An Improved Design and Implementation of the Session-based SAMBO with Parallelization Techniques and MongoDB

Yidan Zhao

Examiner: Patrick Lambrix
Supervisor: Kristian Sandahl, Hongzhi Wang
Upphovsrätt
Detta dokument hålls tillgängligt på Internet – eller dess framtida ersättare – under 25 år från publiceringsdatum under förutsättning att inga extraordinära omständigheter uppstår.
Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för ickekommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns lösningar av teknisk och administrativ art.
Upphovsmannens ideella rätt innefattar rätt att bli nämnd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.
För ytterligare information om Linköping University Electronic Press se förlagets hemsida http://www.ep.liu.se/.

Copyright
The publishers will keep this document online on the Internet – or its possible replacement – for a period of 25 years starting from the date of publication barring exceptional circumstances.
The online availability of the document implies permanent permission for anyone to read, to download, or to print out single copies for his/hers own use and to use it unchanged for non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional upon the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.
According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.
For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its www home page: http://www.ep.liu.se/.

© Yidan Zhao
Abstract

The session-based SAMBO is an ontology alignment system involving MySQL to store matching results. Currently, SAMBO is able to align most ontologies within acceptable time. However, when it comes to large scale ontologies, SAMBO fails to reach the target. Thus, the main purpose of this thesis work is to improve the performance of SAMBO, especially in the case of matching large scale ontologies.

To reach the purpose, a comprehensive literature study and an investigation on two outstanding large scale ontology system are carried out with the aim of setting the improvement directions. A detailed investigation on the existing SAMBO is conducted to figure out in which aspects the system can be improved. Parallel matching process optimization and data management optimization are determined as the primary optimization goal of the thesis work. In the following, a few relevant techniques are studied and compared. Finally, an optimized design is proposed and implemented.

System testing results of the improved SAMBO show that both parallel matching process optimization and data management optimization contribute greatly to improve the performance of SAMBO. However the execution time of SAMBO to align large scale ontologies with database interaction is still unacceptable.

Keywords: Ontology; Large scale ontology alignment; MongoDB; NoSQL database; Parallelization computing; SAMBO
Acknowledgement

It is a great time to conduct my thesis work in ADIT, Linköping University (LiU). Many people have offered valuable help to me during each phase of my thesis work.

First of all, I would like to give my sincere gratitude to my internship supervisor Professor Patrick Lambrigx at ADIT, LiU. He offered me great help by providing research direction, necessary materials, valuable advice and feedback on the thesis. Secondly, I would like to thank you Professor Hongzhi Wang, my supervisor at Harbin Institute of Technology (HIT), who reviewed my report and gave me valuable feedback on thesis writing. I also want to thank you Kristian Sandahl, my supervisor at LiU, who gave comments on my thesis work in each phase of this thesis project.

In addition, I’m grateful to Huanyu Li, who helped me a lot when I encountered problems with the code. I would like to thank you Tea Nygren, Nina Zhang and Jian Zhao, who provided important reminders and helped to answer all kinds of questions during the entire project with great patience.

Lastly, I want to thank you Jiaqi Qin, Zhiming Gao and Lixiang Dong who gave me great support during this two years’ double degree program between LiU and HIT. I really enjoyed the time staying with you.

Linköping in September 2017

Yidan Zhao
## Contents

Abstract......................................................................................................................................... i  
Acknowledgement......................................................................................................................... ii  
Contents.......................................................................................................................................... iii  
Table of Figures.............................................................................................................................. v  
Table of Tables............................................................................................................................... vii  

### Chapter 1 Introduction............................................................................................................. 1  
1.1 Background................................................................................................................................. 1  
1.2 The purpose of project.................................................................................................................. 3  
1.3 The status of related research....................................................................................................... 4  
  1.3.1 Large scale ontology alignment......................................................................................... 4  
  1.3.2 Ontology matching parallelization................................................................................... 9  
  1.3.3 NoSQL databases in ontology matching.......................................................................... 10  
1.4 Research Approach..................................................................................................................... 12  
  1.4.1 Pre-study.............................................................................................................................. 12  
  1.4.2 The development approach.............................................................................................. 13  
1.5 Research question....................................................................................................................... 14  
1.6 Main content and organization of the thesis.............................................................................. 14  
1.7 Limitations................................................................................................................................ 15  

### Chapter 2 System Requirement Analysis................................................................................ 16  
2.1 The goal of the system.................................................................................................................. 16  
2.2 Analysis on the existing SAMBO............................................................................................... 16  
  2.2.1 The overall workflow of the existing SAMBO.................................................................. 16  
  2.2.2 Detailed analysis on data management.......................................................................... 18  
  2.2.3 Detailed analysis on matching process.......................................................................... 19  
2.3 The functional requirements....................................................................................................... 20  
2.4 The non-functional requirements.............................................................................................. 21  
2.5 Brief summary............................................................................................................................ 23  

### Chapter 3 System Design........................................................................................................ 24  
3.1 Architecture design..................................................................................................................... 24  
3.2 Optimization modules design..................................................................................................... 25  
3.3 Database Design........................................................................................................................ 26
Table of Figures

Figure 1.1 An Scenario of Ontology Alignment [5].................................................................2
Figure 1.2 The architecture of AML framework [20].............................................................7
Figure 1.3 The main responsibilities of the loading module in AML [9].................................8
Figure 1.4 The overall architecture of LogMap [21].............................................................8
Figure 1.5 The waterfall methodology..................................................................................13
Figure 2.1 The overall architecture of the session-based SAMBO [17].................................17
Figure 2.2 The primary design of database schema used in computation session[29]............19
Figure 2.3 Use case diagram of SAMBO..............................................................................21
Figure 3.1 Architecture Design............................................................................................24
Figure 3.2 Primary optimization modules of the improved SAMBO..................................25
Figure 3.3 The improved design of primary schema in database in matching process......26
Figure 3.4 The collection structure in MongoDB................................................................28
Figure 4.1 Primary business logic for the improved SAMBO.............................................34
Figure 4.2 Class diagram for chief classes involved in the alignment.................................36
Figure 4.3 Sequence diagram for loading ontologies process...........................................37
Figure 4.4 Sequence diagram for matching ontologies process........................................38
Figure 4.5 Sequence diagram for user validating process................................................39
Figure 4.6 Flowchart of matching workflow in parallel.....................................................40
Figure 4.7 Key codes of function parallelComputation()....................................................41
Figure 4.8 The workflow of computation similarity in function parallelComputation().....42
...........................................................................................................................................43
Figure 4.9 Code of class InstanceFeature...........................................................................43
Figure 4.10 Home page........................................................................................................43
Figure 4.11 Login page.........................................................................................................44
Figure 4.12 Sources of uploaded ontologies.......................................................................44
Figure 4.13 Upload ontologies............................................................................................44
Figure 4.14 Property matching configuration page............................................................45
Figure 4.15 Property matching suggestion page...............................................................45
Figure 4.16 The end of property matching.........................................................................46
Figure 4.17 Concept matching configuration page..............................................................46
Figure 4.18 Property matching suggestion page..................................................................47
Figure 4.19  Alignment result page..................................................................................... 47
Figure 4.20  Final alignment ontology in OWL format.......................................................... 48
Table of Tables

Table 3-1 Data type and description of fields in MongoDB ........................................... 27
Table 5-1 System testing environment.................................................................................. 49
Table 5-2 Test cases and results for loading ontologies.................................................. 50
Table 5-3 Test cases and results for matching ontologies............................................. 50
Table 5-4 Test cases and results for validating suggestions......................................... 51
Table 5-5 Test cases and results for generating alignment file.................................... 51
Table 5-6 Test cases for non-functional testing................................................................. 52
Table 5-7 Test results for old and new database interaction approach............................ 52
Table 5-8 Test results for parallel computing strategy without database interaction...... 53
Table 5-9 Test results for parallel computing strategy with database interaction............ 53
Table 5-10 Test results for the effect of database interaction........................................ 53
Chapter 1 Introduction

1.1 Background

In the past a few decades, the amount of data on the Internet has grown dramatically. Although this explosion of data enriches people’s knowledge in various domains, these data are never made good use of but only presented on millions of web pages. It is difficult to locate, retrieve and integrate relevant information on the Internet [1]. Thus, Semantic Web, as the extension of the existing web, is proposed to solve this kind of difficulties with the aim of automatically processing, integrating and reusing data from different information sources [2]. To achieve these goals, annotations are used to mark the content of web pages and then ontologies are introduced to specify the meanings of these semantic annotations [1].

An ontology typically defines a vocabulary of a specific domain with basic concepts and their relations as well as rules for combining concepts and relations when extending the vocabulary [3], providing the possibility of the interoperability among different data sources [1]. However, in the past a few years, many ontologies have been developed in the same domain with people’s different perspectives and are widely-used in various data sources. These ontologies in the same domain contain plenty of overlapping information, leading to the interoperability problem in retrieving, integrating and analyzing data across different data sources which are annotated with different ontologies [4]. To overcome this semantic heterogeneity issue, ontology alignment is proposed to figure out semantical correspondences between concepts and relations in different ontologies and can be applied in ontology merging, query answering, data translation, etc [3]. Figure 1.1 in [5] shows a scenario of ontology alignment between two ontologies with overlapping information. For example, the concept B-cell activation in GENE ONTOLOGY is equal to the concept B Cell Activation in SIGNAL-ONTOLOGY, while the concept antigen presentation in GENE ONTOLOGY is subsumed in the concept Antigen Processing and Presentation in SIGNAL-ONTOLOGY.

