UNIVERSITÄT LEIPZIG

Faculty of Mathematics & Computer Science
Institute of Computer Science
Department of Business Information Systems

LIMES M/R: Parallelization of the LInk discovery framework for MEtric Spaces using the Map/Reduce paradigm

MASTER'S THESIS

submitted by: Stanley Hillner
Leipzig, 16th March 2011

Examiner: Prof. Dr.-Ing. Klaus-Peter Fähnrich
Supervisor: Dr. Axel-Cyrille Ngonga Ngomo

Acknowledgments

First of all I would like to express my appreciation to my supervising tutor Dr. Axel-Cyrille Ngonga Ngomo for being a great mentor during the last five months. Without Him, this thesis wouldn't have been possible. Thanks for the help in finding a topic and the support and flexibility during the genesis of this thesis.

A further thanks is dedicated to Amazon for providing a huge Research Grant for the Amazon Cloud-Services. This enabled me to test the implementation of LIMES M/R on a stable cluster using the Amazon Elastic Compute Cloud.

I would also like to thank Lars Kolb from the database department of the University of Leipzig. Through his personal commitment, I was able to find an area of interest in which I finally wrote my thesis. He also supported me at the beginning of the test phase by showing me how to use the Amazon Web Services and thus saving me a lot of time. Without his support, it wouldn't have been possible to finish this thesis in time. Thank you very much!

Las but not least, my appreciation goes to my family but especially my fiancée for being so patient with me over the last half year. Thank you so much!

Contents

List of Figures

List of Tables

List of Listings

List of Abbreviations

API	Application Programming Interface
DIP	Dependency Inversion Principle
DNS	Domain Name Service
DOM	Document Object Model
GFS	Google File System
HA	High-Availability
HDFS	Hadoop Distributed File System
HPC	High-Performance-Computing
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
LCL	LIMES Configuration Language
LIMES	LInk discovery framework for MEtric Spaces
LSP	Liskov Substitution principle
MPI	Message Passing Interface
OCP	Open-Closed Principle
OOD	Object-Oriented Design
OpenMP	Open Multi-Processing
RDF	Resource Description Framework
RMI	Remote Method Invocation
RPC	Remote Procedure Call
Silk-LSL	Silk – Link Specification Language
SLB	Server-Load-Balancing
SPARQL	SPARQL Protocol and RDF Query Language
SQL	Structured Query Language
TI	Triangle Inequality
UDF	User-Defined Function
UML	Unified Modeling Language
UPC	Unified Parallel C
URI	Uniform Resource Identifier
W3C	The World Wide Web Consortium
XML	Extensible Markup Language

1 Introduction

1.1 The Web of Data

The World Wide Web is the most important information space in the world. With the change of the web during the last decade, today's Web 2.0 [And07] offers everybody the possibility to easily publish information on the web. For instance, everyone can have his own blog, write Wikipedia articles[1], publish photos on Flickr[2] or post status messages via Twitter[3,4]. All these services on the web offer users all around the world the opportunity to interchange information and interconnect themselves with other users. However, the information, as it is usually published today, does not offer enough semantics to be machine-processable. As an example, Wikipedia articles are created using the lightweight Wiki markup language[5] and then published as HyperText Markup Language (HTML)[6] files whose semantics can easily be captured by humans, but not machines.

To handle this deficiency, various industrial and defacto standards, such as the Extensible Markup Language (XML)[7] or the JavaScript Object Notation (JSON)[8], have been developed so that data can be interchanged and processed by machines. These formats can be applied as long as the consumer of the data knows their semantics which often exists implicitly only. Furthermore, web sites, XML documents and other formats that describe real-world entities (e. g., a person or a city) are usually linked to other documents describing other entities. These links also do not offer an explicit meaning of the relationship of the linked entities, since the data formats that are used to represent these links are not expressive enough to explicitly carry these semantic relationships. The links between the described entities are not typed. These insufficiencies were the driver for the emergence of the Semantic Web as introduced by the director of the World Wide Web Consortium (W3C), Tim Berners-Lee [BLHL01]. He defines the Semantic Web as a *"Web of Data"* where machines can directly process data without missing their semantics.

Since the core idea of the Semantic Web is to extend the World Wide Web by adding semantic, machine-processable information to traditional web pages, there is the need to build on common formats and open standards. This ensures that on the one hand, everyone is able to produce and publish structured data and, on the other hand, these data are expressed, distributed and published in a way which makes them consumable for everyone. For this reason, the Semantic Web architecture builds on the flexible and extensible Web stack [HPPSH05]. By building on the Web stack, the Semantic Web is able to *"introduce 'semantics' by layering on ontologies allowing inferences to be made from the data itself"* [Wal09]. This means that the Semantic Web for instance offers the

[1] http://www.wikipedia.org/

[2] http://www.flickr.com/

[3] http://twitter.com/

[4] There is a huge number of providers of Web 2.0 services and the above mentioned services are to be understood as well-known examples.

[5] http://en.wikipedia.org/wiki/Wikicode

[6] http://www.w3.org/TR/html4/

[7] http://www.w3.org/XML/

[8] http://www.json.org/

ability to do reasoning on the data published on the web (e. g., for semantic web searches). Another important paradigm is called Linked Data which aims at publishing structured (or semi-structured) data on the web and interlinking them [HB11]. These links not automatically contain semantic information but are able to carry and express such, once these semantics are specified. Linked Data, Open Data and the Semantic Web are often referred to as the same things but a meaningful pragmatic distinction is given by Paul Walk in [Wal09]. There, he states four core-characteristics of these paradigms:

1. data can be *open*, while **not** being linked
2. data can be *linked*, while **not** being open
3. data which is both *open* **and** *linked* is increasingly viable
4. the *Semantic Web* can only function with data which is both *open* **and** *linked*

There it becomes clear that the Semantic Web is dependent on, not at least, the Linked Data paradigm which follows the subsequent "rules" for publishing and interlinking on the Web in order to build a single global data space [BHBL09].

URIs Uniform Resource Identifiers (URI) are used to *identify abstract or physical resources* and follow a fixed syntax as specified in [BLFM05]. URIs are used as names for real-world entities.

HTTP The Hypertext Transfer Protocol (HTTP) [BL11] is used as the standard data transfer protocol. Furthermore, HTTP URIs are used as resource identifiers, so that additional information for these names can be looked up.

RDF and SPARQL The Resource Description Framework (RDF) [Gro11a] is used to provide information about the referenced entity when its URI is looked up. Furthermore, the SPARQL Protocol and RDF Query Language (SPARQL) [Gro11b], a graph-based query language for RDF, is used to query information about the referenced entity.

Links Entities of the Web of Data should reference other entities using typed links. This allows users and machines, such as semantic search engines, to explore other data on the Web that are semantically related.

Following these guidelines, a complex network of datasets containing Linked Data has been developed during the past years. This network is commonly known as the Linking Open Data Cloud, as published by the Linked Data community[9]. Figure 1 shows the cloud diagram in its current version (last updated: 2010-09-22). As can be seen, the cloud contains circles of different sizes and unidirectional or bidirectional links between them. The larger a circle, the more triples are contained in the dataset represented by this circle. On the other hand, the arrows between the circles represent a number of links between the connected datasets. The direction of a link thereby indicates the dataset which contains a link. If for instance the two circles A and B are connected through a link from A to B, the link indicates that the knowledge base A contains RDF triples that utilize identifiers from B. The number of links between the datasets is indicated by the thickness of the link[10].

[9] http://linkeddata.org
[10] http://lod-cloud.net/

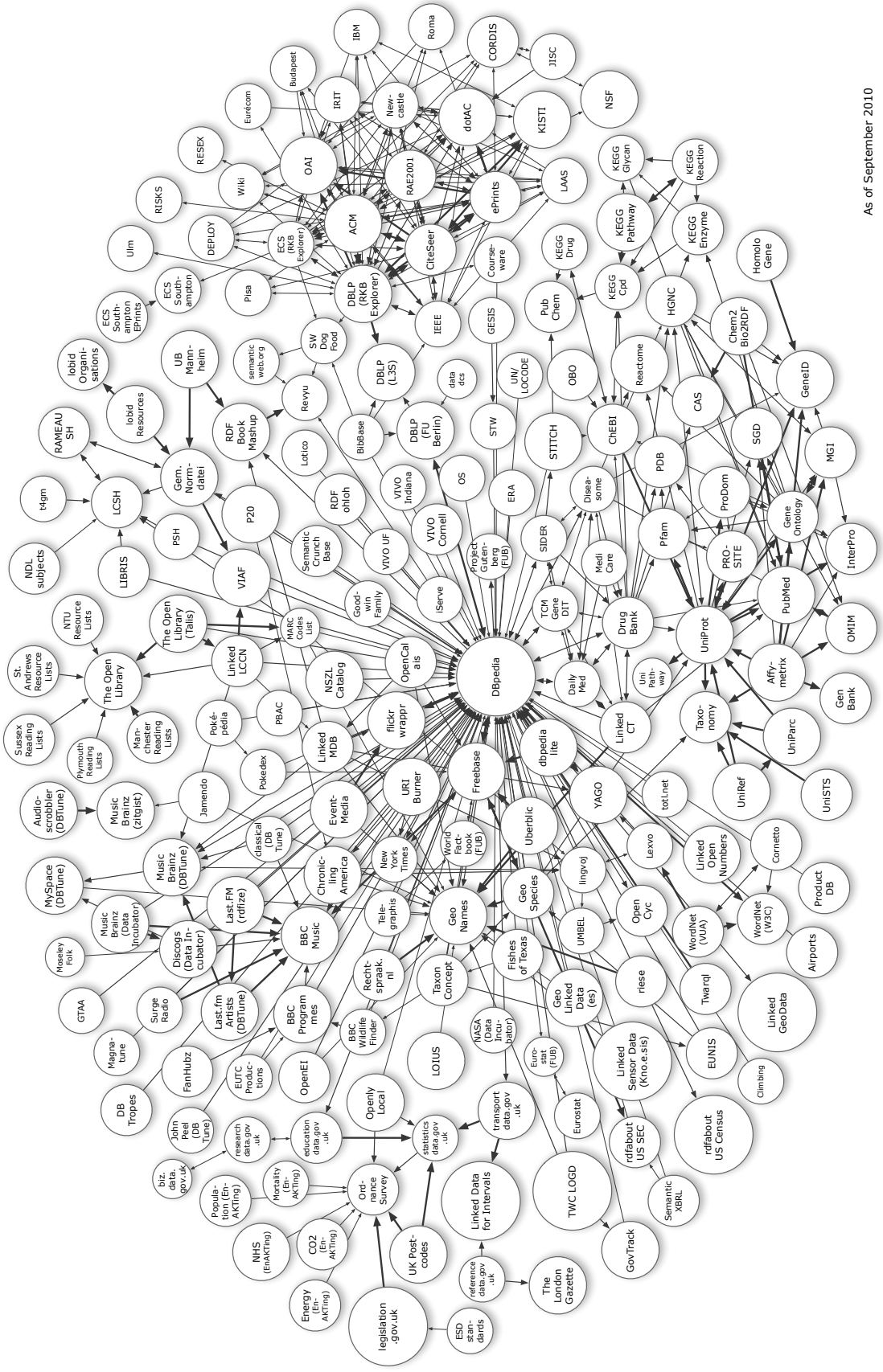


Figure 1: The Linking Open Data cloud diagram, as of September 2010 (http://lod-cloud.net/)

Well-known examples for datasets containing Linked Data are DBpedia[11], which contains structured data extracted from the most popular free online encyclopedia Wikipedia, LinkedGeoData[12], which makes the data of the OpenStreetMap project[13] available as an RDF knowledge base or the Bio2RDF knowledge base[14], a Semantic Web atlas of postgenomic knowledge. As depicted in the diagram, such datasets are often referenced by many other datasets or contain a large number of links to other datasets. The most popular dataset, DBpedia, can thereby be seen as the center of the cloud or something like a hub connecting a huge number of datasets. Identifying those hubs can then, in turn, help discovering links to other datasets and thus extending the cloud and, not at least, simplify the exploration of the Linked Data using for example Semantic Web browsers. With LOD2[15] and LATC[16] there are two current projects that drive the evolution of the Linked Data paradigm forward. Both projects aim at the research in the field of Linked Data as well as some more pragmatic questions, such as how to lower the entrance barrier to get the Linked Data paradigm closer to companies and end users.

1.2 Motivation

The topic of this thesis results from an insufficiency in the process of creating and publishing Linked Data. As depicted in figure 1, the LOD cloud consists of a large number of datasets of different sizes and the number of datasets as well as their RDF triples increases steadily. The LOD cloud currently contains more than 25 billion triples while some of the knowledge bases depicted in the diagram even consist of more than a billion triples. This is the reason why a manual selection of entities for interlinking is nearly impossible and results in a very small subset of possible links. Today, links make up less than 5% of the total number of triples available on the Web of Data [17].

To address this problem, tools for the automated discovery of links on the Web of Data are needed. Yet, while the number of tools and frameworks for creating and publishing Linked Data on the Web increases steadily, there is only a small number of time-efficient tools for the link discovery process. Although it is common praxis to use automatic and semi-automatic link discovery tools besides the manual creation of links [VBGK09a] but looking at the evolution of the LOD cloud, it is crucial to have reliable and time-efficient tool support for this task. Some state-of-the-art frameworks are able to process large datasets for finding links between them and with LIMES [NA10], a really time-efficient approach has been developed that produces exactly the same results as a brute force approach. But even LIMES has limits which can be reached very fast. Considering a common link discovery problem between three datasets of the LOD cloud, ACM, CiteSeer and DBLP, practical boundaries for the application of current link discovery tools are reached. Trying to find links between all publications that are listed in these datasets would require a huge amount of time, due to the size of the knowledge bases. Yet, while ACM contains approximately 12.5 million triples, DBLP about 24 million and CiteSeer only 8 million triples, there are much larger tasks possible, when thinking of LinkedGeoData with its 3 billion triples. Such task can easily require days or even weeks of runtime. But considering such tasks, the current approaches are not just inefficient but sometimes even not applicable, with respect to the current hardware restrictions.

[11] http://dbpedia.org/
[12] http://linkedgeodata.org/
[13] http://www.openstreetmap.org/
[14] http://bio2rdf.org/
[15] http://lod2.eu/
[16] http://latc-project.eu/
[17] http://www4.wiwiss.fu-berlin.de/lodcloud/

One approach of developing tools that require even less time is the parallelization of existing state-of-the-art link discovery frameworks. This thesis treats this task and introduces a first parallelized version of LIMES. LIMES has been chosen since it is the currently most innovative and efficient link discovery framework. Parallel computation has always been an approach for solving large problems and can also be applied to the problem of link discovery. Furthermore, the parallelization of existing frameworks can be a way to address the problem of too large tasks by splitting the problem and distributing it as subtasks to several computers [BM05].

1.3 Structure

The remainder of the thesis is structured as follows. Chapter 2 introduces the state-of-the-art technologies that are necessary to address the problem successfully. Therefore, this chapter is divided into two sections, namely *Mappers for Linked Data* and *Application Parallelization*. The goal of the first subsection is to give a short overview over the Linked Data paradigm before introducing three frameworks for discovering links on the Web of Data. Section 2.2 then gives an introduction to the parallelization of applications by investigating and evaluating state-of-the-art parallel computing approaches as well as some parallel programming models.

Based on these preliminary considerations, chapter 3 (*Architecture and Implementation*) contemplates the architectural concept of LIMES M/R as well as its implementation. To provide a comprehensive introduction into the implementation of LIMES M/R, the first two sections of this chapter contemplate the current architecture of LIMES and some important prerequisites for the parallelization task. The objective of the latter section is to find a decision for a parallel computing approach as well as an appropriate parallel programming model. After that, the internals of the MapReduce programming model are investigated and explained by using the well-known word count example.

The last and most important section of this chapter consists of a description of the parallelization of LIMES using the MapReduce paradigm. There and before describing the implementation, an architectural concept or principle is explained which forms the base of the design of LIMES M/R. Following this concept, the sections 3.3.1 and 3.3.2 then discuss the internal link discovery process and the implementation of LIMES M/R. To enable a detailed look at the internals of LIMES M/R, the implementation section contains many source code listings where the implementation of LIMES M/R is explained.

The last chapter of this thesis presents the evaluation of the LIMES M/R implementation. This chapter proves that LIMES M/R is a suitable and applicable version of LIMES by evaluating the results of several test cases that are run on a test cluster. In order to get a meaningful evaluation, the chapter is structured into two sections. Section 4.1 explains the setup of the test system and the test suite. Furthermore, this section states some questions that allow an interpretation of the test results. Section 4.2 then states and evaluates the results of the individual test cases. It is verified that LIMES M/R is a suitable and applicable approach for the parallel computation of link discovery tasks but it is also shown that although LIMES M/R is applicable for every link discovery tasks, one has to consider the problem size when choosing one of the frameworks in order to achieve the optimal performance.

2 State of the art

2.1 Mappers for Linked Data

The term *Linked Data* is used to describe a set of best practices to publish structured data on the Web with the following properties. The published data instances have an explicitly defined meaning and are machine-readable. Furthermore, the data instances can be interlinked with data instances from other (external) data sets, as source or target of the links [VBGK09a, BL07].

In recent years, the creation and publication of Linked Data became more easy and a wide variety of tools for the publication of Linked Data has come up. The functionality of those tools ranges from data servers which offer storage and query functionality for Linked Data, such as the Virtuoso Universal Server [Sof11], the Talis Platform [Tal11] or the D2R Server [BC11] up to toolkits for extending legacy document-oriented web applications with Linked Data interfaces (Triplify [ADL+09]) or even extracting Linked Data from HTML documents without RDF-annotation, such as SparqPlug [CHM08]. However, in contrast to the creation and publication tasks of Linked Data, there is still a significant lack of tool support for the discovery of links between structured data. The current discovery of links between structured data instances is mostly done manually by the publisher of the data. This process is time consuming and covers only a fraction of the total number of possible links. Current knowledge bases are often huge and their number and sizes are growing steadily, which makes manual link discovery nearly impossible. Therefore, the automation of the link discovery process by means of time-efficient tools is essential. As an example for the performance requirements, one can consider two small knowledge bases with 1 000 instances. A brute-force comparison for finding similar instances would result in 1 000 000 tuples to compare. Changing the number of instances to 10 000 per knowledge base would even result in 100 000 000 comparisons and would increase the runtime of the link discovery tool by an intolerable amount. The time-complexity of the brute force approach is $O(|S| \cdot |T|)$, where $|S|$ is the number of instances in the source knowledge base and $|T|$ the number of instances in the target knowledge base.

A small number of tools and frameworks support the discovery of links on the Web of Data. These frameworks can be classified into two categories, *domain-specific* and *universal* or *domain-independent* link discovery frameworks. The goal of domain-specific link discovery frameworks lies in the discovery of links between knowledge bases of a specific domain, such as the music domain. An example for a domain-specific link discovery framework is RKB-CRS (Resilience Knowledge Base - Co-reference Resolution Service)[GMS+09] which computes links between universities, conferences and authors based on URI lists. GNAT [RSS08] can be named as a second example, which focuses the music domain by using audio fingerprinting methods for the link discovery task. In contrast, universal or domain-independent link discovery frameworks aim at the computation of links between any knowledge bases without restricting them to be part of a specific domain. Examples for universal link discovery frameworks are RDF-AI [SLZ09], which interlinks only local data sets in a five-step process, or LinQuer [HMX11], which introduces its own query language for relational data sources (LinQL) as an extension of the Structured Query Language (SQL). Queries written in LinQL can be transformed to SQL queries in order to provide

link discovery capabilities for relational data sources at query time. Two Further frameworks, that additionally address the time-complexity problem of link discovery, are Silk [VBGK09b] and LIMES [NA10]. Silk offers a large set of similarity metrics and aggregation functions for the link discovery task and improves the runtime by utilizing rough index pre-matching. In contrast to Silk, the primary objective of LIMES lies in the reduction of the problem complexity by utilizing the mathematical characteristics of the similarity metrics used to carry out the matching task. Therefore, LIMES is not able to provide as much similarity metrics as Silk but reduces the number of comparisons significantly.

The subsequent sections contemplate three of the previously mentioned state-of-the-art link discovery frameworks, Silk, LIMES and LinQuer. Since the topic of this work is the parallelization of LIMES, LIMES is one of the three contemplated frameworks. Silk is investigated because of its runtime improvements and its similarities to LIMES. Furthermore, there is already a parallelized version of Silk available that can be used as a benchmark reference for the evaluation of the parallelized version of LIMES. The LinQuer approach differs from Silk's and LIMES' link discovery approach but is considered as well because of its ability to discover links directly at query time. Every of the following frameworks is suitable for the automated discovery of links between knowledge bases but each framework targets different requirements so that it is central to reflect about advantages and disadvantages of each framework before utilizing one of them.

2.1.1 Silk

The link discovery framework Silk is a project at the Free University of Berlin whose key goal is to assist the publisher of Linked Data in finding and setting RDF links to other data sources on the Web [VBGK09b, IJBV10], which can be either local or remote knowledge bases that are queried using the SPARQL [Gro11b] protocol. Silk is one of the state-of-the-art link discovery frameworks because of the diversity of ways to compare item properties, combine comparisons and express semantic relationships as well as its performance improvements compared to the brute force approach.

One benefit of the framework is that Silk comes with an easy to learn but expressive declarative configuration language, the Silk – Link Specification Language (Silk-LSL) [VBGK09b]. Besides the data sources and the RDF link type configuration, Silk-LSL provides a variety of configuration options regarding the comparison process. There is for instance the possibility to aggregate and weight individual comparisons in order to calculate a more precise result value. Furthermore, it is possible to take the RDF graph into account, which makes up the context of a data instance, by using paths to address neighbor RDF nodes [VBGK09b]. Listing 1 contains an extract of an example configuration that interlinks cities of DBpedia[18] and LinkedGeoData[19] using `owl:sameAs` links.

After the definition of some prefixes that can be used in the configuration document, the data sources are specified. Thereafter, the `Interlinks` section contains the link type definition and the aggregation of some individual or already aggregated property comparisons. Here, the `rdfs:label` properties of the instances are compared. At the end of this section, a threshold for accepted instance pairs is set. Listing 22 ,which can be found in the Appendix, shows the full example configuration file.

[18] http://dbpedia.org/
[19] http://linkedgeodata.org/

```
<Silk>
  <Prefixes>
    ..
  </Prefixes>

  <DataSources>
    <DataSource id="dbpedia" type="sparqlEndpoint">..</DataSource>
    <DataSource id="lgdb" type="sparqlEndpoint">..</DataSource>
  </DataSources>

  <Interlinks>
    <Interlink id="cities">
      <LinkType>owl:sameAs</LinkType>
      <SourceDataset dataSource="dbpedia" var="a">..</SourceDataset
         >
      <TargetDataset dataSource="lgdb" var="b">..</TargetDataset>
      <LinkCondition>
        <Aggregate type="average">
          <Aggregate type="max" required="true">
            <Compare metric="jaro">
              <Input path="?a/rdfs:label[@lang='en']"/>
              <Input path="?b/rdfs:label[@lang='en']"/>
            </Compare>
            <Compare metric="jaro">
              <Input path="?a/rdfs:label[@lang='en']"/>
              <Input path="?b/rdfs:label[@lang='']"/>
            </Compare>
          </Aggregate>
          <Aggregate type="max" required="true">
            ..
          </Aggregate>
        </Aggregate>
      </LinkCondition>
      <Filter threshold="0.9" limit="1"/>
    ..
</Silk>
```

Listing 1: Snippet of Silk-LSL example configuration file that interlinks cities of DBpedia and LinkedGeoData [IJBV10]

The current implementation of Silk comes with a wide variety of metrics and aggregation functions that can be used to carry out the individual property comparisons. For the aggregation of result values, one can for instance apply the minimum (MIN), maximum (MAX) or the weighted average (AVG) function. For the individual item property comparisons, there is a whole set of similarity metrics which each calculate a result between 0 and 1. There are for example the equality of URIs, dates and numbers or several string similarity metrics usable, such as the Jaro-Winkler, Levenshtein or the qGram metric. A full list of the similarity metrics and aggregation functions can be found in [VBGK09b].

A second plus of Silk lies in its performance improvements, compared to a brute force link discovery tool. Silk is able to achieve a time-complexity of $O(|S| + |T|)$ instead of the quadratic complexity of $O(|S| \cdot |T|)$ when applying a prematching technique which is often called blocking [VBGK09b, VBGK09a, IJBV10]. The goal of the rough index prematching technique is to lower

the number of comparisons by figuring out a limited subset of entities from the target knowledge base that are likely to match an entity of the source knowledge base under the given conditions. This technique indexes all target entities by comparing them to the source entities using only a few specified properties that are particularly meaningful, such as their labels. During the comparison phase that follows the blocking phase, the algorithm uses the index to search for the limited number of possibly matching resources and compares them to the source entity in detail, using the comparison algorithms that were specified by the user. The subsequent listing contains a simple blocking example which could be added to the configuration file of listing 22. This would result in subsets of a fixed size of 50 resources of the target knowledge base that are considered to be likely to match a source entity. The indexing comparison phase uses the labels of the entities to make assumptions about their similarity.

```
1 <PreMatchingDefinition sourcePath="?a/rdfs:label" hitLimit="50">
2   <Index targetPath="?b/rdfs:label" />
3 </PreMatchingDefinition>
```

Listing 2: Silk-LSL blocking example

The whole process of link detection in Silk is shown in figure 2 below and contains the above mentioned SPARQL data extraction (Data Source) step as well as the blocking and comparison (Link Generation) phase. Two further steps in the link detection process are the filtering of the generated links in order to match the user-defined link conditions and the output phase which serializes the filtered results.

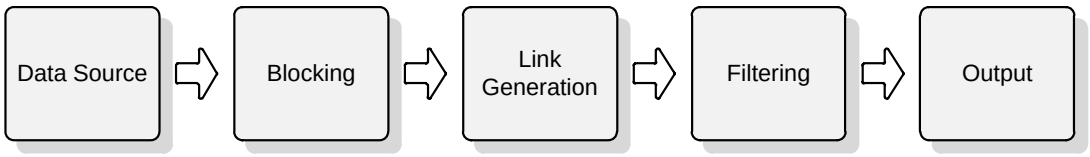


Figure 2: The Silk link detection process [IJBV10]

2.1.2 LIMES

The LInk discovery framework for MEtric Spaces (LIMES) [NA10, NEA10] is, similar to Silk, a runtime-optimized framework that aims in assisting data publishers in finding and setting RDF links to other knowledge bases. LIMES is a project of the AKSW-work group[20] at the University of Leipzig and is currently still in development. Even in its early development phase (currently version 0.3.2), LIMES is one of the state-of-the-art frameworks for link discovery tasks on the Web of Data. The reason for this is the mathematical model that makes up the base of LIMES. The current implementation of LIMES does not provide such a variety of comparison possibilities as Silk does, but has much better performance improvements. There are experiments that proof the runtime of LIMES and compare it to the runtime of Silk, for example the comparison of City-entities from the DBpedia knowledge base (12 701 instances), where LIMES requires only 5% of Silk's computation time without loosing links [NA10].

[20] http://aksw.org/

Similar to Silk, LIMES comes with an easy to learn declarative configuration language in XML format, the LIMES Configuration Language (LCL) [NA10, Ngo10]. Since the framework does not yet provide such complex comparisons as Silk does, LCL is a bit less complex and not as expressive as Silk-LSL. An example of LCL can be found in Listing 23. Listing 3 contains an extract of this configuration file.

```
<LIMES>
  <PREFIX>
    ..
  </PREFIX>

  <SOURCE>
    <ID>dbpedia</ID>
    <ENDPOINT>http://dbpedia.org/sparql</ENDPOINT>
    <VAR>?a</VAR>
    <PAGESIZE>1000</PAGESIZE>
    <RESTRICTION>?a  rdf:type  dbpedia:Settlement</RESTRICTION>
    <PROPERTY>rdfs:label</PROPERTY>
  </SOURCE>

  <TARGET>
    ..
  </TARGET>

  <METRIC>levenshtein(a.rdfs:label,  b.rdfs:label)</METRIC>
  <EXEMPLARS>70</EXEMPLARS>

  <ACCEPTANCE>
    <THRESHOLD>0.9</THRESHOLD>
    <FILE>accepted.nt</FILE>
    <RELATION>owl:sameAs</RELATION>
  </ACCEPTANCE>

  <REVIEW>
    <THRESHOLD>0.85</THRESHOLD>
    <FILE>reviewme.nt</FILE>
    <RELATION>owl:sameAs</RELATION>
  </REVIEW>
</LIMES>
```

Listing 3: LCL configuration file for interlinking cities of DBpedia and LinkedGeoData

LCL configuration files are similarly structured as Silk-LSL files and start with the prefix definitions after the document metadata[21]. These prefixes can be used in the configuration file and will also be written to the result files. After that section, the data sources are configured within the `<SOURCE>` and `<TARGET>` tags where the `<RESTRICTION>` tag allows to define restrictions for the entities to be compared. In this example, there are only entities from the source knowledge base taken into consideration that are of type `dbpedia:Settlement`, where `dbpedia:` is the prefix for the DBpedia ontology URI. Furthermore, the `<PROPERTY>` tag defines the property of the concepts that will be used for the linking of the entities. In contrast to Silk-LSL, the comparison section of

[21] An example of the metadata can be found in the full LCL example file in Listing 23 as well as in the LIMES user manual [Ngo10].

LCL files is much more compact and contains only the <METRIC> tag, which specifies the comparison metric and the properties that have to be compared, as well as the optional <EXEMPLAR> tag. The latter can be used to adjust the comparison performance of LIMES. The last section of the file consists of the acceptance and review definitions, where the user can set the output files as well as the thresholds and link types for accepted and to-be-reviewed instance pairs.

As mentioned in the introduction of this section, the main advantage of LIMES, compared to other mapping approaches, such as Silk, lies in its underlying mathematical principles which significantly reduce the number of comparisons that are necessary to compute similarities of instance pairs[22]. LIMES utilizes metric spaces and the triangle inequality (TI), which is one of the mathematical characteristics of metric spaces [NA10, NEA10], to make pessimistic estimates about similarity values to find possibly matching instance pairs. Figure 3 depicts the link discovery work-flow as it is implemented by LIMES. The LIMES link discovery process consists of the following four steps:

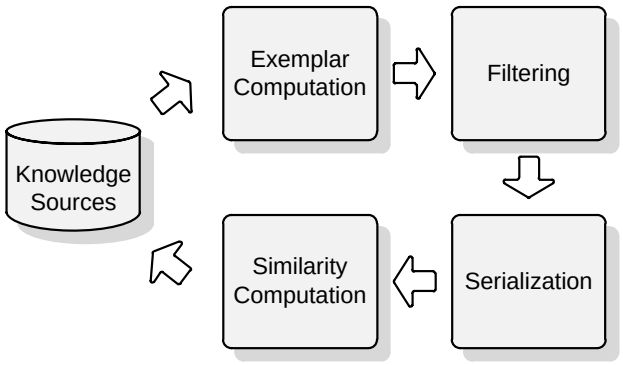


Figure 3: The LIMES link discovery process [NEA10]

Exemplar computation is the first step after fetching the data of the source (S) and target (T) knowledge bases. During this phase, a set of exemplars (E) is computed for T and each instance $t \in T$ is compared to the exemplar e that is the closest to t. This results in a partitioning of T into subspaces, where each subspace is represented by exactly one exemplar $e \in E$. Furthermore, there are distance values known from each instance of a subspace to the subspace-representing exemplar.

