On Matching Large Life Science Ontologies in Parallel

Anika Groß[1,2], Michael Hartung[1,2], Toralf Kirsten[2,3], Erhard Rahm[1,2]

[1] Department of Computer Science, University of Leipzig
[2] Interdisciplinary Centre for Bioinformatics, University of Leipzig
[3] Institute for Medical Informatics, Statistics and Epidemiology, University of Leipzig
{gross, hartung, rahm}@informatik.uni-leipzig.de
tkirsten@izbi.uni-leipzig.de

Abstract. Matching life science ontologies to determine ontology mappings has recently become an active field of research. The large size of existing ontologies and the application of complex match strategies for obtaining high quality mappings makes ontology matching a resource- and time-intensive process. To improve performance we investigate different approaches for parallel matching on multiple compute nodes. In particular, we consider inter-matcher and intra-matcher parallelism as well as the parallel execution of element- and structure-level matching. We implemented a distributed infrastructure for parallel ontology matching and evaluate different approaches for parallel matching of large life science ontologies in the field of anatomy and molecular biology.

Keywords: ontology matching, matching performance, parallel matching

1 Introduction

Ontologies and their applications have become increasingly important especially in the life sciences [19, 5]. Typically they are utilized to semantically annotate molecular-biological objects such as proteins or pathways. For instance, the popular Gene Ontology (GO) [10] is the primary ontology for annotating proteins with information on the functions and processes they are involved in. Other life science ontologies, e.g., in the Open Biomedical Ontologies Foundry (OBO) [31] contain information about anatomical structures for different species (e.g., human, mouse, fly) or diseases. The increasing number and availability of different life science ontologies enables new types of analysis, experiments and applications.

Recently, the development and maintenance of ontology mappings interconnecting different (multiple) related ontologies have gained importance, e.g., to integrate heterogeneous information sources (e.g., [15]), to merge ontologies [18], or to support analysis such as the comparison of expression patterns [2]. Since the manual creation of such ontology mappings is time-consuming or even infeasible their semi-automatic generation called *ontology matching* [24, 9] has become an active research field especially for life science ontologies (e.g., [22, 4, 17, 26]).

Effective ontology matching, i.e. the computation of high quality mappings, typically entails the combined execution of several matchers to determine the similarity between ontology elements based on metadata or instance data (see [24, 9]). For large

ontologies these matchers are often very time-consuming and memory-intensive. This is because metadata-based matchers, e.g., comparing the names of ontology concepts, typically evaluate the Cartesian product of all element pairs leading to a quadratic complexity w.r.t. ontology size. The performance requirements are further multiplied by the number of different matchers or when applying ontology matching on multiple ontology versions [12, 32]. Ontology matching is also memory-intensive for large ontologies because matching is typically performed on memory representations (graph structures) of the ontologies and requires the maintenance of several similarity values for every element pair from the Cartesian product.

The results of previous OAEI contests [20] on matching anatomical ontologies have shown that systems need execution times of up to several hours. This is despite the fact that the considered ontologies are only of medium size of around 3,000 elements (Mouse Anatomy Ontology [13] with ~2,800 elements was matched against the anatomy part of the NCI Thesaurus [30] with ~3,300 concepts). The Cartesian product thus has about $9 \cdot 10^6$ element pairs to be evaluated. Larger ontologies lead to even higher resource requirements. For instance, matching the two sub ontologies Molecular Functions and Biological Processes of GO with 10,000 and 20,000 ontology concepts results in approx. $2 \cdot 10^8$ pairs to compare, i.e., 22 times more than in the OAEI match problem. The memory requirements just for the similarity values are in the order of several GB.

These examples illustrate that it is valuable to have a match system providing high-performance ontology matching especially for interactive (online) applications where fast response times are required or when multiple match configurations have to be evaluated. While improving ontology matching performance has received some attention recently (see Related Work section), to the best of our knowledge the parallel execution of ontology matching on multiple compute nodes has not been studied so far. However, the broad availability of multi-core systems and multiple computing machines makes parallel ontology matching very attractive. Partitioning a large match problem into smaller parallel match tasks also helps to reduce the memory requirements per task. We therefore study strategies for parallel ontology matching and make the following contributions in this paper:

- We propose different strategies for parallel ontology matching, in particular *inter-* and *intra-matcher parallelization*. While the former approach executes independent matchers in parallel, the latter performs an internal parallelization of matchers based on a partitioning of the ontologies to be matched. Both strategies can be combined for additional performance improvements.
- We show how different kinds of matchers (element-level, structure-level, instance-based matchers) can be parallelized.
- We implemented a distributed infrastructure for parallel ontology matching and evaluate different approaches for parallel matching of large life science ontologies in the field of anatomy and molecular biology. The results show the effectiveness and scalability for single matchers and complete match strategies.

The rest of the paper is organized as follows. In Section 2 we introduce our ontology model and provide background information on ontology matching. Section 3

discusses inter- and intra-matcher parallelization and outlines how different matchers can be executed in parallel. The infrastructure for parallel ontology matching is presented in Section 4. We evaluate our approaches in Section 5 and discuss related work in Section 6. Finally, we summarize and outline possibilities for future work.

2 Preliminaries

We first introduce our ontology model. We then discuss the ontology matching problem and common match approaches.