Many matchers have been proposed to generate correspondences between ontologies based on different ontology characteristics [6] and can be generally divided into
metadata-based and instance-based matchers [7]. Metadata-based matchers use characteristics inside ontology, such as concept names, synonyms and structural properties, and can be further divided into element-level and structure-level matchers [8]. Element-level matchers consider the relationships between concepts in isolation and ignore concept structural relations inside an ontology, while structure-level matchers take ontology structure into account, that is, how entities appear in a structure together [6]. On the contrary, instance-based matchers utilize associations between concepts and existing instances and determine correspondences if two concepts share instance overlap [8]. Moreover, some matchers introduce auxiliary information (e.g. WordNet) as reference to help to improve match quality [7].

Figure 1.1 An Scenario of Ontology Alignment [5]

So far, several ontology alignment systems have been set up. A general matching workflow of these alignment systems is composed of four processes, namely preprocessing, matching, combination and filtering [7]. Preprocessing transforms input ontologies into an internal data structure of ontology matching system to prepare for a faster matching computation. Matching utilizes a set of matchers to generate the similarities or dissimilarities between entity pairs. Combination and filtering processes combine and filter results from different matchers with optimized strategies to improve match quality and finally extract correspondences as a final alignment [6]. In addition,
ontology debugging process is also integrated into several advanced alignment systems to ensure coherence of final alignment [9].

Effectiveness and efficiency are two main aspects when evaluating an ontology alignment system. Effectiveness stands for correctness and completeness of alignment while efficiency represents time and memory efficiency [8]. At present, a few ontology alignment systems have been able to achieve high match quality in Ontology Alignment Evaluation Initiative (OAEI) campaigns. OAEI is an annual evaluation event with the purpose of comparing and improving alignment systems [10]. It assesses ontology alignment systems with five main measures: precision, recall, F-measure, coherence and runtime. Precision, recall and F-measure reflects effectiveness of the result alignment compared with a reference alignment [6]. Precision represents the ratio of the number of found relevant mappings to the number of total found mappings. Recall represents the ratio of the number of found relevant mappings to the number of total relevant mappings. F-measure is the harmonic mean of precision and recall. Coherence refers to assess whether there exists unsatisfiable classes in the result alignment when reasoning with the input ontologies. In addition, runtime indicates efficiency of an ontology alignment system which depicts the execution time of aligning ontologies of a given system under a specific running environment.

With the development of ontology matching techniques, several severe challenges have been realized. One of the most serious challenges is large scale ontology matching [6]. As matchers usually compare all entity pairs within two ontologies, typical time complexity of a matcher is Cartesian product of the number of concepts in two ontologies. Thus executing a matcher over large scale ontologies might take dozens of hours which is not acceptable by time requirements of ontology alignment systems. A survey of current state of solving this challenge will be presented in the section 1.3.

1.2 The purpose of project

The session-based SAMBO is an ontology alignment system developed by Linköping University. Currently, SAMBO is capable of performing ontology alignment effectively. The good F-measure results of SAMBO at OAEI in 2009 [11] demonstrates the effectiveness of SAMBO matching strategies. SAMBO is able to execute ontology
matching for small-size ontologies in reachable time, but fails to reach large scale ontology alignment.

The primary purpose of this thesis project is to improve the performance of the existing SAMBO, especially in the case of matching large scale ontologies. The improvement is expected to be achieved from two aspects, parallel matching process and data management optimization. To achieve this improvement, the purpose of the thesis project has been to investigate current status, challenges and trends of large scale ontology alignment. Furthermore, the purpose has been to investigate various essential parallel computing techniques and large scale data management techniques to determine which techniques are suitable for the existing SAMBO. In the end, based on the previous investigation results, the final purpose of this project has been to redesign and implement a paralleled matching process and an efficient data management method to promote the performance of SAMBO.

1.3 The status of related research

This section presents previous research which provide theoretical foundation for this thesis project.

1.3.1 Large scale ontology alignment

This section presents an overview of main challenges in large scale ontology alignment and current large scale ontology matching techniques as well as investigates two leading large scale ontology alignment systems in OAEI campaign.

1.3.1.1 Large scale ontology matching

Nowadays, ontologies in the real world can involve millions of concepts which will generate billions of pairs to be matched if no good matching systems are designed. The quadratic growth of search space and complex semantic heterogeneity between large scale ontologies reduce the effectiveness in achieving semantical correspondences [7]. Besides, as a computation intensive process and being time consuming, large scale ontology matching reduces the efficiency of the alignment process as well. First, large search space consumes large amount of time in matching, combination and filtering steps. Second, current ontology alignment systems usually load preprocessed ontologies as well as store intermediate match results in main memory which limits the efficiency in large
scale ontology matching [7]. According to OAEI results in 2016 [10], only few ontology alignment systems could generate high quality alignment between large scale ontologies under a given time and resource framework. Thus, achieving both good effectiveness and good efficiency is the major challenge in large scale ontology matching [3, 7].

To solve the challenge, various large scale matching techniques have been proposed which can be classified into four categories: reduction of search space of matching, parallel matching, self-tuning match workflow and reuse of previous match results [7].

- **Reduction of search space of matching** mainly adopts two kinds of approaches [7]. One is to early prune entity pairs with low similarity to avoid further matching to the utmost. The other one applies a divide-and-conquer strategy to partition the input ontologies and then divide the whole matching process into several smaller match tasks that each partition of the source ontology only matches with one partition of the target ontology. There are several methods to partition input ontologies. Falcon [12] applies cluster-based algorithm on ontology structures to decompose ontologies respectively and then aligns corresponding blocks based on anchors (similar entity pairs). Taxomap [13] partitions one of the input ontologies using cluster-based algorithm as well and then partitions the other ontology accordingly. Chiatti et al. in [14] implements a method to reduce search space through partitioning ontologies based on clustering techniques and then applying text mining techniques to identify cluster topics to detect mappable clusters. Anchor-Flood [15] and LogMap2 [16] implements a dynamical partition method which starts generating a block with an initial set of anchors and collects structural neighbors surrounding the anchors to find aligned entity pairs. The session-based SAMBO [17] reuses partial reference alignments to obtain mappable blocks based on ontology structures [18]. Both approaches avoid overmuch entity pairs match in large scale matching and thus improve efficiency, whereas might reduce match quality due to the missed entity pairs matching [8]. In addition, the latter approach reduces memory requirements when performing smaller match tasks [7].

- **Parallel matching** is a direct technique to speed up large scale matching [7]. There are two main kinds of parallel matching, namely inter-matcher and intra-matcher parallelization [8]. Inter-matcher parallelization runs independent matchers in
parallel on multi-cores or multiple processors. Intra-matcher parallelization typically partitions the input ontologies to construct a set of smaller match tasks which can be executed in parallel.

- Self-tuning match workflow assists to make decisions on selecting executed matchers and configuring combination and filtering settings [7]. The matcher selections and combination and filtering configurations affect matching effectiveness and efficiency significantly, especially for large scale matching and are difficult to be determined manually, even for experts [7]. Self-tuning match workflow usually applies automatic tuning technique, such as supervised machine learning, to help to find effective matcher selection and configuration. The session-based SAMBO [17] implements three self-tuning methods to recommend alignment strategies based on expert-validated segment pairs [19] or previous validated mapping suggestions.

- Reuse of previous match results saves previous match results and reuses the results of the unchanged ontology parts when a new but similar matching is to be executed [7]. Through reusing, a significant amount of match effort can be saved. In [17, 18], Lambrix et al. reuse a partial reference alignment in different ontology alignment steps to partition ontologies into mappable parts in the preprocessing step, to calculate similarity values on the basis of similar pattern of entity pairs in the reference alignment in the computation step and to filter mapping suggestions in the filtering step.

Due to the fact that the existing techniques have not been widely integrated into current ontology alignment systems, much more work needs to be done in this field.

1.3.1.2 Large scale ontology alignment systems

This section presents two leading alignment systems in large scale ontology alignment, AML and LogMap. These two systems both participated in the recent three years’ OAEI campaigns and completed all the six tasks in the Large biomedical ontologies track (which is the track using large scale ontologies as test cases) successfully.
AML

AML, which is short for AgreementMakerLight, is an automatic matching framework taking computational efficiency and large ontologies matching into account [9]. The framework consists of four main modules, the loading module, the primary matching module, the secondary matching module and the alignment selection and repair module [20]. The architecture of AML framework is presented in figure 1.2.

![Figure 1.2 The architecture of AML framework [20]](image)

The main responsibilities of the loading module is to load the input ontologies and parse each of them into AML ontology objects with two main data structures, the Lexicon and the RelationshipMap, as shown in figure 1.3. The Lexicon stores the lexical information of an ontology and the RelationshipMap stores the structural information of an ontology. These data structures only contain ontology information that is useful in the matching process instead of keeping the whole ontology object. This design efficiently saves a large amount of memory to keep ontologies and thus increases the available memory that can be used in the matching process. Besides, this design regards the lexical information as an index of the ontology which improves the efficiency of the matching process.

The primary and secondary matching modules take the responsibility of aligning ontology objects by utilizing different matchers. The matchers are divided into primary matchers and secondary matchers based on time complexity. Primary matchers employ HashMap cross-search strategy with O(n) time complexity, whereas secondary matchers
compare each term in one ontology with each term in another ontology in $O(n^2)$ time. Thus, primary matchers could achieve efficient matching for large scale ontology alignment.

![Diagram](image)

**Figure 1.3 The main responsibilities of the loading module in AML [9]**

The alignment selection and repair module ensures that all the mappings are above a given similarity threshold and conform to a given cardinality.

**LogMap**

LogMap is an ontology alignment system designed with the purpose of tackling large ontology matching and diagnosing semantic errors [21]. The overall architecture of LogMap is shown in figure 1.4.

![Diagram](image)

**Figure 1.4 The overall architecture of LogMap [21]**

The system parses the input ontologies to construct two efficient data structures, namely an inverted lexical indexation and a structural indexation. The lexical indexation indexes the labels of the concepts as well as their variations of an ontology. These indexations are used to efficiently compute an initial set of candidate mappings based on lexical correspondences between two ontologies. The structural indexation contains the concept hierarchy of an ontology using a data structure called interval labeling schema.
which significantly reduces the cost of computing typical queries on large scale concept hierarchies.

