>25059</td>
<td>25043</td>
<td>25058</td>
<td>25043</td>
</tr>
<tr>
<td>hasWaterPart</td>
<td>208</td>
<td>208</td>
<td>208</td>
<td>208</td>
</tr>
<tr>
<td>hasPopulation</td>
<td>173538</td>
<td>173507</td>
<td>173533</td>
<td>173507</td>
</tr>
<tr>
<td>hasPopulationDensity</td>
<td>50401</td>
<td>50392</td>
<td>50401</td>
<td>50392</td>
</tr>
<tr>
<td>hasGDP PPP</td>
<td>303</td>
<td>303</td>
<td>303</td>
<td>302</td>
</tr>
<tr>
<td>hasNominalGDP</td>
<td>157</td>
<td>157</td>
<td>157</td>
<td>157</td>
</tr>
<tr>
<td>hasHDI</td>
<td>213</td>
<td>213</td>
<td>213</td>
<td>213</td>
</tr>
<tr>
<td>hasGini</td>
<td>114</td>
<td>114</td>
<td>114</td>
<td>105</td>
</tr>
<tr>
<td>hasTLD</td>
<td>227</td>
<td>227</td>
<td>227</td>
<td>227</td>
</tr>
<tr>
<td>hasCallingCode</td>
<td>307</td>
<td>307</td>
<td>307</td>
<td>307</td>
</tr>
<tr>
<td>hasUTCOffset</td>
<td>96552</td>
<td>96484</td>
<td>96551</td>
<td>96483</td>
</tr>
<tr>
<td>publishedOnDate</td>
<td>14411</td>
<td>14407</td>
<td>14405</td>
<td>14407</td>
</tr>
<tr>
<td>hasPages</td>
<td>13028</td>
<td>12991</td>
<td>13027</td>
<td>12991</td>
</tr>
<tr>
<td>relation name</td>
<td>new number</td>
<td>old number</td>
<td>matches</td>
<td>hash set</td>
</tr>
<tr>
<td>--------------------</td>
<td>------------</td>
<td>------------</td>
<td>---------</td>
<td>----------</td>
</tr>
<tr>
<td>hasISBN</td>
<td>7101</td>
<td>7099</td>
<td>7098</td>
<td>7099</td>
</tr>
<tr>
<td>createdOnDate</td>
<td>18660</td>
<td>18656</td>
<td>18653</td>
<td>18656</td>
</tr>
<tr>
<td>isNativeNameOf</td>
<td>29480</td>
<td>19674</td>
<td>19681</td>
<td>19674</td>
</tr>
<tr>
<td>hasEconomicGrowth</td>
<td>51</td>
<td>51</td>
<td>51</td>
<td>51</td>
</tr>
<tr>
<td>hasInflation</td>
<td>71</td>
<td>71</td>
<td>71</td>
<td>71</td>
</tr>
<tr>
<td>hasPoverty</td>
<td>53</td>
<td>53</td>
<td>53</td>
<td>53</td>
</tr>
<tr>
<td>hasExport</td>
<td>68</td>
<td>68</td>
<td>68</td>
<td>68</td>
</tr>
<tr>
<td>hasImport</td>
<td>71</td>
<td>71</td>
<td>71</td>
<td>71</td>
</tr>
<tr>
<td>hasRevenue</td>
<td>4322</td>
<td>4322</td>
<td>4322</td>
<td>4322</td>
</tr>
<tr>
<td>hasExpenses</td>
<td>65</td>
<td>65</td>
<td>65</td>
<td>65</td>
</tr>
<tr>
<td>hasArea</td>
<td>82707</td>
<td>82701</td>
<td>82703</td>
<td>82701</td>
</tr>
<tr>
<td>hasNumberOfPeople</td>
<td>9815</td>
<td>9811</td>
<td>9814</td>
<td>9811</td>
</tr>
<tr>
<td>hasBudget</td>
<td>5777</td>
<td>5775</td>
<td>5777</td>
<td>5775</td>
</tr>
<tr>
<td>hasWeight</td>
<td>5038</td>
<td>5030</td>
<td>5038</td>
<td>5030</td>
</tr>
<tr>
<td>hasLabor</td>
<td>61</td>
<td>61</td>
<td>61</td>
<td>61</td>
</tr>
<tr>
<td>hasUnemployment</td>
<td>79</td>
<td>79</td>
<td>79</td>
<td>79</td>
</tr>
<tr>
<td>hasHeight</td>
<td>5911</td>
<td>5895</td>
<td>5911</td>
<td>5895</td>
</tr>
<tr>
<td>isNumber</td>
<td>1933</td>
<td>1913</td>
<td>1933</td>
<td>1913</td>
</tr>
<tr>
<td>hasImdb</td>
<td>13833</td>
<td>13824</td>
<td>13832</td>
<td>13824</td>
</tr>
</tbody>
</table>

Table 5.2: Satisfied Relations
5. PERFORMANCE

The numbers of facts of old and new YAGO for the relations in table 5.2 are same or have only tiny different. The table 5.3 shows the mistaken relations. The "mistakes" in the first part of the table 5.2 are caused by our new fact id. All the facts of these relations have an fact id as the first entity. Since we have a new "id allocation strategy" for the new YAGO, they can never match. The old YAGO uses some efficient parser for parsing the words, but they are not "safe" for multi threaded YAGO. Therefore we replaced them with some "weaker" but "safer" parsers. The "mistakes" in the second part are the caused by these new parser, especially the parser for name and the parser for detecting plural of a noun. But the point is that the multi threaded YAGO does not trade the accuracy for performance.