2.1 Ontology Model

An ontology $O = (C, R)$ consists of concepts C which are interconnected by directed relationships in R. A special concept called *root* has no relationships to any parent. The directed relationships can be of different type. The most common relationship type in ontologies is '*is_a*' describing an inheritance between two concepts. Furthermore the '*part_of*' relationship type is used to model part-whole relationships between concepts. Life science ontologies use further semantic relationship types, e.g. '*regulates*'. We allow several parents and therefore several root paths per concept. The structural information (context) of concepts is used by structure-based match approaches to determine the concept similarity.

Furthermore, a concept $c \epsilon C$ of an ontology is defined by a set of single- or multi-valued attributes. For instance, the concept name is a single-valued attribute that is frequently used for ontology matching. Some ontologies (e.g., GO) support multi-valued synonym attributes containing alternate names for a concept. Usually there is also an identification attribute or accession number c_{acc}. These concept identifiers are used for annotating biological objects (proteins, genes, etc.) [11] and can be useful for instance-based ontology matching.

2.2 Ontology Matching

Ontology matching is the process of determining a set of semantic correspondences (ontology mapping) between concepts of two related ontologies O_1 and O_2. The correspondences are determined by matcher algorithms determining the similarity $sim(c_1,c_2) \epsilon [0\ldots1]$ between concepts $c_1 \epsilon O_1$ and $c_2 \epsilon O_2$. Matchers can roughly be classified into *metadata- or schema-based* and *instance-based* approaches [24]. Metadata-based matchers do not utilize instance data but focus on ontology information and optionally some background information such as dictionaries. Metadata-based matchers can be further classified into *element-level* and *structure-level* matchers. Element-level matchers utilize information from concept attributes, such as determining the similarity of concept names and synonyms, e.g., based on some string similarity such as ExactMatch, n-Gram or EditDistance. Element-level matchers are almost always used and combined with other approaches. *Structure-level matchers* consider the ontology structure for matching, e.g., to determine the context similarity of concepts.

Typical matchers evaluate the children, leaves, siblings and ancestors of concepts. In contrast, *instance-based matchers* do not depend on the ontology metadata but utilize existing associations between ontology concepts and instances and consider two concepts as similar if they share similar instances. One way to determine instance similarity is to measure the degree of instance overlap between concepts, e.g., based on a Dice or Jaccard measure. The complexity of matchers is usually quadratic by comparing all concepts of the first ontology with all concepts of the second ontology (evaluation of the Cartesian product).

A single matcher is typically not sufficient for high match quality so that one has to combine several matchers within a so-called match strategy or workflow. Match prototypes such as COMA++ therefore provide many matchers and support their flexible combination [1, 6, 7]. The matchers may be sequentially executed so that the results of a first matcher are refined by the following matchers. Alternatively, the matchers are independently executed and combined. Match workflows may use different methods to combine match results of individual matchers, e.g., by performing a union or intersection or by aggregating individual similarity values. The final match result is typically restricted to correspondences for which the similarity values exceed a predetermined threshold. In the next section, we discuss how such match strategies as well as single matchers can be parallelized.

3 Parallelization Strategies

In this section, we discuss possibilities of parallelizing ontology matching workflows consisting of several matchers that are either sequentially or independently executed. We assume that a computing environment of multiple locally interconnected multi-core computing nodes is available for matching.

A straight-forward approach to parallel ontology matching is *inter-matcher parallelism*, i.e., to process independently executable matchers in parallel on different cores or computing nodes. In addition, we want to support *intra-matcher parallelism*, i.e., the internal parallelization of individual matchers. Furthermore, we can combine both kinds of parallelism. In the following, we discuss these parallelization strategies in more detail. For intra-matcher parallelism (Section 3.2) we focus on the parallel similarity evaluation of the Cartesian product of concept pairs according to a partitioning of the input ontologies. In particular we will describe how we can parallelize element-level, structure-level and instance-based matchers.

3.1 Inter-matcher Parallelization

Inter-matcher parallelization enables the parallel execution of independently executable matchers to utilize multiple processors for faster match processing. The example match workflow in Figure 1a utilizes inter-matcher parallelization for n matchers (M_1, …, M_n). The match results can be combined by different aggregation and selection strategies to achieve the final result. Ideally, the inter-matcher parallelization improves the execution time by a factor n if the matchers are of similar complexity. This kind of parallelism is easy to support and can utilize multiple cores of a

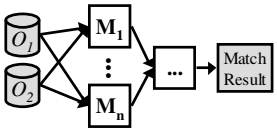


Figure 1a: Inter-matcher parallelization

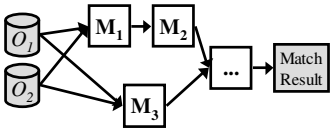


Figure 1b: Combination of inter-matcher parallelization and sequential matching

single computing node or multiple nodes. However, inter-matcher parallelization is limited by the number of independently executable matchers. Furthermore, matchers of different complexity may have largely different execution times limiting the achievable speedup (the slowest matcher determines overall execution time). Moreover, the memory requirements for matching are not reduced since matchers evaluate the complete ontologies.

The degree of parallelism is also limited for sequential matcher execution (e.g., if a structure-level matcher depends on a previously executed element-level matcher) or when the number of available processors is smaller than the number of independently executable matchers. As illustrated in Figure 1b, in such cases inter-matcher parallelism can be applied for a subset of matchers. The shown example assumes that only two cores can be utilized and that the most complex matcher M_3 is assigned to one core while M_1 and M_2 are executed sequentially on the other core.