The matching process of LogMap is an iterative process which alternates two main steps: mapping repair and mapping discovery. The starting point of the matching process is the initial set of candidate mappings mentioned above. The mapping repair step detects and repairs unsatisfiable concepts on the current mapping results and input ontologies. To improve the efficiency of detecting unsatisfiability on large scale ontologies and mappings compared with traditional reasoners, LogMap uses Horn proposition logic to represent concept hierarchy. The mapping discovery step expands mappings by adding neighbor concepts of the expanded mappings in the last iteration and selecting mappings above a specific similarity threshold.

### 1.3.2 Ontology matching parallelization

With the development of ontology matching, a large amount of researches have proposed frameworks to improve efficiency of ontology matching through optimizing matching algorithms or reducing search space. These optimizations improve the efficiency at the price of reducing the effectiveness [22]. In contrast, parallelization takes the advantage of multiple cores to improve the performance of ontology matching, especially when considering runtime of matching, without lowering match quality. Parallel matching achieves the goal through dividing the whole matching task into several smaller ones which can be executed in parallel on a multi-core computer or on a distributed infrastructure. The key technique in parallel matching is to find a suitable size to partition the input ontologies equally which helps to keep load balancing and reduce communication overhead [23]. Furthermore, as a memory-intensive computation, the available memory shared among several cores should also be taken into account [23].

In [8], Gross et al. propose two different parallelization matching strategies, namely inter- and intra-matcher parallelization. Inter-matcher parallelization optimizes the matching workflow by utilizing multiple cores to execute independent matchers in parallel and then combing the results from different matchers [8]. However, due to different time complexity of matchers, the performance of inter-matcher parallelization is limited by the slowest matcher which leads to computation resource waste. Besides, high memory requirement becomes the bottleneck of inter-matcher parallelization because
each matcher loads complete ontology information into main memory. What’s more, redundant matching on candidate mappings is unavoidable unless there exists communication among matchers. Intra-matcher parallelization first partitions the input ontologies and then decomposes the entire matching task into smaller ones which can be executed in parallel [8]. In [8], the authors also describe a general size-based partitioning intra-matcher parallelization approach which is applicable to optimize all element-level, structure-level and instance-based matchers [8]. Compared with inter-matcher parallelization, this approach enables load balancing, limits memory consumption and avoids communication overhead among different cores. Nonetheless, it gives no consideration to the definition of optimal partition size which can maximize the performance improvement.

In [22], Amin et al. implement a parallel matching framework using data parallelism strategy which is similar to intra-matcher parallelization in [8]. Different from [8], this framework only loads necessary ontology information into memory for matchers based on their individual requirements with the aim of reducing memory consumption [22]. Unfortunately, this framework fails to describe the solution to define suitable partition size as well.

Mittra et al. depict another ontology matching framework utilizing parallelization and distribution technique [24]. In contrast with the size-based partitioning approach mentioned above, this framework makes use of clustering mechanism to partition large ontologies into smaller clusters by property-class relationships and subclassof relationships and processes matching tasks between smaller clusters in parallel [24]. The cluster-based partitioning approach shortens runtime and reduces memory consumption through reducing the search space. But it pays no attention to the potential performance reduction caused by the huge difference on cluster size.

1.3.3 NoSQL databases in ontology matching

1.3.3.1 NoSQL databases

RDBMS (Relational DataBase Management System) have been widely used by information systems for decades. RDBMS provides high reliability service based on ACID (Atomic, Consistent, Isolated, Durable) feature, resulting in performance
degradation. With the development of web application and cloud computing, more and more scenarios fail to fit in relational data model and the requirements of dynamic schema and high performance in large scale data processing emerge. The conception of NoSQL (not only SQL) database thus came up to meet the requirements at the price of loosing ACID feature to weaker BASE (Basically Available, Soft-state, Eventually Consistent) feature [25]. The primary scenarios of applying NoSQL databases are as follows [26]: 1) high concurrent of reading and writing with unnoticeable latency time; 2) efficient data storage and access, especially on big data; 3) high scalability in expansion and high availability supporting uninterrupted service.

So far, there has been several implementations of NoSQL databases and can be categorized to four classifications: Key-Value stores, Document databases, Wide-Column databases and Graph databases [27].

- Key-Value stores support a simple data model that data is stored as an array consisting of key-value pairs. Values can be simple data types or complex data structure like objects. They support fast query and high-scalability large scale data storage. Nonetheless, data processing can only be performed through keys and only exact matches are supported.

- Document databases are designed to store and manage documents in standard data exchange format, such as XML, JSON and BSON. Each value column in document databases contains varied attribute name/value pairs from row to row, aiming at supporting flexible schema. Different from Key-Value stores, Document databases allow both keys and values query. Document databases are suitable for use cases storing massive collection of documents as well as semi-structured data [27]. At present, the most popular Document database is MongoDB. MongoDB utilizes BSON documents to store complex data types as well as supports horizontal scalability, high-availability and flexibility to handle semi-structured data.

- Wide-Column databases use table as data model. But different from RDBMS, it doesn’t support table association and column values are stored contiguously. Wide-Column databases are designed to improve performance in large-scale batch-oriented data processing and data analysis. The most widely-used Wide-Column database is
Cassandra, which is an open source database with the following primary characteristics: 1) support linear scalability; 2) support high availability by using fault-tolerance and decentralized strategy to avoid the effect of single node failure.

- Graph databases model the database as structured relational graphs consisting of nodes representing objects and edges reflecting relationship among nodes. Graph databases are useful when an application attaches importance on relationships among data.

### 1.3.3.2 NoSQL databases in ontology alignment system

Currently, typical ontology alignment systems only utilize main memory to complete the matching process and databases are not involved in these systems. To the best of our knowledge, the session-based SAMBO [17] is the only ontology alignment system introducing database management into ontology matching because it supports partial computation when generating candidate mappings through storing and accessing interruptible session information in MySQL database. As the growth of data volume, MySQL, as a RDBMS, cannot afford to retrieve and query session information efficiently. Therefore, based on the requirements of high performance database access, NoSQL could be considered as a solution to this problem. However, there are no such researches on applying NoSQL databases in ontology alignment systems so far.

### 1.4 Research Approach

#### 1.4.1 Pre-study

At the beginning of this thesis project, a pre-study related to ontology, ontology alignment, OWL language and current large scale ontology matching techniques is conducted. The goal of the pre-study is to gain the background knowledge of this thesis project. Then, a comprehensive literature study of two leading ontology matching systems LogMap and AML as well as an investigation of the above systems are conducted to help to figure out their advantages in large scale ontology matching. Next, the existing session-based SAMBO is investigated as well to study its workflow and detailed implementation. In the end, since the improvement of the existing SAMBO focuses on parallel matching and data management optimization, parallel ontology matching techniques and high-performance data storage and query are studied with the purpose of comparing and selecting the techniques might be used in this thesis project.
1.4.2 The development approach

The development approach of this thesis project conforms to waterfall model because the model contains clear steps that are easy to follow. Waterfall model [28] involves a series of development steps with a feedback iteration between each step. The waterfall model divides software development into six steps, namely requirement, analysis, design, implementation, testing and deployment, which are shown in figure 1.5. Software quality is assured in the model by validating the result of each step with an explicit baseline.

To this thesis project, the first step is to analyze the requirements of improving the session-based SAMBO, that is, defining the requirements of the improved system. The second step is to analyze business logic and implementation of the existing SAMBO through a throughout code analysis. The purpose of the analysis is to figure out the defects of the system and find out in which aspects SAMBO can be optimized to reach the requirements. The following step is to conduct a design of optimization approaches before implementation. The baseline of the design is to meet all the purposed requirements. Finally, the improved SAMBO is tested and the optimization results are evaluated and analyzed.

![Figure 1.5 The waterfall methodology](image_url)
1.5 Research question

The overall objective of this project is to enhance the performance of the session-based SAMBO when processing large scale ontology matching. Thus, the research question of this thesis is defined as:

How to improve the performance of the session-based SAMBO for aligning large scale ontologies?

Since there are several aspects can be considered to optimize the performance, two subquestions are proposed to limit the scope of this thesis project. Each subquestion focuses on one improvement aspect and help to fulfill the purpose of this project.

1. How to optimize database design and interaction strategy to improve the performance of the session-based SAMBO?
2. How to use parallelization techniques to improve the performance of the session-based SAMBO?

1.6 Main content and organization of the thesis

The main content of this thesis project is to strengthen the performance of the session-based SAMBO through matching process parallelization and data management optimization. To be specific, matching process parallelization involves enabling the business logic of current matching process to work in parallel and improving the performance of complex matchers with parallelization technique as well. Data management optimization mainly introduces MongoDB to substitute current MySQL database aiming at supporting flexible schema to store session information as well as high-performance data storage and query. Meanwhile, an business logic design to simply database interaction strategy are also conducted to optimize data management.

The thesis report chiefly presents the design, implementation and evaluation of the improved SAMBO. The organization of this report is described as follows. Chapter 1 gives a brief introduction of background knowledge of the thesis project, involving ontology, ontology matchers, general composition of ontology alignment systems and current challenges, as well as the purpose of the project. In addition, the state of art of the
thesis topic is provided to present general approaches in dealing with large scale ontology matching, the design of current leading large scale ontology matching systems. Finally, research approach, research question and main content of the project are presented. Chapter 2 depicts the optimization goal of the project and then defines the requirements of the project based on a detailed analysis on the existing session-based SAMBO. Chapter 3 presents the high-level design and database design of the improved SAMBO, including the overview architecture and primary data schema design. The key techniques which are chosen to implement the improvements are discussed as well. Chapter 4 provides detailed implementation of the new business logic as well as parallel computing matching process to demonstrate how the improvements are implemented. In chapter 5, functional and non-functional tests are carried out and testing results as well as the research method used in this project are discussed in chapter 6. Finally, a conclusion of what have been done in this thesis work is presented in chapter 7.