Filtering is applied before the computation of similarities of the source and target instances and reduces the number of instance pairs to be compared significantly. This step takes advantage of the TI-characteristic of the underlying metric space and approximates the distances between every $s \in S$ and $t \in T$ by computing the distances $m(s,e), \forall s \in S, e \in E$. Since LIMES makes pessimistic distance approximations while filtering instances of T, it is guaranteed that the LIMES-approach discovers links lossless which leads to exactly the same result as carried out by a brute force approach.

Similarity computation is then only carried out for $s \in S$ and $t \in T$ that remained after the filtering process. This step computes the exact distances $m(s,t)$ and can thus be used as the comparison value for the link discovery.

[22] LIMES actually does not reduce the number of comparisons between instance pairs directly but rather the number of instance pairs that are considered to be matching candidates. This reduces the total number of comparisons indirectly.

Serialization is the last phase which writes the discovered links in to the user-defined output stream in the specified format, e. g., NTriples[23].

2.1.3 LinQuer

LinQuer is a research project of the Database Group at the University of Toronto[24], in collaboration with the Thomas J. Watson Research Center, IBM[25] [HMX11]. The main goal of the Linkage Query Writer (LinQuer) lies in assisting users in finding and tuning link discovery methods that suit their needs best, with regard to their domain and application. In a nutshell, LinQuer can be seen as an integration framework for link discovery methods that provides query rewriting to address relational data sources [HLKW09, HXM+09, HKL+09]. LinQuer does not only address the problem of detecting links between entities that refer to the same real-world entity (entity resolution), but also the task of finding other semantic relationships of entities in order to set links between them. The framework consists of three main components, LinQL, a query language for the specification of link conditions, an API and a web interface for the translation of LinQL queries into standard SQL queries and an interface that assists users in writing LinQL queries [HXM+09].

The query language LinQL extends SQL with link discovery capabilities that can be used while querying relational data sources. An extract of the LinQL grammar can be found in the appendix of this work (Listing 24). [HKL+09] describes the LinQL language specification in detail, thus the detailed analysis of the grammar shall not be object of consideration. As a summary, a LinQL query always consists of a set of source data and a set of target data, as well as a set of methods for finding links between the source and target data [HXM+09]. The latter can be either of the various native link discovery methods that come with LinQuer, or custom link discovery methods that can extend the framework as User-Defined Functions (UDF). Link discovery methods in LinQuer can either have a syntactic or a semantic nature, where combinations are allowed, too. Syntactic link discovery methods are for example known from Silk and LIMES, such as similarity measurements or the approximation of similarity values, whereas semantic methods utilize for instance ontologies to find for example taxonomic relationships or dictionaries to determine relationships of words, such as the synonymy or hyponymy of words.

```
1 CREATE LINKSPEC customLink
2 AS customLinkUDF(0.9, 0.85).
3
4 SELECT d.*, l.*
5 FROM DBPEDIA d, LGDB l
6 WHERE d.TYPE = 'Settlement' AND l.type = 'City'
7 LINK d.NAME WITH l.NAME
8 USING customLink
```

Listing 4: LinQL example interlinking cities from DBpedia and LinkedGeoData

[23] http://www.w3.org/2001/sw/RDFCore/ntriples/
[24] http://www.cs.toronto.edu/db/
[25] http://www.watson.ibm.com/index.shtml

Listing 4 contains a small LinQL example which interlinks DBpedia entities of type `Settlement` with LinkedGeoData entities of type `City` using a similarity measurement on their `NAME` properties[26]. The first statement creates a custom link specification which uses a user-defined measurement (`customLinkUDF`). The measurement is implemented as a UDF and takes a threshold for accepted instance pairs and one for the instance pairs to be reviewed as parameters. The subsequent lines query the sources and restrict the entries to be of type `Settlement` or `City`, respectively. This part of the query is written in standard SQL syntax. After that, the properties to link are specified in the `LINK` part of the query, before the linkage method is specified in the `USING` clause. The example of listing 4 references the custom linkage method that was defined in the `CREATE LINKSPEC` section of the query.

One big advantage of using LinQuer for record linkage lies in its performance. Since the LinQL queries are translated into standard SQL queries the link detection process is run at query time and the user can take advantage of several performance improvements that are available in relational databases. This, as well as the ability to use semantic link detection methods instead of syntactical ones only, makes LinQuer to a state-of-the-art framework for link discovery on structured data.

2.2 Application Parallelization

During the last decade, the parallelization of applications has become increasingly important. On the one side, there is the need of the industry and scientists for high-performance computer systems, such as High Performance Computing (HPC) clusters, in order to compute huge amounts of data in reasonable time, to provide load balancing mechanisms, as for instance provided by Server-Load-Balancing (SLB) clusters, or to reach a very high availability for important computer systems which for example can be achieved by using High-Availability (HA) clusters [BM05, Boo02]. On the other side, the rapid progress in the development of computer hardware, especially the evolution of microprocessors, allows for creating hardware that can process data more efficiently. In recent years, the computer industry switched to the development of multi-core processors which also pushed the need for parallel computations in standard software [KMZS08, ABC+06].

The following sections give an introduction to the parallel computation paradigm in order to provide a basis for the decision about the technologies to be used for the parallelization task of the link discovery framework LIMES. For this purpose, the subsequent sections will introduce three different state-of-the-art parallel computing approaches that are suitable before investigating a number of parallel programming models. The conclusion of this section is formed by a summarized evaluation of the contemplated programming models in order to provide a reference for the parallelization task.

2.2.1 Parallel Computing Approaches

In the following, cluster computing, grid computing and cloud computing are presented, three of the most common approaches to parallel computing. This contemplation serves as a reference for the parallelization of LIMES.

[26] The LinQL example is built on fictitious relational data sources that represent the DBpedia and LinkedGeoData knowledge bases, to get a comparison to the LSL and LCL examples of this section.

Cluster Computing

The core idea of cluster computing is to build parallel computers by using a number of standard machines which are connected by a fast network [BM05, Boo02, CB10, Bla10]. Computer clusters can often be seen and interacted with as a single computer and are usually used to improve performance and availability of the system. Furthermore, by using multiple commodity machines, this approach is more cost-effective than the use of a single supercomputer [BM05].

Since the term "Cluster" only describes the architecture and interaction of the elements, one has to distinguish between hardware clusters and software clusters. The aim of software clusters is to provide continuous operation, like the Domain Name Service (DNS), for example by switching the service in case of a failure. Hardware clusters on the other hand provide the infrastructure for the services [BM05, Boo02]. A simple form of a hardware cluster is known as active/passive where services can be migrated from the currently active node to the passive node in the cluster in case of a hardware failure. These clusters mostly consist of two nodes (servers) and are intended to improve the availability and reliability of the system[27] [Boo02].

Other clusters are used for load-balancing over several machines. These clusters are usually deployed in environments with high performance needs, e. g., distributed databases. An important advantage of the cluster approach for balancing the load of a system is the ability to easily improve the performance of the system by adding new nodes to the cluster instead of upgrading a single computer which likely is more expensive and usually cannot be done without disabling the system.

The third and for this work most important type of clusters is used for high-performance computations such as the search for new prime numbers or even the mapping of knowledge bases. Since these clusters are intended to execute computational tasks, the users can submit jobs (the problems to solve) to a central job management system. This system either splits the job into packages and distributes them over the working nodes of the cluster or it distributes several jobs in parallel over the cluster and schedules their execution. Using HPC-clusters, one can efficiently compute huge problems that would take an inacceptable amount of time on single machines, due to the restrictions of current hardware [BM05, Boo02, Bla10].

Grid Computing

In principle, grid computing is a form of distributed computation, where a super computer is built by a cluster of interconnected standard computers [FK03]. However, although the grid computing approach resembles the approach of cluster computing, there are some differences which make up the definition of grid computing. Some important differences are the looser coupling, the geographical dispersion and the heterogeneity of the nodes that are part of the cluster. A definition of grid computing was published in [FK03] by Ian Foster and Carl Kesselman:

> *The sharing that we are concerned with is not primarily file exchange but rather direct access to computers, software, data, and other resources, as is required by a range of collaborative problem-solving and resource-brokering strategies emerging in industry, science, and engineering. This sharing is, necessarily, highly controlled, with resource providers and consumers defining clearly and carefully just what is*

[27] Active-passive and active-active failover clusters are f. i. described in the virtuozzo documentation: http://download.swsoft.com/virtuozzo/virtuozzo4.0/docs/en/win/VzWindowsClustering/28086.htm and http://download.swsoft.com/virtuozzo/virtuozzo4.0/docs/en/win/VzWindowsClustering/28060.htm

> *shared, who is allowed to share, and the conditions under which sharing occurs. A set of individuals and/or institutions defined by such sharing rules form what we call a virtual organization.*

In the above definition, Foster and Kesselman define the grid computing approach as the sharing of resources but do not restrict the term resource to files or memory. They rather extend the term resource to be for example the direct access to computers and their software or other resources needed to solve problems. Furthermore, they state another difference to clusters. While clusters are commonly controlled and used by a single corporation or institution, a grid is controlled and used by a virtual organization which is made up of a number of individuals and/or institutions that clearly define which resources are shared under which conditions. In general, grids have the following three characteristics:

- A grid coordinates resources that are typically not maintained by a central instance. This means that grids integrate and coordinate resources from different administrative domains with respect to safety, security and billing.
- Grid computing typically applies open and standardized protocols and interfaces for the communication, authentication and resource administration.
- A grid combines existing resources and reconfigures resources to provide services with the aim of a significantly higher business value than the sum of its components.

Cloud Computing

A third approach for distributed computation is called cloud computing. The primary goal of cloud computing is the abstraction of all details about the IT-infrastructure from the customers so that they do not have the need to worry about computing capacity, memory, software or similar things [Dan08]. The IT-infrastructure will be available over a network (typically the Internet) depending on the needs of the user. Similar to the previously described grid computing approach, cloud computing offers a consumption-based billing model with an automated service provision, based on the concept of virtual computer centers and under the usage of current web technologies, such as web services [FK03]. Thereby, the concept of supplying "everything" as a dynamically usable service goes back to the illusion of the infinite supply whose idea can be described as follows: Resources (in the sense of cloud computing[28]) are infinitely available and will be dynamically supplied over a network without any delay, just like electricity coming from the socket [FK03].

An actual definition for cloud computing comes from the National Institute for Standards and Technology (NIST)[29] [MG09] and contains the above mentioned core characteristics. This definition additionally identifies three core service models for cloud computing, which are Cloud Software as a Service (SaaS), Cloud Platform as a Service (PaaS) and Cloud Infrastructure as a Service (IaaS). Without describing these service levels in detail, they indicate that the cloud provides everything as a service, starting at the hardware, which is necessary to store data or compute problems, ending at the software level like web-based email services. A very good example for cloud services are the Amazon Web Services[30] which are also utilized in this thesis.

The last part of this definition states four deployment models that are often referred to as types and of which the first two types, private and community clouds, are exclusively operated for one

[28] Resources are not just files or memory to store data but rather everything the customer needs to solve problems.
[29] http://www.nist.gov
[30] http://aws.amazon.com

(private) or several (community) organizations that have shared concerns, for example their security requirements. Public clouds, however, are publicly accessible and owned by an organization that sells the cloud services. The last type is called hybrid clouds and formed as compositions of two or more clouds (private, community or public). Each sub-cloud remains a unique entity but their composition enables portability between them [MG09].

To summarize the cloud computing approach, one can identify a main difference to, for instance, the grid computing paradigm. Grid computing aims at the community usage of common resources. As the word common indicates, there is no centralized maintenance of these resources. In contrast, cloud computing has Providers and consumers of the cloud services. There, the resources and services are maintained by the cloud provider.

2.2.2 Parallel Programming Models

Building programs for parallel computation has been a challenge since parallel computation was invented, which is one reason why the parallel computation approach had only been used by, for instance, researchers which computed huge amounts of data. But during the last decade, for example driven by the steadily decreasing hardware prices and the rapid technological development (such as the ubiquity of multi-core processors[31]), parallel programming became more important in order to use the full potential, even of standard computers [ABC+06, CB10]. In [KMZS08], two main approaches for application parallelization are identified, namely *auto parallelization* an *parallel programming*.

On the one hand, the auto parallelization approach, as for example represented by parallel compilers, can be applied to automatically parallelize sequentially programmed applications, which offers some important advantages. Developers for instance do not have the need to learn new programming paradigms and they can simply parallelize legacy applications without developing a whole new architecture. There is only the need to recompile the sources with an appropriate parallel compiler. However, as positive as this approach affects the conceptional effort of the development of parallel applications, it can not be applied to applications with high performance requirements. Automatically parallelized applications are limited in their degree of parallelism which lowers the speedup[32] of the application.

To ensure a higher parallelism, developers will have to apply special parallel programming techniques. Generally, parallel programming is a complex task that involves, among other things, the partitioning of the workload and data as well as the task-to-worker mapping and the data distribution and management. This likely increases the workload on the developers and therefore the development time and costs, but usually results in a higher performance increment for the parallelized application.

There are various parallel programming models from which an application developer can choose [KMZS08, CB10, EB09, ABC+06]. Software architects often select the programming model which will probably deliver the most performing end product. This is admittedly the most important criterion for the decision about the programming model, but there are various other criteria that should also be taken into account during the design process, like the abstraction of parallelism

[31] Recently, with the LG Optimus Speed (http://www.lg.com/global/press-release/article/lg-launches-world-first-and-fastest-dual-core-smartphone.jsp) the first dual core smartphone appeared.

[32] Speedup is a measurement that describes the improvements of a computer system that can be achieved through optimizations, such as its parallelization. The speedup of a parallelized application mathematically describes the relationship between the processing speed on a single machine and multiple machines or processors [HP07].

to the developer. The authors of [KMZS08] identified seven evaluation criteria for parallel programming models and evaluated six models against these criteria. In [ABC+06], five critical tasks during the development of parallel applications are named and ten programming models are evaluated against them. Five important criteria for parallel programming models have been extracted from both papers in order to serve as evaluation criteria for the parallel programming models that are contemplated in this chapter. These criteria are:

Task-to-Worker Mapping defines which task shall be executed on which worker. One can distinguish automatic (done by the runtime-system) and manual (to be managed by the developer) worker mapping.

Worker Management is the management of the workers, e. g., their life cycles. Automatic management indicates that the system controls the creation or termination of, for example, threads or processors. In manual management, the developer has to maintain the worker life cycles manually.

Data Distribution describes the task of distributing data over the parallel system to feed the workers with input data to be processed and collect the results from the workers. Again, one can distinguish automatic data distribution and manual data distribution where the developer has to manage the passing of data or the current state of the data.

Synchronization is the task of managing the order of the workers which access shared data. Automatic synchronization indicates that the developer does not need to worry about synchronization details.

Programming Methodologies defines how the parallelism is abstracted, as an example, the available API and the language specification. This can be a main factor in development time when a developer, for instance, uses a new programming model that does not offer an intuitive language.

In the following, a number of current parallel programming models are enumerated, briefly explained and evaluated against the criteria that are explained above, so as to allow to determine the best suited approach for the task at hand. Note that most of the information about the programming models below are extracted from [KMZS08, CB10, EB09] and the project websites or project describing papers. Therefore, each subsection names the project source and then omits the sources that have been cited before.

OpenMP

Open Multi-Processing (OpenMP)[33] is a representative of the programming models for shared memory architectures. It is an open specification which extends C, C++ and Fortran with parallelization capabilities, for example by offering a set of compiler directives and callable runtime libraries. In OpenMP, the workers are realized as threads whose management is implicit. This means that the developer does not need to manage the life cycle of the threads. Instead, OpenMP provides special compiler directives which indicate that a code section, e. g., a for-loop, shall be run in parallel. This includes the task-to-worker mapping, which is also done by OpenMP.

Since OpenMP is a programming model for shared-memory architectures, the effort for distributing data between the workers is small. Each worker can access the data in the shared address space. Yet, another critical task while working on shared memory computers is the synchroniz-

[33] http://openmp.org/wp/

ation between the workers. One problem is the synchronization of the access of the shared data. The programmer or the system has to ensure that no conflicts between workers arise when accessing the same data. In OpenMP, the synchronization is done implicitly by several mechanisms and the programmer only has to declare sections where a synchronization is needed.

```
1  int main(int argc, char *argv[]) {
2    const int N = 100000;
3    int i, a[N];
4
5    #pragma omp parallel for
6    for (i = 0; i < N; i++) {
7      a[i] = 2 * i;
8    }
9
10   return 0;
11 }
```

Listing 5: OpenMP example: parallelized initialization of a large array in C (from: http://en.wikipedia.org/wiki/Openmp)

The last evaluation criterion is the abstraction of the parallelism to the programmer (Programming Methodologies). As previously described, the developer does not need to handle any parallelization tasks explicitly, most tasks are done by specifying parallel sections using compiler directives. Thus, OpenMP abstracts away the most critical tasks, like the the workload partitioning or the synchronization mechanisms, so that the programmer can focus on the development of the core functionality of the program.

MPI

The Message Passing Interface (MPI)[34] is a standard for inter-process communication in distributed computer systems by sending and receiving messages over a shared network. Developers are able to use MPI for the communication in distributed memory environments as well as shared memory computers. Similar to OpenMP, MPI provides language support for e. g. C, C++ and Fortran, but there are also libraries for other programming languages available, such as Java[35].

In contrast to OpenMP, workers in MPI are instantiated as processes and the worker management is, similar to OpenMP, mostly done by the MPI runtime. The programmer does not need to implement code for the worker life cycle, such as the creation or destruction of processes. There is only the need to tell MPI how much processes are needed for the current job. But unlike the worker management, the developer has to take care of the task-to-worker mapping and workload partitioning, which means that the programmer has to specify which tasks have to be computed on which processors. Furthermore, the programmer has to partition the data manually into subsets that are distributed to the workers.

Synchronization in MPI is done implicitly by the system but the developer still has to define where synchronized execution is needed. This can for instance be indicated by using synchronous operations for sending data in peer-to-peer communication mode, such as `MPI_Ssend`, or by using the `MPI_Barrier` operation for data exchange between all processes. The latter operation blocks

[34] http://www.mcs.anl.gov/research/projects/mpi/
[35] http://www.hpjava.org/mpiJava.html

until all processes that are grouped together have reached this part of the program. The following example illustrates the communication of processes in MPI and uses the `MPI_Barrier` operation to ensure that process 1 waits for process 0 to finish, which writes data to the output file, before reading data from this file. Furthermore, there are several calls of the `MPI_File_Sync` operation to ensure that the data written by process 0 is transferred to the storage device and that each process of the group can see the changes.

```
1  /* Process 0 */
2  int i, a[10];
3  for (i=0;i<10;i++)
4    a[i] = 5 ;
5
6  MPI_File_open(MPI_COMM_WORLD, "workfile", MPI_MODE_RDWR |
7          MPI_MODE_CREATE, MPI_INFO_NULL, &fh0);
8  MPI_File_set_view(fh0, 0, MPI_INT, MPI_INT, "native", MPI_INFO_NULL
       );
9  MPI_File_write_at(fh0, 0, a, 10, MPI_INT, &status);
10 MPI_File_sync(fh0);
11 MPI_Barrier(MPI_COMM_WORLD);
12 MPI_File_sync(fh0);
13
14 /* Process 1 */
15 int b[10];
16 MPI_File_open(MPI_COMM_WORLD, "workfile", MPI_MODE_RDWR |
17         MPI_MODE_CREATE, MPI_INFO_NULL, &fh1);
18 MPI_File_set_view(fh1, 0, MPI_INT, MPI_INT, "native", MPI_INFO_NULL
       );
19 MPI_File_sync(fh1);
20 MPI_Barrier(MPI_COMM_WORLD);
21 MPI_File_sync(fh1);
22 MPI_File_read_at(fh1, 0, b, 10, MPI_INT, &status);
```

Listing 6: MPI example: two-process-communication with synchronization (from: http://www.mcs.anl.gov/research/projects/mpi/mpi-standard/mpi-report-2.0/node215.htm)

UPC

The Unified Parallel C (UPC)[36] programming language is an extension of the C programming language and is intended to be used for parallel programming on HPCs. It supports both, shared memory machines as well as distributed memory environments, where UPC adopts the concept of partitioned memory where each processor can access each variable directly. Since the address space is logically partitioned into a number of per-thread address spaces, there are two different types of memory accesses which share the same syntax, access to the private address space and to the address spaces of the other threads. These memory accesses are handled differently by UPC to increase the performance.

In UPC, the workers are threads and the worker management is done implicitly, the programmer still has to specify the number of threads when running the application. The workload partitioning and task-to-worker mapping can be carried out in implicit mode as well as in explicit

[36] http://upc.lbl.gov/

mode. The implicit task-to-worker mapping and workload partitioning is provided by an API called `upc_forall` which automatically distributes the execution of loops through all workers in parallel. In explicit mode, the programmer has to specify which tasks are to be executed on which thread, similar to MPI. This includes not only the mapping of tasks to workers, but also the partitioning of the initial data to balance the workload of the threads.

The synchronization in UPC is more complex than in OpenMP or MPI and offers implicit and explicit synchronization constructs. UPC provides for instance memory consistency mechanisms like the `upc_fence` statement (implicit) which implies a strict write operation followed by a strict read operation. Other statements for ensuring the memory consistency are `upc_wait` & `upc_notify` (implicit) or `upc_lock` & `upc_unlock` (explicit), where the shared data can be protected against other processors by requiring for exapmle a strict read operation before returning from the method (in case of `upc_lock`). Another blocking synchronization expression is `upc_barrier` (implicit) which is similar to `MPI_Barrier` in MPI and blocks the further program execution until all threads have reached the defined part of the program. Furthermore, UPC distinguishes between two memory consistency models, relaxed and strict. The programmer can utilize them by including their headers: `upc_strict.h` and `upc_relaxed.h`.

From the view of the programming methodologies, UPC abstracts for example from the memory architecture and presents the memory as a partitioned global address space to the programmer who must therefore make no distinction between different system architectures. Furthermore, the UPC-API and the language specification often provides various mechanisms for implicit and explicit synchronization, worker mapping or workload partitioning. This has the advantage that the developer can choose to handle single problems manually, e. g., the memory access synchronization, but does not have the need to.

MapReduce

MapReduce is a programming model for high-performance clusters. The main goal of MapReduce is to abstract from common parallel development tasks, such as the worker management, the task-to-worker mapping, or the data distribution for allowing the programmer to set the focus of development onto the business logic of the application. It has been designed by Google[37] [DG04] to simplify the development process of parallel applications for the processing and generation of huge amounts of data. The core idea of this programming model is based on the *Map* and *Reduce* primitives of functional programming languages, such as Lisp or Haskell, where *Map* is a higher-order function that iterates over a list of elements, applies a given function to each element and builds a list of results. In functional programming, *Reduce* also is a higher-order function that iterates over a list of elements and applies a given function to them. However, in contrast to *Map*, *Reduce* builds a single return value instead of a list of values.

Since MapReduce is a programming model, there are various implementations available, like Google's original implementation, Apache Hadoop[38] or Greenplum[39]. The workers in MapReduce are usually implemented as processes and managed by the MapReduce cluster. The developer does not have to care about the creation or destruction of workers but there are various options that can be used to configure the cluster, such as the number of map, combine, or reduce tasks. Furthermore, there is no need to specify a task-to-worker mapping during development

[37] http://www.google.com
[38] http://hadoop.apache.org/mapreduce/
[39] http://www.greenplum.com/resources/mapreduce/

time. MapReduce maps each task to an available worker and re-executes tasks on other workers if necessary. This ensures a very high fault tolerance and can speed-up the computation in case of slow workers being part of the cluster.

The distribution of the input and the generated intermediate data can be carried out fully automatically, but there are also possibilities to influence the distribution over the cluster. The programmer only has to split the input data into the given number of chunks which then will be distributed to the mappers. The latter task is the workload partitioning and can either be done manually or automatically for a bunch of standard input formats. Depending on the implementation, such standard input formats can be plain text files, structured text files or even database tables. In MapReduce, a synchronization between map or reduce tasks is usually not required since the model provides no communication between worker tasks. Manual synchronization between workers could be required if the developer for example processes data that contains dependencies to other data which is changed during the computation.

Looking at the programming methodologies, MapReduce hides nearly every task of parallel programming by providing Interfaces against which the programmer can develop his application. This simplifies the whole development process, from the design and programming phase up to the test and maintenance phase and offers the possibility to focus on the core features of the application.

Summary

Table 1 shows a summary for the evaluation of the parallel programming models made above. Consequently, this table contains the four programming models which have been evaluated as well as the five evaluation criteria. Each criterion is rated as *automatic* if this task is fulfilled by the system, *manual* if the developer has to handle this task manually or *both* if the task is handled by the system but the programmer can choose to take over the task. An exception to the evaluation is the criterion of the programming methodologies where only positive and negative scorings are possible. There for instance, a scoring of "++" means that the programming model hides most of the parallelization tasks which enables the developer to get a quick start with the programming model and sets the main focus to the development of the core-features of the application.

	Worker management	**Task-to-worker mapping**	**Data distribution**	**Synchro-nization**	**Programming methodologies**
OpenMP	automatic	automatic	automatic	automatic	++
MPI	automatic	manual	manual	automatic	+/-
UPC	automatic	both	automatic	both	+
MapReduce	automatic	automatic	automatic	manual	++

Table 1: Evaluation of parallel programming models

The evaluation of Table 1 shows that there are two programming models which are suitable, OpenMP and MapReduce. Both models are rated withe the maximum value for the programming methodologies criterion, even if the synchronization of MapReduce is listed as manual. Due to the nature of the UPC model (it is based on the C programming language) and the intended design principles of LIMES M/R, UPC is not suitable for the parallelization of LIMES. The final decision for MapReduce has been made with respect to the actuality of the approach as well as the minimum effort that is required for the cluster setup and learning the model.

3 Architecture and Implementation

3.1 LIMES – Current Architecture

In addition to section 2.1.2, which gives a short introduction to the LIMES framework, this section highlights the architecture and internal processes when carrying out a link discovery task. At first, figure 4 shows the structure of the LIMES framework in its current stand-alone implementation. The framework consists of seven main modules which are described in the following [NEA10]. Furthermore, the activity diagram of figure 5 represents the sequence of activities that are examined during the link discovery process. In order to establish a relationship between the subsequent module explanation and the activity diagram, the activities of the diagram are framed by gray boxes that indicate the module responsibilities.

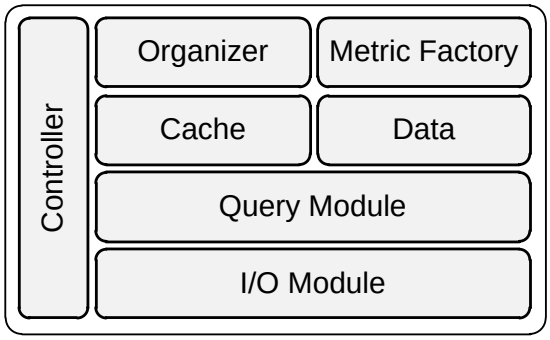


Figure 4: The LIMES architecture [NEA10]

Controller Module The controller is responsible for the coordination of the whole link discovery process as it is implemented by LIMES. Therefore, the controller integrates all six submodules (I/O, Query, Cache, Data, Organizer and Metric Factory) in order to realize the work-flow as it is depicted in figure 3 (see section 2.1.2). The general LIMES link discovery process starts with the validation of the passed configuration file and the extraction of the configuration parameters that are specified. Then, the input data is fetched from the specified knowledge bases into internal caches before the target cache, which holds the data from the target knowledge base, is reorganized under usage of the organizer module. After that, the similarity computation can be carried out much faster when using the reorganized target cache instead of the original target cache [NA10]. The last step of the process is the serialization of the linked instance pairs for further usage.

I/O Module The I/O module controls several tasks that occur over the whole link discovery process. Therefore, it builds the base of the LIMES architecture. One task of the I/O module is the validation of the configuration file. If the validation fails, the I/O module stops the execution with the appropriate error message. If the validation succeeds, the module parses the configuration file and provides different objects that contain the configuration properties. Later, the I/O module is used to serialize the results of the link discovery process. For this purpose, the module contains an interface (`de.uni_leipzig.simba.io.Serializer`) that can be implemented to plug new serialization formats into the framework.

Query Module Another important module is the query module. This part of the framework is utilized to fetch the data from the specified data sources. The current implementation contains a SPARQL query engine that is used to query remote knowledge bases. This component implements the interface `de.uni_leipzig.simba.query.QueryModule` and is therefore replaceable by custom query modules.

Cache Module While fetching the data from the source and target knowledge bases, the query module stores these data instances into temporary local caches for later processing tasks. The local caches must implement the interface `de.uni_leipzig.simba.cache.Cache` and typically offer methods to iterate over all cached instances or to add or remove instances. The current version of LIMES comes with a memory-based cache implementation. But since this may cause problems with scalability on low-memory machines, a file cache version is planned.

Data Module The data module can be seen as a module which contains helper classes. Classes of this module are used in the cache module of the application as well as the I/O module during the serialization phase. The data module contains for example the class `de.uni_leipzig.simba.data.Instance` which is a data structure for the fetched data instances.

Organizer Module The main advance that is introduced by the LIMES link discovery approach is implemented in the organizer module. By realizing a reorganization method which utilizes the triangle inequality as one of the characteristics of the metrics that are used for the instance comparisons, LIMES improves the runtime-complexity of the link discovery process without lowering the recall[40] of the process as it likely happens when applying blocking mechanisms [NA10]. The organizer module can be extended by implementing the interface `de.uni_leipzig.simba.organizer.Organizer`. LIMES comes with a standard organizer which implements the brute force comparison approach as well as several optimized organizers which take advantage of the mathematical characteristics of the applied metrics.

Metric Factory Module The metric factory module is a small one, that is applied by the organizer in order to calculate the similarity of two instances. For this purpose, the metric factory is parametrized with the metric expression which is extracted from the configuration. The current LIMES implementation comes with an implementation of the metric factory which offers a few different metrics. One can extend the metric factory module by implementing the interface `de.uni_leipzig.simba.metricfactory`.