3.2 Intra-matcher Parallelization

Intra-matcher parallelization deals with the internal decomposition of individual matchers or matcher parts (e.g., tokenization of concept names) into several match tasks that can be executed in parallel. We focus on a general approach to support intra-matcher parallelism based on partitioning the input data (the ontologies). Such a partitioning is very flexible and scalable and can be used to generate many match tasks of limited complexity. Furthermore, intra-matcher parallelism can be applied for sequential as well as independently executable matchers, i.e., it can also be combined with inter-matcher parallelism.

Figure 2 illustrates intra-matcher parallelization for n matchers that are executed in parallel (i.e., in combination with inter-matcher parallelism). For each matcher the input ontologies are first partitioned followed by the generation of multiple match tasks $M_{i1}, \ldots, M_{ik}$ $(i = 1, \ldots, n)$. These match tasks are executed in parallel, the union

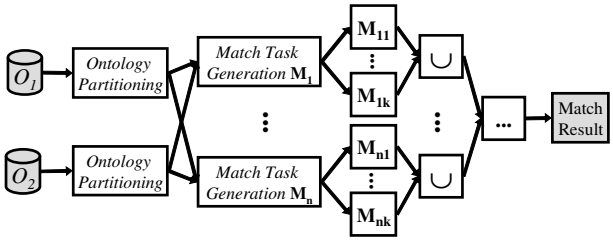


Figure 2: Intra-matcher parallelization

of the match task results gives the complete match result. In the example, all match tasks of the n matchers can be concurrently executed on the available compute nodes to achieve a maximal reduction of the execution time. Note that the match tasks only match partitions of the two ontologies and have thus reduced memory and processing requirements compared to a complete matcher. Hence, intra-matcher parallelization is especially promising for matching large ontologies.

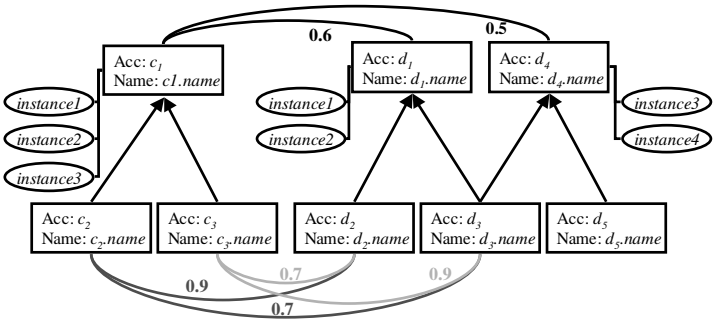
Before we discuss how we can parallelize element-level, structure-level and instance-based matchers we first outline our approach for *ontology partitioning*. In this initial study of parallel ontology matching we focus on a simple but yet flexible size-based approach that enables the parallel matching of the Cartesian product of the concepts from the two input ontologies O_1 and O_2. To generate match tasks of similar complexity we partition both ontologies into partitions of equal size (number of concepts); the partition size is a parameter that can be chosen according to the size of input ontologies and the complexity of the utilized matcher. Each task matches one O_1 partition with one O_2 partition so that we generate $p_1 \cdot p_2$ match tasks for p_1 (p_2) equally sized partitions of O_1 (O_2). For instance, if we partition two ontologies of 10,000 concepts into 10 partitions each, we generate 10·10=100 match tasks. As we will discuss in Section 4, generated match tasks are managed in job queues from where they are scheduled for parallel execution.

This size-based ontology partitioning has significant advantages besides its simplicity: (1) it is scalable to large ontologies by choosing manageable partition sizes and thus enables unproblematic processing and reduced memory requirements per match task, (2) it supports good load balancing because of equally sized partitions and match tasks, (3) it helps optimizing performance without sacrificing match quality since the full Cartesian product is evaluated, and (4) it can be utilized for element-level, structure-level and instance-based matchers as we will discuss in the following.

3.2.1 Parallelization of Element-level Matchers

To parallelize element-level matching approaches based on the introduced size-based partitioning is relatively easy. This is because element-level matchers compare ontology concepts with each other by utilizing metadata from the concepts themselves, i.e., their attribute values such as the name or synonyms. By partitioning the ontologies into subsets of concepts we retain the information needed for matching the concepts. Hence, element-level matchers can easily be applied to ontology partitions.

Figure 3 shows a running example for matching two ontology parts $c_1, \ldots, c_3 \in O_1$ and $d_1, \ldots, d_5 \in O_2$. As shown, concept c_1 has two children c_2 and c_3. The concept d_3 of O_2 is assumed to have two parent concepts d_1, d_4 (multiple inheritance). Some concepts have associated instances that will be considered later for instance-based matching. We assume that the concepts should be matched with each other by a string-based name matcher. The name matcher evaluates the string similarity (e.g., TriGram) for all (3·5=15) concept pairs. The result set (shown on the right of Figure 3) contains six correspondences with similarities ranging from 0.5 to 0.9; all other concept pairs are assumed to have similarity 0, i.e., they do not match.