1.7 Limitations

SAMBO is an ontology alignment system with many different versions. This project only optimizes the session-based SAMBO.

The main optimization objective of this project is to enable SAMBO to process large scale ontology matching in acceptable time, thus execution time of the matching process is the primary consideration when optimizing the system. The optimization of other evaluation criteria for ontology alignment system, such as precision, recall and F-measure, is not included in this project. In the system testing, execution time testing is also the chief aspect to evaluate the optimization.

This project focuses on optimizing the workflow and database interaction of SAMBO. No improvement is implemented on the essence of how matchers work to compute similarity values, that is, the effectiveness of matchers are not enhanced.
Chapter 2 System Requirement Analysis

This section mainly depicts the requirements of the thesis project. First, the optimization goal of the thesis project is defined. Next, an analysis of the session-based SAMBO is conducted to present defects of the existing implementation. In the end, both functional requirements and non-functional requirements are proposed with the purpose of improving the design.

2.1 The goal of the system

The existing SAMBO employs a matching process which can obtain match results with high quality. However, the current process cannot provide match results in acceptable time for large scale ontology alignment. Thus, the goal of this thesis project is to optimize the existing SAMBO to obtain the ability to process large scale ontology matching. At the meantime, the optimization approaches should also ensure that the improved system is easy to be extended and maintained in the future.

2.2 Analysis on the existing SAMBO

To achieve the goal of improving the existing SAMBO, it is imperative to conduct an analysis on the existing SAMBO. The analysis will be performed in two levels. First, an analysis of the overall workflow of the session-based SAMBO will be presented. Second, detailed analyses focused on data management and matching process will be presented respectively.

2.2.1 The overall workflow of the existing SAMBO

SAMBO is an ontology alignment and merging system for biomedical ontologies [4]. The system handles input ontologies in OWL format and generates an alignment after preprocessing, matching, combining, filtering and user validation [17]. With the awareness that user involvement is able to avoid many mapping errors in the matching process and then improve match quality, the latest version of SAMBO is redesigned with the introduction of session mechanism. The overall architecture of the session-based SAMBO is shown in figure 2.1, consisting of three kinds of sessions: computation, validation and recommendation sessions.
The computation sessions automatically generate mapping suggestions through preprocessing, matching, combing and filtering processes after user defines computation session settings.

- The preprocessing process is an optional process determined by user-defined settings. It partitions the input ontologies into corresponding mappable parts based on ontology structure to improve performance by reducing the search space.

![Figure 2.1](image)

**Figure 2.1 The overall architecture of the session-based SAMBO [17]**

- The matching process utilizes user-selected matchers to calculate similarity between entity pairs. The primary matchers in SAMBO are Edit-Distance, NGram, TermBasic, TermWN, UMLSKSearch and Naive Bayes. Edit-Distance and NGram are string-based matchers calculating similarities by edit distance and 3-grams algorithms. TermBasic is also a string-based matcher combining edit distance, 3-grams and Porter stemming algorithms. TermWN and UMLSKSearch obtain mapping suggestions through making use of third-party resources WordNet and UMLS (Unified Medical Language System) as thesaurus. Naive Bayes is an instance-based learning matcher based on the similarity of literature researches referring to entity pairs. All the user-selected matchers is executed for each entity pair at one time and the similarity results are stored in each matcher’s own similarity table in MySQL database to support fast combined and filtered results when the computation session is interrupted as well as to avoid re-computation in the case of iterative alignment.
The combining process combines the similarity values obtained in the matching process using weighted-sum or maximum-based approach. The combined similarity values are then filtered based on user-defined threshold(s) and the retained entity pairs are mapping suggestions that need to be validated by users.

The validation sessions provide an user interface for users to accept or reject mapping suggestions. All the validated mappings, both accepted and rejected ones, are stored in the mapping decisions table. Besides, a reasoner is used to check the consistency of the accepted mappings and then the final alignment is output.

The recommendation sessions give users suggestions about the settings of matching, combining and filtering processes, aiming to perform the best match quality.

2.2.2 Detailed analysis on data management

Since interruptible session mechanism is the key feature of the existing SAMBO, database becomes an indispensable part in the system to store and provide computed similarity values. Thus, the performance of data storage plays an important role in affecting the performance of SAMBO. The most common scenario of using database storage is storing and querying similarity values between entity pairs in computation session. The primary design of schema in database used in computation session in the latest version of session-based SAMBO are shown in figure 2.2. When computing similarity value for an entity pair, checking whether similarity values of all the user-selected matchers are available in the database will be executed first. In current database design, each matcher has its individual similarity value table and thus queries on each table are needed to get all the similarity values. Test results comparing execution time of matching the same ontology pair with or without database interaction on the existing SAMBO is shown in table 2-1 and table 2-2. It is obvious to see that current database interaction strategy takes a great proportion of execution time in the matching process. Current data management design leads to high database query workload and degrades performance. It is better to use one table to store similarity values for all matchers, whereas this solution will cause sparse data in a fixed schema table. Therefore, database supports flexible schema should be considered.
Besides, in large scale ontology matching, millions or billions of records will be stored in one table. The use of foreign keys in relational database will degrade performance when inserting or updating records in the table.

![Database Schema Diagram](image)

**Figure 2.2 The primary design of database schema used in computation session[29]**

### 2.2.3 Detailed analysis on matching process

A typical workflow of the mapping process in the existing SAMBO is first generating a list of all the entity pairs that needed to be matched. Then, each entity pair performs the matching process in sequence. The serial matching workflow doesn’t take full advantage of computation resources and becomes slow when matching millions of entity pairs. Utilizing multi-threads to match entity pairs in parallel could contribute to the speed of the matching process.

Moreover, the implementation of some complex matchers doesn’t benefit from parallelization techniques as well. For instance, Naive Bayes matcher is a complex instance-based learning matcher based on the intuition that similarity values between entity pairs can be defined based on the probability that documents about one concept are also about the other concept and vice versa [30]. Naive Bayes matcher employs a training process to generate a document classifier for each ontology. In the training process, the algorithm calculates conditional probability of each word from the corpus of documents appearing in different concept categories [30]. When matching large scale ontologies, the
scale of documents corpus is huge and the number of conditional probability to be calculated would be tremendous. Thus, it would take a long time to generate the document classifier. Due to the calculation of conditional probability of each word is independent of each other, parallel training making use of multi-threads can be a possible approach to improve training speed.

2.3 The functional requirements

Since the primary goal of this thesis project is to improve performance of the existing SAMBO when it comes to large scale ontology matching, the improved SAMBO must fulfill all the functional requirements of the old system. After analyzing the system, the main use cases of the system are presented in figure 2.3.

Based on the use case diagram, the functional requirements are defined as follows:

- The user shall be able to upload ontologies to the system. The source of ontologies could be from local file system, web address or the server.
- The user shall be able to match two ontologies with a user-defined matching strategy. A user-defined matching strategy indicates that the user shall be able to select matchers and define their weight in the matching process. The user shall be able to set combination and filtering method as well.
- The system shall be able to store matching results in the database.
- The system shall be able to interrupt an on-going matching process when the system reaches the interrupted condition set by user.
- The user shall be able to validate mapping suggestions after matching process is completed and mapping suggestions are displayed.
- After user validates all the mapping suggestions, the system shall be able to create an alignment file in OWL format and the user shall be able to download this alignment file.
2.4 The non-functional requirements

This section lists the non-functional requirements of the system.

- Data management optimization
  Based on the detailed analysis of the database design in the above section, it is better to apply a database supporting flexible schema and high-performance large scale data storage and query. So far, these requirements are not well supported by RDBMS, while NoSQL databases are able to reach the target. Thus, NoSQL database, such as MongoDB, should be introduced to substitute currently used MySQL database.

- Matching workflow optimization
  To improve efficiency of the matching process, the match workflow shall be designed with parallelization strategy. Parallel matching techniques, such as intra-matcher parallelization, should be integrated in the existing SAMBO. In addition, the internal
implementation of complex matchers should also be optimized with parallelization techniques.

- **Execution time**
  Execution time is a crucial non-functional requirement when evaluating an ontology matching system. Execution time of the session-based SAMBO should only consider the execution time of computation session and exclude user involvement time. The execution time of the improved SAMBO should be shorter than that of the current session-based SAMBO and be close to the execution time of leading large scale ontology alignment systems.

- **Matching results quality**
  Precision, recall and F-measure are used as measures to evaluate the effectiveness of match results. Due to the fact that the improvements don’t modify the internal logic of the existing matchers and the nature of matching strategies, the improved SAMBO should ensure that the matching results quality is no lower than that of the existing SAMBO.

- **Be able to process large scale ontology matching**
  Due to the fact that the size of ontologies in the real world come to be large-scale, it is critical for ontology matching system could deal with this challenge. Since the current SAMBO doesn’t work well when the size of ontologies increases, the ability of processing large scale ontology matching becomes an important and fatal requirement of the improvement.

- **Maintainability**
  The improved SAMBO should be implemented with readable code and necessary documents, contributing to the continuous improvement in the future.

- **Extensibility**
  The design of the optimization approaches should make sure the system structure is easy to be extended. For instance, when adding or modifying matchers in the matching module, other modules in the system should not be affected.
2.5 Brief summary

This section introduces the current SAMBO system and the requirements of the thesis project. The analysis of the session-based SAMBO shows the functions of the system which implies the functional requirements of the improved system. The analysis also presents defects of the existing implementation showing the directions that can be enhanced. In the end, both functional requirements and non-functional requirements are listed.
Chapter 3 System Design

This chapter shows the overview optimization design in the project. First, the architecture of the session-based SAMBO is presented in section 3.1. In section 3.2, an overview of optimization modules are demonstrated. Database design for the improved SAMBO is described in section 3.3. Finally, key techniques utilized in the project are discussed in section 3.4.