[40] Recall is a measure for the hit rate of f. i. classification algorithms in information retrieval and is generally used in combination with the precision measure. A possible definition for both is given below:

> *In information retrieval, precision and recall are the fundamental parameters defining the behavior of an information retrieval system. Precision is defined as the proportion of relevant documents in the set of all documents returned by a search. Recall is defined as the number of relevant documents retrieved as fraction of all relevant documents.*

PlanetMath.org, January 2011, http://planetmath.org/encyclopedia/PrecisonRecall.html

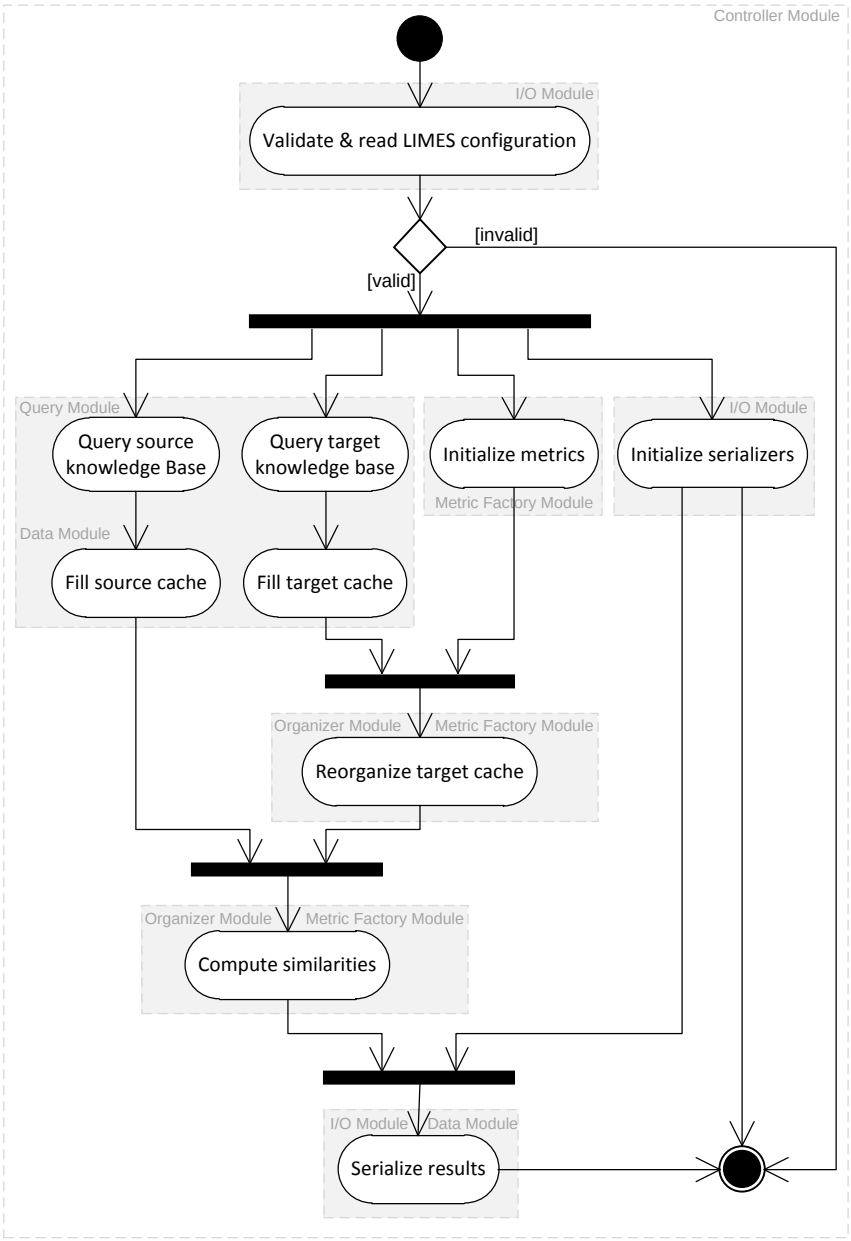


Figure 5: Internal link discovery process of LIMES

The above description covers all main parts of the activity diagram of figure 5. Therefore, the diagram is not further explained but will be extended in section 3.3 (Architecture and Implementation) to visualize the new architecture of the parallelized version of LIMES. The next task that has to be done before examining the parallelization is to decide about the parallelization approach as well as the parallel programming model which shall be used to parallelize LIMES. The next section accomplishes this task before the used programming model (MapReduce) is contemplated in detail.

3.2 Parallelization Prerequisites

A critical task that has to be carried out with great diligence as a preliminary for the parallelization of an existing application or conceptualizing a new application, is the decision about the parallel computing approach and the programming model to be used. This task is mainly influenced by the requirements the software should meet. A wrong requirement specification can lead to a suboptimal or even wrong decision about the parallelization approach which is likely to cause a large project delay or even a cancellation of the project. Therefore, the next section examines this task for the parallelization of LIMES with regard to chapter 2.2 which builds a solid basis for this still pending decision.

3.2.1 Selection of a Parallelization Approach

Link discovery is a computationally intensive task that can easily require several billion instance comparisons and thus a huge amount of time. Considering two knowledge bases S and T with about 350 000 instances[41], a full cross product of both knowledge bases, as used in the brute force approach, would lead to 122.5 billion instance comparisons. Even runtime optimized approaches like Silk or LIMES would result in several billion comparisons which would still require several hours to be carried out, depending on the hardware settings. Furthermore, a mapping task of this size could lead to memory problems when executed on standard hardware, which would also affect the runtime of the link discovery task[42]. For this reason, link discovery can be classified as an HPC problem which are typically parallelized using clusters rather than using the cloud or grid approach. Furthermore, link discovery doesn't require load balancing or a high availability, as for example required by parallel databases, and only needs the hardware as a shared resource instead of services that are typically provided in cloud or grid environments.

Because of the previously mentioned requirements, the decision about the parallel computer architecture and thus the computation approach fell to cluster computing. A further benefit of this approach is the level of control over the used hardware. Clusters are typically maintained and used by a single institution. This provides full control over the hardware and software of the cluster and simplifies for instance the measurement of the speedup which is central for scientific projects.

[41] For example, the DBpedia knowledge base contains approximately 364 000 instances of type `http://dbpedia.org/ontology/Person`

[42] This phenomenon occurred for instance during the execution of a duplicate detection task using LIMES. There, all instances of type `http://dbpedia.org/ontology/Animal` of the DBpedia knowledge base have been compared. After the reorganization of T, the mapping required approx. 1 billion comparisons but needed to swap data from the memory to the hard drive and vice versa. This affected the runtime significantly, so that LIMES required more than 7 hours for the execution, using 1GB of Memory.

Once, the decision about the parallelization approach is made, one has to consider different parallel programming models in order to reach a good compromise between the degree of parallelization, and thus the speedup, and for instance the conceptional and programming effort. Since the LIMES framework is implemented in Java, it is necessary to utilize parallel programming models that are available as Java libraries. This eliminates the effort for reimplementing LIMES for the usage with another language. In Chapter 2.2.2, a number of programming models for parallel applications are explained and evaluated. For the parallelization of LIMES, three of theses four models can be applied: OpenMP, MPI and, since the parallelization will utilize the cluster architecture, MapReduce. Each of the three models is available as a Java implementation and can thus be applied. In contrast, UPC cannot be applied since this model is based on C and would require a reimplementation of LIMES. Next, MPI is excluded from the list of possible models, since it requires too much manual programming effort, for example for the data distribution or the task-to-worker mapping as well as a lower degree of abstraction of the underlying architecture, compared to OpenMP and MapReduce. Finally, the decision for the MapReduce approach can not only be justified by the better results of the evaluation of section 2.2.2 this decision bases for example on the scientific relevance of this approach and its analogy to the executed link discovery task[43]. Further criteria that were decisive are the comparability of the parallelized LIMES implementation to Silk's parallel version which is also based on MapReduce, as well as the simplicity of the cluster setup using MapReduce.

3.2.2 MapReduce

After the decision about the parallel computer architecture and the programming model to be used has been made and justified in the previous section, this part of the thesis makes detailed contemplations about the used programming model, which is the MapReduce approach. Therefore, the focus of this section lies in highlighting the most important internal concepts, properties and processes of the MapReduce approach. In order to illustrate the development of parallel applications using MapReduce, this section on the one hand utilizes the open-source MapReduce implementation Apache Hadoop[44] and refers to its API in the current version 0.21.0[45,46]. On the other hand, the development is explained at the example of a small application, the word count example, which is often used in MapReduce tutorials[47].

As stated in section 2.2.2, MapReduce has been introduced by Google [DG04] with the goal of simplifying the development process of parallel programs for the handling of huge amounts of data. Possible problems that can efficiently be solved by applying the MapReduce programming model are for instance the processing of crawled documents or web server log files in order to build for example inverted indices on these data. Since most of those computations are conceptually not very complex but straight forward, the amount of work that is necessary for parallelizing such computations would exceed the effort that is necessary for development of the core functionality many times over. Applying the MapReduce framework on such problems reduces the parallelization effort to an absolute minimum, the implementation of two functions (or methods in object-oriented programming style, as applied when using Apache Hadoop), *Map* and *Reduce*.

[43] An analogy can for instance be found with the mapping phase. MapReduce maps key-value pairs which can be transferred to the instance pair comparison of the link discovery task.

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

[45] Hadoop-Mapred 0.21.0 API (http://hadoop.apache.org/mapreduce/docs/current/api/index.html)

[46] Hadoop-common 0.21.0 API (http://hadoop.apache.org/common/docs/r0.21.0/api/index.html)

[47] MapReduce tutorial with word count (http://hadoop.apache.org/mapreduce/docs/r0.21.0/mapred_tutorial.html

One main feature of MapReduce, besides its high level of abstraction, is the type of input and output data that can be processed and produced. The functions *Map* and *Reduce* require a set of key/value pairs as input and produce a set of key/value pairs as output. This may limit the number of problems that can be computed using the MapReduce approach but many real-world problems, such as a distributed grep, URL access frequency count, distributed sort or even the building of a reverse web-link graph, as they for instance arise at search engine operators like Google [DG04], are easily expressible using this model. The following listing 7 contains the method/function signatures of the two main functions that are to be implemented by the user, in order to compute data using MapReduce. The listing is based on the Hadoop MapReduce 0.21.0 API.

```
1 public void map(K1 key, V1 value, OutputCollector<K2,V2> output,
    Reporter reporter) throws IOException;
2
3 public void reduce(K2 key, Iterator<V2> values, OutputCollector<K3,
    V3> output, Reporter reporter) throws IOException;
```

Listing 7: Map and Reduce method signatures of Hadoop MapReduce 0.21.0 API [Fou11]

As shown in listing 7, the map function takes a single key/value pair as input and produces a set of intermediate output key/value pairs which are collected by the `OutputCollector`. The input and output keys can be of different types what shows that the MapReduce framework isn't limited to the processing of specific data types, such as Strings or Integers. Hadoop comes with a large number of different standard classes to be usable as keys and values (e. g., `Text`, `IntWritable`, `FloatWritable`, or `ArrayWritable`) but can easily be extended by user-defined data types to be used as keys or values. As the example list of data types indicates, each custom data type must be writable to enable the framework the transmission of these data. The user has to implement the `Writable` interface for this purpose. The `OutputCollector` collects all intermediate key/value pairs that are produced by the mapper while processing the input data. After the mapper has finished its work, the framework groups together (combines) all intermediate values that are associated with the same intermediate key before passing them to the reducers of the framework. The reduce function then processes sets of intermediate values that are associated with the same intermediate key. The reducers typically merge all the values for the same key in order to produce a possibly smaller set of values. The number of outputs per reduces is typically just one or even zero. Passing the intermediate values via an iterator is one mechanism of the MapReduce model to achieve a high scalability. The user is able to work on lists of intermediate values that are too large to fit in memory and would therefore result in a high degree of swapping[48] or even an application crash.

[48] Swapping can be described as follows:

> To move a program from fast-access memory to a slow-access memory ("swap out"), or vice versa ("swap in"). The term often refers specifically to the use of a hard disk (or a swap file) as virtual memory or "swap space".

Free On-Line Dictionary of Computing, February 2011, http://foldoc.org/swapping

Listing 8 shows a minimal but running word count example written in Java using the Apache Hadoop MapReduce implementation. Looking at the code confirms the advantages of the MapReduce model. Instead of writing hundreds of lines of code, one is able to design implement and run simple parallel applications by just implementing two methods, *Map* and *Reduce*. There is no need to deal with communication, synchronization or data distribution tasks, instead, the developer can focus on the core functionality of the application. The next paragraph explains the word count example to get a pictorial introduction to the internal processes of a MapReduce application.

```
1  package org.myorg;
2
3  import java.io.IOException;
4  import java.util.*;
5
6  import org.apache.hadoop.fs.Path;
7  import org.apache.hadoop.conf.*;
8  import org.apache.hadoop.io.*;
9  import org.apache.hadoop.mapreduce.*;
10 import org.apache.hadoop.mapreduce.lib.input.*;
11 import org.apache.hadoop.mapreduce.lib.output.*;
12 import org.apache.hadoop.util.*;
13
14 public class WordCount extends Configured implements Tool {
15
16   public static class Map extends Mapper<LongWritable, Text, Text,
       IntWritable> {
17     private final static IntWritable one = new IntWritable(1);
18     private Text word = new Text();
19
20     public void map(LongWritable key, Text value, Context context)
         throws IOException, InterruptedException {
21       String line = value.toString();
22       StringTokenizer tokenizer = new StringTokenizer(line);
23       while (tokenizer.hasMoreTokens()) {
24         word.set(tokenizer.nextToken());
25         context.write(word, one);
26       }
27     }
28   }
29
30   public static class Reduce extends Reducer<Text, IntWritable,
       Text, IntWritable> {
31     public void reduce(Text key, Iterable<IntWritable> values,
         Context context) throws IOException, InterruptedException {
32       int sum = 0;
33       for (IntWritable val : values) {
34         sum += val.get();
35       }
36       context.write(key, new IntWritable(sum));
37     }
38   }
39
40   public int run(String [] args) throws Exception {
41     Job job = new Job(getConf());
```

```
42      job.setJarByClass(WordCount.class);
43      job.setJobName("wordcount");
44
45      job.setOutputKeyClass(Text.class);
46      job.setOutputValueClass(IntWritable.class);
47
48      job.setMapperClass(Map.class);
49      job.setCombinerClass(Reduce.class);
50      job.setReducerClass(Reduce.class);
51
52      job.setInputFormatClass(TextInputFormat.class);
53      job.setOutputFormatClass(TextOutputFormat.class);
54
55      FileInputFormat.setInputPaths(job, new Path(args[0]));
56      FileOutputFormat.setOutputPath(job, new Path(args[1]));
57
58      boolean success = job.waitForCompletion(true);
59      return success ? 0 : 1;
60    }
61
62    public static void main(String[] args) throws Exception {
63      int ret = ToolRunner.run(new WordCount(), args);
64      System.exit(ret);
65    }
66  }
```

Listing 8: WordCount.java [Fou11]

Listing 8 contains two classes that implement the mapper and reducer to be used for processing the input data. The class `Map` extends the Hadoop class `org.apache.hadoop.mapreduce.Mapper` and is typed with `org.apache.hadoop.io.LongWritable`, `org.apache.hadoop.io.Text`, `org.apache.hadoop.io.Text` and `org.apache.hadoop.io.IntWritable`. This means that the custom word count mapper is able to process input key/value pairs with long `LongWritable` keys and `Text` values. The mappers will then produce intermediate key/value pairs that consist of `Text` /`IntWritable` combinations. The mapper of this example processes one line of the input data at a time, as provided by the `org.apache.hadoop.mapreduce.lib.input.TextInputFormat` (line 52) which splits the input file into single lines and distributes them as key/value pairs consisting of the line number and the line contents. The map method then splits each line into a list of tokens using the `StringTokenizer` and emits a key/value pair of `<<word>, 1>` for each token, the so-called intermediate key/value pairs. These intermediate key-value pairs are now passed to the reducer of the application, `Reduce`. `Reduce` extends the class `org.apache.hadoop.mapreduce.Reducer` and is able to process intermediate key/value pairs of `Text` and `IntWritable`, thus the output of the mapper. The reducer now emits for each word of the input text the total number of occurrences that are written to the specified output file. Furthermore, there is an option to define a combiner class that is used to do local key/value combinations on the map task worker node after the map tasks which avoids the transmission of duplicate intermediate keys. In this example, the combiner uses the same class as the reducer (line 49). The following listings show the resulting output key/value pairs of an example run of the word count application after each step, the mapper, combiner and reducer steps.

```
>>Input File 1:<<
=================
Hello World
Hello
Bye World



>>Input File 2:<<
=================
Hello Hadoop
Hello World
Goodbye Hadoop
Bye
```

Listing 9: WordCount sample: input data

```
>>Map Task 1:<<
===============
<Hello, 1>
<World, 1>
<Hello, 1>
<Bye, 1>
<World, 1>


>>Map Task 2:<<
===============
<Hello, 1>
<Hadoop, 1>
<Hello, 1>
<World, 1>
<Goodbye, 1>
<Hadoop, 1>
<Bye, 1>
```

Listing 10: WordCount sample: mapper output

```
>>Combiner 1:<<
===============
<Bye, 1>
<Hello, 2>
<World, 2>


>>Combiner 2:<<
===============
<Bye, 1>
<Goodbye, 1>
<Hadoop, 2>
<Hello, 2>
<World, 1>
```

Listing 11: WordCount sample: combiner output

```
>>Global Reducer:<<
===================
<Bye, 2>
<Goodbye, 1>
<Hadoop, 2>
<Hello, 4>
<World, 3>
```

Listing 12: WordCount sample: reducer output

Besides the implementation of the two main methods, *Map* and *Reduce*, there is just a very small organizational overhead for setting up and scheduling a MapReduce job. This configuration work is done in the `run` method of the applications main class which implements the interface `org.apache.hadoop.util.Tool`. There a new `org.apache.hadoop.mapreduce.Job` is set up and configured, for instance with the job jar which contains the word count example code, a job name, the classes to be used as mappers, combiners and reducers and the types of the input and output key/-value pairs[49]. Once, the job is configured, it can be run with or without waiting for its completion to return.

[49] There are much more properties that can be configured for a MapReduce job, for example the number of mappers or reducers of the job.

After this small example-based introduction into the MapReduce-programming model and its internal processes, figure 6 depicts the general work-flow of MapReduce applications. This execution overview contains numbered labels that correspondent to the steps below.

(1) When a user schedules a MapReduce program, the MapReduce library in this program first splits up the input data, which is typically file-based input (here 5 input files), into M input splits. After this splitting is done, the user-program is distributed (copied) to a number of machines in the cluster.

(2) A MapReduce cluster consists of two different types of nodes (machines), one master node and a number of worker nodes. The cluster for the user program consists of M map tasks (worker nodes) and R reduce tasks (worker nodes) that process the data. The master node controls the job flow and organizes the communication within the cluster. It picks for example idle worker nodes from the cluster and assigns map and reduce tasks to them.

(3) Workers with an assigned map task first read the contents of the input splits that are assigned to them and extract input key/value pairs of that input splits. These input pairs are then passed to the map function of the program. After that the workers start processing the input data and buffer the produced key/value pairs in their local memory.

(4) In order to avoid memory overflows and to forward the intermediate results to the reducers, the buffered intermediate key/value pairs are periodically written to the local disks of the workers and partitioned into R regions. The workers then pass back the locations of the serialized intermediate pairs to the master node who forwards these paths to the reduce workers of the cluster.

(5) After the master node has notified the reduce workers about the intermediate pair locations, the reduce workers need to fetch these data from the map worker nodes. Therefore, the reducers utilize Remote Procedure Calls (RPC)[50] or Remote Method Invocation (RMI)[51] to read the data from the local disks of the map workers. After a reduce worker has read all intermediate data, a sorting by the intermediate keys is performed in order to group all pairs that are associated with the same key are grouped together. This ensures that each reduce task invokes the reduce method only once per unique intermediate key with the whole set of intermediate values that are associated with that key.

(6) The last step is the iteration of the reduce worker over the sorted intermediate key/value pairs and passing each unique key with its associated set of values to the reduce method. There, the intermediate data is processed and the output of the reduce methods is appended to the final output file of this reduce partition. The global output of the MapReduce job is then available in the R output files (one per reduce task) and can be read from the distributed file system for further use.

[50] RPC

> A protocol which allows a program running on one host to cause code to be executed on another host without the programmer needing to explicitly code for this. RPC is an easy and popular paradigm for implementing the client-server model of distributed computing.

Free On-Line Dictionary of Computing, February 2011, http://foldoc.org/remote+procedure+call

[51] RMI

> Part of the Java programming language library which enables a Java program running on one computer to access the objects and methods of another Java program running on a different computer.

Free On-Line Dictionary of Computing, February 2011, http://foldoc.org/rmi

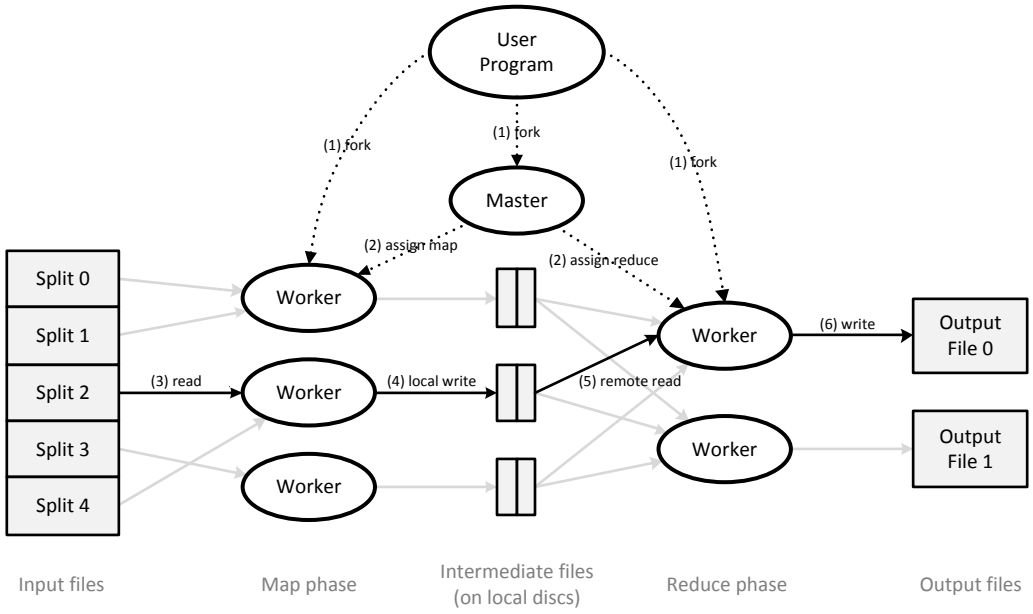


Figure 6: Work-flow of a MapReduce program [DG04]

As a conclusion of the MapReduce framework contemplations, three of the many further properties and Mechanisms of the MapReduce model are explained, the fault tolerance mechanism, the locality of data on the cluster and the backup of running tasks. These are mechanisms that typically have to be implemented by the developer and are associated with a large amount of conceptional work.

Fault tolerance is a critical task since MapReduce clusters usually run on a large number of standard machines. There are two types of failures that occur in a MapReduce cluster: worker failures and master failures. The failure-handling strategy for worker failures is based on re-execution and ensures that the cluster is resistant against a large-scale worker failure rate. To check the availability of worker nodes, the master node periodically pings every worker with a fixed timeout. If a worker does not answer within the given time, the master marks this worker as failed and resets all completed and running map tasks and all running reduce tasks that are associated with this worker to their initial state. These tasks can then be re-executed on working nodes in order to complete the job. In the case of map tasks, the master needs to reset even completed tasks, because their output data is stored in their local file system and thus unreachable. The output of reduce tasks is stored in a global distributed file system and therefore still reachable. In contrast to the worker failure handling strategy, the handling of master failures is checkpoint based. Even if master failures are unlikely to happen, since there is only one master node in a cluster, the master node can periodically write checkpoints of its internal data structures. in case of a failure, another running node becomes the new master node and the execution can be started from the latest checkpoint[52].

Locality of data is another task to be considered while developing cluster applications. This can reduce the network load significantly and thus speed up the whole computation. MapReduce

[52]The handling of master failures as described is just a proposal in the initial introduction of the MapReduce model and usually not implemented in the different distributions.

clusters typically come with an implementation of a distributed file system, such as the Google File System (GFS) or the Hadoop Distributed File System (HDFS), which stores input and output data replicated and distributed over the nodes of the cluster[53]. The data distribution and replication is taken into account by the master node when scheduling map tasks on the machines of the cluster. The master node usually tries to schedule map tasks on nodes that contain a replica of the corresponding input data or at least on a node in the near of such a node. This results in local reads of a majority of the input data avoiding a time consuming transport of the data.

Backup tasks are used by MapReduce to avoid "stragglers"[54] slowing down the execution of a MapReduce job by a significant amount of time. For this purpose, the master node schedules backup tasks of the remaining tasks that are still in progress. Whenever one of these executions finishes, the primary or the backup execution, the task is marked as completed.

3.3 LIMES M/R – Architecture and Implementation

The goal of this section is to explain the architecture and implementation of the parallelized version of LIMES, namely LIMES M/R. The first part of this section describes the design principles on which the design and implementation of the application are based. After this introduction, the section continues with the description of the inner structure of the MapReduce-based implementation of LIMES and presents some details of the implementation.

A preliminary for the design and implementation of LIMES M/R has been to determine a strategy in which way to extend the existing implementation to retrieve a parallelized version of LIMES. In order to find an adequate solution, a deeper analysis of the current design and its implementation was necessary. During this reverse engineering[55] process it became clear that the current LIMES implementation mainly follows some essential principles of Object-Oriented Design (OOD) which makes it easily extensible [Mar00a]. LIMES largely meets for example the Open-Closed Principle (OCP) [Mar00b] whose essence targets the extensibility of the system. The key message of OCP is the following: The modules of a software should be written in a manner that makes them extensible without requiring them to be changed, thus without touching their source code. As stated in [Mar00a], the key to OCP lies in abstraction. There are several methods for achieving the goals of

[53] The GFS for instance divides files into blocks of 64MB and stores several (typically 3) copies of each block [DG04].

[54] A machine that takes an unusually long time to complete one of the last few map or reduce tasks in the computation. [DG04]

[55] Reverse Engineering

> The process of analyzing an existing system to identify its components and their interrelationships and create representations of the system in another form or at a higher level of abstraction. Reverse engineering is usually undertaken in order to redesign the system for better maintainability or to produce a copy of a system without access to the design from which it was originally produced.

Free On-Line Dictionary of Computing, February 2011, http://foldoc.org/reverse+engineering

the OCP, for instance by using polymorphisms[56] instead of hard-coding object calls, which makes objects replaceable. This can for example be achieved by specifying interfaces or abstract classes and programming against these abstracted types instead of their concrete implementations. LIMES applies this method in nearly every of its modules to achieve a high degree of extensibility. As an example, the Organizer module provides an interface `Organizer` whose code is shown in listing 13. This interface declares methods which are used to compute the exemplars that are responsible for the large speedup of the mapping process as well as methods for the retrieval of similar instances for a given one. The whole framework then only references this organizer-abstraction and neglects any concrete implementations of this interface. This can also be associated with the Liskov Substitution Principle (LSP) [Mar96b] and makes the organizers easily exchangeable and thus lets the user add new or improved functionality.

```
1 public interface Organizer {
2   public void computeExemplars(Cache c, MetricFactory m, int
       nrExemplars);
3   public void computeExemplars(Cache c, MetricFactory m);
4   public HashMap<String, Float> getSimilarInstances(Instance
       instance, float threshold, MetricFactory metric);
5   public int getComparisons();
6     public float getComparisonTime();
7   public HashMap<Instance, TreeSet<Instance>> getExampleMap();
8   public void setExampleMap(HashMap<Instance, TreeSet<Instance>>
       map);
9 }
```

Listing 13: The organizer interface (`de.uni_leipzig.simba.organizer.Organizer`)

Another design principle that is strongly related to the OCP is the Dependency Inversion Principle (DIP) [Mar96a]. In short, the core message of this principle is to depend upon abstractions rather than concretions. Therefore, the DIP states the primary mechanism for reaching the goals of OOD that are expressed by the OCP. As stated in the previous section, LIMES makes extensive use of abstraction in order to achieve extensibility as well as stability and maintainability.

As seen in the previous paragraphs, the architecture of LIMES follows some important object-oriented design principles which makes it easily extensible by new features. This makes a minimum-invasive extension approach for LIMES M/R feasible and the first-choice strategy. The advantages associated with this strategy are obvious: The changeability and maintainability of LIMES remains after the parallelization. When extending a system without touching its code, there is no need to open a new development branch for the existing components. The evolution of LIMES and LIMES M/R can thus run in parallel without requiring any awkward and error-prone synchronization and integration tasks. Additionally, this approach saves development time and ensures the software quality to stay on a high level. A further highly important reason in favor for this approach is to

[56] Polymorphism

> A concept first identified by Christopher Strachey (1967) and developed by Hindley and Milner, allowing types such as list of anything. ... This is known as parametric polymorphism. Polymorphic typing allows strong type checking as well as generic functions. ... Ad-hoc polymorphism (better described as overloading) is the ability to use the same syntax for objects of different types, ... Parametric polymorphism allows the same object code for a function to handle arguments of many types but overloading only reuses syntax and requires different code to handle different types.

Free On-Line Dictionary of Computing, February 2011, http://foldoc.org/polymorphism

keep the framework extensible after the parallelization and especially to avoid breaks with current extensions of LIMES. The latter means that existing extensions of the LIMES framework should also be working with the parallelized version, without any change being necessary, which is another goal of the design and implementation of the parallel version of LIMES.

3.3.1 Internal Processes

For describing the internal link discovery process of LIMES M/R, this section contains an activity diagram (figure 7). Since the LIMES M/R implementation is realized as an extension of the original framework, this diagram extends the activity diagram describing the LIMES process (figure 5) with some MapReduce-related activities. Therefore and in contrast to figure 5, this diagram consists of two Swimlanes (LIMES M/R & Hadoop) that show up which activity is executed in which responsibility area. Activities that are included in the LIMES M/R-Swimlane are controlled and executed by LIMES M/R while the MapReduce framework (here Apache Hadoop) is responsible for the execution of activities in the Hadoop-Swimlane. In the following, the internal link discovery process is explained with respect to the diagram.

When running a MapReduce-based link discovery task, LIMES M/R first checks the MapReduce configuration file that is passed in addition to the LIMES configuration file. A LIMES MapReduce configuration file is based on XML and must conform to the DTD of listing 15. The MapReduce configuration of LIMES is explained in detail in the next section (Implementation). In any case, the first step or activity (in the remainder of this section, activities are also referred to as steps) consists of the validation of the MapReduce configuration file against the document type definition, as well as the extraction of the MapReduce settings of this configuration, which are for example the number of mappers for the MapReduce job. If the configuration is valid, the next step of the process creates and configures the MapReduce job using these information before scheduling the job for execution at the MapReduce framework. In case of a failing validation of the configuration file, the link discovery process is aborted by LIMES M/R. Once, the job is scheduled for execution, Hadoop starts the job by executing the MapReduce work-flow as described in the MapReduce section as depicted in figure 6. During this job execution the responsibilities for the atomic activities switch between LIMES M/R and Hadoop several times.