$corr(c,d)$	$sim_{Name}(c,d)$
$c_1 - d_1$	0.6
$c_1 - d_4$	0.5
$c_2 - d_2$	0.9
$c_2 - d_3$	0.7
$c_3 - d_2$	0.7
$c_3 - d_3$	0.9

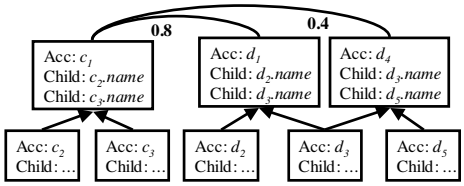
Figure 3: Element-level matching on *Name* attribute

3.2.2 Parallelization of Structure-level Matchers

Structure-level matchers are more difficult to parallelize than element-level matchers since they utilize information from the structural context or neighborhood of concepts (e.g., children, parents, siblings) or even the whole ontology. Hence, an ontology partition consisting of a certain number of concepts does generally not provide all information needed for structure matching. Even more difficult is the parallelization of iterative structural matchers such as Similarity Flooding [21] that start with initial element-level similarities and iteratively propagate these along the concept relationships across the whole ontologies. For such matchers parallelization is inherently difficult and has likely to be restricted to the initial element-level matching.

We therefore focus on structural matchers that utilize information from a restricted neighborhood (local context) of concepts. To limit the resource and memory requirements we do not want the match tasks to work on the whole ontologies but to restrict them to input partitions of restricted size similar to parallel element-level matching. This can be achieved by extending the concept-level information, within special multi-valued *context attributes*, by information from the local context that is needed for structure-level matching. The values for these context attributes, e.g., *Child*, *Parents*, *NamePath,* are determined in a preprocessing step by traversing the input ontologies once (linear effort) to collect the necessary context information about children, parents, etc. Concepts with these additional context attributes can then be partitioned as for element-level matching. Each match task performs structure matching for a pair of partitions utilizing information from the context attributes.

Figure 4 illustrates the context attribute approach for a Children matcher for our running example of Figure 3. The matcher determines the similarity between two concepts by calculating the average element (e.g., name) similarity between their children, i.e. it takes the sum of the name similarities between any two children and divides by the total number of child pairs. Note that this is only one possibility to compute the children similarity, used for illustration. For the example of Figure 4, we obtain that c_1 is more similar to d_1 than to d_4 as c_1 and d_2 share more similar children (using the similarity values of Figure 3). To execute this matcher we use a multi-valued *Child* context attribute for each (non-leaf) concept and populate it during the preprocessing step, in our case with the name values of child concepts. A child match task matches each concept c of an O_1 partition with each concepts d of an O_2 partition



$corr(c,d)$	$sim_{Children}(c,d)$
$c_1 - d_1$	(0.9 + 0.7 + 0.7 + 0.9) / (2·2) = 0.8
$c_1 - d_4$	(0.7 + 0 + 0 + 0.9) / (2·2) = 0.4

Figure 4: Attribute-based child matching

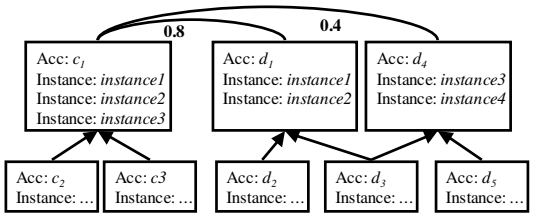
by merely comparing all *Child*-attributes of c with all *Child*-attributes of concept d w.r.t. their string (name) similarity and dividing it by number of possible child pairs: $sim_{Children}(c,d) = \sum_{i,j} sim_{Name}(c.child_i, d.child_j) / (|c.child| \cdot |d.child|)$.

The context attribute approach can similarly be applied for other local context matchers such as Parents, Siblings or NamePath. For instance, to realize the NamePath matcher we determine a concept's predecessors in a root path and store their concatenated names in a multi-valued *NamePath* context attribute during preprocessing. Matching is then similar to name element-matching but uses the *NamePath* attribute and its structural information about the names of the predecessor concepts. In previous evaluations [7], NamePath was shown to be one of the most effective single matchers so that it is valuable to have a parallel implementation of it.

3.2.3 Parallelization of Instance-based Matchers

Finally, we discuss how instance-based matching approaches can be parallelized. One common approach evaluates the instances associated to ontology concepts and considers two concepts as similar if they largely share similar instances [17]. Since instances are directly associated to concepts, we can determine the concept similarity using concept-specific information. This allows us to apply a similar parallelization strategy as for local-context structure matching and element-level matching.

As illustrated in Figure 5 instances are mapped to a multi-valued attribute *Instance* during preprocessing. For example, *Instance* may contain the accessions of biological objects associated to a GO concept. Size-based partitioning is applied to the input ontologies and the associated instances. A Dice-based measuring of the instance overlap similarity [17] would count the common *Instance* attribute values N_{cd} of two concepts $c \epsilon O_1$, $d \epsilon O_2$, and compute the similarity $sim_{Dice}(c,d) = 2 \cdot N_{cd} / (N_c + N_d)$ where N_c (N_d) is the number of instances of concept c (d). In our example (Figure 5) the match result contains two correspondences with a higher similarity for concepts c_1-d_1 sharing more common instances than c_1-d_4.



$corr(c,d)$	$sim_{Dice}(c,d)$
$c_1 - d_1$	2·2 / (3+2) = 0.8
$c_1 - d_4$	2·1 / (3+2) = 0.4

Figure 5: Attribute-based instance-based matching

4 Infrastructure for Parallel Ontology Matching

To execute ontology matching workflows in parallel we have implemented a distributed and service-based infrastructure illustrated in Figure 6. It consists of several services including a central *workflow service*, a *data service*, and multiple *match services* that are implemented in Java. These services run on different loosely coupled servers or workstations. While the workflow service coordinates the execution of the complete match workflow, match services compute the ontology mapping for two ontologies or ontology partitions. The data service manages all ontology and instance data forming the input of a match workflow and stores the final ontology mapping as result. The data service implements the repository schema proposed in [16] to efficiently store ontology and mapping versions.