3.1 Architecture design

The architecture design of the system conforms to three-tier architecture which is shown in figure 3.1. Three tiers are presentation tier, business logic tier and data access tier, respectively.

![Figure 3.1 Architecture Design](image)

The presentation tier exhibits user interface to users, enabling users to upload ontologies, configuring matching strategies and validating mapping suggestions. All the requests from users are sent to the Servlet classes in business logic tier as HttpServletRequest based on Java Servlet technique.

The business logic tier is responsible for dealing with different alignment modules through calling Servlet classes in the server side. The primary jobs in this tier are to
compute matching suggestions, to validate mapping suggestions, to generate matching strategy configuration recommendations and to store or query validated mapping suggestions with accessing the database to through data access tier.

### 3.2 Optimization modules design

According to the detailed analysis of the existing SAMBO in section 2.2, the optimization modules of the project are designed as shown in figure 3.2. The primary optimization task is divided into three modules, which are business logic optimization, parallel matching optimization and data management optimization. The purpose of business logic optimization is to speed up SAMBO through reducing unnecessary business procedures. Parallel matching optimization is further subdivided into two aspects: one is to execute the holistic match workflow in parallel, the other one is to parallelize the internal implementation of complex matching algorithms. Data management optimization aims to optimize the performance of data query and storage in SAMBO through introducing MongoDB to store session information and reduce database interaction through designing a reasonable schema for MongoDB.  

![Figure 3.2 Primary optimization modules of the improved SAMBO](image-url)
3.3 Database Design

The main purpose of using database during the matching process is to store similarity results of different matchers which can be retrieved to offer mapping suggestions when a computation session is interrupted and to avoid re-computation in iterative alignment. The idea of the optimized design is to reduce the workload of querying and storing similarities which contributing to improve database performance. The new design employs flexible schema characteristic in MongoDB to minimize queries. The primary schema in database involved in matching process is shown in figure 3.3.

```
_id: ObjectID("ZZZZ"),
scname: "SOURCE_CONCEPT_NAME",
tcname: "TARGET_CONCEPT_NAME",
type: ENTITY_PAIR_TYPE,
m0: SIMILARITY-VALUE,
m1: SIMILARITY-VALUE,
...
m8: SIMILARITY-VALUE,
```

Figure 3.3 The improved design of primary schema in database in matching process

In this design, for each entity pair, SAMBO generates a Bson document containing all the similarity results that have been calculated by diverse matchers for one entity pair. The _id field is automatically generated by MongoDB to distinguish different entity pairs. The scname and tcname fields represent the names of two entities that have been matched. The type field is a flag to identify the source of an entity pair, whether it belongs to concept or relation. The remaining fields store similarity values of matchers. Each of m0 to m8 fields stands for a specific matcher, which is listed in table 3-1. Different from fixed schema in the existing system, the new design only store similarity values of matchers which have been used to match the entity pair. This means that in MongoDB the schema of different entity pairs’ documents vary from each other and is easy to be extended. When matching entity pair with a new matcher, database only needs to add one more field in the end of the document. Due to all the information of one entity pair is stored in one document, retrieving existing similarity values of one entity pair can be executed with only one query which sharply reducing the number of queries compared with the existing system. The data type and detailed description of each field for a document is listed in table 3-1.
When the user uploads two ontologies, the system creates a unique collection for the ontology pair to store all entity pair documents related to it. The structure of collections in MongoDB is shown in figure 3.4. Compared with the current data model used in MySQL, this design avoids to store all similarity results of different ontology pairs together and makes use of collection to separate them which aims to reduce the search space when retrieving data. To improve the performance of data retrieval, a compound index on scname field and tname is created at the meantime when the collection is created. Besides, for large scale ontologies, the system creates several collections for one ontology pair. The idea of this design is to avoid storing too many documents in one collection which will lead to decreasing database query performance dramatically. The system uses scname field to map different entity pairs to different collections.

Table 3-1 Data type and description of fields in MongoDB

<table>
<thead>
<tr>
<th>Field</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_id</td>
<td>ObjectId</td>
<td>The unique identification for a document. Automatically generated by MongoDB.</td>
</tr>
<tr>
<td>scname</td>
<td>String</td>
<td>The name of source ontology concept.</td>
</tr>
<tr>
<td>tname</td>
<td>String</td>
<td>The name of target ontology concept.</td>
</tr>
<tr>
<td>type</td>
<td>Int</td>
<td>A flag to distinguish whether an entity pair is conceptual or structural.</td>
</tr>
<tr>
<td>m0</td>
<td>Double</td>
<td>Similarity result of matcher Edit Distance.</td>
</tr>
<tr>
<td>m1</td>
<td>Double</td>
<td>Similarity result of matcher NGram.</td>
</tr>
<tr>
<td>m2</td>
<td>Double</td>
<td>Similarity result of matcher WL.</td>
</tr>
<tr>
<td>m3</td>
<td>Double</td>
<td>Similarity result of matcher WN.</td>
</tr>
<tr>
<td>m4</td>
<td>Double</td>
<td>Similarity result of matcher TermBasic.</td>
</tr>
<tr>
<td>m5</td>
<td>Double</td>
<td>Similarity result of matcher TermWN.</td>
</tr>
<tr>
<td>m6</td>
<td>Double</td>
<td>Similarity result of matcher UMLSKSearch.</td>
</tr>
<tr>
<td>m8</td>
<td>Double</td>
<td>Similarity result of matcher BayesLearning.</td>
</tr>
</tbody>
</table>

3.4 Key techniques

3.4.1 Parallel matching

Parallelization is a key aspect to optimize the existing SAMBO with the aim of making full use of computing resources and reducing the execution time of matching process. To implement parallelization, it is dispensable to employ multithreading programming technique to process matching in parallel. Several techniques have been investigated to find the most suitable technique to improve current matching strategy.
ExecutorService framework is firstly introduced in java.util.concurrent package of Java 5 and then completed in Java 6. ExecutorService provides methods to manage progress and termination of asynchronous tasks executing in multiple threads, simplifying thread-handling when programming in Java [31, 32]. ExecutorService creates a thread pool with several threads initiated in it and then assigns tasks to available threads in the pool. Once a task is completed, the thread is returned to the pool and the Callable interface can be used to return a Future object containing task result. The Callable and Future interfaces can be used in the project to achieve similarity results calculated by matchers.
ExecutorCompletionService provides similar service compared with ExecutorService framework. ExecutorService calls function get() in Future interface to obtain task result. However, if the task is not completed when calling function get() to obtain result, function get() will block until current task is completed. This leads to those completed results of subsequent tasks can not be processed in time. ExecutorCompletionService solves this problem by decoupling the production of new asynchronous tasks from the consumption of the results of completed tasks [33]. ExecutorCompletionService manages an internal completion queue where completed tasks can be inserted into it. Then the Future interface obtains task results from this queue ensuring that all the completed results can be processed in time which improves the efficiency of ExecutorService.

Fork/Join framework is introduced into java.util.concurrent package from Java 7 [34]. Fork/Join framework in nature is an implementation of divide and conquer strategy. It divides a big task into several subtasks recursively until the subtasks are small enough. Then Fork/Join processes these subtasks in parallel and merges the results [35]. Fork/Join employs work-stealing mechanism to boost computation performance [34]: a thread finds and processes subtasks in other threads’ scheduling queue after completing all the subtasks in its own scheduling queue. However, according to [35], when memory is allocated in a high rate, the efficiency of Fork/Join will decrease dramatically because threads have to stop all the time to collect garbage produced by a great number of subtasks.

After comparing the multithreading programming techniques described above, ExecutorCompletionService is finally chosen to be utilized in the thesis project. The reason is that ExecutorCompletionService is more efficient than ExecutorService since it enables to process completed task results in time, while Fork/Join is more suitable for recursive problem and might lose efficiency when processing large scale problem.

3.4.2 Thread safety in parallel matching

As described section 3.2, a parallel matching strategy is involved in SAMBO with the expectation to speed up the matching process. However, this also introduces thread safety problem into the system. Shared data is the most common reason leading to thread
unsafety. To make sure the matching process work in a correct way, shared data are avoided to be used as much as possible. In the design of parallel matching, shared container can not be avoided in some cases. Therefore, some thread safety containers are investigated.

- Hashtable implements a thread-safe hash table [36]. Every element in Hashtable is a key-value pair. Hashtable implements thread safety by enabling all the public functions of Hashtable synchronized. This approach ensures thread safety by limiting only one thread can access Hashtable at one time which makes Hashtable show poor performance in multi-thread access.

- synchronizedMap and synchronizedList provides a synchronized wrapper for Map and List, respectively. The wrapper ensures synchronization of the wrapped object for any single operations. However, for compound operations, the wrapper might fail to support thread safety. So, to ensure absolute thread safety, the user must add additional synchronized block manually[37].

- ConcurrentHashMap is a thread-safe version of HashMap and supports the same functions as Hashtable [38]. Unlike Hashtable processing synchronization through locking the entire table, ConcurrentHashMap utilizes lock with lower granularity through using separate locks for separate buckets. Thus, multiple reads and writes can always be executed concurrently which guaranteeing high performance in multi-thread access.

- CopyOnWriteArrayList and CopyOnWriteSet is designed with the idea that when performing a mutative operation, the container creates a copy of itself where the mutative operation is executed on and then makes a reference to the copy after mutation [39, 40]. Due to the fact that the cost of copying a container is huge, CopyOnWrite containers is suitable for the situation with fewer write and insert operations.

Given that the requirement of high performance is the key point of optimizing SAMBO, ConcurrentHashMap is chosen as shared container in the project.
Besides, thread safety of the third-party library is also considered. For example, TermWN matcher employs JWNL API of WordNet to look up index words. When multiple threads use a shared Dictionary instance of JWNL to execute query, deadlock occurs. Then synchronized block should be added to keep thread safety and obtain correct result from the library.