Some tasks that are executed when running a MapReduce job are not included in the chart, such as the distribution of the program to the master and worker nodes, since this are internals that are not important for the contemplation of the LIMES M/R process. The job execution starts here with the retrieval of the input splits that are later passed to the mappers of the job in order to find links between both knowledge bases. This activity is fully executed on the master node of the cluster and calls some routines of LIMES M/R which do the following. First, the LIMES configuration file, as described in section 3.1, is validated against its DTD and read. If the validation fails, the framework stops the execution of the job and returns with an appropriate error message. In the other case, the configuration data are extracted and LIMES M/R starts producing input splits for the mappers of the job. Therefore, the source and target knowledge bases are queried and instances matching the restrictions are extracted to the source and target caches. Furthermore, the underlying metrics to be used for the instance comparisons are initialized. As soon as the target cache is filled and the metrics are initialized, LIMES M/R starts with the reorganization of the target cache. The initial paper for the LIMES approach [NA10] describes this process and its purpose in detail. After the reorganization of the target cache has finished, the production of the input splits is started. The number and thus size of the input splits is thereby mainly influenced by the number of mappers of the job. The current implementation of LIMES M/R, which is also used for the evaluation, splits

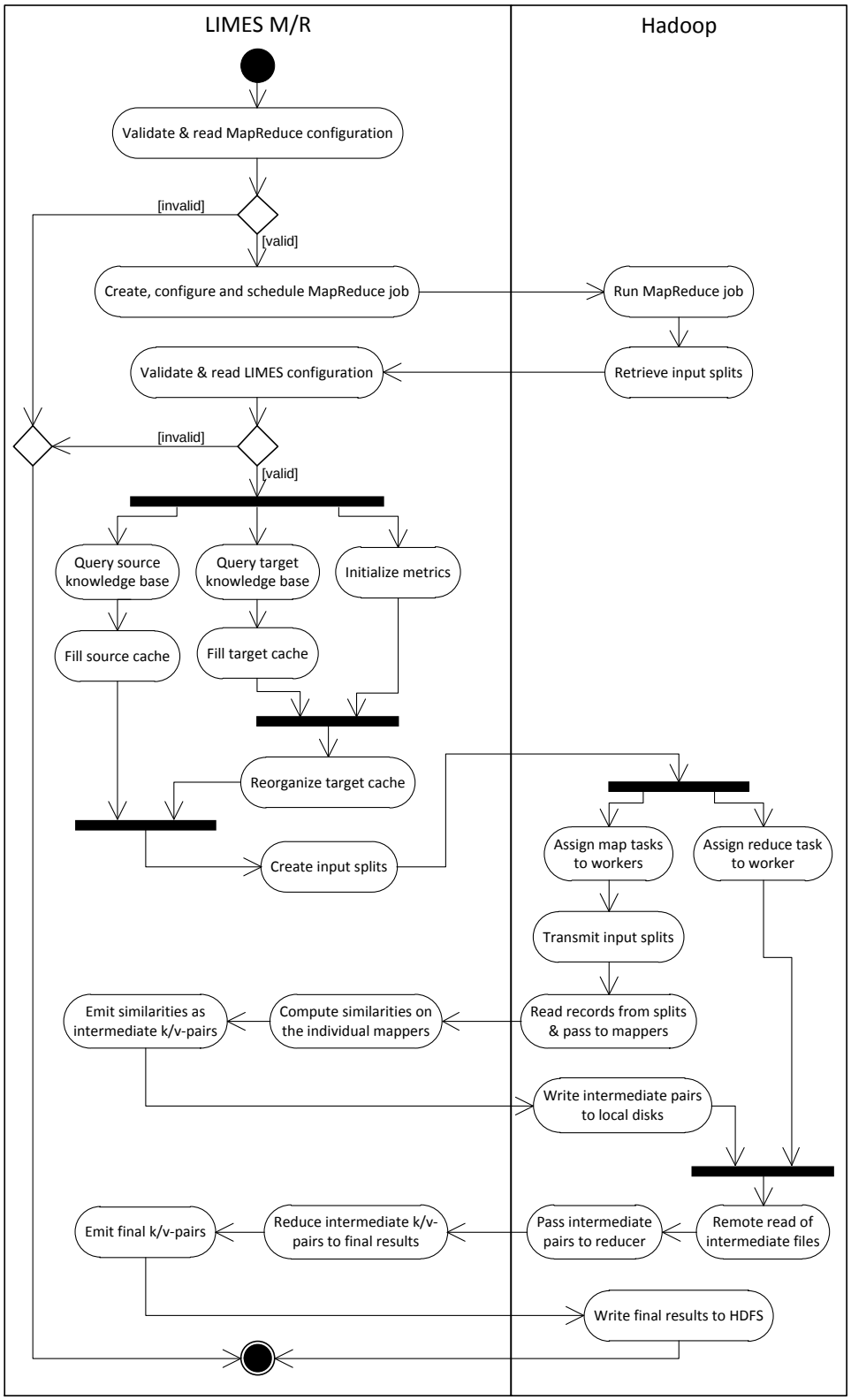


Figure 7: Internal link discovery process of LIMES M/R

the source cache into a number of cache fragments which equals the number of mappers defined by the user. These source cache fragments are then used as the input keys for the map tasks of the job. Furthermore, there are two different implementations of the input split generator that either uses the whole reorganized target cache as the value for each input key/value pair or splits the reorganized target cache into 10 fragments and uses these fragments as values for the input splits. In the latter case, the current implementation creates 10 times more map tasks than specified by the user. This input split generator has been implemented for evaluation purposes only. A final implementation should then calculate the number of source and target cache splits depending on the specified number of map tasks to give the user more control over the job settings. A detailed analysis of the split generator is done in the implementation section (section 3.3.2).

After the input splits are produced, the further few steps of the job are executed by the MapReduce framework, so they are depicted in the Hadoop Swimlane. The next steps, the MapReduce framework goes through, are at first the assignment of map and reduce tasks to the worker nodes of the cluster. The implemented standard setting for the number of reducers that are used to join the intermediate key/value pairs coming from the mappers, uses just one reducer in order to join the mapper output into a single file lying in the distributed file system. In contrast to the reducers, a MapReduce cluster (here, the running jobs are meant) usually consists of a large number of map tasks with the aim at distributing the work over the whole cluster and ensuring a good work load balance, so does LIMES M/R. As stated in the previous section, the number of map tasks can be influenced by the user. However, Hadoop uses the generated input splits to feed the assigned map tasks with input data, but in contrast to most other MapReduce programs, LIMES M/R does not store its queried input data or the generated input splits in the distributed file system and passes their locations to the mappers. LIMES M/R transfers the generated input splits directly to the map tasks in order to build in-memory caches for the mappers. This is even for huge knowledge bases possible since the problem size is reduced significantly for the single map tasks by means of the input splitting. Due to the direct transferring of the input splits, the activity diagram contains an activity named *"Transmit input splits"* instead of a *"Read input splits"* activity. Next, Hadoop calls the `RecordReader` for each map task and input split that shall extract the single key/value pairs from the splits in order to be consumed by the mappers. The current implementation does not apply any optimizations to this step and simply creates the caches out of the passed binary input data coming from the input splits. It is planned for a future version to implement some optimizations, such as the further splitting of the input splits into single key-value pairs instead of passing the whole splits to the map tasks.

The next two steps are executed under the responsibility of LIMES M/R and are therefore depicted in the LIMES M/R Swimlane. The similarity computation process follows the one that has been introduced by LIMES [NA10] and is executed in each map task for the input split that has been passed to it. In short, each instance of the key-cache (a fragment of the original source cache) is at first compared to each exemplar of the reorganized target cache or its fragment, depending on the version of LIMES M/R, as described for the input split generation. If the similarity lies above a given threshold and the triangle inequality-property of the metric space can be held, the instance is compared to each instance in the subspace of the exemplar, as long as the inequality can be applied (please refer to [NA10] for the detailed description of the LIMES-concept). As soon as the similarity computation for one instance of the source cache fragment is completed, the mapper emits the resulting key/value pairs as intermediate pairs to the Hadoop output collector which then writes the results to the local disks of the worker that is executing the current map task. In the following, the reduce task is applied as soon as one map task produces intermediate key/value pairs. The job tracker now sends the locations of the intermediate results in the distributed file

system to the worker node that runs the reduce task of LIMES M/R which then reads these pairs from the file system using RMI. This is all done internally by Hadoop and is therefore depicted in the Hadoop Swimlane of the activity diagram. Once, the intermediate files are fetched, the reduce worker passes the intermediate key/value pairs to the reduce task where these pairs are reduced to the final output of the framework. The reducer of LIMES M/R has to execute just one simple task, the combination of the incoming intermediate results to just one final result file. For this reason, the reducer simply passes all incoming pairs as plain text to the output collector of Hadoop which writes them into a single file on the distributed file system. If no further error occurs during the link discovery process, the execution of the LIMES M/R job successfully terminates after the last map task has finished its similarity computation and the so produced intermediate results have passed the reducer.

3.3.2 Implementation

The purpose behind this section is the analysis of the LIMES M/R implementation in order to demonstrate how applications are parallelized using the MapReduce approach using Hadoop and to get a deeper look into the implementation details of LIMES M/R, for example how the input splits are generated and transferred between the nodes of the cluster. This section therefore contains a number of code listings and Unified Modeling Language (UML)[57] diagrams by which the implementation is described. Furthermore, the source code reflection follows the outline which is given by the earlier described process, as depicted in figure 7. Based on this process, the implementation of the classes that are responsible for the activities of the chart are discussed.

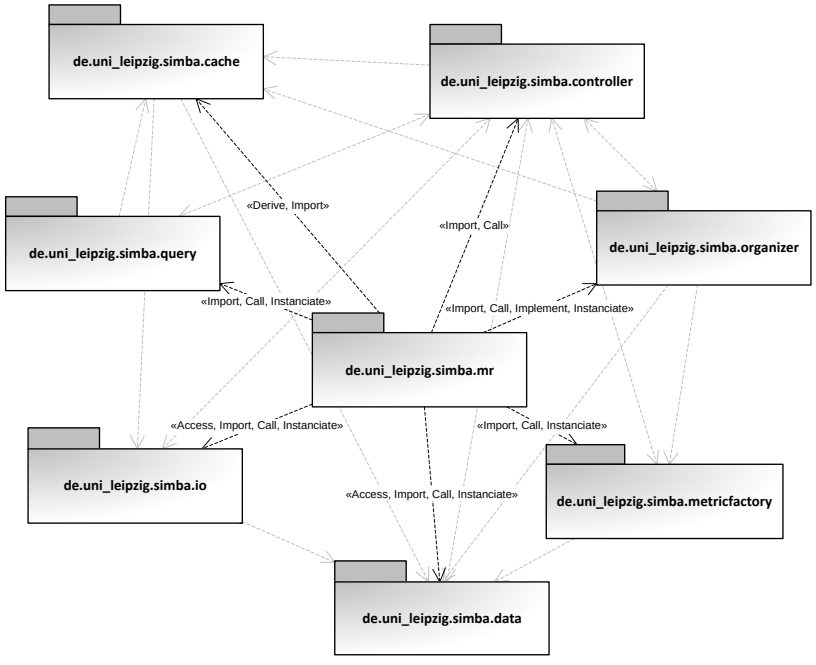


Figure 8: LIMES M/R – Package diagram

[57] http://www.uml.org/

To get a good introduction to the source code analysis, figure 8 depicts a package diagram of LIMES M/R. As described in section 3.3, LIMES M/R is just an extension of the framework in order to avoid breaks with other extensions and run configurations. This is also confirmed by the package diagram. The current LIMES M/R implementation is limited to a single package, `de.uni_leipzig.simba.mr`, and is contained in the program jar file. Yet, for a better separation of LIMES M/R and LIMES, it is planned to outsource the package into a standalone program archive and restructure it. The diagram of figure 8 shows the interaction of each of the original LIMES-packages (gray arrows) and the integration of the LIMES M/R-package into this structure. The imports, calls, etc. between `de.uni_leipzig.simba.mr` and the LIMES-packages are depicted as dashed black arrows that are labeled with the appropriate operations. As the diagram shows, there are only outgoing arrows from `de.uni_leipzig.simba.mr` to the other packages which is a first indicator that LIMES M/R is realized as a simple extension of LIMES instead of reimplementing the framework. LIMES M/R uses every of the original LIMES packages while implementing a new work-flow.

The package `de.uni_leipzig.simba.mr` of the current implementation consists of 12 classes that are depicted by the class diagram in figure 9. Besides the classes, the diagram contains their dependencies and interactions to visualize their structure. Since this diagram only depicts the class names, a full diagram of the package can be found in the appendix of this work (figure 21). This chart is intended to complement the package chart of figure 8 and shall help to keep the overview during the code reflection later in this chapter.

Looking at the chart, it is noticeable that there are very few dependencies and interactions between the classes of the package. For example, the class `LimesMRMapper` has only two outgoing arrows, one to the class `LimesMRCache` and one to `LimesMROrganizer`. There is even a class that does not seem to be integrated into the MapReduce-based link discovery process, `LimesMRReducer`. This class neither has outgoing links, nor any incoming links that indicate an interaction with this class. This comes from the integration of LIMES M/R with the Hadoop MapReduce framework and reflects the great benefit of using the MapReduce programming model for application parallelization tasks. As discussed earlier and shown by the activity diagram of LIMES M/R (figure 7), the developer can focus on the development of the core application than investing too much work into tasks such as communication, synchronization, etc. which makes MapReduce to one of the state-of-the-art parallel programming models. In the case of the reducer class of LIMES M/R, there are no dependencies to other classes of the package because of Hadoop interacting with this class exclusively. Hadoop kicks off the reduce task of the framework which is implemented by `LimesMRReducer` and organizes the whole communication with this task until the link discovery process has finished.

The first step of the process is the validation of the MapReduce configuration file and the extraction of its configuration parameters. This step, as well as the setup and scheduling of the MapReduce Job is executed by the controller of the LIMES M/R framework, `LimesMRController`. The `run` method of this class is shown in listing 14. There, the first statements are used to read the MapReduce configuration data before setting up a MapReduce job using these parameters. Please note that the configuration reader is described in the next section in order to explain the job configuration and scheduling in a consecutive way. After the extraction of the configuration parameters, the next statements are just setting up the logger for the framework and are therefore partly omitted. After that, the second step of the activity chart (figure 7) is executed, the setup and scheduling of the MapReduce job. For this purpose, the controller creates a new `JobConf` and

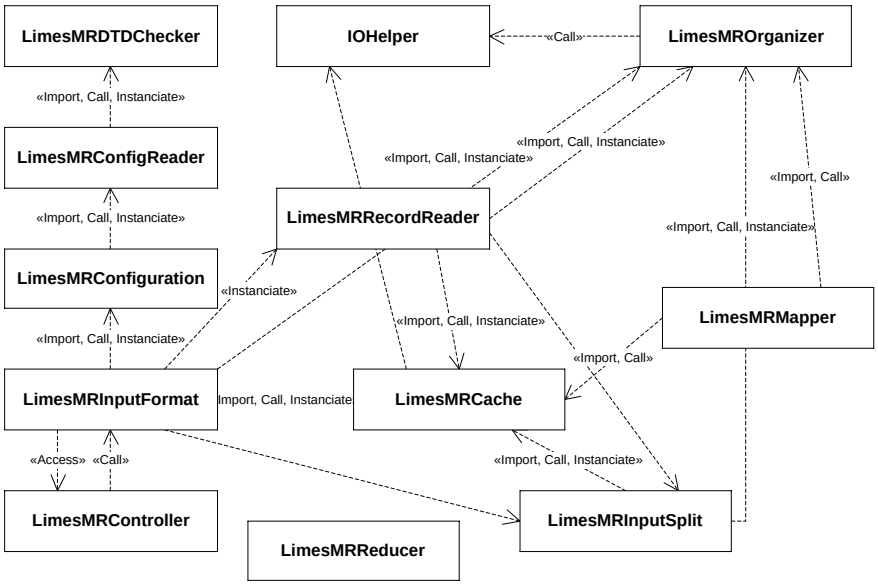


Figure 9: Class diagram of the LIMES M/R package

passes an example class to its constructor (line 10). By passing an example class to the `JobConf`[58], one can easily specify the jar file that contains the program logic, which is necessary since this jar has to be transferred to the worker nodes of the cluster in order to execute the job. Afterwards, the controller configures several properties of the job as specified in the MapReduce configuration file. There are for example the name of the job (line 11) or the number of map and reduce tasks for the job (lines 13–16). Line 12 sets a custom property of the job (here the Organizer class to be used for the similarity computations) in order to be used later in the job flow. It is necessary to use this feature of the `JobConf` class to be able to use custom properties. Passing those options directly to other methods or providing getter-methods for retrieving them will not work since for instance map tasks on the worker nodes of the cluster are executed withing different Java Virtual Machines (JVM). The next two lines set the `InputFormat` of the job which is responsible for the creation of the input splits and prepare it by passing the configuration. Afterwards, the types of the output keys and values (that are written by the reducer) are set, since Hadoop needs to know how to handle the final key/value pairs for writing them to the distributed file system. Then, the controller specifies the mapper and reducer class for this job. The mapper class is the class which implements the comparison of the source and target knowledge bases while the reducer class just passes the intermediate results of the mappers, as described in section 3.3.1. The lines 27–29 specify the output path on the distributed file system where the results are written to and set a custom output format (here `TextOutputFormat` to enable Hadoop writing the output correctly. At the end of the method, the controller just schedules the job and blocks until its completion.

[58] `org.apache.hadoop.mapred.JobConf` is part of the Hadoop framework. In the remainder of this thesis, several classes and interfaces of the Hadoop framework are used but will not explicitly declared as such. Please refer to the API documentation of Hadoop at http://hadoop.apache.org/mapreduce/docs/current/api/index.html and http://hadoop.apache.org/common/docs/current/api/index.html.

```
1  public void run(String limesConfigPath, String MRConfigPath, String
       organizerClazz) {
2    LimesMRConfigReader cr = new LimesMRConfigReader();
3    cr.validateAndRead(MRConfigPath);
4
5    logger = Logger.getLogger(this.getClass());
6    if (cr.needsLogger()) {
7      //initialization of the logger
8    }
9
10   JobConf conf = new JobConf(new Configuration(), LimesMRController
         .class);
11   conf.setJobName(cr.getJobName());
12   conf.set(LimesMRConfiguration.CONFIG_ORGANIZER_CLASS,
         organizerClazz);
13   if (cr.getNumMapTasks() > -1) {
14     conf.setNumMapTasks(cr.getNumMapTasks());
15   }
16   conf.setNumReduceTasks(cr.getNumReduceTasks());
17
18   conf.setInputFormat(LimesMRInputFormat.class);
19   LimesMRInputFormat.setInput(conf, limesConfigPath);
20
21   conf.setOutputKeyClass(Text.class);
22   conf.setOutputValueClass(Text.class);
23
24   conf.setMapperClass(LimesMRMapper.class);
25   conf.setReducerClass(LimesMRReducer.class);
26
27   Path outputPath = new Path(cr.getMrOutputPath());
28   TextOutputFormat.setOutputPath(conf, outputPath);
29   conf.setOutputFormat(TextOutputFormat.class);
30
31   if (cr.getNumRuns() == 1) {
32     try {
33       JobClient.runJob(conf);
34     } catch (IOException e) {
35       logger.log(Level.FATAL, "The LIMES Map/Reduce job couldn't be
             executed.", e);
36     }
37   } else {
38     // multiple runs for testing only
39   }
40 }
```

Listing 14: `run` method of class `LimesMRController`

An important prerequisite, and thus the first activity to be executed by LIMES M/R, is the extraction of user-defined parameters for the MapReduce job. As already mentioned above, this task is done by the class `LimesMRConfigReader`. Due to the fact that this class just verifies and extracts parameters from an XML-based configuration file using standard Java XML features (the Document Object Model (DOM) implementation), a listing of this class is omitted. Instead, listing 15 contains the DTD of the newly added configuration options. The decision was made for this additional DTD, and thus the need for a second configuration file when starting the MapReduce-based

version of LIMES, in order to follow the targeted design principle, realizing LIMES M/R as an extension of LIMES without restructuring or reimplementing any of its components. The DTD is partitioned into two sections, MapReduce-related configuration parameters and a debugging section. The current implementation expects a job name and the output path in the distributed file system as mandatory parameters. Furthermore, one can configure the number of map and reduce tasks used by the job. The second part of the configuration file, the debugging section, offers the possibility to specify a few output and runtime-related parameters that are used to keep track of the program execution. This section is fully optional and one can for example configure the log file path (either console or a file path) or the number of runs of the current job. This parameter is for instance used for the evaluation of the framework which is discussed in chapter 4. This section then also lists the example configuration files that are used for the evaluation task.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <!ELEMENT JOB (MAPPER?, REDUCER?, OUTPUT, DEBUG?)>
3  <!ATTLIST JOB NAME CDATA #REQUIRED>
4  <!ELEMENT MAPPER (NUMBER)>
5  <!ELEMENT NUMBER (#PCDATA)>
6  <!ELEMENT REDUCER (NUMBER)>
7  <!ELEMENT OUTPUT (PATH)>
8  <!ELEMENT PATH (#PCDATA)>
9
10 <!ELEMENT DEBUG (LOG?, RUNS?)>
11 <!ELEMENT LOG (OUT)>
12 <!ATTLIST LOG LEVEL (ALL|DEBUG|OFF|INFO|WARN|ERROR|FATAL|TRACE) "
       ALL">
13 <!ELEMENT OUT (CONSOLE|FILE)>
14 <!ELEMENT CONSOLE EMPTY>
15 <!ELEMENT FILE (#PCDATA)>
16 <!ATTLIST FILE APPEND (true|false) "true">
17 <!ELEMENT RUNS (#PCDATA)>
```

Listing 15: LIMES M/R configuration DTD

The next set of steps (starting at *"Retrieve input splits"*) is triggered by Hadoop (see figure 7) and comprises several LIMES M/R classes that utilize, in turn, several classes of the LIMES framework. For retrieving the input splits for a MapReduce job, Hadoop requires a class of the user program that either implements the interface `InputFormat` or extends the class `InputFormat`[59]. Although there are many different standard implementations that split up for example plain text files in a line oriented way (`TextInputFormat`) or extract data directly from a database (`DBInputFormat`), LIMES M/R implements its own input format class, `LimesMRInputFormat`. Listing 16 shows the most important method of this class, `getSplits`. Hadoop utilizes this method of the `InputFormat` implementing classes to retrieve a number of input splits as input for the mappers of the job. The next section describes the `InputSplit` generation and will therefore also refer to and explain some other classes of the LIMES M/R package, as there are `LimesMRInputSplit`, `LimesMRCache` and `LimesMROrganizer`.

[59] With Hadoop 0.21.0, the API has been changed and split up into several parts. In order to be compatible with previous versions, Hadoop keeps and wraps classes and interfaces of previous API versions. For this reason, some classes are available in several implementations or as interfaces as well.

```
1  public InputSplit[] getSplits(JobConf job, int numSplits) throws
       IOException {
2    List<LimesMRInputSplit> splits = new ArrayList<LimesMRInputSplit
       >();
3
4    SparqlQueryModule sourceQm = new SparqlQueryModule(this.limesConf
       .getLimesConfigReader().getSourceInfo());
5    LimesMRCache sourceCache = new LimesMRCache();
6    sourceQm.fillCache(sourceCache);
7
8    SparqlQueryModule targetQm = new SparqlQueryModule(this.limesConf
       .getLimesConfigReader().getTargetInfo());
9    LimesMRCache targetCache = new LimesMRCache();
10   targetQm.fillCache(targetCache);
11
12   SimpleMetricFactory mf = new SimpleMetricFactory();
13   mf.setExpression(this.limesConf.getLimesConfigReader().
       metricExpression);
14
15   job.set(LimesMRConfiguration.CONFIG_METRIC_EXPRESSION, this.
       limesConf.getLimesConfigReader().metricExpression);
16   job.setClass(LimesMRConfiguration.CONFIG_METRIC_FACTORY_TYPE,
       SimpleMetricFactory.class, MetricFactory.class);
17   job.setFloat(LimesMRConfiguration.CONFIG_THRESHOLD_ACCEPTANCE,
18       this.limesConf.getLimesConfigReader().acceptanceThreshold);
19   job.setFloat(LimesMRConfiguration.CONFIG_THRESHOLD_VERIFICATION,
20       this.limesConf.getLimesConfigReader().verificationThreshold);
21   job.set(LimesMRConfiguration.CONFIG_RELATION_ACCEPTANCE, this.
       limesConf.getLimesConfigReader().acceptanceRelation);
22
23   SimpleMetricFactory organizerMf = new SimpleMetricFactory();
24   String var1 = this.limesConf.getLimesConfigReader().source.var.
       replaceAll("\\?", "");
25   String var2 = this.limesConf.getLimesConfigReader().target.var.
       replaceAll("\\?", "");
26   organizerMf.setExpression(mf.foldExpression(this.limesConf.
       getLimesConfigReader().metricExpression, var2, var1));
27
28   LimesMROrganizer organizer = new LimesMROrganizer(
29       mapOrganizerString(job.get(LimesMRConfiguration.
           CONFIG_ORGANIZER_CLASS)));
30   if (this.limesConf.getLimesConfigReader().exemplars < 2) {
31     organizer.computeExemplars(targetCache, organizerMf);
32   } else {
33     organizer.computeExemplars(targetCache, organizerMf, this.
         limesConf.getLimesConfigReader().exemplars);
34   }
35
36   List<Instance> sourceInstances = sourceCache.getAllInstances();
37   int splitSize = Math.round(sourceInstances.size() / (float)
       numSplits);
38   for (int i = 0; i < numSplits; i++) {
39     LimesMRCache sourceSplit = new LimesMRCache();
40     int finalSplitSize = splitSize;
```

```
41      if (i == numSplits - 1) {
42        finalSplitSize = sourceInstances.size() - (i * splitSize);
43      }
44      for (int j = 0; j < finalSplitSize; j++) {
45        sourceSplit.addInstance(sourceInstances.get((i * splitSize) +
              j));
46      }
47      splits.add(new LimesMRInputSplit(sourceSplit, organizer));
48    }
49
50    return splits.toArray(new LimesMRInputSplit[splits.size()]);
51  }
```

Listing 16: `getSplits` method of class `LimesMRInputFormat`

As can be seen in the listing, the desired number of input splits for the job is passed as a parameter of the method (`numSplits`, line 1). It is noteworthy, that this number is to be understood as a hint, as the documentation of the class `InputFormat` confirms. The developer is free in his decision about the number of splits returned by this method. Nevertheless, LIMES M/R takes this parameter into account, since it can be influenced by the number of map tasks, which is a parameter of the LIMES M/R configuration file passed to the program. When starting to calculate the input splits, the method at first creates a temporary container that collects these splits (here, a `List` which is typed with the `LimesMRInputSplit` class). Then the internal process continues with the validation and reading of the LIMES configuration file, which is executed by the standard LIMES configuration reader and wrapped by `LimesMRConfiguration`. This task is executed before querying the knowledge base endpoints and is integrated into the query expression in line 4 (`this.limesConf.getLimesConfigReader().getSourceInfo()`). A listing of the configuration class can be found in the appendix (listing 25). If this configuration is valid and all parameters are extracted, the method continues querying the source and target knowledge bases and filling the source and target cache with the retrieved instances (lines 4–10). To achieve a high degree of reuse, the query modules of LIMES are used for the retrieval of the input data. Furthermore, the cache class (`LimesMRCache`) is implemented as a wrapper for the class `MemoryCache` of LIMES in order to make that cache transferable by Hadoop. `LimesMRCache`, `LimesMROrganizer` and `LimesMRInputSplit` are explained after the analysis of the current method.

The next step in the process of LIMES M/R is the initialization of the metrics to be used for the instance comparison. This task is done after querying the endpoint (lines 12–13 and 23–26). Additionally, some parameters are passed to the job configuration (lines 15–21) to be usable in the map tasks that are executed on the worker nodes of the cluster, and thus in different JVMs. As described previously, Hadoop passes these parameters with the job configuration object to the remote machines where they can be extracted and used to configure the workers. However, the main innovation of the LIMES approach, the computation of exemplars for the target cache which speeds up the instance comparison by using the triangle inequality [NA10], is carried out by the `LimesMROrganizer` and triggered in lines 28–34. Line 28 first instantiates a new organizer and configures it with the specified organizer class by mapping the organizer-identifying string which is specified by the user to a specific organizer class[60]. Then the exemplars for the target cache are computed depending on the user-specified configuration. Calculating the exemplars can be an expensive task for huge knowledge bases but will speed up the comparisons significantly. Since the computation of the input splits is done on the master node of the cluster, this task is also executed

[60] The method `mapOrganizerString` is omitted because it does only simple string-to-class mappings.

on this node. Although a parallelization of the reorganization of the target cache seems obvious, this task cannot be parallelized at all, at least not using the MapReduce approach. This is motivated by the fact that the reorganization would require extensive synchronization between the different workers, which is not possible using MapReduce and could not be implemented efficiently using most other parallel programming models. As a consequence, the execution of the reorganization will affect the run time of LIMES M/R and will thus reduce the speed up that can be achieved.

The last task to be executed is the generation of the input splits to be used as input for the map tasks. The current implementation is tested with two different implementations of this task, one that only splits up the source cache and one that additionally splits up the target organizer (which contains the reorganized cache) into a fixed number (currently 10) of splits. The second implementation is listed in the appendix (listing 26) and is referred to as LIMES M/R - V2 during the evaluation of the implementation. The lines 36–48 of listing 16 split up the source cache into a set of splits containing a balanced number of instances and then creates a `LimesMRInputSplit` for each of this cache fragments with the whole target organizer as the value. Thereby, the number of splits matches the `numSplits` parameter that is passed to the method. In contrast to this implementation, LIMES M/R - V2 also splits the target organized into a set of currently 10 fragments based on the exemplars of the organizer and then creates 10 input splits for each fragment of the source cache containing the appropriate fragment of the target organizer as the value. By this means, the number of input splits produced by the method is 10 times higher than the `numSplits` parameter which also affects the number of map tasks in the cluster. An alternative implementation of the split generator has been made for evaluation purposes in order to refine the granularity of the input splits with the aim at reducing the overhead when transmitting the target cache to the worker nodes[61].