Match applications (e.g., matching tools such as COMA++ [1]) use the workflow service to centrally access the match infrastructure. We assume that these applications configure a concrete match workflow, i.e., they specify the ontologies and instance data (or versions of both) as input data as well as utilized matchers and steps to preprocess the matcher input and to post-process match results (e.g., ontology partitioning and mapping manipulations including union and majority as well as filtering). Within this specification the match and manipulation steps are interconnected such that the workflow defines which matchers can be executed in parallel (inter-matcher parallelization) or in sequential order. The workflow service takes this configuration as input and processes the specified match workflow.

The workflow service performs ontology preprocessing if necessary, in particular for determining the values of context attributes for structure-level matching (see Sec-

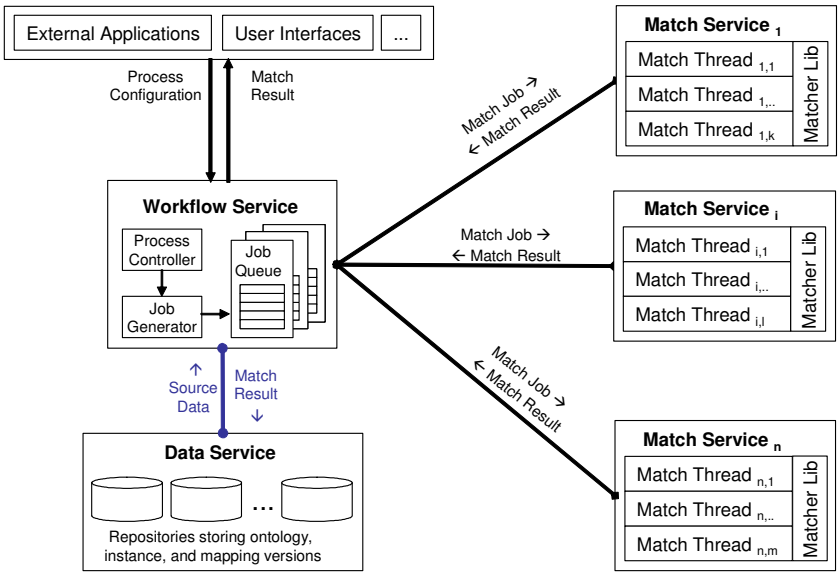


Figure 6: Distributed infrastructure for matching ontologies in parallel

tion 3.2). The workflow service executes the matchers in the workflow in the specified order for sequential matchers or in parallel. For this purpose, it maintains a job queue for each matcher. For intra-matcher parallelism the workflow service generates all match tasks and stores them in the matcher-specific job queue. Without intra-matcher parallelization the job queues consist of only a single match(er) job. The workflow service sends the queued match jobs to available match services as long as there are unprocessed jobs available. The match services execute the jobs and send their results back to the workflow service which unifies the partial match results. For efficiency reasons, the match jobs are restricted by a similarity threshold so that they only return concept correspondences exceeding the minimal similarity.

The match services run on dedicated nodes to fully exploit their compute power. Each match service contains several concurrently working *match threads* executing the match jobs (one thread per job at a time). The number of match threads per service can vary according to the number of available cores on the node. Hence, the infrastructure can cope with heterogeneously configured computing environments, i.e., servers and workstations with different number of cores and speed can be used for the proposed infrastructure. The match threads obtain their input data (ontology partitions) from the match job and execute the specified matcher implementation from a comprehensive *matcher library*.

5 Evaluation

We used the ontology matching infrastructure to evaluate the proposed parallelization strategies. We first describe the evaluation setup, in particular the considered ontologies and matchers. In Section 5.2 we show results for parallel matching on a single multi-core node. We then analyze the scalability of parallel ontology matching on multiple compute nodes.

5.1 Evaluation Setup

We use up to four nodes for running match services each consisting of four cores, i.e., we utilize up to 16 cores. Each node has an Intel(R) Xeon(R) W3520 4x2.66GHz CPU, 4GB memory and runs a 64-bit Debian GNU/Linux OS with a 64-bit JVM. We use 3GB main memory (heap size) per node. The workflow and data services run on additional nodes.

In our experiments we consider a medium-scale as well as a large-scale match problem. For the medium-scale problem we match the AdultMouseAnatomy (MA) (2,737 concepts) with the anatomical part of the NCI Thesaurus (NCIT) (3,289 concepts) as in the OAEI 2009 contest. The large-scale match problem computes an ontology mapping between the two GO sub ontologies Molecular Functions (MF) and Biological Processes (BP) consisting of 9,395 and 17,104 concepts, respectively (versions of June 2009). For intra-matcher parallelism we use different partition sizes for the two match problems. For the medium-scale problem we set the maximum partition size to 500 concepts resulting in 6 (7) partitions for MA (NCIT) and thus 42

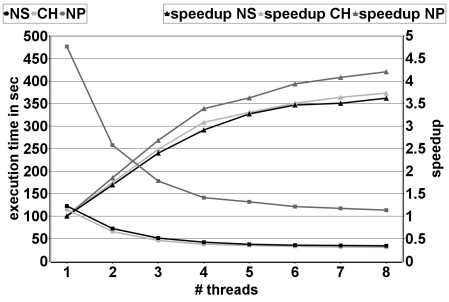


Figure 7: Intra-matcher parallelization on 1 node: medium-scale problem

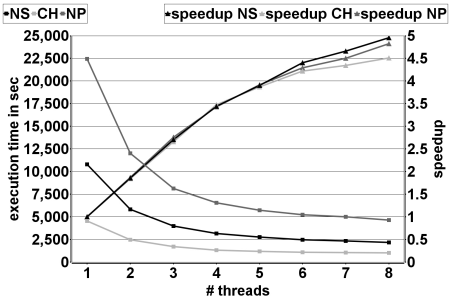


Figure 8: Intra-matcher parallelization on 1 node: large-scale problem

match tasks. For the large-scale match problem the max. partition size is set to 1,500. Hence, MF (BP) is split into 7 (12) partitions which results in 84 match tasks.