3.4.3 Database design optimization

The session-based SAMBO frequently stores and retrieves data in database to support its session characteristic. To large scale ontology matching, the system needs to deal with millions of database operations in extremely short time which is difficult for relational database to afford. Besides, fixed data schema and normalized database design of relational database also aggravate the burden of database query. Thus, with the purpose of meeting time requirements, replacing current MySQL database with MongoDB providing higher performance and flexible data schema [41] is considered.

MongoDB is an open source document database originally released in 2009, initiated by 10gen Company [42]. It is designed to be scalable and is written in C++. MongoDB stores data into collections of documents in the format of BSON (Binary Serialized Document Format) instead of normalized relational tables in relational databases. Thus, MongoDB is schema free and is easy to add or change the existing document structure. A document represents a record in the database containing a set of attributes without predefined schema. Each document can involve diverse data structures or even embedded documents and arrays, which is capable of showing complex hierarchy within a record. MongoDB supports high-speed access to massive data through using memory mapped files structure [43]. To make full use of available memory to improve the performance, MongoDB uses B-Tree to index the files. MongoDB also offers powerful query language on read and write operations as well as data aggregation. To retrieve data from a collection, a query document is created with the attributes that the matched documents should contain.

The most frequent database operation scenario in SAMBO is that SAMBO queries the database to obtain existing similarity value for every single entity pair. For database storing billions of similarity documents for large scale ontologies, executing a query needs to traverse all the documents in a collection which results in unacceptable time.
consumption. To optimize this situation, index is created to speed up query operation. MongoDB provides rich types of index, such as single field index, compound index and text index [44]. When implementing the improved SAMBO, all the three kinds of indexes mentioned above were tested and finally compound index is chosen to be used in the system because of its higher performance in SAMBO’s scenario compared with the other ones.

3.5 Summary

This section presents the high-level design as well as a new MongoDB data schema design in the improved SAMBO. In the latter part, the primary technique difficulties faced in system implementation are depicted followed by several optional solutions as well as reason for the chosen solution.
Chapter 4 System Implementation

In this section, detailed system implementation is described. Section 4.1 lists the development environment and tool of the thesis project. Next, detailed business logic optimization approach is presented with flowcharts, class diagram and sequence diagrams in section 4.2, followed by the implementation of parallel matching in section 4.3. In the end, key interfaces of the improved SAMBO is presented in section 4.4.

4.1 The environment of system implementation

The development environment of the improved SAMBO is conducted on a laptop with Intel i5 CPU (two cores, four threads, 2.4GHz), running Windows 10 Operating System. NetBeans IDE 8.2 is used as development tool during the development. The running environment is Java JDK 1.8. The system is deployed on Apache Tomcat Server 8.0.27, using MongoDB 3.4.3 as database. In addition, SourceTree, a free Git client, is used as version control tool.

4.2 Business logic optimization

The new business logic is shown in figure 4.1. The system first loads ontologies to construct essential data structures used in the alignment and checks the database to see whether the ontology pair has been matched before. If there exists no similarity collections for the ontology pair, the system creates a collection in MongoDB for the ontology pair. Next, the system creates a task list for all entity pairs generated from the ontology pair and then the matching process starts. For each entity pair, the matching process works in the following way. If the ontology pair has been matched before, the system retrieves database to fetch similarities for the entity pair. If the query results contain available similarity results for all the matchers selected by user, the results will be stored in memory. If the query results contain partial or no similarity results of user-selected matchers, the system will use matchers to compute missing similarity values and then generate an update document for the database. If the ontology pair hasn’t been matched before, the system will skip the step of retrieving database, process all user-selected matchers and then create an insert document for the database. After all the entity pairs obtaining entire similarity results, the system will first update the database in batch and then combine and filter matching results with user-defined strategy to generate
mapping suggestions. Finally, the mapping suggestions are shown to the user to be validated.

![Flowchart of primary business logic for the improved SAMBO](image)

**Figure 4.1 Primary business logic for the improved SAMBO**

In the existing SAMBO, for each entity pair, the system needs to process database interaction for three times: fetching previous computed similarity values before computing, updating database with new similarities and fetching similarities again to process combine and filter step. As described in section 2.2.2, database interaction occupies a large proportion of time in the matching process. This will lead to
unacceptable time consumption when aligning large scale ontologies. Thus, the new business logic implementation focuses on reduce database interaction times.

Compared with the existing system, the optimization of business logic is implemented mainly in two aspects. One is that after loading ontologies, the system will query the database to check whether the similarity collection for an ontology pair exists. If the collection does not exist, which means that no available similarity values is stored in the database, the query before computing similarities will be ignored. This saves lots of time for the first-time aligned ontology pair. The other one is that the similarity results will be kept in memory before combination step which cuts the last database interaction in the old system.

In figure 4.2, the class diagram presents chief classes involved in the alignment process. For each class, only the most relevant variables and functions are listed in this diagram. The optimization for business logic is mainly located in class MergerManager, class SimValueConstructor and class Task. Class MergerManager is responsible for managing the workflow of the entire alignment process. Each time when user sends requests to Servlet, the Servlet will call the instance of MergerManager to process the requests. Class SimValueConstructor is the class calculating similarity values for an ontology pair and interacting with database. In the matching process, two SimValueConstructor instances are employed to process the computation for conceptual and structural entity pairs respectively. Class Task is the elementary unit used in similarity computation. One Task object is correspondence to one entity pair. When the computation of a Task object is finished, the combined result will be stored in a Pair object and once it fulfills the filter condition, the Pair object will be inserted into a suggestion vector.

The entire ontology alignment process is mainly composed of three sub-processes: loading ontologies, matching ontologies and user validation. The following sections will present detailed implementation of these processes.
4.2.1 Loading ontologies process

The detailed description of loading ontologies process is presented in sequence diagram 4.3. When the user upload ontologies, the system will create a HttpServletRequest to activate LoadFileServlet. The LoadFileServlet will instantiate an MergeManager object which manages the entire alignment process for an ontology pair. The MergeManager object will instantiate an OntManager object who is responsible to manage both source and target ontologies. The OntManager object creates two MOntology instances for each ontology respectively. It calls function in MOntology to load ontology files from local disk, web address or server and build classes, properties and relationship structure which will be used in the matching process. After building, an OntManager object containing source and target MOntology instances is returned to the MergerManager object. In the following, the MergerManager object will call function in class MapOntologyGenerate to check whether the similarity collection for the ontology pair exists in database. Class MapOntologyGenerate provides functions related to database collection operations. If the collection for the ontology pair doesn’t exist, it will call internal function to create the collection and its index. A boolean value will be returned to the MergerManager object indicating the query result. In the end of the loading process, the LoadFileServlet will get a returned MergerManager object.
4.2.2 Matching ontologies process

After loading ontologies, user selects matchers and configures combination and filtering parameters for the matching process. When user starts matching, all these configurations will be sent to MainServlet through a HttpServletRequest. Then the MainServlet will call the MergerManager object created in loading process to manage matching and return candidate suggestions. The MergerManager object will call the instance of SimValueConstructor to instantiate all the user-selected matchers at the beginning of matching process and then generate a list of Tasks through calling its internal function. The SimValueConstructor object will compute similarity values for these Tasks in parallel and stores new similarities in database. Finally, all the entity pairs passing filtering will be added into a suggestion vector and returned to the MainServlet. The sequence diagram of matching process is presented in figure 4.4.

![Sequence diagram for loading ontologies process](image-url)
4.2.3 User validation process

In user validation process, the MainServlet will display suggestions obtained from the matching process to user. If there are several suggestions containing the same entity from source ontology, these suggestions will be displayed to user at the same time. User can determine the relationship of a suggestion as Equivalence, Sub-Concept or Super-Concept as well as reject the suggestion based on user’s background knowledge. The typical sequence diagram when user accepts the relationship of a suggestion as equivalence is displayed in figure 4.5.
4.3 Parallel matching optimization

4.3.1 Parallel matching workflow optimization

For large scale ontology matching, the number of entity pairs to be matched is huge. Thus, to improve the performance, as discussed in section 3.3.1, the optimized matching workflow employs ExecutorCompletionService framework to execute matching task in parallel. The implementation of parallel matching workflow is shown in figure 4.6.
The matching process starts with generating a task list of input ontologies. With the consideration that the size of this list for large scale ontologies would be on the order of billions, the number of tasks submitted to be executed in parallel will be limited in a pre-
defined size. The system submits every computation task parallelComputation() to ExecutorCompletionService and Executor performs these tasks in parallel. The function of task parallelComputation() is to execute similarity computation and access similarity result from database for a specific Task. Function parallelComputation() is executed in parallel through calling an override function call() in interface Callable<Pair>. Key codes of function parallelComputation() is presented in figure 4.7. ExecutorCompletionService calls function take() to get the returned computation result of function call() as a Future object. The process of how function parallelComputation() executing computation for a Task is shown in figure 4.8. Each parallelComputation() generates an insert or update document containing similarity result which will be committed to database in batch after all parallelComputation() are completed. Since several parallelComputation() running on multiple threads put the documents in a shared container in parallel, ConcurrentHashMap is used to ensure thread safety as discussed in section 3.3.2.

```java
public Callable<Pair> parallelComputation(Task task,
    ConcurrentHashMap<Task,MongoDBWriteModel> updateRequests)
{
    Callable<Pair> computationResults = new Callable<Pair>(){
        @Override
        public Pair call() throws Exception {
            ...
        }
    };
    return computationResults;
}
```

Figure 4.7 Key codes of function parallelComputation()

### 4.3.2 Parallel matching algorithms optimization

The improved SAMBO implements parallel computation of training process of Naive Bayes matcher. There are three steps in the training process optimized. The first one is generating features for each document in the training set. Since the each document’s features are independent from each other, the computation can be executed in parallel. Class InstanceFeature implementing the interface Callable<Instance> overrides function call() to compute feature for a document and return the feature as an Instance object. The code of class InstanceFeature is presented in figure 4.9. Similarly, there are two other steps is processed in parallel. One is calculating the conditional probability of each
feature in the different categories. The other one is calculating posterior probability of each document in the testing set. All of these steps are implemented by calling ExecutorCompletionService.

