Application of High Performance Computing Techniques to the Semantic Data Transformation

José Antonio Bernabé-Díaz[1,3(✉)], María del Carmen Legaz-García[2], José M. García[3], and Jesualdo Tomás Fernández-Breis[1]

[1] Departamento de Informática y Sistemas, Universidad de Murcia, IMIB-Arrixaca, 30100 Murcia, Spain
{joseantonio.bernabe1,jfernand}@um.es

[2] Biomedical Informatics and Bioinformatics Platform, Fundación para la Formación e Investigación Sanitarias de la Región de Murcia, IMIB-Arrixaca, Calle Luis Fontes Pagán, n 9, 30003 Murcia, Spain
mcarmen.legaz@ffis.es

[3] Departamento de Ingeniería y Tecnología de Computadores, Universidad de Murcia, 30100 Murcia, Spain
jmgarcia@um.es

Abstract. The growth of the Life Science Semantic Web is illustrated by the increasing number of resources available in the Linked Open Data Cloud. Our SWIT tool supports the generation of semantic repositories, and it has been successfully applied in the field of orthology resources, helping to achieve objectives of the Quest for Orthologs consortium. However, our experience with SWIT reveals that despite the computational complexity of the algorithm is linear with the size of the dataset, the time required for the generation of the datasets is longer than desired.

The goal of this work is the application of High Performance Computing techniques to speed up the generation of semantic datasets using SWIT. For this purpose, the SWIT kernel was reimplemented, its algorithm was adapted for facilitating the application of parallelization techniques, which were finally designed and implemented.

An experimental analysis of the speed up of the transformation process has been performed using the orthologs database InParanoid, which provides many files of orthology relations between pairs of species. The results show that we have been able to obtain accelerations up to 7000x.

The performance of SWIT has been highly improved, which will certainly increase its usefulness for creating large semantic datasets and show that HPC techniques should play an important role for increasing the performance of semantic tools.

Keywords: Semantic web · Data transformation
High Performance Computing

Á. Rocha et al. (Eds.): WorldCIST′18 2018, AISC 745, pp. 691–700, 2018.
https://doi.org/10.1007/978-3-319-77703-0_69

1 Introduction

Biomedical research is producing vast amounts of data, which are usually stored in databases. According to the 2017 Molecular Biology Database Update [7] there are more than 1500 biological resources. Such resources are usually represented in heterogeneous ways [3], which makes data retrieval and management hard for life scientists and complicates the realization of the Web of Data [2] pursued by the Semantic Web community. The growth of the Life Science Semantic Web is illustrated by the increasing number of resources available in the Linked Open Data Cloud, although the generation of semantic versions of biological datasets require to develop specific scripts for each resource as in Bio2RDF [1], showing limited possibility of reuse and standardization of the process.

An example is the biological subdomain of orthology, where the Quest for Orthologs (QfO) consortium[1] not only has realized of the potential of semantic web technologies for promoting data sharing and interoperability but also tries to standardize the process of generating RDF orthology datasets [13]. For this purpose, QfO has developed the Orthology Ontology [6] and has used the SWIT tool for implementing the standardized transformation process [6,9], which has been applied to a few orthology resources.

The SWIT method uses an ontology to drive the transformation process and to guarantee the logical consistency of the transformed dataset, and requires to perform a series of time-consuming tasks to obtain the final RDF dataset. Experiments performed with large orthology datasets show that the computational complexity of the SWIT method is linear with respect to the number of data instances to generate, but this may take longer than expected for some use cases. Consequently, investigating ways for accelerating the execution of SWIT transformation processes could contribute to increase the practical usefulness of SWIT for supporting the transformation of large datasets.

High-Performance Computing (HPC) has been identified as a key player for optimizing and accelerating scientific applications in biomedicine [4,5,8]. HPC is usually associated with supercomputers but HPC techniques can be applied in less powerful servers by exploiting the capability of parallelization and distributed computing of current servers, and even personal computers. HPC techniques have demonstrated to obtain extremely high gains in speed of biomedical applications and the acceleration factor depends on the degree of parallelization and distributed computing of a given application or problem.