In this first evaluation analysis we focus on element-level and structure-level matchers. We applied three different single matchers namely NameSynonym (NS), Children (CH) and NamePath (NP). NS determines the maximal TriGram similarity for the name (label) and multi-valued synonym attribute values between concepts. CH and NP use TriGram similarity on the context attributes *Child* and *NamePath*, respectively (see Section 3.2.2). NamePath is restricted to at most three ancestor levels including '*is_a*' and '*part_of*' paths. These matchers are evaluated individually as well as within combined match strategies. In this study we focus on evaluating the efficiency (execution times) and not the matching effectiveness (e.g., precision, recall). This is because our parallel match approaches only target efficiency but do not affect quality since we always evaluate the Cartesian product, e.g. when using size-based partitioning (as described in Section 3.2).

5.2 Individual Matcher Parallelization on a Multi-core Node

We first analyze intra-matcher parallelization of individual matchers (NS, CH, NP) on a single multi-core node. Figures 7 and 8 show the execution time and speedup results for parallelizing the three matchers for up to eight parallel match threads for the medium-scale and large-scale match problems, respectively. We observe that execution times can be significantly improved by increasing the degree of parallelism for all matchers and both match problems. The NP matcher with its long concatenated name strings is by far the most expensive matcher with about four times longer execution times than CH; for the large-scale problem it takes more than 6 hours without parallelism. For the medium-scale match problem NS and CH take about the same time while NS takes much more time for the large-scale problem. This is because GO has many synonyms per concept so that for every concept pair about 11 (instead of 3 in the medium-scale problem) comparisons have to be computed.

For all matchers we achieve excellent speedup values of up to 3.6-4.2 for the medium-scale problem and even 4.5-5 for the large-scale problem. For up to four threads (= number of cores) we achieve almost linear speedup (up to 3.5). Increasing the number of threads brings further improvements (especially for the large-scale prob-

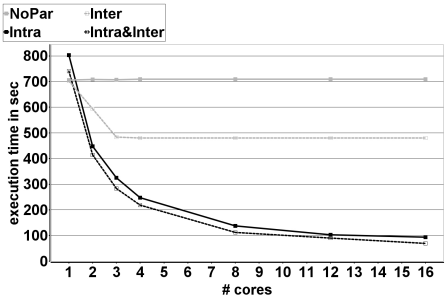


Figure 9: Parallelization strategies for medium-scale problem

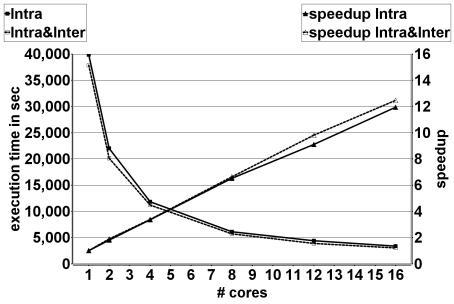


Figure 10: Intra-matcher parallelization strategies for large scale problem

lem) but at a reduced level. This is likely because the additional threads can utilize the cores when other match threads are waiting for new tasks to process.

5.3 Parallel Ontology Matching on Multiple Nodes

We now evaluate parallelization strategies using up to four compute nodes (16 cores) running up to four threads per node. In this experiment we combine the three individual matchers NP, CH and NS according to the following parallelization strategies: no parallelization *(NoPar*), inter-matcher parallelization (*Inter*), intra-matcher parallelization (*Intra*) as well as the combination of both intra- and inter-matcher parallelization (*Intra&Inter*).

Figure 9 shows the execution time results for these strategies on the medium-scale match problem. *NoPar* is the base case that does not benefit from multiple threads and cores. The other parallelization strategies lead to a performance improvement using more than one core. However, there are differences. *Inter* benefits only to a small degree since we do not apply intra-matcher but only inter-matcher parallelism. Since we apply three matchers we can only improve execution times for up to three cores/threads, i.e., multiple cores are not utilized for our match strategy. The total execution time is limited by the slowest matcher (NP). In contrast, *Intra* and *Intra&Inter* are very effective and achieve matching times of under 100 s. The combined *Intra&Inter* parallelization is slightly better than only using *Intra* and achieves a speedup of up to 10.6 (vs. 8.6). This is because *Intra* executes the three matchers sequentially resulting in some execution delays between matchers that are avoided for the combined approach.