```
1  public class LimesMRCache extends MemoryCache implements Writable {
2
3    public void readFields(DataInput in) throws IOException {
4      int size = in.readInt();
5      this.instanceMap = new HashMap<String, Instance>(size);
6
7      for (int i = 0; i < size; i++) {
8        int length = in.readInt();
9        char[] buffer = new char[length];
10
11       for (int j = 0; j < buffer.length; j++) {
12         buffer[j] = in.readChar();
13       }
14
15       String s = new String(buffer);
16       Instance instance = IOHelper.parseInstance(s);
17       this.instanceMap.put(instance.getUri(), instance);
18     }
19   }
20
21   public void write(DataOutput out) throws IOException {
22     out.writeInt(this.instanceMap.size());
23
```

[61] Actually, the overhead for transmitting the cached instances is not reduced but using smaller splits, the workers can start comparing the instances much earlier and the load balancing capabilities of the cluster can be applied in a more efficient way.

```
24      for (Instance i : this.instanceMap.values()) {
25        String s = IOHelper.getSerialization(i);
26        out.writeInt(s.length());
27        out.writeChars(s);
28      }
29    }
30  }
```

Listing 17: Class `LimesMRCache`

Listing 17 contains the most relevant part of the class `LimesMRCache`, the methods that are used to serialize and deserialize a cache for the transmission to other nodes of the cluster, `write` and `readFields`. The cache extends the class `MemoryCache` of LIMES to utilize its features and implements the `Writable` interface of Hadoop to be transferable to the nodes of the cluster. When transferring objects between nodes of the cluster, Hadoop calls the `write` method of these objects at their source JVM to retrieve a serialized form of this object before creating an empty object of this type at the target JVM and calling the `readFields` method to restore the state of the object. In the case of the `LimesMRCache`, the `write` method first writes an integer to the `DataOutput` which indicates the size of the cache (the number of instances in the cache), followed by the instances of the cache. The method thereby writes the size of the string representation for each instance and its string representation as a character array. Although there are methods for directly writing UTF-8 encoded strings to the output these methods are not applicable for the problem to solve, since there are many data sources that contain instance properties that use non-UTF characters, such as Chinese names. Using the character-based transmission method requires a bit more effort when reading the instances but ensures a failure-free transmission of the input data. When restoring the state of the cache at the target JVM, the `readFields` method first determines the number of instances of the cache by reading the first integer value and creates a new container for the instances to be read. Then the method reads the instances as long as the end of the cache is not reached. An instance of the cache is read by fetching the size of its character array and then filling a buffer with characters read from the `DataInput`. Afterwards, the character array is converted to a string, parsed as an `Instance` and added to the cache.

Listing 18 contains a part of the source code of the organizer class which is also transferable. To ensure this, the organizer follows the same principle but writes and reads a more complex structure. At first, the number of exemplars is written to the output. Then, the method writes an exemplar-instance, as described for the cache before the number of instances in the subspace of this exemplar is serialized, followed by all the instances of the subspace. This procedure is done as long as there are exemplars left in the organizer. Reading this structure is equivalent to the `readFiels` method of the cache but has been adapted to the structure written by the organizer (see listing 18). As mentioned earlier, the `LimesMROrganizer` is configurable with the class of the organizer of the LIMES framework that should be used to carry out the reorganization and comparisons. For this purpose, the class `LimesMROrganizer` does not extend a specific organizer implementation but holds the organizer as a variable and implements the `Organizer` interface to be compatible with the LIMES framework. Since the reorganization task is executed on the master node before the object is transferred to the worker nodes, the internal data structure of the organizer is established so that Hadoop is able to create an untyped organizer object at the worker nodes and restore exactly the same state as on the master node. For this reason, the computation of the map task can rely on the correct internal state of the organizer even if the object has been initialized using a standard implementation of the organizer interface, the data structures are not changed afterwards.

```
1  public class LimesMROrganizer implements Organizer, Writable {
2    private Organizer organizer;
3
4    public <T extends Organizer> LimesMROrganizer(T arg0) {
5      this.organizer = arg0;
6    }
7
8    public LimesMROrganizer() {
9      this.organizer = new LimesOrganizer();
10   }
11
12   public void readFields(DataInput in) throws IOException {
13     int numExemplars = in.readInt();
14     HashMap<Instance, TreeSet<Instance>> map = new HashMap<Instance
         , TreeSet<Instance>>(numExemplars);
15     for (int i = 0; i < numExemplars; i++) {
16       int length = in.readInt();
17       char[] buffer = new char[length];
18       for (int j = 0; j < buffer.length; j++) {
19         buffer[j] = in.readChar();
20       }
21       String s = new String(buffer);
22       Instance exemplar = IOHelper.parseInstance(s);
23
24       int subSetSize = in.readInt();
25       TreeSet<Instance> subSet = new TreeSet<Instance>();
26       for (int j = 0; j < subSetSize; j++) {
27         length = in.readInt();
28         buffer = new char[length];
29         for (int k = 0; k < buffer.length; k++) {
30           buffer[k] = in.readChar();
31         }
32         s = new String(buffer);
33         Instance instance = IOHelper.parseInstance(s);
34         subSet.add(instance);
35       }
36       map.put(exemplar, subSet);
37     }
38     this.organizer.setExampleMap(map);
39   }
40
41   public void write(DataOutput out) throws IOException {
42     out.writeInt(this.organizer.getExampleMap().keySet().size());
43     for (Instance exemplar : this.organizer.getExampleMap().keySet
         ()) {
44       String s = IOHelper.getSerialization(exemplar);
45       out.writeInt(s.length());
46       out.writeChars(s);
47
48       TreeSet<Instance> subSet = this.organizer.getExampleMap().get
           (exemplar);
49       out.writeInt(subSet.size());
```

```
50        for (Iterator<Instance> i = subSet.iterator(); i.hasNext();)
              {
51          s = IOHelper.getSerialization(i.next());
52          out.writeInt(s.length());
53          out.writeChars(s);
54        }
55      }
56    }
57 }
```

Listing 18: Class `LimesMROrganizer`

To complete the contemplations before investigating the mapper class that runs on the worker nodes, this paragraph has a short look at the class `LimesMRInputSplit` and just mentions the class `LimesMRRecordReader` which passes the input key/value pairs to the map tasks. The input split class (listing 19), consists of a fragment of the source cache and the whole target organizer, or a fragment of it, depending on the implementation. These are held as variables. Furthermore, a split must be writable by Hadoop in order to pass it to the worker nodes. For this reason, the class implements the interface `InputSplit` and, when called redirects its `write` and `readFields` method calls to the appropriate methods of the cache and organizer. Besides some other methods, the class provides two getter-methods for the cache and the organizer to enable the record reader (`LimesMRRecordReader`) to create input key/value pairs from a split. The record reader of the current implementation creates only one key/value pair per split and is therefore not further considered.

```
1  public class LimesMRInputSplit implements InputSplit {
2    private LimesMRCache sourceCache;
3    private LimesMROrganizer targetOrganizer;
4
5    public void readFields(DataInput in) throws IOException {
6      this.sourceCache = new LimesMRCache();
7      this.sourceCache.readFields(in);
8      this.targetOrganizer = new LimesMROrganizer();
9      this.targetOrganizer.readFields(in);
10   }
11
12   public void write(DataOutput out) throws IOException {
13     this.sourceCache.write(out);
14     this.targetOrganizer.write(out);
15   }
16
17   public LimesMRCache getSourceCache() {
18     return this.sourceCache;
19   }
20
21   public LimesMROrganizer getTargetOrganizer() {
22     return this.targetOrganizer;
23   }
24 }
```

Listing 19: Class `LimesMRInputSplit`

The conclusion of this chapter shall be the investigation of the mapper class of LIMES M/R, `LimesMRMapper`, which is listed below followed by a short explanation of the invocation of LIMES M/R and the retrieval of the comparison results. The class `LimesMRMapper` implements the comparisons of the knowledge bases, that are cached by the program, and is executed on the worker nodes of the cluster. As listing 20 shows, the mapper class extends the class `MapReduceBase` which offers default implementations for some interfaces of the Hadoop framework, and implements the `Mapper` interface typed with the input and output classes, `LimesMRCache`, `LimesMROrganizer` and `Text`. The class implements the `configure` method coming from the `JobConfigurable` interface where the mapper class is configured with the parameters that were passed previously using the job configuration. There, the class retrieves parameters such as the comparison thresholds or the `MetricFactory` as a prerequisite for the instance comparisons. However, the core of this class is formed by the `map` method where the instance comparisons are carried out. When Hadoop calls the `map` method for an input key/value pair (a `LimesMRCache`/`LimesMROrganizer` combination), the method retrieves an iterator of the source cache and iterates over all instances of this cache. For each instance of the cache, a set of instances from the target cache is computed by the target organizer (line 35) whose similarities to the source instance lie above the verification threshold specified by the user. Afterwards, the method iterates over the result set and, depending on the similarity value of both instances, passes the instance pair as an accepted of to-be-reviewed RDF-triple to the `OutputCollector`. The so produced intermediate key/value pairs (key="a"|"v" for accepted or to be verified, value=RDF-triple) are then written to the local file system of the map worker and in the following read by the reduce task using RMI calls. Since the reducer of LIMES M/R just passes the incoming intermediate key/value pairs with the goal to write them to a single output file, the listing of this class is omitted.

```
1  public class LimesMRMapper extends MapReduceBase implements Mapper<
       LimesMRCache, LimesMROrganizer, Text, Text> {
2    private MetricFactory mf;
3    private String acceptanceRelation;
4    private float acceptanceThreshold;
5    private float verificationThreshold;
6    private JobConf job;
7
8    public void configure(JobConf job) {
9      super.configure(job);
10     this.job = job;
11     String metricExpression = job.get(LimesMRConfiguration.
         CONFIG_METRIC_EXPRESSION);
12
13     try {
14       this.mf = job.getClass(LimesMRConfiguration.
           CONFIG_METRIC_FACTORY_TYPE, SimpleMetricFactory.class,
           MetricFactory.class).newInstance();
15       this.mf.setExpression(metricExpression);
16     } catch (InstantiationException e) {
17       e.printStackTrace();
18     } catch (IllegalAccessException e) {
19       e.printStackTrace();
20     }
21
22     this.acceptanceRelation = job.get(LimesMRConfiguration.
```

```
        CONFIG_RELATION_ACCEPTANCE);
23    this.acceptanceThreshold = job.getFloat(LimesMRConfiguration.
        CONFIG_THRESHOLD_ACCEPTANCE, 0.9f);
24    this.verificationThreshold = job.getFloat(LimesMRConfiguration.
        CONFIG_THRESHOLD_VERIFICATION, 0.75f);
25  }
26
27  public void map(LimesMRCache key, LimesMROrganizer value,
      OutputCollector<Text, Text> output, Reporter reporter) throws
      IOException {
28    List<String> sourceUris = key.getAllUris();
29    HashMap<String, Float> results;
30    Iterator<String> resultIterator;
31    String s;
32
33    for (int i = 0; i < sourceUris.size(); i++) {
34      reporter.progress();
35      results = value.getSimilarInstances(key.getInstance(
          sourceUris.get(i)), this.verificationThreshold, this.mf);
36      resultIterator = results.keySet().iterator();
37
38      while (resultIterator.hasNext()) {
39        s = resultIterator.next();
40        if (results.get(s) >= this.acceptanceThreshold) {
41          output.collect(new Text("a"), new Text(getStatement(
              sourceUris.get(i), this.acceptanceRelation, s)));
42        } else if (results.get(s) >= this.verificationThreshold) {
43          output.collect(new Text("v"), new Text(getStatement(
              sourceUris.get(i), this.acceptanceRelation, s)));
44        }
45      }
46    }
47  }
48
49  private String getStatement(String subject, String predicate,
      String object) {
50    return "<" + subject + "> " + predicate + " <" + object + "> ."
        ;
51  }
52 }
```

Listing 20: The Mapper class of LIMES M/R `LimesMRMapper`

To speed up a comparison of huge knowledge bases using the MapReduce-based implementation of LIMES, as it has been described in this thesis, one simply has to create a MapReduce configuration file that conforms to the DTD of listing 15. Listing 21 contains a minimum example for a configuration that specifies 16 map tasks to be used for the instance comparisons. This configuration sets *"Limes M/R"* as the job name which can then be used for identifying the job in the web-based job tracker interface. Furthermore, the `JOB` tag contains two child tags, `MAPPERS` and `OUTPUT` that define the number of map tasks (here 16) and the path for the output file of the job (a path in the distributed file system, HDFS).

Assuming the existence of a LIMES configuration file with the name `config.xml`, the name of the MapReduce configuration file to be `mr_16.xml` and `$hadoop_home` being the installation directory of Hadoop, the link discovery process can be scheduled using the following command:

```
$hadoop_home/bin/hadoop jar Limes_MR.jar config.xml mr_16.xml Limes
```

The last parameter specifies the organizer class to be used during the process and can be any String matching the names of the organizer classes contained in the package `de.uni_leipzig.simba.organizer`. Omitting this parameter indicates the usage of the default organizer (`LimesOrganizer`) as described in [NA10]. Since the generated output file lies at the specified output path on the distributed file system, the following command can be used to copy this file to the local file system. `$hadoop_home/out` is assumed to be the desired local output directory.

```
$hadoop_home/bin/hadoop fs -copyToLocal output/limes $hadoop_home/out
```

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE JOB SYSTEM "limes_MR.dtd">
3 <JOB NAME="Limes M/R">
4   <MAPPER>
5     <NUMBER>16</NUMBER>
6   </MAPPER>
7   <OUTPUT>
8     <PATH>output/limes</PATH>
9   </OUTPUT>
10 </JOB>
```

Listing 21: Example MapReduce configuration file for 16 map tasks

4 Evaluation

4.1 Experimental setup

After the MapReduce-based implementation of LIMES has been introduced and described in the last chapter, the purpose of this chapter now is to evaluate this implementation against the original LInk discovery framework for MEtric Spaces and the MapReduce implementation of Silk, one of the other state-of-the-art link discovery frameworks, which has been introduced in chapter 2. In order to not simply state a set of test results and explain their meaning with respect to speedup, the following questions form the basis of this evaluation to give some practical guidelines for the usage of LIMES and its MapReduce-based version, LIMES M/R.

Q_1 : *Are there any restriction in the usage of LIMES M/R?* This question targets the functionality of LIMES M/R. The evaluation of the approach will find out whether there are any link discovery tasks that are processable using LIMES or other link discovery frameworks to which LIMES M/R cannot or should not be applied.

Q_2 : *How fast is LIMES M/R?* That is the most important question for end users as well as scientific contemplations. The primary answer to this question will be given as the comparison of the runtimes of LIMES and LIMES M/R as well as the speedup compared to LIMES.

Q_3 : *How does LIMES M/R compare to other, similar frameworks?* In order to evaluate the suitability of the approach, the implementation should not only be compared to LIMES but also to other MapReduce-based or otherwise parallelized link discovery approaches. This question will be answered by using the MapReduce implementation of Silk as a benchmark during the tests.

Q_4 : *Are there any recommendations for the usage of LIMES M/R?* The core message of this question is whether there are any hints that may assist users in figuring out the right link discovery approach and when decided for LIMES M/R, whether there are practical experiences for determining the right configuration of the framework.

To answer these questions, several test cases have been run using different setups. The following sections describe the technical setup within which the test cases were carried out as well as the test configurations themselves.

Hardware: The test cluster was launched using the Amazon Elastic Compute Cloud (EC2) Web Service[62] and consisted of 9 computers[63] (nodes). One of these computers was exclusively used as the master node of the cluster and the other 8 nodes were designated as worker nodes of the cluster (the slaves). Since the master node was not intended to perform large computations, this node was launched as a standard small EC2 instance (*m1.small*), while the slave nodes were launched as high-CPU medium EC2 instances (*c1.medium*) in order to provide enough computation resources.

A small instance provides $1.7GB$ of memory and has 1 EC2 Compute Unit[64] which is provided by 1 virtual core with 1 EC2 Compute Unit. The high-CPU medium instances had $1.7GB$ of memory, too, but in contrast to the master node, the slaves could use up to 5 EC2 Compute Units which were provided by 2 virtual cores with 2.5 EC2 Compute Units each. Both instance types are built on a 32-bit platform architecture and provided a moderate I/O performance[65].

Operating System and Software: The cluster nodes were launched with the 32-bit version of the Ubuntu 10.04 LTS (Lucid Lynx) Server Edition operating system which is available as the Amazon Machine Image *ami-4a34013e*[66]. Furthermore, the MapReduce cluster was set up with Apache Hadoop in version 0.20.2 as it had been released in January 2010. The cluster was configured to run at most 2 map tasks per node in parallel which results in a total number of 16 parallel map tasks for the whole cluster. This value has been chosen due to the hardware restrictions of the cluster nodes. To avoid memory problems carrying out larger link discovery tasks, the JVMs of the child nodes were started with a maximum of $1024MB$ of memory. This setting was made in the MapReduce configuration file of Hadoop.

The test cases were run using the following versions of the link discovery frameworks. LIMES was used in its current version 0.3.2 as it can be downloaded on the project website[67]. All of the link discovery tasks were carried out using the standard LIMES organizer instead of one of the further optimized organizers. As a representative for other state-of-the-art link discovery frameworks, the MapReduce implementation of Silk[68] was used in version 2.2. The versions of LIMES M/R match the description of section 3.3.2 and both versions (LIMES M/R and LIMES M/R - V2) were utilized to carry out the tests.

The following subsections contemplate the test cases and the test configurations. The test suite consists of a total of 6 test cases. Each of these tasks could occur as real-world problems, either as real link discovery problems or as duplicate detection tasks. Each of the tests was executed 5 times by each tool and, in case of LIMES M/R, with each configuration. This allowed the calculation of an average of the runtimes which was needed to complete the task. Each test case has furthermore been executed on a single Amazon EC2 c1.medium instance using LIMES. The measure of Silk MapReduce, LIMES M/R and LIMES M/R - V2 was done on the Amazon cluster as described above. The LIMES M/R and LIMES M/R - V2 tests have thereby been carried out using 6 different

[62] http://aws.amazon.com/ec2/

[63] Amazon EC2 provides compute capacity on demand. Using this cloud-based service, it is possible to create virtual computer instances matching the personal requirements.

[64] Amazon has developed several measures for the CPU capacity of the instances. The term *EC2 Compute Unit* has been introduced in order to make the EC2 instance types comparable. An EC2 Compute Unit provides the equivalent CPU capacity of a $1.0 - 1.2GHz$ AMD Opteron or Intel Xeon Processor of 2007.

[65] http://aws.amazon.com/ec2/instance-types/

[66] http://aws.amazon.com/amis/

[67] http://aksw.org/Projects/LIMES

[68] http://www4.wiwiss.fu-berlin.de/bizer/silk/mapreduce/

configurations which vary in the number of mappers for the task. For retrieving meaningful and representative results, the numbers of the map tasks had been chosen to be powers of two, up to 64 mappers per task. Thus, the tests were carried out using 2, 4, 8, 16, 32 and 64 map tasks. In the case of LIMES M/R - V2, these numbers are multiplied by 10 due to the additional split of the target organizer, as described in section 3.3.2. The subsequent sections describe each of the link discovery tasks and the intention behind these tests. All LIMES and Silk configuration files can be found in the appendix of this thesis, in order to be able to reproduce the results of these tests. In the case of the very similar MapReduce configurations for LIMES M/R, which only differ in the number of map tasks, the appendix only contains one configuration file (for 2 map tasks) which can be customized for further setups.

Test case 1: Organisms

The goal of the organisms test case was to establish links between "Species" of DBpedia[69] and "organisms" of STITCH[70], a dataset containing chemical-protein interactions. After querying the SPARQL-endpoint of the source knowledge base (DBpedia)[71], the source cache constisted of about 146 thousand instances of the class `dbpedia-o:Species`, where the prefix `dbpedia-o:` maps to the URI `http://dbpedia.org/ontology/`. Due to the restrictions of the SPARQL-endpoint of the STITCH knowledge base[72], it was only possible to retrieve exactly 100 of the 373 instances of the class `stitch:organisms`, the prefix `stitch:` maps to the URI `http://www4.wiwiss.fu-berlin.de/stitch/resource/stitch/`. Summarized: $|S| \approx 146\,000$, $|T| = 100$, what results in $|S \times T| \approx 14\,600\,000$ instance pairs.

This test case has been chosen for the following reasons. First, this task simulates a standard link discovery task which occurs every day when publishing linked data. If LIMES M/R is applicable to this problem, a first part of the proof of the correctness of LIMES M/R is done. Furthermore, this task represents smaller link discovery tasks which are already processed very efficiently using LIMES. The intention behind this test is to prove that applying the parallelized version of LIMES to a link discovery task is only worth above certain problem sizes, since the organizational overhead of the cluster negatively affects the speedup.

Test case 2: Diseases

This test case consists of the task of discovering links between the MeSH dataset[73] of the Bio2RDF project[74] and the LinkedCT knowledge base[75], containing data about clinical trials. The goal is the establishment of links between similar diseases of both datasets. The SPARQL-endpoint of the source knowledge base has been queried for instances of the type `meshr:Concept`, where `meshr:` maps to `http://bio2rdf.org/ns/mesh#`. The target cache consists of instances of the class `linkedct:condition`, where `linkedct:` maps to

[69] http://dbpedia.org/About
[70] http://stitch.embl.de/
[71] http://dbpedia.org/sparql
[72] http://www4.wiwiss.fu-berlin.de/stitch/sparql
[73] http://mesh.bio2rdf.org/sparql
[74] http://bio2rdf.org/
[75] http://linkedct.org/

`http://data.linkedct.org/resource/linkedct/`. The task shows up the following properties: $|S| \approx 23\,600$, $|T| = 5000$ and thus $|S \times T| \approx 118\,000\,000$ instance pairs.

There are two grounds for choosing this setup as a test case. First, with the 118 million instance pairs when building a cross product of the source and target cache, this task represents a medium real-world link discovery task. Applying LIMES M/R to problems of this size should result in a reasonable speedup. This should give a first idea of the problems to which LIMES M/R can be applied. Second, this test case is chosen to validate the LIMES M/R implementation handling non UTF-8 characters correctly. The retrieved instances sometimes contain characters that are not UTF-8 encoded. As described in section 3.3.2, all writable objects of LIMES M/R should be able to handle those characters correctly. This is an important second step to prove the correctness of LIMES M/R.

Test case 3: Similar Cities

The third test case is designated as a benchmark test case. This test can occur as a duplicate detection task in the everyday work, as mentioned in [NA10]. The test cases 4 and 5 are similar but differ in their cache sizes. The goal of this test is to find links between cities of the DBpedia knowledge base (instances of type `dppedia-o:City`). Therefore, the source and the target endpoints are equal, as well as the queries that are executed on them. The outline of the task looks like the following: $|S| \approx 12\,600$, $|T| = 12\,600$ and thus $|S \times T| \approx 159\,000\,000$.

This setup has been chosen as a test case in order to represent duplicate detection tasks. Furthermore, this test is the first of the four real benchmark tests that examine the speedup of LIMES M/R and LIMES M/R - V2. Even if the size of this problem is in the range of the diseases test case, this setup has been chosen as a separate test of a medium problem size, since in contrast to diseases, the size of the caches is balanced. This makes investigations on the influence of significantly different cache sizes unnecessary.

Test case 4: Similar Films

As mentioned before, test case 4 serves as a benchmark test as well. This setup differs from test case 3 in its problem size and represents larger link discovery (or here duplicate detection) tasks. The goal of the similar films test is to compare the films of the DBpedia knowledge base in order to find duplicate entries. For this purpose, the caches of the tools are filled with instances of the class `dbpedia-o:Film`. As in the previous example, the source endpoint equals the target endpoint which results in an equal size of the source and target cache. $|S| \approx 49\,000$, $|T| = 49\,000$ and thus for the cross product: $|S \times T| \approx 2\,400\,000\,000$ instance pairs.

The decision for this setup was made in order to have a runtime and speedup comparison for large problem sizes. The speedup of LIMES M/R should be even better than for medium link discovery problems as represented by the similar cities test case. Furthermore, LIMES M/R - V2 should be even better since the percentage of its larger organizational overhead decreases with the increasing complexity of the problem.

Test case 5: Similar Animals

Test case 5 of the test suite follows the example of the previous two test cases and represents even larger duplicate detection tasks. This test carries out the comparison of all animals of the DBpedia knowledge base. Thereby, all instances of the class `dbpedia-o:Animal` are compared in order to find duplicate entries. Again, the source and target knowledge bases are equal so that the cache size is balanced, too. In the following the setup: $|S| \approx 96\,500$, $|T| = 96\,500$ which results in a cross product of $|S \times T| \approx 9\,300\,000\,000$ possible instance pairs. Similar to test case 4, the speedup of this test should be very high, since the portion of the parallelizable computations increases with increasing problem sizes.

Test case 6: Thresholds

The LIMES link discovery approach is based on thresholds. By reorganizing the target cache and using the triangle inequality to exclude instance pairs that do not match the user-defined comparison parameters, LIMES takes advantage of the mathematical principles of the underlying metrics [NA10]. Using a high threshold for the comparisons results in a lower number of possible links and thus a lower number of comparisons. Instances with too high distances to the exemplars of the target cache are skipped during the comparison phase. Conversely, lowering the thresholds increases the runtime of LIMES. The intention of this test case is to check if there is a different behavior as for the previous tests when running the same task with different configurations.

This investigation is made by the similar cities example of test case 3. The similar Cities test case has been run using a minimum threshold of 0.95 for to-be-reviewed link triples (see listing 33). To evaluate the speedup of LIMES M/R when using different thresholds for the same configuration, the current test case consists of 6 different configurations, switching the minimum threshold between the values of 0.95, 0.9, 0.85, 0.8, 0.75 and 0.5. The latter configuration for instance produces a result set containing more than 3.65 million links. All 6 configurations have been run 5 times using LIMES and both versions of LIMES M/R. Silk has not been applied since Silk's runtime does not change for different thresholds. Furthermore, the core idea of this test case is the contemplation of the speedup trends for different configurations of the same task.

Summary of the test configurations

Table 2 summarizes all important parameters for the test cases explained before. Thereby, *SM* identifies the similarity metric property which was the same (levenshtein) for all test cases. P_S and P_T identify the properties of the source and the target instances to be compared. The last two parameters, T_A and T_R, represent the thresholds that have been used for the comparisons. T_A identifies the similarity threshold for accepted instance pairs whereas T_R represents the threshold property for to-be-reviewed instance pairs. For test case 6, all minimum thresholds are listed.

	Test case 1	Test case 2	Test case 3	Test case 4	Test case 5	Test case 6
SM	levenshtein					
P_S	rdfs:label	dc:title	rdfs:label	rdfs:label	rdfs:label	rdfs:label
P_T	rdfs:label	linkedct: condition_ name	rdfs:label	rdfs:label	rdfs:label	rdfs:label
T_A	0.95	0.98	1.0	1.0	1.0	1.0
T_R	0.9	0.75	0.95	0.95	0.95	0.5, 0.75, 0.8, 0.85, 0.9, 0.95

Table 2: Summary of the test configurations

4.2 Results

The objective of this final section is the evaluation of the results that have been obtained during the test runs. All frameworks produced exactly the same set of links for each test case which indicates a successful implementation of LIMES M/R. In the following, the results of all test cases are analyses in separate subsections. Therefore, each subsection states a table containing the individual results of the test case. The results are listed for each configuration of each framework and each run of the test case. The rows of the tables contain the runtime in seconds of the five runs in an ascending order as well as the average runtime for each configuration. Every sections furthermore contain two diagrams which visualize the runtimes and speedup-graphs representing the results. In contrast to the runtime charts, the speedup graphs are only drawn for the two LIMES M/R implementations compared to the optimum speedup curve.

The end of the evaluation section and thus the thesis is then formed by answering the questions that have been asked in the previous chapter (Experimental setup). This shall serve as a final summary of the LIMES M/R implementation and may help users in making a decision about the most appropriate link discovery framework for their needs as well as choosing the right configuration parameters when choosing LIMES M/R.

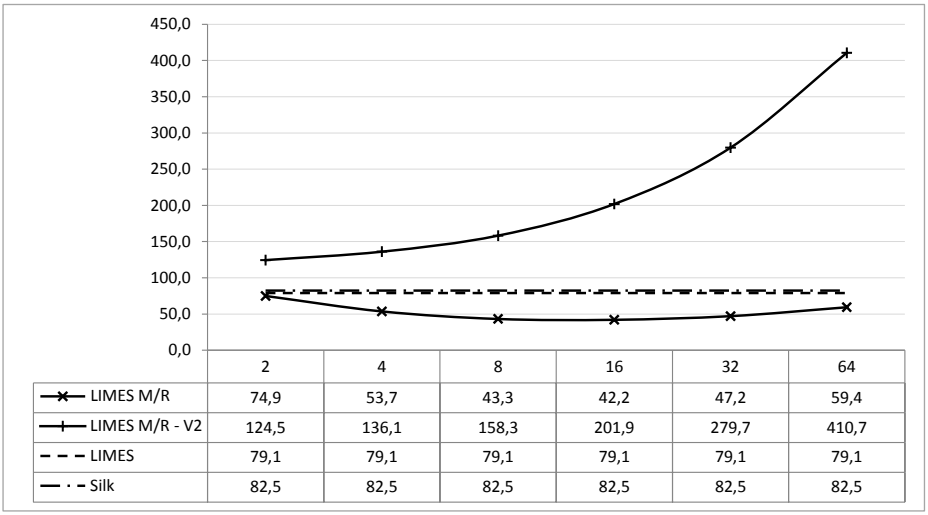
Organisms

	LIMES M/R						LIMES M/R - V2						LIMES	Silk
	2	4	8	16	32	64	20	40	80	160	320	640		
1	73.0	51.9	41.3	39.1	46.3	57.2	118.5	132.1	156.4	196.9	277.0	407.0	78.5	81.9
2	74.5	53.1	41.7	41.1	46.3	58.4	124.9	134.0	156.9	198.8	279.8	410.4	78.7	82.0
3	74.6	53.7	43.9	41.3	47.1	60.4	125.1	135.0	158.3	201.8	279.8	410.7	78.9	82.7
4	74.6	54.4	44.0	42.8	48.0	60.4	126.4	138.1	159.4	203.0	280.4	412.2	79.3	82.9
5	77.9	55.4	45.4	46.6	48.1	60.5	127.4	141.3	160.3	209.0	281.7	413.0	79.9	83.0
∅	74.9	53.7	43.3	42.2	47.2	59.4	124.5	136.1	158.3	201.9	279.7	410.7	79.1	82.5

Table 3: Test results – Organisms

The first result that can be retrieved from the above table is that LIMES M/R is fully applicable to standard link discovery problems. Thus the first hurdle for proving the correctness of LIMES M/R is taken, which will also be confirmed by the next test cases.