In this paper we describe how SWIT has been re-engineering with the application of HPC techniques in order to reduce the time required for generating large RDF datasets. Re-engineering SWIT has required to modify the original transformation algorithm and to adapt it for distributing the workload in parallel processes. The importance of semantic web technologies and HPC is growing in the areas of bioinformatics and biomedical informatics and this work sheds light on how they can be combined for the progress of biomedical and computational sciences.

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

The paper is structured as follows: Sect. 2 presents the functionality of SWIT, along the optimizations performed on the tool. The results obtained in several datasets are described in Sect. 3. In Sect. 4 the conclusions are presented.

2 Methods

This section describes SWIT functionality and all the procedures used for its optimization and parallelization.

2.1 SWIT

SWIT transforms and integrates heterogeneous biomedical data with the purpose of generating open semantic repositories. Figure 1 shows the architecture of SWIT. The transformation approach requires the prior definition of mapping rules between an input schema and a OWL ontology. Next, we describe the SWIT inputs:

- Data instances: SWIT is able to process XML and relational data, from which RDF/OWL can be generated.
- OWL Ontology: It supplies the domain knowledge for the transformation. The ontology also contains constraints that are used for ensuring the generation of logically consistent data.
- Transformation rules: They define how the content of the input dataset is transformed into a semantic format. SWIT distinguishes between mapping rules and identity rules.

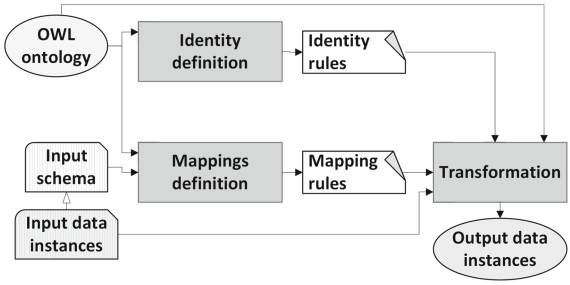

Fig. 1. SWIT architecture.

Mapping Rules. They provide the links between the entities of the input data schema and the entities of the OWL ontology. Next we provide an example of use of these rules to transform input data into the Orthology Ontology. Listing 1.1 shows the representation of the homology of two genes from *Escherichia coli* and *Nematocida parisii* in orthoXML format [12], which is a format used by various orthology resources. The mapping rule shown in Listing 1.2 is an entity rule that links the <*genes*> in the input data with the *Gene* class of the ontology. The XPath query defined in <*nodepath*> is processed by the *libxml* library[2] and SWIT transforms

[2] http://xmlsoft.org/.

each "id" attribute from the input file into an instance of the class *Gene*. For this example we get the individuals "gene_1" and "gene_2".

Additional mapping rules for transforming the datatype property *dct:identifier* of class *Gene* (attribute rule) and for the object property *sio:codifies* that links *Gene* and *Protein* classes (relation rule) produce the result shown in Listing 1.3.

```
<database name="UniProt" version="Comp_Proteomes_2013_06"
protLink="http://www.uniprot.org/uniprot/">
  <genes>
    <gene id="1" protId="P63284" geneId="clpB"/>
    <gene id="2" protId="I3EL83" geneId="NEPG_02529"/>
  </genes>
</database>
<groups>
  <orthologGroup id="1">
    <score id="bit" value="617"/>
    <geneRef id="1">
      <score id="inparalog" value="1"/>
      <score id="bootstrap" value="1"/>
    </geneRef>
    <geneRef id="2">
      <score id="inparalog" value="1"/>
      <score id="bootstrap" value="1"/>
    </geneRef>
  </orthologGroup>
</groups>
```

Listing 1.1. Example of an input use case

```
<map>
<type>Arch2Class</type>
  <class><id>http://purl.org/net/orth#Gene</id></class>
  <arch>
    <nodepath>/database/genes/gene/@id</nodepath>
  </arch>
</map>
```