Figure 10 shows the execution time and speedup results for the two parallelization strategies *Intra* and *Intra&Inter* for the large-scale match problem. Due to the large ontology sizes we omit the cases without intra-matcher parallelism and partitioning (*NoPar*, *Inter*). The sequential match time for the three matchers is 11h. Using 16 cores *Intra* and *Intra&Inter* reduce the overall execution time to 55 and 50 min and achieve thus an impressive speedup of 11.9 and 12.5, respectively. So, the speedup could be increased compared to the medium-scale match case, similar to the parallelization on a single node. This shows *Intra* and *Intra&Inter* are especially valuable for parallel matching of large ontologies.

6 Related Work

Matching life science ontologies has attracted considerable interest, particularly the matching of anatomy ontologies [22, 33] and molecular biological ontologies [4, 17, 26]. Typically, these studies aim at improving the quality of match results while efficiency aspects found only little attention. The performance of matching large schemas and ontologies in general is considered an open issue [3, 29]. In the past different algorithmic optimizations and fragmentation techniques for improving ontology as well as schema matching performance have been proposed.

Some approaches aim at reducing the search space compared to the Cartesian product for improved performance. Several divide-and-conquer approaches have been proposed where only parts of the input ontologies are matched against each other. [25,7] propose a fragment-based schema matching approach for COMA++ [1] where only similar fragments / sub-schemas need to be matched with each other. [14] partition entities of the input ontologies into sets of clusters and construct blocks which are matched based on pre-calculated anchors. The authors assess that the anchor pre-calculation consumes a main part of the overall runtime. The Anchor-Flood algorithm proposed in [28] also uses anchors (pairs of look-alike concepts) to gradually explore neighboring concepts in order to match only between ontology segments. In [27] nodes are clustered based on a linguistic label similarity and performance can be improved through minimization of the search space.

[23] propose a rule-based optimization technique to rewrite match strategies for improved performance. In particular, newly added filter operators allow a reduction of a matcher output and can thus speedup subsequently executed matchers. QOM [8] uses heuristics to reduce the number of candidate mappings to avoid the complete pair-wise comparison. These candidate mappings are classified into promising and less promising pairs by exploiting the ontological structures.

All these optimizations rely on algorithmic optimizations or partitioning/ fragmentation strategies to reduce the number of comparisons for improved performance. However, these approaches often lead to reduced match quality because relevant correspondences can be missed. Furthermore, the applicability of the approaches is dependent on the considered ontologies and match techniques. In contrast our parallelization strategies are orthogonal and general techniques to improve the performance of matchers and match strategies. They are especially valuable for large-scale match problems. We have shown their usefulness for evaluating the Cartesian product but they should also be usable in combination with other performance optimizations such as reduced search spaces.

7 Conclusion and Future Work

We propose general strategies for parallel ontology matching on multiple compute nodes, namely *inter-* and *intra-matcher parallelization* and their combination. They allow us to execute whole matchers in parallel and to parallelize matchers internally using data partitioning. For intra-matcher parallelism we propose a *size-based partitioning* enabling good load balancing, scalability and limited memory consumption

without reducing the quality of match results. We described how element-level, structure-level, instance-based matchers can be parallelized and use multi-valued *context attributes* for structural matching. We implemented a distributed infrastructure that enables parallel ontology matching and evaluated our approach for large life science ontology match problems. The results show the efficiency and scalability for single matchers as well as combined match strategies, especially for large match problems and for the combination of inter- and intra-matcher parallelism.

There are several opportunities for future work. Parallel ontology matching can be investigated for additional matchers. Furthermore, parallelization can be combined with algorithmic performance optimizations and advanced fragmentation strategies proposed in previous work. Moreover, parallel ontology matching may be extended to larger configurations such as cloud infrastructures.

Acknowledgments. This work is supported by the German Research Foundation (DFG), grant RA 497/18-1 ("Evolution of Ontologies and Mappings").

References

1. Aumueller, D., Do, H.H., Massmann, S., Rahm, E.: Schema and ontology matching with COMA++. In: Proc. of ACM SIGMOD Intl. Conference on Management of Data, pp. 906-908 (2005)
2. Bastian, F., Parmentier, G., Roux, J. et al.: Bgee: Integrating and Comparing Heterogeneous Transcriptome Data Among Species. In: Bairoch, A., Cohen-Boulakia, S., Froidevaux, C. (eds.) DILS 2008. LNCS (LNBI), vol. 5109, pp. 124-131. Springer Heidelberg (2008)
3. Bernstein, P.A., Melnik, S., Petropoulos, M., Quix, C.: Industrial Strength Schema Matching. ACM SIGMOD Record 33(4), 38-43 (2004)
4. Bodenreider, O., Burgun, A.: Linking the Gene Ontology to other biological ontologies. In: Proc. of 8th ISMB Meeting on Bio-Ontologies, pp. 17-18 (2005)
5. Bodenreider, O., Stevens, R.: Bio-ontologies: current trends and future directions. Briefings in Bioinformatics 7(3), 256-274 (2006)
6. Do, H.H., Rahm, E.: COMA – A System for Flexible Combination of Schema Matching Approaches. In: Proc. of the 28th Intl. Conference on Very Large Databases (VLDB), pp. 610-621 (2002)
7. Do, H.H., Rahm, E.: Matching large schemas: Approaches and evaluation. Information Systems 32(6), 857-885 (2007)
8. Ehrig, M., Staab, S.: QOM – Quick Ontology Mapping. In: Proc. of the 3rd Intl. Semantic Web Conference (ISWC), pp. 683-697 (2004)
9. Euzenat, J., Shvaiko, P.: Ontology Matching. Springer, 2007
10. The Gene Ontology Consortium: The Gene Ontology project in 2008. Nucleic Acids Research 36(Database issue), D440-D444 (2008)
11. Gross, A., Hartung, M., Kirsten, T., Rahm, E.: Estimating the Quality of Ontology-Based Annotations by Considering Evolutionary Changes. In: Paton, N.W., Missier, P., Hedeler, C. (eds.) DILS 2009. LNCS (LNBI), vol. 5647, pp. 71-87. Springer Heidelberg (2009)
12. Hartung, M., Kirsten, T., Rahm, E.: Analyzing the Evolution of Life Science Ontologies and Mappings. In: Bairoch, A., Cohen-Boulakia, S., Froidevaux, C. (eds.) DILS 2008. LNCS (LNBI), vol. 5109, pp. 11-27. Springer Heidelberg (2008)