To start with the evaluation of the runtimes, the results show an average runtime of 79.1 seconds for LIMES when executed on an Amazon EC2 c1.medium instance. The reference framework Silk required in its parallelized version a total runtime of 82.5 seconds when splitting the input data into several hundred map tasks from which 16 where run in parallel at a time. This, once more, shows the runtime-optimized comparison approach of LIMES. Looking at the results of LIMES M/R and LIMES M/R - V2 and considering the small size of the problem, it becomes clear that the organizational overhead for the input split generation, their transfer to the map tasks, as well as the distribution of LIMES M/R over the cluster and the task initialization on each of the cluster nodes does not pay off very well. With the minimum of 42.2 seconds for 16 map tasks, that can all run in parallel, LIMES M/R still requires about 53% of the runtime that was measured for LIMES which results in a speedup of approximately 1.87 where the optimum speedup would be 16. Due to the technical setup, it is not surprising that the runtime minimum is reached when running 16 mappers in parallel. After this point the runtime goes slightly up while the speedup decreases since only 16 map tasks could be run in parallel. The remaining map tasks were blocked and scheduled as soon as any map task slots became free.



	2	4	8	16	32	64
LIMES M/R	74,9	53,7	43,3	42,2	47,2	59,4
LIMES M/R - V2	124,5	136,1	158,3	201,9	279,7	410,7
LIMES	79,1	79,1	79,1	79,1	79,1	79,1
Silk	82,5	82,5	82,5	82,5	82,5	82,5

Figure 10: Test results – Organisms

As shown in the chart of figure 10 and listed in the results table, the results of LIMES M/R - V2 are worse than the ones of LIMES. The reason for this lies in the higher overhead which results from the additional organizer splitting. In this context, it is noteworthy that although the results of LIMES M/R - V2 are given for 2, 4, 8, 26, 32 and 64 map tasks, due to the organizer splitting, these results have originally been obtained for a 10 times higher number of map tasks. The intention of reducing the network load by splitting the target organizer did not result in a faster setup of the map tasks. A further and last note in this section concerns the comparability of the obtained results. As explained in the setup,

the LIMES test cases have been carried out on an Amazon EC2 c1.medium instance with a total of 5 EC2 Compute Units. However, the LIMES M/R, LIMES M/R - V2 and Silk test cases have been executed on the cluster with a master node working on an Amazon EC2 m1.small instance with only 1 EC2 Compute Unit. In the case of Silk, this does not affect the measure because the computations of Silk MapReduce are carried out on the worker nodes. However, LIMES M/R needs to execute the reorganization of the target cache on the master node which is much slower on the m1.small instance. A difference of factor 2.7–2.85 has been measured for the reorganization task which negatively affects the overall runtime and thus the speedup. For this test case, the measure of the reorganization resulted in less than 1 second and is thus negligible. For the evaluations of the further test cases, this loss in speedup has not been included in the contemplations (except for test case 5) but the differences will be stated in order to get a better comparison.

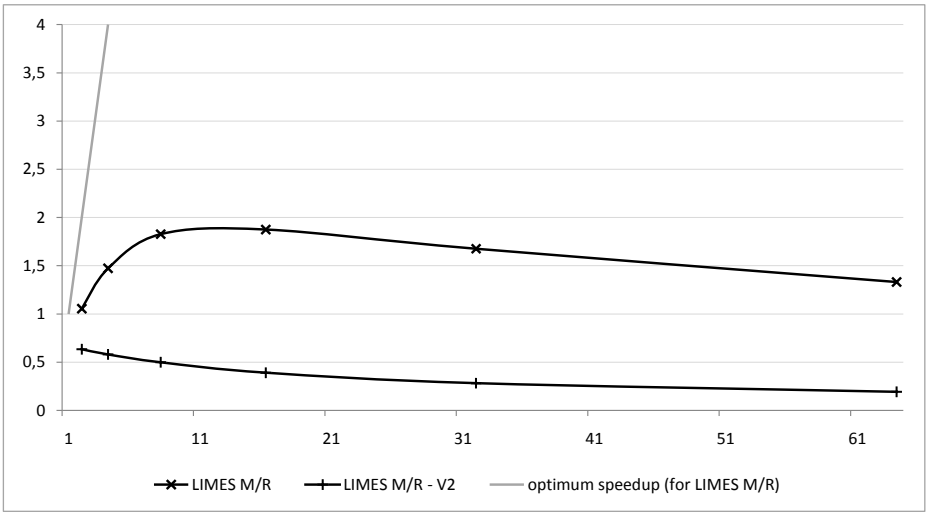


Figure 11: Speedup graph – Organisms

Diseases

The primary goal of this test case was to prove the correct handling of non UTF-8 characters by LIMES M/R. This is necessary since the transfer mechanisms for string data, as provided by Hadoop, can only handle UTF-8 encoded strings. As explained in the implementation chapter, LIMES M/R is dependent on these transfer mechanisms in order to distribute the generated input splits. The result table now confirms that all LIMES M/R runs completed and, as stated in the introduction of this evaluation, with exactly the same output as LIMES. With this result, the second hurdle for proving the correctness of LIMES M/R was overcome.

Looking at the results of this test case confirms the assumption that was made during the description of the test case. With the increase of the problem size, LIMES M/R is able to utilize the higher computing power provided by the cluster and reduces the required

	LIMES M/R						LIMES M/R - V2						LIMES	Silk
	2	4	8	16	32	64	20	40	80	160	320	640		
1	1 228.5	649.6	349.8	215.9	212.6	247.2	317.6	217.3	215.3	239.8	292.2	401.7	2 325.9	330.1
2	1 234.0	652.4	356.2	216.7	218.8	248.7	328.9	221.5	215.4	240.3	293.3	407.3	2 327.5	334.9
3	1 234.3	654.3	358.6	218.5	221.3	249.1	331.8	229.1	216.5	241.0	295.6	419.1	2 328.8	337.9
4	1 240.3	699.5	359.0	220.4	224.9	251.0	361.2	231.0	218.1	241.0	298.3	419.1	2 352.8	338.3
5	1 243.6	732.0	362.8	221.8	231.4	254.3	366.7	234.3	222.4	241.6	301.0	424.1	2 388.4	339.0
∅	1 236.1	677.6	357.3	218.6	221.8	250.1	341.2	226.6	217.5	240.7	296.1	414.3	2 344.7	336.0

Table 4: Test results – Diseases

runtime significantly. Due to the fact that the diseases test case is just a representative for link discovery tasks of medium size, it is obvious and can be confirmed by the result graphs that the relative speedup decreases with an increasing number of processors. As described previously, the speedup curve starts declining at the number of 16 map tasks, with respect to the technical setup. Furthermore, it is apparent that the speedup curve of LIMES M/R - V2 rises much faster than the one of LIMES M/R but it has to be considered that LIMES M/R - V2 creates 20 map tasks where LIMES creates only 2 tasks. Reaching a speedup of 6.9 at the first measuring point therefore has to be seen in relation to the number of map tasks (20). Taking this into account results in a relative speedup of 0.345 for LIMES M/R and 0.95 for LIMES (reaching a speedup of 1.9 for two map tasks).

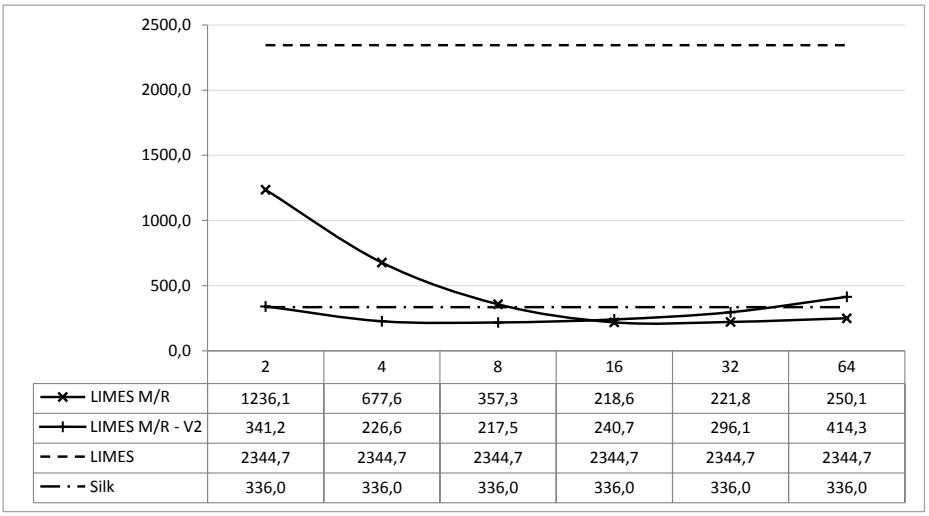


Figure 12: Test results – Diseases

Considering Silk MapReduce as a benchmark, it can be said that this parallel version of Silk is much faster than the single-core version of LIMES but LIMES M/R can even be faster than Silk when using the right configuration. At the point of 16 parallel running map tasks, the LIMES M/R graph reaches its lowest point and surpasses Silk by approximately 35%. The reason for Silk becoming so fast lies in the data preparation that is done when fetching the data. With the distribution of the input data over the nodes of the cluster, Silk is able to take advantage of the locality of the input data and thus reduce its runtime significantly. This process is not included in the measurement of the runtimes.

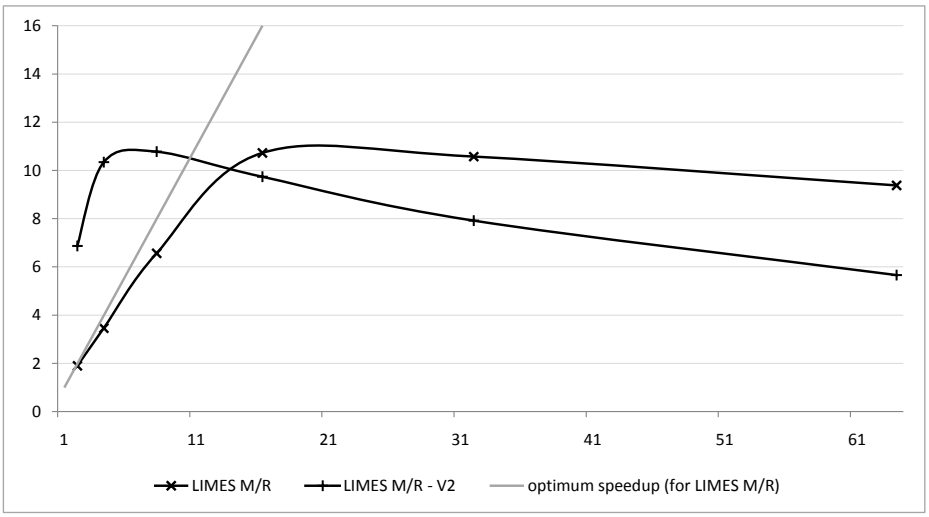


Figure 13: Speedup graph – Diseases

Similar Cities

With similar cities, there is another test case representing medium link discovery problems. Similar Cities is also the first of the four benchmark test cases that are used to visualize the speedup of LIMES M/R for different problem sizes and configurations. When analyzing the test results, the significantly different runtimes of LIMES for this and the diseases test case are striking. Although this test case contains approximately 26% more instances in the cross product of the caches (159 million to 118 million), LIMES completes much faster for the current test case. This phenomenom can be justified by the structure of the problems. At first, there are the differing sizes of the caches. Since LIMES reorganizes the target cache, it has to process the larger source cache for the instance comparisons (diseases). The similar cities test case has caches of equal size and thus, the reorganization of the target cache has greater impact on the number of comparisons. A second and the most important reason for the larger runtime is the minimum threshold of the LIMES configuration. The diseases test case applies a minimum threshold of 0.75 for instance pairs that shall be reviewed while similar cities applies a minimum threshold of 0.95 for to-be-reviewed instance pairs. As described in [NA10], lowering the thresholds results in a larger number of comparisons using the LIEMES approach. However, Silks results are, as expected, nearly similar to the diseases test case, since the number of comparisons does not depend on the thresholds or similar parameters.

The results of LIMES M/R now begin to prove that the parallelization of LIMES was successfully, when looking at the speedup graph and the total runtime in comparison to, not just LIMES, but also other tools like Silk MapReduce. During this and the last test case, the speedup curve of LIMES M/R has come closer to the optimum speedup curve but needs to be more stable for the larger number of map tasks.

With respect to the graphs, it can be resumed that the curves begin to stabilize with increasing problem sizes. Even LIMES M/R - V2, with its larger overhead is able to produce better and better results and constantly stays below the original LIMES implementation. Assuming this to be trend, the next two test cases should draw a similar picture with even better results.

	LIMES M/R						LIMES M/R - V2						LIMES	Silk
	2	**4**	**8**	**16**	**32**	**64**	**20**	**40**	**80**	**160**	**320**	**640**		
1	469.7	269.4	165.4	130.4	149.3	199.0	424.1	254.9	166.0	191.2	268.9	433.8	842.0	344.4
2	473.9	270.9	165.8	130.8	150.0	202.1	431.1	255.8	169.7	192.6	272.4	441.8	847.1	344.5
3	475.1	270.9	166.2	130.8	151.7	202.6	431.5	256.1	170.3	193.5	274.5	444.2	848.3	345.4
4	484.9	274.5	170.1	136.6	154.6	203.1	432.1	271.5	173.8	193.8	275.8	447.5	848.7	347.8
5	490.3	276.3	170.2	136.9	156.5	205.7	437.0	275.5	176.1	194.5	276.9	450.3	849.6	350.7
∅	478.8	272.4	167.5	133.1	152.4	202.5	431.2	262.7	171.2	193.1	273.7	443.6	847.1	346.5

Table 5: Test results – Similar Cities

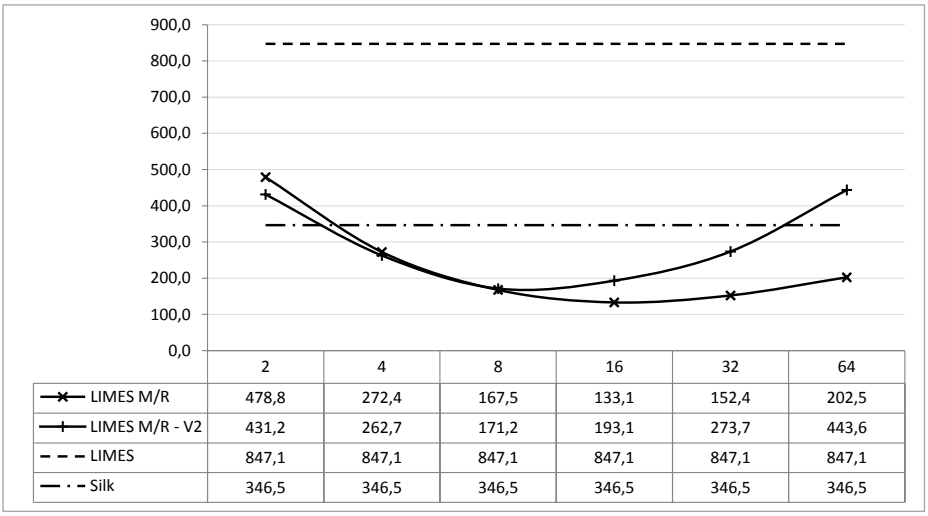


Figure 14: Test results – Similar Cities

Similar Films

	LIMES M/R						LIMES M/R - V2						LIMES	Silk
	2	**4**	**8**	**16**	**32**	**64**	**20**	**40**	**80**	**160**	**320**	**640**		
1	5 184.2	2 830.6	1 702.7	1 180.7	1 268.2	1 685.8	2 927.5	1 769.4	1 212.0	1 252.2	1 414.4	2 075.0	9 783.3	5 591.4
2	5 301.5	2 849.4	1 707.7	1 181.8	1 280.8	1 703.4	2 945.4	1 799.8	1 232.4	1 253.7	1 421.4	2 119.8	9 787.1	5 602.4
3	5 347.3	2 920.1	1 708.4	1 197.4	1 287.2	1 708.4	2 983.2	1 843.1	1 257.1	1 254.0	1 425.8	2 156.8	9 823.1	5 605.7
4	5 348.3	2 926.6	1 709.9	1 198.7	1 291.6	1 716.3	2 997.8	1 844.6	1 257.2	1 261.0	1 430.9	2 169.3	9 825.0	5 608.6
5	5 374.2	3 311.4	1 715.9	1 264.2	1 300.8	1 735.5	3 411.8	1 860.4	1 469.1	1 266.8	1 433.1	2 183.9	9 855.5	5 627.3
∅	5 311.1	2 967.6	1 708.9	1 204.5	1 285.7	1 709.9	3 053.1	1 823.4	1 285.6	1 257.5	1 425.1	2 141.0	9 814.8	5 607.1

Table 6: Test results – Similar Films

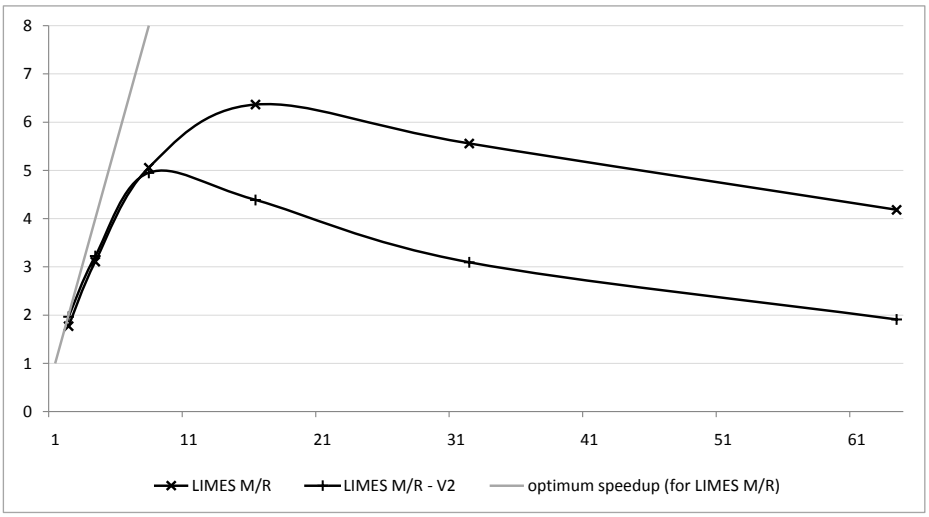
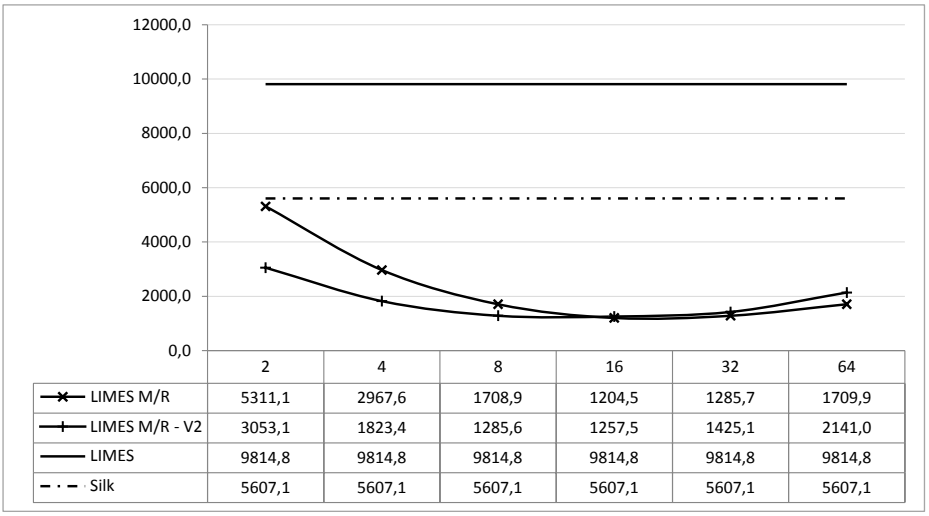


Figure 15: Speedup graph – Similar Cities

Similar Films, with its cross product size of 2.4 billion possible instance pairs, represents larger link discovery problems. As expected, LIMES M/R is able to achieve better results for this increased problem size. At the lowest point of the graph, LIMES M/R requires only 12% of the runtime of LIMES (when using 16 mappers) and furthermore, the speedup curve is approaching the optimum speedup curve more and more with increasing problem sizes.

A further improvement of LIMES M/R is achieved for the second half of the graphs. At first, the runtime graphs (LIMES M/R as well as LIMES M/R - V2) seem to be more stable than for smaller problems. This can be seen when looking at the graphs starting at 16 mappers. There, the runtime increase, which is caused by the organizational overhead, is much smaller than in the previous test cases. This is also confirmed by the speedup graphs, where the decrease of LIMES M/R - V2 comes later and less rapidly. This is justified by the fact that the larger overhead slightly looses its impact.

The last point to be mentioned here are the different runtimes for the reorganization of the target cache. As described in the evaluation of the first test case, large differences can be the result of the two different node types on which these tasks are run. As stated, the master node of the test cluster ran on an Amazon EC2 m1.small instance while the LIMES tests have been executed on an Amazon EC2 c1.medium node. This results in a difference of factor 2.7–2.85, as measured during the tests. While this difference had not enough impact for the previous test cases (the differences have not exceeded 25 seconds), the performance of the similar films test case had been influenced by the smaller master node. While the reorganization of the target cache took an average of 186 seconds on the c1.medium instance, an average of 526 seconds had been measured on the m1.small instance which makes up a difference of 340 seconds or 3.5% of the overall runtime of LIMES. Reducing the results by this amount would yield a new lowest point of the graph of 864.5 seconds as well as a higher speedup.



	2	4	8	16	32	64
LIMES M/R	5311,1	2967,6	1708,9	1204,5	1285,7	1709,9
LIMES M/R - V2	3053,1	1823,4	1285,6	1257,5	1425,1	2141,0
LIMES	9814,8	9814,8	9814,8	9814,8	9814,8	9814,8
Silk	5607,1	5607,1	5607,1	5607,1	5607,1	5607,1

Figure 16: Test results – Similar Films

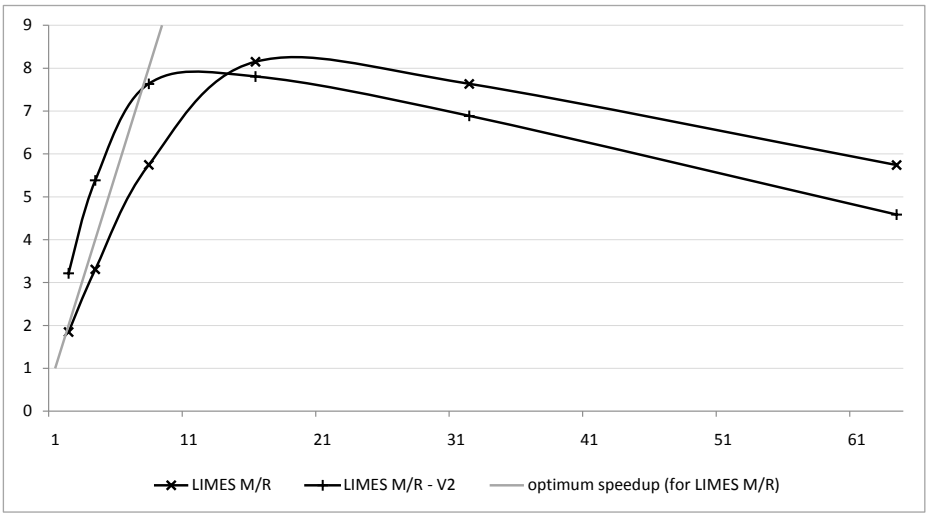


Figure 17: Speedup graph – Similar Films

Similar Animals

	LIMES M/R						LIMES M/R - V2						LIMES	Silk
	2	4	8	16	32	64	20	40	80	160	320	640		
1	4 725.3	3 116.7	2 297.5	1 982.7	2 188.4	2 573.6	2 576.4	2 289.9	2 396.3	2 552.5	2 633.5	2 843.9	7 551.0	16 305.9
2	4 758.3	3 153.3	2 306.2	1 987.3	2 219.8	2 586.7	2 518.4	2 291.5	2 426.2	2 552.8	2 643.7	2 865.9	7 611.8	16 395.4
3	4 827.4	3 168.7	2 308.2	2 033.4	2 243.3	2 612.7	2 426.5	2 310.1	2 367.1	2 526.7	2 648.6	2 865.9	7 628.7	16 401.6
4	4 887.6	3 169.6	2 310.3	2 037.0	2 265.8	2 623.8	2 464.8	2 293.3	2 413.1	2 612.4	2 682.2	2 868.6	7 645.9	16 408.1
5	4 899.1	3 177.1	2 314.1	2 038.8	2 284.9	2 635.0	2 409.7	2 329.2	2 375.1	2 590.8	2 768.9	2 886.4	7 654.1	16 416.8
∅	4 819.6	3 157.1	2 307.3	2 015.8	2 240.5	2 606.4	2 479.2	2 302.8	2 395.6	2 567.0	2 675.4	2 863.7	7 618.3	16 385.6

Table 7: Test results – Similar Animals

The purpose of this test case is to benchmark LIMES M/R for large link discovery tasks. Although the problem seems to be suitable with its 9.3 billion instance pairs, at the first appearance, the results of this test case draw another picture. At first, the significant optimizations of the LIMES approach against other frameworks become more and more obvious, the larger the problem size is. While the already parallelized version of Silk used nearly 10 thousand map tasks and still required an average of 16 385 seconds, LIMES took just 7 618 seconds. Given these conditions, it seems hard to achieve a speedup near the optimum speedup curve. The results and speedup graphs confirm this circumstance. The runtime improvement and thus the speedup is much lower than for the similar films test case which gives the impression that LIMES M/R does not scale well with the problem size. The maximum speedup of LIMES M/R is for instance reached for 16 map tasks with a value of 3.8 where the optimum would be 16. Although it is clear that the optimum speedup is not reachable, due to the amount of organizational overhead as well as the portion of the not-parallelizable (sequential) computations, LIMES M/R reached just a quarter of the optimum speedup for 16 map tasks.

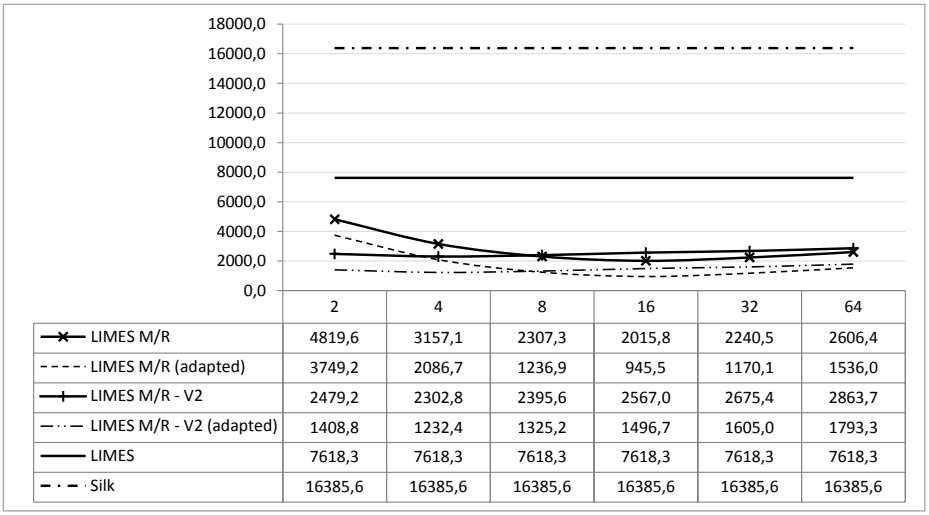


Figure 18: Test results – Similar Animals

The problem with this measure and thus the reason for the two additional curves per diagram is the neglect of the higher computation time for the reorganization task of LIMES. As described previously, the reorganization is executed on the approximately 2.7–2.85 times slower master node. With increasing problem size, the reorganization takes more comparisons and thus runtime. While the impact of this difference was comparatively small for the first tests, with the similar films test case, it became clear that the slower reorganization has to be taken into account for larger problems. During the current test case, it has been figured out that LIMES required an average total of 614 seconds for the reorganization while LIMES M/R required 1684 seconds. Contemplating the test results, the difference of 1070 seconds makes up 53% of the total runtime of the fastest configuration (16 map tasks). For this reason, the adapted runtime and speedup graphs are drawn for the current test case.

By considering and compensating the inequal reorganization task, these graphs draw another picture of the improvements of LIMES M/R. Especially when looking at the speedup curves, an enormous correction of the results can be seen. LIMES M/R now achieves a highest speedup of 8.1 when using 16 map tasks which matches the results of the similar films test case. Furthermore, the new speedup curve is steeper and closer to the optimum speedup curve for LIMES M/R. Although there is a collapse of the speedup graph for more than 16 map tasks, it can be assumed that the speedup can be further increased when using larger clusters, since the overhead for the program and data distribution looses impact.

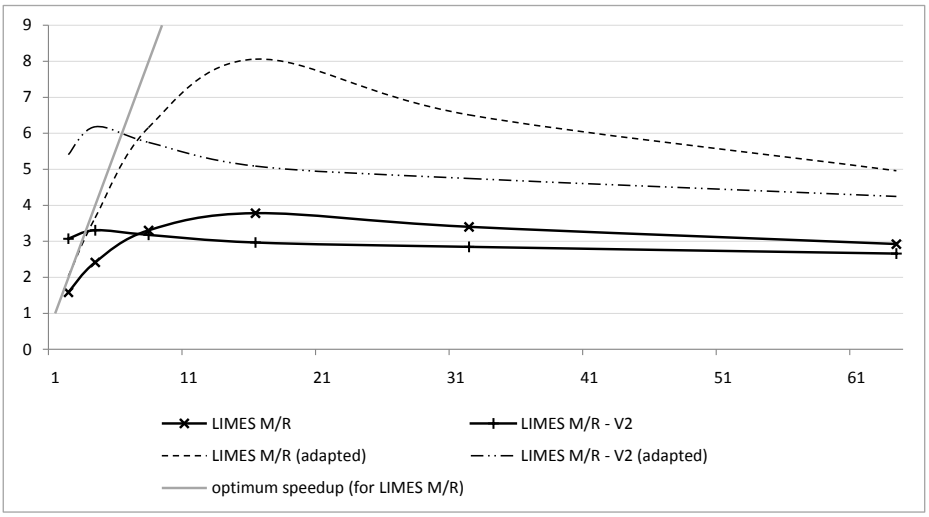


Figure 19: Speedup graph – Similar Animals

Thresholds

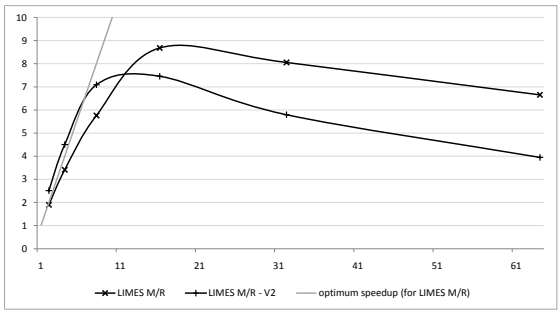
The results of this test case are depicted in table 8. In contrast to all previous test cases, this table contains only the average runtimes for each configuration and each minimum threshold. This ensures the clarity of the results table. Furthermore, this test case does not graphically visualize the runtimes but all 6 speedup graphs.