The wikipedia file is provided in the bzip2 form, which has a size of 6GB. If the wikisource reader reads the unzipped wikipedia file, the total run time of our frame work will increase by 3 hours. This is inconvenient, as the heuristics require adjusting after each run. We will lost about 3 hours each time for running the YAGO. Thus we unpack the wikipedia file once, and work on the unzipped file.

5.1 Summary

The performance of the multi threaded YAGO is better than the single threaded. There are some difference between the results, but it is caused by the "weakness" of the new parsers. We believe that with "powerful" parser the performance will not be changed much. Even it works slower with other parsers, we can always add new threads and more CPUs to improve the performance, because the bottleneck of the system is reading and splitting the wikipedia file.
<table>
<thead>
<tr>
<th>relation name</th>
<th>new number</th>
<th>old number</th>
<th>matches</th>
<th>hash set</th>
</tr>
</thead>
<tbody>
<tr>
<td>foundIn</td>
<td>32560355</td>
<td>26776246</td>
<td>0</td>
<td>26776246</td>
</tr>
<tr>
<td>using</td>
<td>32560355</td>
<td>26776246</td>
<td>0</td>
<td>26776246</td>
</tr>
<tr>
<td>during</td>
<td>32658293</td>
<td>26874161</td>
<td>0</td>
<td>26874161</td>
</tr>
<tr>
<td>inLanguage</td>
<td>5852454</td>
<td>5663468</td>
<td>0</td>
<td>5663468</td>
</tr>
<tr>
<td>since</td>
<td>81998</td>
<td>83574</td>
<td>0</td>
<td>83573</td>
</tr>
<tr>
<td>until</td>
<td>48524</td>
<td>49142</td>
<td>0</td>
<td>49121</td>
</tr>
<tr>
<td>inUnit</td>
<td>150251</td>
<td>150173</td>
<td>0</td>
<td>150173</td>
</tr>
<tr>
<td>means</td>
<td>7311296</td>
<td>4199795</td>
<td>7306793</td>
<td>3752125</td>
</tr>
<tr>
<td>isOfGenre</td>
<td>281</td>
<td>150645</td>
<td>1</td>
<td>150645</td>
</tr>
<tr>
<td>exports</td>
<td>1</td>
<td>145</td>
<td>0</td>
<td>144</td>
</tr>
<tr>
<td>imports</td>
<td>0</td>
<td>84</td>
<td>0</td>
<td>84</td>
</tr>
<tr>
<td>musicalRole</td>
<td>1952</td>
<td>27127</td>
<td>0</td>
<td>27127</td>
</tr>
</tbody>
</table>

**Table 5.3:** Mistaken Relations
5. PERFORMANCE
Conclusion

YAGO is a semantic knowledge with high coverage and precision. YAGO itself can be represented as a list of facts, which are extracted from Wikipeidia and WordNet. The frequently updated news resource require a high performance of extraction, and the information of Wikipedia that has not been extracted require extensibility of the extractor. Therefore we present our multi threaded framework, which consists of reader, extractor and writer. As shown in chapter 5, by separating I/O from parsing, the performance has been considerably improved. The extensibility has been showed in chapter 4. When we need new data source we can extend it by new SourceReader. Adding new technique and remove old technique can be done without modifying the existing source code. By applying different writer YAGO can be stored into different storage mediums. The following goals have achieved.

- Infobox, Type, Word and Category heuristics have been translated into the new framework. About 20 million facts has been extracted from WordNet and the latest Wikipedia.

- Each part of the framework is extendable.

- Comparing with the single threaded YAGO extractor, the performance of the extraction has been considerably improved comparing with the old extractor.
6. CONCLUSION

6.1 Future Work

If we want to provide to the users high quality information, the quality control must be applied to the results that we collected from the Wikipedia and WordNet. In order to do this, each entity must be compared with each unchecked fact. But both set of entities and set of unchecked facts contain more than million elements. If we can find efficient way to do this or avoid this, the performance can be further improved.

The extract method of the Technique methode takes a list of existing facts as parameter. Instead of returning a list of facts it adds the new facts into that list. It allows us to remove the existing facts from the list. With this feature the "filter technique" which for example can be used for quality control, can also be applied to the wikipedia pages.

Currently, if any article of Wikipedia has been changed, we have to download the new Wikipedia file and parse it. RSS has been used to feed changes to any Wikipedia article. As RSS files are essentially XML formatted plain text, it is possible to extend the extraction with a RSS parser, such that we can get a "real time YAGO".
Declaration

I herewith declare that I have produced this thesis without the prohibited assistance of third parties and without making use of aids other than those specified; notions taken over directly or indirectly from other sources have been identified as such. This paper has not previously been presented in identical or similar form to any other German or foreign examination board.

2. NOV, Saarbruecken