13. Hayamizu, T.F., Mangan, M., Corradi, J.P. Kadin, J.A., Ringwald, M.: The Adult Mouse Anatomical Dictionary: a tool for annotating and integrating data. Genome Biology 6(3), R29 (2005)
14. Hu, W., Qu, Y., Cheng, G.: Matching large ontologies: A divide-and-conquer approach. Data & Knowledge Engineering 67(1), 140-160 (2008)
15. Jakoniene, V., Lambrix, P.: Ontology-based integration for bioinformatics. In: Proc. VLDB Workshop on Ontologies-based techniques for Databases and Information Systems (ODBIS), pp. 55-58 (2005)
16. Kirsten, T., Hartung, M., Gross, A., Rahm, E.: Efficient Management of Biomedical Ontology Versions. In: Meersman, R., Herrero, P., Dillon, T.S. (eds.): On the Move to Meaningful Internet Systems Workshops. Proceedings. LNCS, vol. 4544, pp. 172-187. Springer (2007)
17. Kirsten, T., Thor, A., Rahm, E.: Instance-based matching of large life science ontologies. In: Cohen-Boulakia, S., Tannen, V. (eds.) DILS 2007. LNCS (LNBI), vol. 5109, pp. 11-27. Springer Heidelberg (2008)
18. Lambrix, P., Edberg, A.: Evaluation of ontology merging tools in bioinformatics. In: Proc. of the 8th Pacific Symposium on Biocomputing, pp. 589-600 (2003)
19. Lambrix, P., Tan, H., Jakoniene, V., Strömbäck, L.: Biological Ontologies. In: Semantic Web: Revolutionizing Knowledge Discovery in the Life Sciences, pp. 85-99 (2007)
20. Ontology Alignment Evaluation Initiative: http://20.ontologymatching.org/
21. Melnik, S., Garcia-Molina, H., Rahm, E.: Similarity Flooding: A Versatile Graph Matching Algorithm and Its Application to Schema Matching. In: Proc. of the 18th Intl. Conference on Data Engineering (ICDE), pp. 117-128 (2002)
22. Mork, P., Bernstein, P.A.: Adapting a Generic Match Algorithm to Align Ontologies of Human Anatomy. In: Proc. of the 20th Intl. Conference on Data Engineering (ICDE), pp. 787-790 (2004)
23. Peukert, E., Berthold, H., Rahm, E.: Rewrite Techniques for performance Optimization of Schema Matching Processes. In: Proc. 13th Intl. Conference on Extending Database Technology (EDBT), pp. 453-464 (2010)
24. Rahm, E., Bernstein, P.A.: A survey of approaches to automatic schema matching. VLDB Journal 10(4), 334-350 (2001)
25. Rahm, E., Do, H.H., Massmann, S.: Matching large XML schemas. ACM SIGMOD Record 33(4), 26-31 (2004)
26. Rance, B., Gibrat, J.F., Froidevaux, C.: An Adaptive Combination of Matchers: Application to the Mapping of Biological Ontologies for Genome Annotation. In: Paton, N.W., Missier, P., Hedeler, C. (eds.) DILS 2009. LNCS (LNBI), vol. 5647, pp. 113-126. Springer Heidelberg (2009)
27. Saleem, K., Bellahsene, Z., Hunt, E.: PORSCHE: Performance ORiented SCHEma mediation. Information Systems 33(7-8), 637-657 (2008)
28. Seddiqui, H., Aono, M.: An efficient and scalable algorithm for segmented alignment of ontologies of arbitrary size. Web Semantics: Science, Services and Agents on the World Wide Web 7(4), 344-356 (2009)
29. Shvaiko, P., Euzenat, J.: Ten challenges for ontology matching. In: Proc. of On the Move to Meaningful Internet Systems (OTM), pp. 1164-1182 (2008)
30. Sioutos, N., de Coronado, S., Haber, M.W. et al.: NCI Thesaurus: A semantic model integrating cancer-related clinical and molecular information. Journal of Biomedical Informatics 40(1), 30-43 (2007)
31. Smith, B., Ashburner, M., Rosse, C. et al.: The OBO Foundry: coordinated evolution of ontologies to support biomedical data integration. Nature Biotechnology 25(11), 1251-1255 (2007)

32. Thor, A., Hartung, M., Gross, A., Kirsten, T., Rahm, E.: An evolution-based approach for assessing ontology mappings - A case study in the life sciences. In: Proc. Conference of the Business, Technology and Web (BTW), pp. 277-286 (2009)
33. Zhang, S., Bodenreider, O.: Aligning Representations of Anatomy using Lexical and Structural Methods. In: Proc. of AMIA Annual Symposium, pp. 753-757 (2003)