	LIMES M/R						LIMES M/R - V2						LIMES
	2	**4**	**8**	**16**	**32**	**64**	**20**	**40**	**80**	**160**	**320**	**640**	
0.5	1 103.4	613.0	363.3	241.0	259.9	314.6	833.5	464.7	295.2	280.7	361.2	530.2	2 093.2
0.75	987.6	548.0	306.2	201.8	224.7	271.3	684.4	402.6	252.8	243.4	320.3	554.7	1 875.4
0.8	924.2	497.3	283.7	192.9	213.1	261.5	675.8	388.9	247.0	237.5	316.0	475.5	1 711.7
0.85	808.0	430.3	259.3	171.2	195.0	142.1	639.1	362.7	225.4	224.1	302.8	463.7	1 464.0
0.9	630.9	359.8	210.1	154.1	173.6	224.8	539.0	321.3	209.6	208.9	287.5	464.1	1 177.7
0.95	478.8	272.4	167.5	133.1	152.4	202.5	431.2	262.7	171.2	193.1	273.7	443.6	847.1

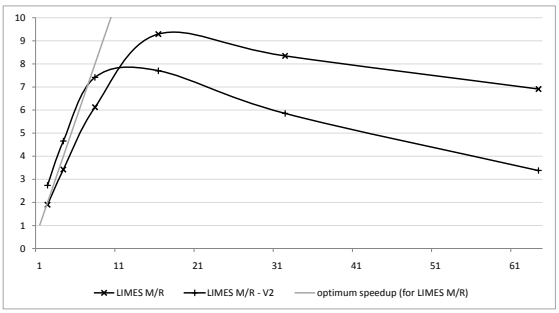
Table 8: Test results – Thresholds

As table 8 shows, the runtime of LIMES decreases with increasing minimum thresholds. While LIMES required an average of nearly 2 100 seconds for the task when using a minimum threshold of 0.5, the runtime for a threshold of 0.95 is decreased to about 850 seconds. This observation matches the assumption that was made in the description of this test case. Another observation is that both versions of LIMES M/R show the typical behavior, which has been figured out during the tests, with regard to their runtimes. The runtimes are decreasing rapidly until a slight slow down is recognized when reaching the number of 16 map tasks. Applying more than 16 map tasks then results in a small increase of the average runtimes of both LIMES M/R versions. This flattening of the runtime graphs as well as their small increase when applying more map tasks than processors in the cluster can be justified by the increasing overhead of the job and the need of the master node to block some map tasks until some computing resources become free.

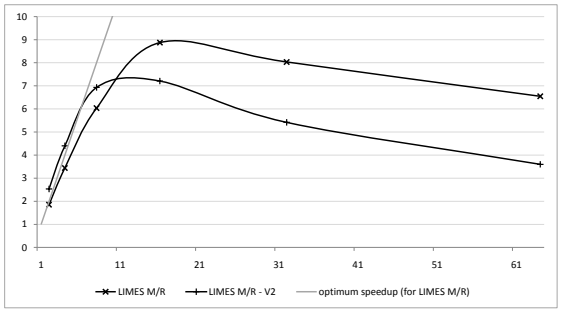
The speedup graphs of figure 20 draw a clear picture. The speedup values of a specific configuration decrease with an increasing minimum threshold, the curves become more flat and drift away from the optimum speedup curve. The reason for this behavior lies in the change of the portion of computations for the instance comparison subtask when changing the thresholds. As described in the introduction of this test, the number of comparisons increases with the decrease of the minimum threshold. This also results in a lower impact of the communication overhead for the link discovery task when applying lower thresholds, since each map task carries out more comparisons for the same input split. The primary conclusion that can be drawn from these speedup graphs is that LIMES M/R is not only applicable to large link discovery tasks. The advantages of the parallel computation can even be used for medium and small tasks when using configurations with small thresholds.



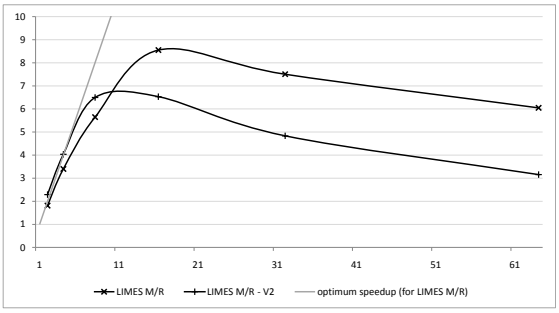
(a) Threshold = 0.5



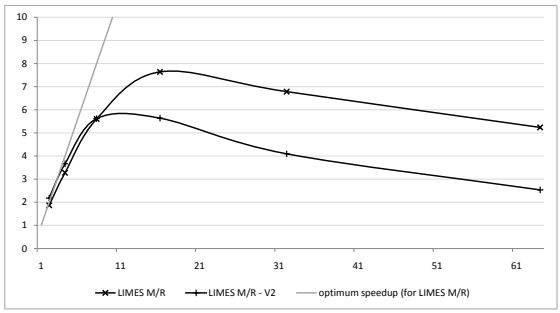
(b) Threshold = 0.75



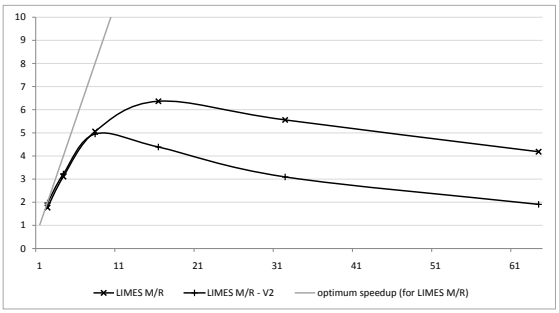
(c) Threshold = 0.8



(d) Threshold = 0.85



(e) Threshold = 0.9



(f) Threshold = 0.95

Figure 20: Speedup graphs – Thresholds

Questions Answering

Q_1 : *Are there any restriction in the usage of LIMES M/R?*

A_1 : No, there are no known restrictions for the usage of LIMES M/R. This means that LIMES M/R is fully applicable for all standard link discovery and duplicate detection tasks. Furthermore, LIMES M/R produces exactly the same results as LIMES does.

Q_2 : *How fast is LIMES M/R?*

A_2 : As the evaluation of the test cases shows, LIMES M/R can be very fast and retrieve speedups near the optimum. But the speed of LIMES M/R mainly depends on the size of the problem. While LIMES is even faster than its parallelized version when applying to really small tasks, LIMES M/R is able to decrease the overall runtime to approximately 10% of the runtime of LIMES for larger problems[76]. Besides the scale of the link discovery task, the configuration of LIMES, especially the minimum threshold, affects the runtime and thus the reachable speedup, as proved in test case 6.

Q_3 : *How does LIMES M/R compare to other, similar frameworks?*

A_3 : It is shown that LIMES M/R is able to surpass Silks runtime for each test case. But since LIMES becomes more effective for increasing problem sizes, there are only small differences for smaller tasks. In contrast, for large tasks, as represented by the similar animals test case, LIMES M/R can easily achieve runtimes of 5.8% of Silks runtime.

Q_4 : *Are there any recommendations for the usage of LIMES M/R?*

A_4 : Regarding the problems to which LIMES M/R should be applied, the answer can be given based on the previous answers: The larger the problem and the smaller the minimum threshold, the more time can be saved using LIMES M/R. Thus, LIMES M/R should be applied to medium and large tasks or even smaller tasks when using small threshold values. Another recommendation concerns the configuration of LIMES M/R. It has turned out that the optimal number of map tasks matches the number of map tasks that can be run in parallel on the cluster.

[76] It is also possible to achieve even smaller runtimes and thus higher speedups when applying LIMES M/R to really huge problems on larger clusters.

5 Conclusion and Outlook

This thesis investigated the parallelization of the LInk discovery framework for MEtric Spaces (LIMES) and proposed a first implementation. Since the Linked Data paradigm is relatively new, the introduction of the thesis gave a short summary of the idea of the Semantic Web and Linked Data. After that, the second chapter introduced current and state-of-the-art technologies for link discovery and application parallelization. Therefore, this chapter introduced different frameworks for the discovery of links between different knowledge bases and contemplated especially LIMES and Silk more in detail. Section 2.2 then gave an introduction to state-of-the-art technologies and methods for the parallelization of applications. This section contemplated three different and well-known parallel computing approaches (cluster computing, grid computing and cloud computing) and investigated four important parallel programming models (OpenMP, MPI, UPC and MapReduce). The latter were then evaluated based on different criteria which for example comprise tasks like the worker management, synchronization or the programming methodologies. This evaluation was then the basis for the decision about how to parallelize LIMES which was described in chapter 3.

There, some important prerequisites, like the investigation of LIMES' current architecture, the decision for a programming model and the contemplation of the MapReduce paradigm, were created in order to achieve adequate results for the parallelization task which was then described in section 3.3. This section comprised the description of the adapted internal process, the design principles of LIMES M/R and its current implementation. Then, the last chapter made an evaluation of the new framework by analyzing the results of a test suite that had been executed on a test cluster running on a set of virtual computers in the Amazon Elastic Compute Cloud. Based on these results, it can be summarized that the parallelization of LIMES was successful, even if further optimizations regarding the achievable speedup are possible. However, it is noteworthy that since LIMES already is a time-optimized link discovery framework, and with respect to the architecture and the reorganization task of LIMES, the absolute speedup will always differ from the optimal speedup.

Further investigations should comprise the lowering of the data distribution overhead in order to achieve higher speedups. This could for instance be done by trying to partition the input data and putting them directly into the distributed file system in order to use the locality of the data for the map tasks. Furthermore, the target organizer could be distributed to the file system using the `DistributedCache` of Hadoop. This could also enable the map tasks taking advantage of the locality and could reduce the higher overhead of splitting this organizer. A last possible optimization to be stated here is the change of the record reader usage. This could be possible when distributing the organizer to the file system. Then, the granularity of the map tasks could be refined in order to take advantage of the load balancing mechanisms provided by MapReduce.

Bibliography

[ABC+06] Krste Asanovic, Ras Bodik, Bryan Christopher Catanzaro, Joseph James Gebis, Parry Husbands, Kurt Keutzer, David A. Patterson, William Lester Plishker, John Shalf, Samuel Webb Williams, and Katherine A. Yelick. The Landscape of Parallel Computing Research: A View from Berkeley. Technical Report UCB/EECS-2006-183, EECS Department, University of California, Berkeley, Dec 2006.

[ADL+09] Sören Auer, Sebastian Dietzold, Jens Lehmann, Sebastian Hellmann, and David Aumüller. Triplify – Light-Weight Linked Data Publication from Relational Databases. In *18th International World Wide Web Conference*, pages 621–630, April 2009.

[Akl97] Selim G. Akl. *Parallel computation: models and methods*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1997.

[And07] Paul Anderson. What is Web 2.0: Ideas, technologies and implications for education. Technical report, JISC, 2007.

[BC11] Chriz Bizer and Richard Cyganiak. D2R Server – Publishing Relational Databases on the Semantic Web. http://www4.wiwiss.fu-berlin.de/bizer/d2r-server/, January 2011.

[BHBL09] Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked Data - The Story So Far. *International Journal on Semantic Web and Information Systems (IJSWIS)*, 2009.

[BL07] Tim Berners-Lee. Linked Data – Design Issues. http://www.w3.org/DesignIssues/LinkedData.html, July 2007.

[BL11] Tim Berners-Lee. The Original HTTP as defined in 1991. http://www.w3.org/Protocols/HTTP/AsImplemented.html, February 2011.

[Bla10] Barney Blaise. Introduction to Parallel Computing. https://computing.llnl.gov/tutorials/parallel_comp/, October 2010.

[BLFM05] T. Berners-Lee, , R. Fielding, and L. Masinter. RFC 3986: Uniform Resource Identifier (URI): Generic Syntax. http://www.ietf.org/rfc/rfc3986.txt, January 2005.

[BLHL01] Tim Berners-Lee, James Hendler, and Ora Lassila. The Semantic Web. http://www.scientificamerican.com/article.cfm?id=the-semantic-web, May 2001.

[BM05] Heiko Bauke and Stephan Mertens. *Cluster Computing: Praktische Einführung in das Hochleistungsrechnen auf Linux-Clustern (german)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[Boo02] Charles Bookman. *Linux Clustering: Building and Maintaining Linux Clusters*. New Riders Publishing, Thousand Oaks, CA, USA, 2002.

[CB10] Liang T. Chen and Deepankar Bairagi. Developing Parallel Programs – A Discussion of Popular Models. Technical report, Oracle Corporation, September 2010.

[CHM08] Peter Coetzee, Tom Heath, and Enrico Motta. SparqPlug: Generating Linked Data from Legacy HTML, SPARQL and the DOM. In *Proceedings of the WWW2008 Workshop on Linked Data on the Web*, Beijing, China, April 2008.

[Dan08] Krissi Danielson. Distinguishing Cloud Computing from Utility Computing. http://www.ebizq.net/blogs/saasweek/2008/03/distinguishing_cloud_computing/, March 2008.

[DG04] Jeffrey Dean and Sanjay Ghemawat. MapReduce: simplified data processing on large clusters. In *Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, pages 10–10, Berkeley, CA, USA, 2004. USENIX Association.

[EB09] Ali Ebnenasir and Rasoul Beik. Developing parallel programs: A design-oriented perspective. In *Proceedings of the 2009 ICSE Workshop on Multicore Software Engineering*, IWMSE '09, pages 1–8, Washington, DC, USA, 2009. IEEE Computer Society.

[FK03] Ian Foster and Carl Kesselman. *The Grid 2: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

[Fou11] Apache Software Foundation. Hadoop MapReduce 0.21.0. http://hadoop.apache.org/mapreduce/docs/r0.21.0/, January 2011.

[GMS+09] Hugh Glaser, Ian C. Millard, Won-Kyung Sung, Seungwoo Lee, Pyung Kim, and Beom-Jong You. Research on Linked Data and Co-reference Resolution. In *International Conference on Dublin Core and Metadata Applications*, pages 113–117, October 2009.

[Gro11a] W3C (RDF Working Group). Resource Description Framework (RDF) . http://www.w3.org/2001/sw/wiki/RDF, February 2011.

[Gro11b] W3C (SPARQL Working Group). SPARQL Query Language for RDF. http://www.w3.org/2001/sw/wiki/SPARQL, February 2011.

[HB11] Tom Heath and Christian Bizer. *Linked Data: Evolving the Web into a Global Data Space*. Synthesis Lectures on the Semantic Web: Theory and Technology. Morgan & Claypool, 1st edition, 2011.

[HKL+09] Oktie Hassanzadeh, Anastasios Kementsietsidis, Lipyeow Lim, Renée J. Miller, and Min Wang. A Framework for Semantic Link Discovery over Relational Data. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management, CIKM 2009, Hong Kong, China, November 2-6, 2009*, pages 1027–1036, 2009.

[HLKW09] Oktie Hassanzadeh, Lipyeow Lim, Anastasios Kementsietsidis, and Min

Wang. A Declarative Framework for Semantic Link Discovery over Relational Data. In *18th International World Wide Web Conference (WWW2009)*, April 2009.

[HMX11] Oktie Hassanzadeh, Renée J. Miller, and Reynold Xin. LinQuer: Linkage Query Writer. http://dblab.cs.toronto.edu/project/linquer/index.html#description, January 2011.

[HP07] John L. Hennessy and David A. Patterson. *Computer Architecture - A Quantitative Approach*. Morgan Kaufmann, fourth edition, 2007.

[HPPSH05] Ian Horrocks, Bijan Parsia, Peter Patel-Schneider, and James Hendler. Semantic Web Architecture: Stack or Two Towers? In *PPSWR*, pages 37–41. 2005.

[HXM+09] Oktie Hassanzadeh, Reynold Xin, Renée J. Miller, Anastasios Kementsietsidis, Lipyeow Lim, and Min Wang. Linkage Query Writer. *PVLDB*, 2(2):1590–1593, 2009.

[IJBV10] Robert Isele, Anja Jentzsch, Chris Bizer, and Julius Volz. Silk – A Link Discovery Framework for the Web of Data. http://www4.wiwiss.fu-berlin.de/bizer/silk/, November 2010.

[KMZS08] Henry Kasim, Verdi March, Rita Zhang, and Simon See. Survey on Parallel Programming Model. In Jian Cao, Minglu Li, Min-You Wu, and Jinjun Chen, editors, *Network and Parallel Computing*, volume 5245 of *Lecture Notes in Computer Science*, pages 266–275. Springer Berlin / Heidelberg, 2008.

[Mar96a] Robert C. Martin. The Dependency Inversion Principle. http://www.objectmentor.com/resources/articles/dip.pdf, May 1996.

[Mar96b] Robert C. Martin. The Liskov Substitution Principle. http://www.objectmentor.com/resources/articles/lsp.pdf, March 1996.

[Mar00a] Robert C. Martin. Design Principles and Design Patterns. http://www.objectmentor.com/resources/articles/Principles_and_Patterns.pdf, 2000.

[Mar00b] Robert C. Martin. The Open Closed Principle, 2000.

[MG09] Peter Mell and Tim Grance. The NIST Definition of Cloud Computing. Technical report, National Institute for Standards and Technology (NIST), July 2009.

[MMT95] B. M. Maggs, L. R. Matheson, and R. E. Tarjan. Models of parallel computation: a survey and synthesis. In *Proceedings of the 28th Hawaii International Conference on System Sciences*, pages 61–71, Washington, DC, USA, 1995. IEEE Computer Society.

[NA10] Axel-Cyrille Ngonga Ngomo and Sören Auer. LIMES – A Time-Efficient Approach for Large-Scale Link Discovery on the Web of Data. In draft, December 2010.

[NEA10] Axel-Cyrille Ngonga Ngomo, Timofey Ermilov, and Sören Auer. LIMES – LInk discovery framework for MEtric Spaces. http://aksw.org/Projects/LIMES, November 2010.

[Ngo10] Axel-Cyrille Ngonga Ngomo. LIMES User Manual, Version 0.3.2. http://aksw.org/Projects/LIMES/files?get=limes_manual.pdf, October 2010.

[RSS08] Yves Raimond, Christopher Sutton, and Mark Sandler. Automatic Interlinking of Music Datasets on the Semantic Web. *1st Workshop about Linked Data on the Web*, 2008.

[SLZ09] François Scharffe, Yanbin Liu, and Chuguang Zhou. RDF-AI: an Architecture for RDF Datasets Matching, Fusion and Interlink. In *Proc. IJCAI 2009 workshop on Identity, reference, and knowledge representation (IR-KR), Pasadena (CA US)*, 2009.

[Sof11] OpenLink Software. Virtuoso RDF. http://www.openlinksw.com/dataspace/dav/wiki/Main/VOSRDF, January 2011.

[Tal11] Talis. talis®Platform – data, connected. http://www.talis.com/platform/, January 2011.

[VBGK09a] Julius Volz, Christian Bizer, Martin Gaedke, and Georgi Kobilarov. Discovering and Maintaining Links on the Web of Data. In Abraham Bernstein, David R. Karger, Tom Heath, Lee Feigenbaum, Diana Maynard, Enrico Motta, and Krishnaprasad Thirunarayan, editors, *International Semantic Web Conference*, volume 5823 of *Lecture Notes in Computer Science*, pages 650–665. Springer, 2009.

[VBGK09b] Julius Volz, Christian Bizer, Martin Gaedke, and Georgi Kobilarov. Silk – A Link Discovery Framework for the Web of Data. In *18th International World Wide Web Conference*, April 2009.

[Wal09] Paul Walk. Linked, open, semantic? http://blog.paulwalk.net/2009/11/11/linked-open-semantic/, November 2009.

Appendix

```
<Silk>
  <Prefixes>
    <Prefix id="rdf" namespace="http://www.w3.org/1999/02/22-rdf-
       syntax-ns#"/>
    <Prefix id="rdfs" namespace="http://www.w3.org/2000/01/rdf-
       schema#"/>
    <Prefix id="owl" namespace="http://www.w3.org/2002/07/owl#"/>
    <Prefix id="dbpedia" namespace="http://dbpedia.org/ontology/"/>
    <Prefix id="lgdp" namespace="http://linkedgeodata.org/property/
       "/>
    <Prefix id="georss" namespace="http://www.georss.org/georss/"/>
  </Prefixes>

  <DataSources>
    <DataSource id="dbpedia" type="sparqlEndpoint">
      <Param name="endpointURI" value="http://dbpedia.org/sparql"/>
      <Param name="graph" value="http://dbpedia.org"/>
      <Param name="retryCount" value="100"/>
      <Param name="retryPause" value="1000"/>
    </DataSource>
    <DataSource id="lgdb" type="sparqlEndpoint">
      <Param name="endpointURI" value="http://linkedgeodata.org/
         sparql"/>
    </DataSource>
  </DataSources>

  <Interlinks>
    <Interlink id="cities">
      <LinkType>owl:sameAs</LinkType>
      <SourceDataset dataSource="dbpedia" var="a">
        <RestrictTo>
          ?a rdf:type dbpedia:Settlement
        </RestrictTo>
      </SourceDataset>
      <TargetDataset dataSource="lgdb" var="b">
        <RestrictTo>
          { ?b lgdp:place "city" }
          UNION
          { ?b lgdp:place "town" }
        </RestrictTo>
      </TargetDataset>
      <LinkCondition>
        <Aggregate type="average">
          <Aggregate type="max" required="true">
            <Compare metric="jaro">
```

```
42                <Input path="?a/rdfs:label[@lang='en']"/>
43                <Input path="?b/rdfs:label[@lang='en']"/>
44              </Compare>
45              <Compare metric="jaro">
46                <Input path="?a/rdfs:label[@lang='en']"/>
47                <Input path="?b/rdfs:label[@lang='']"/>
48              </Compare>
49            </Aggregate>
50            <Aggregate type="max" required="true">
51              <Compare metric="wgs84">
52                <Input path="?a/georss:point"/>
53                <Input path="?b/georss:point"/>
54                <Param name="unit" value="km"/>
55                <Param name="threshold" value="50"/>
56                <Param name="curveStyle" value="linear"/>
57                <Param name="longitudeFirst" value="false"/>
58              </Compare>
59            </Aggregate>
60          </Aggregate>
61        </LinkCondition>
62
63        <Filter threshold="0.9" limit="1"/>
64
65        <Outputs>
66          <Output minConfidence="0.9" type="file">
67            <Param name="file" value="dbpedia_lgdb.nt"/>
68            <Param name="format" value="ntriples"/>
69          </Output>
70        </Outputs>
71      </Interlink>
72    </Interlinks>
73  </Silk>
```

Listing 22: LSL example configuration file that interlinks cities of DBpedia and LinkedGeoData [IJBV10]

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!DOCTYPE LIMES SYSTEM "limes.dtd">
3  <LIMES>
4    <PREFIX>
5      <NAMESPACE>http://www.w3.org/1999/02/22-rdf-syntax-ns#</
           NAMESPACE>
6      <LABEL>rdf</LABEL>
7    </PREFIX>
8    <PREFIX>
9      <NAMESPACE>http://www.w3.org/2000/01/rdf-schema#</NAMESPACE>
10     <LABEL>rdfs</LABEL>
11   </PREFIX>
12   <PREFIX>
13     <NAMESPACE>http://www.w3.org/1999/02/22-rdf-syntax-ns#</
           NAMESPACE>
14     <LABEL>rdf</LABEL>
15   </PREFIX>
```

```
  <PREFIX>
    <NAMESPACE>http://www.w3.org/2000/01/rdf-schema#</NAMESPACE>
    <LABEL>rdfs</LABEL>
  </PREFIX>
  <PREFIX>
    <NAMESPACE>http://www.w3.org/2002/07/owl#</NAMESPACE>
    <LABEL>owl</LABEL>
  </PREFIX>
  <PREFIX>
    <NAMESPACE>http://dbpedia.org/ontology/</NAMESPACE>
    <LABEL>dbpedia</LABEL>
  </PREFIX>
  <PREFIX>
    <NAMESPACE>http://linkedgeodata.org/property/</NAMESPACE>
    <LABEL>lgdb</LABEL>
  </PREFIX>
  <PREFIX>
    <NAMESPACE>http://www.georss.org/georss/</NAMESPACE>
    <LABEL>georss</LABEL>
  </PREFIX>

  <SOURCE>
    <ID>dbpedia</ID>
    <ENDPOINT>http://dbpedia.org/sparql</ENDPOINT>
    <VAR>?a</VAR>
    <PAGESIZE>1000</PAGESIZE>
    <RESTRICTION>?a  rdf:type  dbpedia:Settlement</RESTRICTION>
    <PROPERTY>rdfs:label</PROPERTY>
  </SOURCE>

  <TARGET>
    <ID>lgdb</ID>
    <ENDPOINT>http://linkedgeodata.org/sparql</ENDPOINT>
    <VAR>?b</VAR>
    <RESTRICTION>?b  rdf:type  lgdb:city</RESTRICTION>
    <PROPERTY>rdfs:label</PROPERTY>
  </TARGET>

  <METRIC>levenshtein(a.rdfs:label,  b.rdfs:label)</METRIC>
  <EXEMPLARS>70</EXEMPLARS>
  <ACCEPTANCE>
    <THRESHOLD>0.9</THRESHOLD>
    <FILE>accepted.nt</FILE>
    <RELATION>owl:sameAs</RELATION>
  </ACCEPTANCE>
  <REVIEW>
    <THRESHOLD>0.85</THRESHOLD>
    <FILE>reviewme.nt</FILE>
    <RELATION>owl:sameAs</RELATION>
  </REVIEW>
</LIMES>
```

Listing 23: LCL example configuration file that interlinks cities of DBpedia and LinkedGeoData

```
linkspec_stmt:= CREATE LINKSPEC linkspec_name
                AS link_method opt_args opt limit;

linkindex_stmt:= CREATE LINKINDEX opt_idx_args linkindex_name
                 ON table(col)
                 USING link_method;

link_method:= native_link | link_clause_expr | UDF;

native_link:= synonym | hyponym | stringMatch;

link_clause expr:= link_clause AND link_clause_expr
                 | link_clause OR link_clause_expr
                 | link_clause;

link_clause:= LINK source WITH target
              USING link_terminal opt_limit;

link_terminal:= native_link | UDF | linkspec_name

opt_limit:= LINKLIMIT number;
```

Listing 24: Extract of the LinQL grammar specification [HXM+09]

```
package de.uni_leipzig.simba.mr;

import org.apache.hadoop.conf.Configuration;
import de.uni_leipzig.simba.io.ConfigReader;
import de.uni_leipzig.simba.metricfactory.MetricFactory;

public class LimesMRConfiguration {
  public static final String CONFIG_FILE_PATH = "limes.config.file.
     path";
  public static final String CONFIG_METRIC_EXPRESSION = "limes.
     config.metric.expression";
  public static final String CONFIG_METRIC_FACTORY_TYPE = "limes.
     config.metric.factory.type";
  public static final String CONFIG_THRESHOLD_VERIFICATION = "limes
     .config.threshold.verification";
  public static final String CONFIG_THRESHOLD_ACCEPTANCE = "limes.
     config.threshold.acceptance";
  public static final String CONFIG_RELATION_ACCEPTANCE = "limes.
     config.relation.acceptance";
  public static final String CONFIG_ORGANIZER_CLASS = "limes.config
     .organizer.class";
  private Configuration conf;
  private String configFilePath;
  private ConfigReader configReader;

  public LimesMRConfiguration(Configuration conf) {
    this.conf = conf;
    this.configFilePath = conf.get(CONFIG_FILE_PATH);
```

```
  }

  public Configuration getConfiguration() {
    return this.conf;
  }

  public ConfigReader getLimesConfigReader() {
    if (this.configReader == null) {
      this.configReader = new ConfigReader();
      this.configReader.validateAndRead(this.configFilePath);
    }
    return this.configReader;
  }
}
```

Listing 25: LimesMRConfiguration.java

```
public InputSplit[] getSplits(JobConf job, int numSplits) throws
    IOException {
  List<LimesMRInputSplit> splits = new ArrayList<LimesMRInputSplit
      >();

  SparqlQueryModule sourceQm = new SparqlQueryModule(this.limesConf
      .getLimesConfigReader().getSourceInfo());
  LimesMRCache sourceCache = new LimesMRCache();
  sourceQm.fillCache(sourceCache);

  SparqlQueryModule targetQm = new SparqlQueryModule(this.limesConf
      .getLimesConfigReader().getTargetInfo());
  LimesMRCache targetCache = new LimesMRCache();
  targetQm.fillCache(targetCache);

  SimpleMetricFactory mf = new SimpleMetricFactory();
  mf.setExpression(this.limesConf.getLimesConfigReader().
      metricExpression);

  job.set(LimesMRConfiguration.CONFIG_METRIC_EXPRESSION, this.
      limesConf.getLimesConfigReader().metricExpression);
  job.setClass(LimesMRConfiguration.CONFIG_METRIC_FACTORY_TYPE,
      SimpleMetricFactory.class, MetricFactory.class);
  job.setFloat(LimesMRConfiguration.CONFIG_THRESHOLD_ACCEPTANCE,
      this.limesConf.getLimesConfigReader().acceptanceThreshold);
  job.setFloat(LimesMRConfiguration.CONFIG_THRESHOLD_VERIFICATION,
      this.limesConf.getLimesConfigReader().verificationThreshold);
  job.set(LimesMRConfiguration.CONFIG_RELATION_ACCEPTANCE, this.
      limesConf.getLimesConfigReader().acceptanceRelation);

  SimpleMetricFactory organizerMf = new SimpleMetricFactory();
  String var1 = this.limesConf.getLimesConfigReader().source.var.
      replaceAll("\\?", "");
  String var2 = this.limesConf.getLimesConfigReader().target.var.
      replaceAll("\\?", "");
```

```
organizerMf.setExpression(mf.foldExpression(this.limesConf.
    getLimesConfigReader().metricExpression, var2, var1));

LimesMROrganizer organizer = new LimesMROrganizer(
    mapOrganizerString(job.get(LimesMRConfiguration.
        CONFIG_ORGANIZER_CLASS)));
if (this.limesConf.getLimesConfigReader().exemplars < 2) {
  organizer.computeExemplars(targetCache, organizerMf);
} else {
  organizer.computeExemplars(targetCache, organizerMf, this.
      limesConf.getLimesConfigReader().exemplars);
}

int numExemplars = organizer.getExampleMap().keySet().size();
  LimesMROrganizer[] organizerSplits = new LimesMROrganizer[Math.
      min(10, numExemplars)];
  Iterator<Instance> exemplarIterator = organizer.getExampleMap()
      .keySet().iterator();
  int organizerSplitSize = numExemplars / organizerSplits.length;

  for (int i = 0; i < organizerSplits.length; i++) {
    LimesMROrganizer o = new LimesMROrganizer(
        mapOrganizerString(job.get(LimesMRConfiguration.
            CONFIG_ORGANIZER_CLASS)));
    HashMap<Instance, TreeSet<Instance>> map = new HashMap<
        Instance, TreeSet<Instance>>();
    if (i == organizerSplits.length - 1) {
      while (exemplarIterator.hasNext()) {
        Instance exempar = exemplarIterator.next();
        map.put(exempar, organizer.getExampleMap().get(exempar));
      }
    } else {
      for (int j = 0; j < organizerSplitSize; j++) {
        Instance exempar = exemplarIterator.next();
        map.put(exempar, organizer.getExampleMap().get(exempar));
      }
    }
    o.setExampleMap(map);
    organizerSplits[i] = o;
  }