Figure 4.8 The workflow of computation similarity in function parallelComputation()
4.4 Key Interfaces of the software system

This section presents the key interfaces [17] when an user uses SAMBO to align ontologies in an usual way.

Figure 4.10 shows the home page of SAMBO when an user starts to use the system.

When an user wants to sign in the system, she needs to click “Login” and the system will go to the Login page shown in Figure 4.11.

After inputting username and password and clicking on “login” button, the user can start to upload ontologies. User could choose different sources of uploaded ontologies, including URL of web address, local file system address or ontologies on the server, as shown in figure 4.12. In figure 4.13, user could set the name of aligned ontology file and

```java
class InstanceFeature implements Callable<Instance> {
    FileDocument doc = null;
    int category = -1;

    public InstanceFeature(FileDocument doc, int category) {
        this.category = category;
        this.doc = doc;
    }

    @Override
    public Instance call() throws Exception {
        Instance instance = new Instance(doc.featureVector(),
                                           category, doc.file.getName(), doc);
        return instance;
    }
}
```

Figure 4.9 Code of class InstanceFeature
decide whether to use database in the matching process. If user doesn’t select database, the matching process will match ontology pair without querying database to obtain previous similarity results and the matching results this time will not store in the database as well.

Figure 4.11 Login page

Figure 4.12 Sources of uploaded ontologies

Figure 4.13 Upload ontologies

When user clicks on “Upload” button, the ontologies will be uploaded and the system prepares to process property matching. In process property matching, TermBasic matcher (a combination of Edit Distance matcher, Ngram matcher and porter stemming matcher)
is the default matcher and user could set threshold which is used to filter suggestions in figure 4.14. After user clicks on “Start” button, the system processes property matching and presents suggestions one by one in figure 4.15. For each suggestion, user could decide to accept or reject the suggestion as equivalent entity.

After evaluating all the suggestions, the user could click on “Finalize” button in figure 4.16 to end property matching and start concept matching as shown in figure 4.17. Similar to property matching configuration, users could decide matchers to be used, threshold and combination strategy and then click on “Start Computation” button to execute concept matching.
Since concept matching takes most of time in ontology matching, user usually needs to wait for a moment to get matching suggestions. All the suggestions will be presented one by one and suggestions related to the same concepts will be displayed together as shown in figure 4.18. User could decide the relation of a suggestion as equivalence, sub-class or sup-class.

After validating all the suggestions, the system will go back to concept matching configuration page in figure 4.17. This is because ontology matching is always an iterative process and allow users to select different matchers or combination and filtering strategies to align a pair of ontologies for more than one time. The matching process will be finished after user clicks on “Finish” button in figure 4.17 and the system will go to alignment result page in figure 4.19.
As shown in figure 4.19, user could click on the link “The Alignment in OWL file” and then the system will present the final alignment ontology in OWL format as shown in figure 4.20.

4.5 Brief summary

This section describes detailed implementation approach to improve SAMBO, including the implementation of optimized business logic and parallel matching approaches. Detailed flowcharts, class diagrams and sequence diagrams are presented aiming at making the implementation clear. Key interfaces for the improved SAMBO are presented in the end of this section.
Figure 4.20 Final alignment ontology in OWL format
Chapter 5 System Testing

This section describes detailed system testing results of the improved SAMBO. Both functional and non-functional tests are processed. Section 5.1 lists the operating environment of the tests followed by all testing results presented in section 5.2.

5.1 Testing environment

This testing environment of the improved SAMBO, which is the same with development environment, is listed in table 5-1.

<table>
<thead>
<tr>
<th>Operation system</th>
<th>Windows 10 OS, Intel i5 CPU(two cores, four threads, 2.4GHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory</td>
<td>8GB</td>
</tr>
<tr>
<td>Server</td>
<td>Apache Tomcat Server 8.0.27</td>
</tr>
<tr>
<td>Browser</td>
<td>Opera 47.0</td>
</tr>
<tr>
<td>Database</td>
<td>MongoDB 3.4.3</td>
</tr>
<tr>
<td>IDE</td>
<td>Netbeans 8.2</td>
</tr>
<tr>
<td>JRE</td>
<td>1.8.0_121</td>
</tr>
</tbody>
</table>

5.2 Testing results

The purpose of this thesis project is to improve the performance of the session-based SAMBO for large scale ontology matching. To validate whether the improved SAMBO fulfills the goal, a thorough system testing is performed. The system testing is executed in two phases: functional testing and non-functional testing. Functional testing aims to verify whether the improved system achieves the functions of use cases described in section 2.3. Non-functional testing for this project focuses on testing runtime of the matching process, validating whether the work in this project contributes to improve the performance of SAMBO, especially in the case of large scale ontology matching.

5.2.1 Functional testing

Functional testing involves four main steps when running SAMBO: loading ontologies, matching ontologies, validating suggestions and generating merged alignment file. Test cases are designed based on functional requirements. Each set of test cases is corresponding to one main step mentioned above. For each test case, different kinds of ontologies with different scales are all tested to make sure the improved SAMBO could work smoothly in practical use.
Test cases for loading ontologies step are designed to verify whether the system could load ontologies from different sources. The expectation result of this step is SAMBO constructs data structure for ontologies and creates collections in database successfully and then forwards user to the matching step. The test cases for loading ontologies with their corresponding results are listed in table 5-2.

Test cases for matching ontologies step aims to verify whether the system could match ontologies with user-defined strategies, including selected matchers, combination method, filtering strategy and interrupt condition. The typical expectation result of this step is SAMBO processes ontology matching successfully with storing similarity results in database and generating a vector of suggestions as computation results and moves to the validating step to display candidate suggestions to user. The test cases with their corresponding results are listed in table 5-3.

### Table 5-2 Test cases and results for loading ontologies

<table>
<thead>
<tr>
<th>Test case</th>
<th>Scenario</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>User chooses ontologies in local file system and clicks “Upload”.</td>
<td>Ontology structure and database collection is related.</td>
</tr>
<tr>
<td>2</td>
<td>User inputs web addresses of ontologies and clicks “Upload”.</td>
<td>Ontology structure and database collection is related.</td>
</tr>
<tr>
<td>3</td>
<td>User chooses ontologies on the server and clicks “Upload”.</td>
<td>Ontology structure and database collection is related.</td>
</tr>
</tbody>
</table>

### Table 5-3 Test cases and results for matching ontologies

<table>
<thead>
<tr>
<th>Test case</th>
<th>Scenario</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>User selects matcher EditDistance, sets Single threshold as 0.6 with Maximum-based combination and clicks “Start Computation”.</td>
<td>The system processes matching, stores similarities in database and goes to suggestion validation page.</td>
</tr>
<tr>
<td>2</td>
<td>User selects matcher EditDistance and NGram, sets Single threshold as 0.6 with Weighted-sum combination and clicks “Start Computation”.</td>
<td>The system processes matching, stores similarities in database and goes to suggestion validation page.</td>
</tr>
</tbody>
</table>

Test cases for validating suggestions step mainly verify whether the system could get the relationship of validated suggestions from user. The expectation result of this step is SAMBO obtains relationship, name and comment for a validated suggestion, filters remaining suggestions with current validated suggestion’s relationship and then displays next candidate suggestions to user. The test cases with their corresponding results are listed in table 5-4.
Test cases for generating merged alignment file step mainly verify whether the system could provide an alignment file to user after validating all candidate suggestions. The test cases with their corresponding results are listed in table 5-5.

<table>
<thead>
<tr>
<th>Test case</th>
<th>Scenario</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>User inputs name and comment and clicks “Accept an Equivalence Relation” when validating a suggestion. User inputs name and comment and clicks “Accept a Sub-concept Relation” when validating a suggestion.</td>
<td>The system records information for current accepted suggestion, filters remaining candidate suggestions containing the same concept and displays next candidate suggestion to user. The system records information for current accepted suggestion and displays next candidate suggestion to user.</td>
</tr>
<tr>
<td>2</td>
<td>User clicks “Reject” when validating a suggestion.</td>
<td>The system displays next candidate suggestion to user.</td>
</tr>
</tbody>
</table>

5.2.2 Non-functional testing

Due to the fact that reducing runtime is the key point of the purpose of this project as well as the improvement doesn’t modify logic of matchers, the non-functional testing only focuses on runtime of the matching process of the improved SAMBO. The tested ontology pairs are listed in table 5-6. Test cases 1 and 2 consist of small-size ontologies which have been used as test cases in [4]. Test case 3 contains medium-size anatomy ontologies [45, 46]. Test case 4 includes two large scale ontologies [45] aiming at testing the ability of tackling large-scale ontology matching with the optimization design. To exclude the influence of combination method and filtering strategy, all the tests are executed with weight-sum combination method and filtered with single threshold set as 0.6. To compare improvement effects, the tests are executed on the old system as well.

The new system implements improvements mainly on two aspects. One is achieving a new database interaction approach though replacing MySQL with MongoDB. The other one is utilizing parallelization technique to speed up the matching process. To evaluate
the improvement effect on each of these two aspects, non-functional testing is executed in two aspects.