  List<Instance> sourceInstances = sourceCache.getAllInstances();
  int splitSize = Math.round(sourceInstances.size() / (float)
      numSplits);
  LimesMRCache[] cacheSplits = new LimesMRCache[numSplits];

  for (int i = 0; i < numSplits; i++) {
    LimesMRCache sourceSplit = new LimesMRCache();
    int finalSplitSize = splitSize;
    if (i == numSplits - 1) {
      finalSplitSize = sourceInstances.size() - (i * splitSize);
    }
    for (int j = 0; j < finalSplitSize; j++) {
      sourceSplit.addInstance(sourceInstances.get((i * splitSize)
```

```
            + j));
      }
      cacheSplits[i] = sourceSplit;
    }

    for (int i = 0; i < cacheSplits.length; i++) {
      for (int j = 0; j < organizerSplits.length; j++) {
        splits.add(new LimesMRInputSplit(cacheSplits[i],
            organizerSplits[j]));
      }
    }

    return splits.toArray(new LimesMRInputSplit[splits.size()]);

  return splits.toArray(new LimesMRInputSplit[splits.size()]);
}
```

Listing 26: `getSplits` method of class `LimesMRInputFormat` – alternative implementation

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE JOB SYSTEM "limes_MR.dtd">
<JOB NAME="Limes M/R">
  <MAPPER>
    <NUMBER>2</NUMBER>
  </MAPPER>
  <OUTPUT>
    <PATH>output/limes</PATH>
  </OUTPUT>
  <DEBUG>
    <LOG LEVEL="ALL">
      <OUT>
        <FILE APPEND="true">logs/LIMES_MR.log</FILE>
      </OUT>
    </LOG>
    <RUNS>5</RUNS>
  </DEBUG>
</JOB>
```

Listing 27: `mr_2.xml` – MapReduce configuration file for LIMES M/R (2 map tasks)

```
<?xml version="1.0" encoding="utf-8" ?>
<Silk>
    <Prefixes>
        <Prefix id="rdf" namespace="http://www.w3.org/1999/02/22-
            rdf-syntax-ns#"/>
        <Prefix id="rdfs" namespace="http://www.w3.org/2000/01/rdf-
            schema#"/>
        <Prefix id="owl" namespace="http://www.w3.org/2002/07/owl#"
            />
        <Prefix id="dbpedia-owl" namespace="http://dbpedia.org/
            ontology/"/>
```

```
        <Prefix id="dbpedia-prop" namespace="http://dbpedia.org/
            property/"/>
        <Prefix id="stitch" namespace="http://www4.wiwiss.fu-berlin
            .de/stitch/resource/stitch/"/>
    </Prefixes>
    <DataSources>
        <DataSource id="dbpedia" type="sparqlEndpoint">
            <Param name="endpointURI" value="http://dbpedia.org/
                sparql"/>
            <Param name="retryCount" value="100"/>
        </DataSource>
        <DataSource id="stitch" type="sparqlEndpoint">
            <Param name="endpointURI" value="http://www4.wiwiss.fu-
                berlin.de/stitch/sparql"/>
        </DataSource>
    </DataSources>
    <Interlinks>
        <Interlink id="organisms">
            <LinkType>owl:sameAs</LinkType>
            <SourceDataset dataSource="dbpedia" var="a">
                <RestrictTo>
                    ?a rdf:type dbpedia-owl:Species
                </RestrictTo>
            </SourceDataset>
            <TargetDataset dataSource="stitch" var="b">
                <RestrictTo>
                    ?b rdf:type stitch:organisms
                </RestrictTo>
            </TargetDataset>
            <LinkCondition>
                <Aggregate type="max">
                    <Compare metric="levensthein">
                        <TransformInput function="lowerCase">
                            <Input path="?a/rdfs:label[@lang = 'en
                                ']"/>
                        </TransformInput>
                        <TransformInput function="lowerCase">
                            <Input path="?b/rdfs:label" />
                        </TransformInput>
                    </Compare>
                </Aggregate>
            </LinkCondition>
            <Filter threshold="0.90" />
            <Outputs>
                <Output type="file" maxConfidence="0.95">
                  <Param name="file" value="
                      stitch_dbpedia_verify_links.xml"/>
                  <Param name="format" value="alignment"/>
                </Output>
                <Output type="file" minConfidence="0.95">
                  <Param name="file" value="
                      stitch_dbpedia_accepted_links.nt"/>
                  <Param name="format" value="ntriples"/>
                </Output>
```

```
            </Outputs>
        </Interlink>
    </Interlinks>
</Silk>
```

Listing 28: Test case 1 – Silk configuration file

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE LIMES SYSTEM "limes.dtd">
<LIMES>
  <PREFIX>
    <NAMESPACE>http://dbpedia.org/ontology/</NAMESPACE>
    <LABEL>dbpedia-o</LABEL>
  </PREFIX>
  <PREFIX>
    <NAMESPACE>http://dbpedia.org/property/</NAMESPACE>
    <LABEL>dbpedia-p</LABEL>
  </PREFIX>
    <PREFIX>
    <NAMESPACE>http://dbpedia.org/resource/</NAMESPACE>
    <LABEL>dbpedia-r</LABEL>
  </PREFIX>
  <PREFIX>
    <NAMESPACE>http://www.w3.org/1999/02/22-rdf-syntax-ns#</
       NAMESPACE>
    <LABEL>rdf</LABEL>
  </PREFIX>
  <PREFIX>
    <NAMESPACE>http://www.w3.org/2000/01/rdf-schema#</NAMESPACE>
    <LABEL>rdfs</LABEL>
  </PREFIX>
  <PREFIX>
    <NAMESPACE>http://www.w3.org/2002/07/owl#</NAMESPACE>
    <LABEL>owl</LABEL>
  </PREFIX>
  <PREFIX>
    <NAMESPACE>http://www4.wiwiss.fu-berlin.de/stitch/resource/
       stitch/</NAMESPACE>
    <LABEL>stitch</LABEL>
  </PREFIX>
  <SOURCE>
    <ID>dbpedia</ID>
    <ENDPOINT>http://dbpedia.org/sparql</ENDPOINT>
    <VAR>?x</VAR>
    <PAGESIZE>2000</PAGESIZE>
    <RESTRICTION>?x rdf:type dbpedia-o:Species</RESTRICTION>
    <PROPERTY>rdfs:label</PROPERTY>
  </SOURCE>
  <TARGET>
    <ID>stitch</ID>
    <ENDPOINT>http://www4.wiwiss.fu-berlin.de/stitch/sparql</
       ENDPOINT>
```

```
    <VAR>?y</VAR>
    <PAGESIZE>100</PAGESIZE>
    <RESTRICTION>?y rdf:type stitch:organisms</RESTRICTION>
    <PROPERTY>rdfs:label</PROPERTY>
  </TARGET>
  <METRIC>levenshtein(a.rdfs:label, b.rdfs:label)</METRIC>
  <ACCEPTANCE>
    <THRESHOLD>0.95</THRESHOLD>
    <FILE>organisms-limes-a.nt</FILE>
    <RELATION>owl:sameAs</RELATION>
  </ACCEPTANCE>
  <REVIEW>
    <THRESHOLD>0.9</THRESHOLD>
    <FILE>organisms-limes-r.nt</FILE>
    <RELATION>owl:sameAs</RELATION>
  </REVIEW>
</LIMES>
```

Listing 29: Test case 1 – LIMES configuration file

```
<?xml version="1.0" encoding="utf-8" ?>
<Silk>
  <Prefixes>
        <Prefix id="rdf" namespace="http://www.w3.org/1999/02/22-
           rdf-syntax-ns#"/>
        <Prefix id="rdfs" namespace="http://www.w3.org/2000/01/rdf-
           schema#"/>
        <Prefix id="owl" namespace="http://www.w3.org/2002/07/owl#"
           />
        <Prefix id="linkedct" namespace="http://data.linkedct.org/
           resource/linkedct/"/>
        <Prefix id="dc" namespace="http://purl.org/dc/elements/1.1/
           "/>
    <Prefix id="meshr" namespace="http://bio2rdf.org/ns/mesh#"/>
    </Prefixes>
    <DataSources>
        <DataSource id="linkedct" type="sparqlEndpoint">
            <Param name="endpointURI" value="http://lod.openlinksw.
               com/sparql"/>
        </DataSource>
    <DataSource id="mesh" type="sparqlEndpoint">
      <Param name="endpointURI" value="http://mesh.bio2rdf.org/
         sparql"/>
    </DataSource>
    </DataSources>
    <Interlinks>
        <Interlink id="disease">
            <LinkType>owl:sameAs</LinkType>
            <SourceDataset dataSource="linkedct" var="a">
                <RestrictTo>?a rdf:type linkedct:condition</
                   RestrictTo>
            </SourceDataset>
```

```
            <TargetDataset dataSource="mesh" var="b">
                <RestrictTo>?b rdf:type meshr:Concept</RestrictTo>
            </TargetDataset>
      <LinkCondition>
        <Aggregate type="min">
          <Compare metric="levenshtein">
            <Input path="?a/linkedct:condition_name"/>
            <Input path="?b/dc:title"/>
          </Compare>
        </Aggregate>
      </LinkCondition>
            <Filter threshold="0.75" limit="1"/>
      <Outputs>
                <Output maxConfidence="0.979" type="file">
          <Param name="file" value="diseases-silk-r.nt"/>
          <Param name="format" value="ntriples"/>
                </Output>
                <Output minConfidence="0.98" type="file">
          <Param name="file" value="disease-silk-a.nt"/>
          <Param name="format" value="ntriples"/>
                </Output>
            </Outputs>
        </Interlink>
    </Interlinks>
</Silk>
```

Listing 30: Test case 2 – Silk configuration file

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE LIMES SYSTEM "limes.dtd">
<LIMES>
  <PREFIX>
    <NAMESPACE>http://www.w3.org/1999/02/22-rdf-syntax-ns#</
       NAMESPACE>
    <LABEL>rdf</LABEL>
  </PREFIX>
  <PREFIX>
    <NAMESPACE>http://www.w3.org/2000/01/rdf-schema#</NAMESPACE>
    <LABEL>rdfs</LABEL>
  </PREFIX>
  <PREFIX>
    <NAMESPACE>http://www.w3.org/2002/07/owl#</NAMESPACE>
    <LABEL>owl</LABEL>
  </PREFIX>
  <PREFIX>
    <NAMESPACE>http://data.linkedct.org/resource/linkedct/</
       NAMESPACE>
    <LABEL>linkedct</LABEL>
  </PREFIX>
  <PREFIX>
    <NAMESPACE>http://purl.org/dc/elements/1.1/</NAMESPACE>
    <LABEL>dc</LABEL>
```

```
  </PREFIX>
  <PREFIX>
    <NAMESPACE>http://bio2rdf.org/ns/mesh#</NAMESPACE>
    <LABEL>meshr</LABEL>
  </PREFIX>
  <SOURCE>
    <ID>mesh</ID>
    <ENDPOINT>http://mesh.bio2rdf.org/sparql</ENDPOINT>
    <VAR>?y</VAR>
    <PAGESIZE>5000</PAGESIZE>
    <RESTRICTION>?y rdf:type meshr:Concept</RESTRICTION>
    <PROPERTY>dc:title</PROPERTY>
  </SOURCE>
  <TARGET>
    <ID>linkedct</ID>
    <ENDPOINT>http://data.linkedct.org/sparql</ENDPOINT>
    <VAR>?x</VAR>
    <PAGESIZE>-1</PAGESIZE>
    <RESTRICTION>?x rdf:type linkedct:condition</RESTRICTION>
    <PROPERTY>linkedct:condition_name</PROPERTY>
  </TARGET>
  <METRIC>levenshtein(y.dc:title, x.linkedct:condition_name)</
     METRIC>
  <ACCEPTANCE>
    <THRESHOLD>0.98</THRESHOLD>
    <FILE>diseases-limes-a.nt</FILE>
    <RELATION>owl:sameAs</RELATION>
  </ACCEPTANCE>
  <REVIEW>
    <THRESHOLD>0.75</THRESHOLD>
    <FILE>diseases-limes-r.nt</FILE>
    <RELATION>owl:sameAs</RELATION>
  </REVIEW>
</LIMES>
```

Listing 31: Test case 2 – LIMES configuration file

```
<?xml version="1.0" encoding="utf-8" ?>
<Silk>
  <Prefixes>
    <Prefix id="rdf" namespace="http://www.w3.org/1999/02/22-rdf-
       syntax-ns#"/>
    <Prefix id="rdfs" namespace="http://www.w3.org/2000/01/rdf-
       schema#"/>
    <Prefix id="owl" namespace="http://www.w3.org/2002/07/owl#"/>
    <Prefix id="dbpedia" namespace="http://dbpedia.org/ontology/"/>
  </Prefixes>
  <DataSources>
    <DataSource id="dbpedia" type="sparqlEndpoint">
      <Param name="endpointURI" value="http://dbpedia.org/sparql"/>
      <Param name="graph" value="http://dbpedia.org"/>
    </DataSource>
```

```
    <DataSource id="dbpedia" type="sparqlEndpoint">
      <Param name="endpointURI" value="http://dbpedia.org/sparql"/>
      <Param name="graph" value="http://dbpedia.org"/>
    </DataSource>
  </DataSources>
  <Interlinks>
    <Interlink id="cities">
      <LinkType>owl:sameAs</LinkType>
      <SourceDataset dataSource="dbpedia" var="a">
        <RestrictTo>?a rdf:type dbpedia:City</RestrictTo>
      </SourceDataset>
      <TargetDataset dataSource="dbpedia" var="b">
        <RestrictTo>?b rdf:type dbpedia:City</RestrictTo>
      </TargetDataset>
      <LinkCondition>
        <Aggregate type="max">
          <Compare metric="levenshtein">
            <Input path="?a/rdfs:label"/>
            <Input path="?b/rdfs:label"/>
          </Compare>
        </Aggregate>
      </LinkCondition>
      <Filter threshold="0.95" limit="1"/>
      <Outputs>
        <Output type="file" maxConfidence="0.999">
          <Param name="file" value="simCities-silk-r.nt"/>
          <Param name="format" value="ntriples"/>
        </Output>
        <Output type="file" minConfidence="1">
          <Param name="file" value="simCities-silk-a.nt"/>
          <Param name="format" value="ntriples"/>
        </Output>
      </Outputs>
    </Interlink>
  </Interlinks>
</Silk>
```

Listing 32: Test case 3 – Silk configuration file

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE LIMES SYSTEM "limes.dtd">
<LIMES>
  <PREFIX>
    <NAMESPACE>http://dbpedia.org/ontology/</NAMESPACE>
    <LABEL>dbpedia-o</LABEL>
  </PREFIX>
  <PREFIX>
    <NAMESPACE>http://dbpedia.org/property/</NAMESPACE>
    <LABEL>dbpedia-p</LABEL>
  </PREFIX>
    <PREFIX>
    <NAMESPACE>http://dbpedia.org/resource/</NAMESPACE>
```

```
    <LABEL>dbpedia-r</LABEL>
  </PREFIX>
  <PREFIX>
    <NAMESPACE>http://www.w3.org/1999/02/22-rdf-syntax-ns#</
       NAMESPACE>
    <LABEL>rdf</LABEL>
  </PREFIX>
  <PREFIX>
    <NAMESPACE>http://www.w3.org/2000/01/rdf-schema#</NAMESPACE>
    <LABEL>rdfs</LABEL>
  </PREFIX>
  <PREFIX>
    <NAMESPACE>http://www.w3.org/2002/07/owl#</NAMESPACE>
    <LABEL>owl</LABEL>
  </PREFIX>
  <SOURCE>
    <ID>dbpedia</ID>
    <ENDPOINT>http://dbpedia.org/sparql</ENDPOINT>
    <VAR>?x</VAR>
    <PAGESIZE>1000</PAGESIZE>
    <RESTRICTION>?x rdf:type dbpedia-o:City</RESTRICTION>
    <PROPERTY>rdfs:label</PROPERTY>
  </SOURCE>
  <TARGET>
    <ID>dbpedia</ID>
    <ENDPOINT>http://dbpedia.org/sparql</ENDPOINT>
    <VAR>?x</VAR>
    <PAGESIZE>1000</PAGESIZE>
    <RESTRICTION>?x rdf:type dbpedia-o:City</RESTRICTION>
    <PROPERTY>rdfs:label</PROPERTY>
  </TARGET>
  <METRIC>levenshtein(a.rdfs:label, b.rdfs:label)</METRIC>
  <ACCEPTANCE>
    <THRESHOLD>1</THRESHOLD>
    <FILE>simCities-limes-a.nt</FILE>
    <RELATION>owl:sameAs</RELATION>
  </ACCEPTANCE>
  <REVIEW>
    <THRESHOLD>0.95</THRESHOLD>
    <FILE>simCities-limes-r.nt</FILE>
    <RELATION>owl:sameAs</RELATION>
  </REVIEW>
</LIMES>
```

Listing 33: Test case 3 – LIMES configuration file

```
<?xml version="1.0" encoding="utf-8" ?>
<Silk>
  <Prefixes>
    <Prefix id="rdf" namespace="http://www.w3.org/1999/02/22-rdf-
       syntax-ns#"/>
    <Prefix id="rdfs" namespace="http://www.w3.org/2000/01/rdf-
```

```
      schema#"/>
    <Prefix id="owl" namespace="http://www.w3.org/2002/07/owl#"/>
    <Prefix id="dbpedia" namespace="http://dbpedia.org/ontology/"/>
  </Prefixes>
  <DataSources>
    <DataSource id="dbpedia" type="sparqlEndpoint">
      <Param name="endpointURI" value="http://dbpedia.org/sparql"/>
      <Param name="graph" value="http://dbpedia.org"/>
    </DataSource>
    <DataSource id="dbpedia" type="sparqlEndpoint">
      <Param name="endpointURI" value="http://dbpedia.org/sparql"/>
      <Param name="graph" value="http://dbpedia.org"/>
    </DataSource>
  </DataSources>
  <Interlinks>
    <Interlink id="cities">
      <LinkType>owl:sameAs</LinkType>
      <SourceDataset dataSource="dbpedia" var="a">
        <RestrictTo>?a rdf:type dbpedia:Film</RestrictTo>
      </SourceDataset>
      <TargetDataset dataSource="dbpedia" var="b">
        <RestrictTo>?b rdf:type dbpedia:Film</RestrictTo>
      </TargetDataset>
      <LinkCondition>
        <Aggregate type="max">
          <Compare metric="levenshtein">
            <Input path="?a/rdfs:label"/>
            <Input path="?b/rdfs:label"/>
          </Compare>
        </Aggregate>
      </LinkCondition>
      <Filter threshold="0.95" limit="1"/>
      <Outputs>
        <Output type="file" maxConfidence="0.999">
          <Param name="file" value="simFilms-silk-r.nt"/>
          <Param name="format" value="ntriples"/>
        </Output>
        <Output type="file" minConfidence="1">
          <Param name="file" value="simFilms-silk-a.nt"/>
          <Param name="format" value="ntriples"/>
        </Output>
      </Outputs>
    </Interlink>
  </Interlinks>
</Silk>
```

Listing 34: Test case 4 – Silk configuration file

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE LIMES SYSTEM "limes.dtd">
<LIMES>
  <PREFIX>
```

```
    <NAMESPACE>http://dbpedia.org/ontology/</NAMESPACE>
    <LABEL>dbpedia-o</LABEL>
  </PREFIX>
  <PREFIX>
    <NAMESPACE>http://dbpedia.org/property/</NAMESPACE>
    <LABEL>dbpedia-p</LABEL>
  </PREFIX>
    <PREFIX>
    <NAMESPACE>http://dbpedia.org/resource/</NAMESPACE>
    <LABEL>dbpedia-r</LABEL>
  </PREFIX>
  <PREFIX>
    <NAMESPACE>http://www.w3.org/1999/02/22-rdf-syntax-ns#</
       NAMESPACE>
    <LABEL>rdf</LABEL>
  </PREFIX>
  <PREFIX>
    <NAMESPACE>http://www.w3.org/2000/01/rdf-schema#</NAMESPACE>
    <LABEL>rdfs</LABEL>
  </PREFIX>
  <PREFIX>
    <NAMESPACE>http://www.w3.org/2002/07/owl#</NAMESPACE>
    <LABEL>owl</LABEL>
  </PREFIX>
  <SOURCE>
    <ID>dbpedia</ID>
    <ENDPOINT>http://dbpedia.org/sparql</ENDPOINT>
    <VAR>?x</VAR>
    <PAGESIZE>1000</PAGESIZE>
    <RESTRICTION>?x rdf:type dbpedia-o:Film</RESTRICTION>
    <PROPERTY>rdfs:label</PROPERTY>
  </SOURCE>
  <TARGET>
    <ID>dbpedia</ID>
    <ENDPOINT>http://dbpedia.org/sparql</ENDPOINT>
    <VAR>?x</VAR>
    <PAGESIZE>1000</PAGESIZE>
    <RESTRICTION>?x rdf:type dbpedia-o:Film</RESTRICTION>
    <PROPERTY>rdfs:label</PROPERTY>
  </TARGET>
  <METRIC>levenshtein(a.rdfs:label, b.rdfs:label)</METRIC>
  <ACCEPTANCE>
    <THRESHOLD>1</THRESHOLD>
    <FILE>simFilms-limes-a.nt</FILE>
    <RELATION>owl:sameAs</RELATION>
  </ACCEPTANCE>
  <REVIEW>
    <THRESHOLD>0.95</THRESHOLD>
    <FILE>simFilms-limes-r.nt</FILE>
    <RELATION>owl:sameAs</RELATION>
  </REVIEW>
</LIMES>
```

Listing 35: Test case 4 – LIMES configuration file

```
<?xml version="1.0" encoding="utf-8" ?>
<Silk>
  <Prefixes>
    <Prefix id="rdf" namespace="http://www.w3.org/1999/02/22-rdf-
       syntax-ns#"/>
    <Prefix id="rdfs" namespace="http://www.w3.org/2000/01/rdf-
       schema#"/>
    <Prefix id="owl" namespace="http://www.w3.org/2002/07/owl#"/>
    <Prefix id="dbpedia" namespace="http://dbpedia.org/ontology/"/>
  </Prefixes>
  <DataSources>
    <DataSource id="dbpedia" type="sparqlEndpoint">
      <Param name="endpointURI" value="http://dbpedia.org/sparql"/>
      <Param name="graph" value="http://dbpedia.org"/>
    </DataSource>
    <DataSource id="dbpedia" type="sparqlEndpoint">
      <Param name="endpointURI" value="http://dbpedia.org/sparql"/>
      <Param name="graph" value="http://dbpedia.org"/>
    </DataSource>
  </DataSources>
  <Interlinks>
    <Interlink id="cities">
      <LinkType>owl:sameAs</LinkType>
      <SourceDataset dataSource="dbpedia" var="a">
        <RestrictTo>?a rdf:type dbpedia:Animal</RestrictTo>
      </SourceDataset>
      <TargetDataset dataSource="dbpedia" var="b">
        <RestrictTo>?b rdf:type dbpedia:Animal</RestrictTo>
      </TargetDataset>
      <LinkCondition>
        <Aggregate type="max">
          <Compare metric="levenshtein">
            <Input path="?a/rdfs:label"/>
            <Input path="?b/rdfs:label"/>
          </Compare>
        </Aggregate>
      </LinkCondition>
      <Filter threshold="0.95" limit="1"/>
      <Outputs>
        <Output type="file" maxConfidence="0.999">
          <Param name="file" value="simAnimals-silk-r.nt"/>
          <Param name="format" value="ntriples"/>
        </Output>
        <Output type="file" minConfidence="1">
          <Param name="file" value="simAnimals-silk-a.nt"/>
          <Param name="format" value="ntriples"/>
        </Output>
      </Outputs>
    </Interlink>
  </Interlinks>
</Silk>
```

Listing 36: Test case 5 – Silk configuration file

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE LIMES SYSTEM "limes.dtd">
<LIMES>
  <PREFIX>
    <NAMESPACE>http://dbpedia.org/ontology/</NAMESPACE>
    <LABEL>dbpedia-o</LABEL>
  </PREFIX>
  <PREFIX>
    <NAMESPACE>http://dbpedia.org/property/</NAMESPACE>
    <LABEL>dbpedia-p</LABEL>
  </PREFIX>
    <PREFIX>
    <NAMESPACE>http://dbpedia.org/resource/</NAMESPACE>
    <LABEL>dbpedia-r</LABEL>
  </PREFIX>
  <PREFIX>
    <NAMESPACE>http://www.w3.org/1999/02/22-rdf-syntax-ns#</
       NAMESPACE>
    <LABEL>rdf</LABEL>
  </PREFIX>
  <PREFIX>
    <NAMESPACE>http://www.w3.org/2000/01/rdf-schema#</NAMESPACE>
    <LABEL>rdfs</LABEL>
  </PREFIX>
  <PREFIX>
    <NAMESPACE>http://www.w3.org/2002/07/owl#</NAMESPACE>
    <LABEL>owl</LABEL>
  </PREFIX>
  <SOURCE>
    <ID>dbpedia</ID>
    <ENDPOINT>http://dbpedia.org/sparql</ENDPOINT>
    <VAR>?x</VAR>
    <PAGESIZE>1000</PAGESIZE>
    <RESTRICTION>?x rdf:type dbpedia-o:Animal</RESTRICTION>
    <PROPERTY>rdfs:label</PROPERTY></SOURCE>
  <TARGET>
    <ID>dbpedia</ID>
    <ENDPOINT>http://dbpedia.org/sparql</ENDPOINT>
    <VAR>?x</VAR>
    <PAGESIZE>1000</PAGESIZE>
    <RESTRICTION>?x rdf:type dbpedia-o:Animal</RESTRICTION>
    <PROPERTY>rdfs:label</PROPERTY></TARGET>
  <METRIC>levenshtein(a.rdfs:label, b.rdfs:label)</METRIC>
  <ACCEPTANCE>
    <THRESHOLD>1</THRESHOLD>
    <FILE>simAnimals-limes-a.nt</FILE>
    <RELATION>owl:sameAs</RELATION></ACCEPTANCE>
  <REVIEW>
    <THRESHOLD>0.95</THRESHOLD>
    <FILE>simAnimals-limes-r.nt</FILE>
    <RELATION>owl:sameAs</RELATION></REVIEW>
</LIMES>
```

Listing 37: Test case 5 – LIMES configuration file

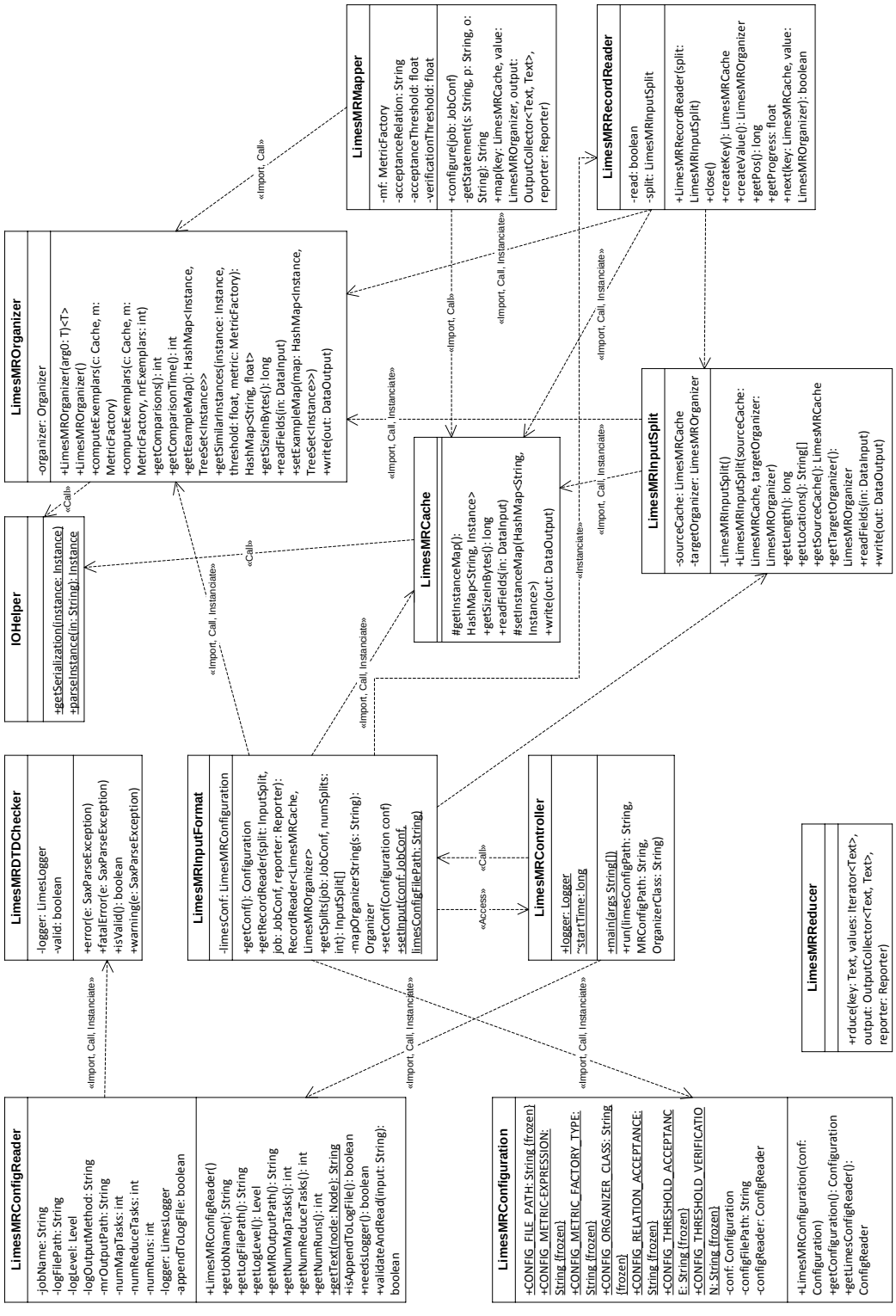


Figure 21: Full class diagram of the LIMES M/R package

Selbständigkeitserklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe, insbesondere sind wörtliche oder sinngemäße Zitate als solche gekennzeichnet. Mir ist bekannt, dass Zuwiderhandlung auch nachträglich zur Aberkennung des Abschlusses führen kann.

Leipzig, März 2011 Stanley Hillner