Table 5-6 Test cases for non-functional testing

<table>
<thead>
<tr>
<th>Test case</th>
<th>Source Ontology</th>
<th>Target Ontology</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ear_MA.owl (78 concepts, 1 relation)</td>
<td>ear_MeSH.owl (39 concepts, 1 relation)</td>
</tr>
<tr>
<td>2</td>
<td>eye_MA.owl (113 concepts, 1 relation)</td>
<td>eye_MeSH.owl (45 concepts, 1 relation)</td>
</tr>
<tr>
<td>3</td>
<td>human.owl (3298 concepts, 1 relation)</td>
<td>mouse.owl (2737 concepts, 2 relations)</td>
</tr>
<tr>
<td>4</td>
<td>oaei2014_FMA_whole_ontology.owl (78988 concepts, 54 relations)</td>
<td>oaei2014_NCI_whole_ontology.owl (66724 concepts, 190 relations)</td>
</tr>
</tbody>
</table>

The first aspect aims to test the improvement effect of the new database interaction approach. Thus, the test is executed with no parallel computing strategy. Besides, the other precondition is assuming that there exists similarity results storing in the database. The matcher used in this test is EditDistance. The testing results are listed in table 5-7 (“Unacceptable time” in the test results represents the execution time will be more than one day and the same below).

Table 5-7 Test results for old and new database interaction approach

<table>
<thead>
<tr>
<th>Test case</th>
<th>Matcher</th>
<th>Execution time of matching process</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>No parallel computing + MySQL</td>
</tr>
<tr>
<td>1</td>
<td>EditDistance</td>
<td>16829ms</td>
</tr>
<tr>
<td>2</td>
<td>EditDistance</td>
<td>49629ms</td>
</tr>
<tr>
<td>3</td>
<td>EditDistance</td>
<td>Unacceptable time</td>
</tr>
<tr>
<td>4</td>
<td>EditDistance</td>
<td>Unacceptable time</td>
</tr>
</tbody>
</table>

The second aspect aims to test the improvement effect of parallel computing strategy. Since database is involved in SAMBO, the effect of parallel computing strategy is evaluated with database interaction and without database interaction, respectively. Thus, two sets of tests are processed. The first set is processed under the condition that no database interaction is involved, which means that SAMBO doesn’t query existing similarity results in database. The test results is listed in table 5-8. The second set executes matching process with the new optimized database interaction strategy. Complete similarity results are stored in the database before processing the tests. Test results for parallel computing strategy with database interaction are presented in table 5-9.
An additional set of tests is processed to evaluate the effect of database interaction strategy in ontology matching. Two matchers are selected to calculate similarities in this test. One is EditDistance, a simple matching algorithm, which can finish calculation in extremely short time. The other one is UMLSKSearch, which is a matcher achieving similarity results through getting access to online third-party resource UMLS Terminology Service. Test results are listed in table 5-10.

<table>
<thead>
<tr>
<th>Test case</th>
<th>Matcher</th>
<th>Execution time of matching process</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>No parallel computing + MongoDB (Without similarities in database)</td>
</tr>
<tr>
<td>1</td>
<td>EditDistance</td>
<td>44ms</td>
</tr>
<tr>
<td></td>
<td>UMLSKSearch</td>
<td>216812ms</td>
</tr>
<tr>
<td>2</td>
<td>EditDistance</td>
<td>36ms</td>
</tr>
<tr>
<td></td>
<td>UMLSKSearch</td>
<td>217748ms</td>
</tr>
</tbody>
</table>
Chapter 6 Discussion

In this section, system testing results are explained and discussed as well as the research method applied in this thesis project is discussed. In the end, some directions of future work to optimize SAMBO motivated by the improvement result and research process is presented.

6.1 Result Discussion

The core purpose of this project is to improve the performance of the session-based SAMBO, consequently enabling SAMBO to achieve the ability of matching large scale ontologies. To achieve this goal, efforts on database interaction optimization and parallelize matching process are made. To evaluate the results clearly, the results of both aspects will be explained and discussed respectively.

Test results listed in table 5-7 clearly show the improvement effect of optimized database interaction. The runtime of matching process decreases sharply after integrating MongoDB into SAMBO. This result is mainly achieved through four aspects: high-performance data access of MongoDB, new schema design aiming at reducing search space and the number of database query, the construction of a reasonable index and the optimization on business logic of matching process. Although the new database interaction improves the performance observably, it is not enough to achieve the goal of processing large scale ontologies.

Test results listed in table 5-8 and table 5-9 present the improvements achieved by parallelization strategy. No matter with or without database involved, parallel matching process decreases execution time successfully. In the case of processing matching without database access, the improved SAMBO gains the ability to match large scale ontology. Comparing the results in table 5-9 with the result in table 5-7, the improved SAMBO has decreased the runtime by over 90%. However, even with such a great improvement, the runtime of matching large scale ontologies with database, under the testing environment described in section 5.1, is unacceptable. Since the parallelization strategy shows its ability to improve the performance of SAMBO, running SAMBO on a
computer with more CPUs might complete large scale ontology matching with database interaction in acceptable time.

By comparison the results in table 5-8 and 5-9, it is obvious to see that database interaction still takes a large proportion of runtime when matching ontologies with simple matchers. This implies more efforts can be done to optimize database interaction. From the results in table 5-10, it is easy to observe that storing similarity results for simple matcher, such as EditDistance, doesn’t contribute to enhance the performance of SAMBO. While for those complex, time-consuming matcher, such as UMLSKSearch, storing results for reuse is effective. Thus, avoid querying similarity results for simple matchers might be a good way to enhance user experience.

6.2 Method Discussion

As mentioned in section 1.4, this thesis work started with a comprehensive literature study of related theory to achieve a solid knowledge base for the thesis topic. After learning of state of art, a general direction of improvements was made. The detailed analysis of the old system helped to master the general implementation of SAMBO and motivated the ideas on how to improve the system. The requirements of this project were gathered in a very simple way. The functional requirements were just to stay the same with those of the old system. The non-functional requirements mainly focused on reducing runtime of the matching process which were the core goal of this project. To achieve this goal, several alternative techniques were compared and evaluated during the design phase. The selection criteria were higher performance of processing large scale data and less running time. Since too much attention were paid on reducing time consumption, some disadvantages of the selected technique were neglected. For instance, integrating MongoDB into SAMBO did help to reduce the runtime, while huge space usage caused by MongoDB became another problem of the improved system. The technique selection method should be improved to consider more aspects and make a balanced decision.

During the implementation, the development went well as a whole based on the detailed designed described in section 4. Besides, lots of efforts were made to ensure thread safety when transforming a system developed for single thread to a parallel system. This procedure was achieved through reading existing codes line by line to find the codes
needed to be modified. This method were easy to miss some points and caused potential incorrectness of the results of computation, while a better method were not found to avoid this risk. In this project, the method to prove of the correctness of the improvements were to compare the number of suggestions generated by the new system with the number purposed by the old system. To make the results more convicive, a more throughout test to validate the effectiveness (such as precision, recall and F-measure) of the system should be done.

6.3 Future work

Although the improvements achieved in this thesis project have decreased the runtime of ontology matching, it is still impossible to align large scale ontologies with database interactions in acceptable time. More improvements need to be done to optimize the session-based SAMBO. From the literature study and improvement results, some ideas on how to improve SAMBO in the future are motivated.

● Reducing the scale of the entity pairs to be matched

All the improvements have been done aim to reduce the execution time of matching each entity pairs from the input ontologies. For large scale ontology matching, the scale of entity pairs are on the order of billions which is the root cause of the heavy work in ontology matching. Thus, applying some algorithms to reduce the number of entity pairs to be matched could be a valuable further research aspect to improve the performance of SAMBO.

● Parallel computing on distributed systems

This thesis project implements a parallel matching strategy to process computing for the session-based SAMBO. However, due to the computing capability of one computer is limited, the results cannot be achieved in short time. Currently, several parallel computing techniques on distributed systems for large scale data have been purposed and widely used, such as Hadoop and Spark. These techniques accelerate computing through integrating computing capabilities from a cluster. Deploying SAMBO on a distributed system and designing a suitable parallel strategy could be another valuable direction to optimize SAMBO in the future.
Chapter 7 Conclusion

The session-based SAMBO is an ontology alignment system with the characteristic of utilizing database to store similarity results for recording interrupted session and for reuse results in iterative alignment. The existing SAMBO is effective to align ontologies, while it loses its efficiency when the scale of ontologies increase. This thesis work is conducted with the purpose of designing and implementing effective approach to improve the performance of SAMBO for aligning large scale ontologies. To fulfill this purpose, two specific subquestions are proposed to guide the direction of the research.

1. How to optimize database design and interaction strategy to improve the performance of the session-based SAMBO?
2. How to use parallelization technique to improve the performance of the session-based SAMBO?

To answer the first subquestion, a detailed analysis for the old SAMBO is carried out to figure out the defects of current data management approach. As a result, the disadvantages of fixed database schema as well as redundant database interaction are listed. To implement the optimization, MongoDB with features of high performance data access and flexible schema was introduced into SAMBO. The design of new flexible data schema lightens the workload of data queries. Meanwhile, the workflow of business logic were optimized to decrease unnecessary database interaction. Besides, index of MongoDB were also optimized to accelerate query speed. From the testing results, the new design of data management are confirmed to be effective to improve the performance of SAMBO, while it is still insufficient to process large scale ontology with database interaction.

For the second subquestion, a thorough review of the old SAMBO is conducted to find which part of work in SAMBO can be executed in parallel. A redesign of the workflow of the matching process were performed with the consideration of balancing the workload on each threads to maximum benefits gained from parallel computing. All the code involved in the matching processed were reviewed to ensure no bugs caused by thread safety when running the system with new parallel workflow. The testing results
show that the new parallel strategy does contribute to improve the performance of SAMBO.

To sum up, both data management optimization and parallel computing optimization designed and implemented in this thesis project is effective to reduce runtime of the session-based SAMBO. The improved SAMBO is able to deal with large scale ontologies without database interaction. Although it fails to process alignment with database interaction in acceptable time, the improvement achieved in this project have reduced the runtime of matching process to a great extent according to the testing results. Since involving database in ontology alignment is the key feature of the session-based SAMBO, further work on other aspects to optimize the performance of processing large scale ontology matching with database interaction are expected.
References