Listing 1.2. Example of a mapping rule

```
<rdf:Description rdf:about="http://purl.org/net/orth#gene_1">
  <dct:identifier rdf:datatype="string">
    clpB
  </dct:identifier>
  <sio:codifies rdf:resource="http://purl.org/net/orth#
      protein_1"/>
  <rdf:type rdf:resource="http://purl.org/net/orth#Gene"/>
</rdf:Description>
```

Listing 1.3. Output example

Identity Rules. The input data might contain redundant entities that might be considered different due to heterogeneity of the sources and the definition of the mappings. For example, the application of the basic SWIT rules would generate two different individuals ("gene_1" and "gene_2") for the Listing 1.4 if the transformation is driven by the "id" property, despite the fact that they refer to the same gene. SWIT provides identity rules for preventing such situation. They define the set of datatype properties and object properties that permit to distinguish each individual in the ontology. The Listing 1.5 shows a rule which states that a gene is equivalent to another one if they have the same value for the property identifier (case insensitive checking). In this example, it should be noted that http://purl.org/dc/terms/identifier is mapped to the OrthoXML geneId attribute and not to "id".

```
<gene id="1" protId="P63284" geneId="clpB"/>
<gene id="2" protId="p63284" geneId="clpb"/>
```

Listing 1.4. Two alike genes

```
<class><id>http://purl.org/net/orth#Gene</id></class>
<and>
  <requirement>
    <scope>ALL</scope>
    <dataproperty>
      http://purl.org/dc/terms/identifier
    </dataproperty>
    <value>EQUALS IGNORE CASE</value>
  </requirement>
</and>
```

Listing 1.5. Identity rule

2.2 Optimizations

The experience in the application of SWIT to the transformation of large datasets has revealed that despite the computational complexity increases linearly with the number of entities to be transformed, it is slower than expected, and we have identified some limitations to the performance: (1) use of an interpreted language; (2) inefficient memory management; (3) the execution of identity rules using SPARQL queries create an execution bottleneck; and (4) the sequential execution of the transformation. In this section we describe the optimizations designed and implemented to overcome this limitations. The code modernization[3] will be based on three independent steps: Scalar Tuning, Memory and Threading. These steps are performed at the level of single-node.

Scalar Tuning. The first optimization has been to re-implement the SWIT kernel in C/C++, since compiled code languages provide faster executions than interpreted ones such as Java, which is experimentally demonstrated in [10].

[3] https://software.intel.com/en-us/articles/what-is-code-modernization.

Memory. SWIT creates ontology individuals as a result of the execution of the mapping rules, and they are frequently used by SWIT during the process when axioms have to be added to them. The original SWIT algorithm stores them using hash maps of vectors, which is not optimal. The renovated algorithm provides a faster access by using two hash maps hashed differently by integers, both maps are composed of pointers to individuals and not copies, so keeping the coherence. One of the maps grants that no failure happens when searching for an individual, if this exists, while the other map could miss in some searches since it acts as a greedy algorithm. The speed up of the execution of searches with the two maps is up to 2x.

The optimization of memory management also affects the process of identifying and merging equivalent individuals through the identity rules, which is one of the SWIT bottlenecks. The new method uses **two hash maps of vectors** where pointers to individuals are stored: a map for *AND* conditions and another one for *OR*s. The memory optimizations required the modification of the SWIT algorithm for the creation of RDF/OWL individuals. For example, in order to process an *AND* condition the pseudocode is:

```
checkDuplicate(Individual ind, Identity identityAND,
    IndividualMap andMap) {
 //Extract and concatenate the individual's properties
 string resume = extractProperties(ind, identityAND);
 int hash = doHash(propertiesConcat);
 vector<Individual> indVector = andMap[hash];
 //Individuals with the same hash
 for (int i = 0; i < indVector.size(); i++) {
  Individual coincidence = indVector[i];
  string coincidenceResume = extractProperties(
      coincidence, identityAND);
  if (coincidence.ontologyClass() == ind.ontologyClass
      () && coincidenceResume == resume) {
   ind.setCoincidence(coincidence);
   return;
  }
 }
 andMap[hash].add(ind);
}
```

The same is applied on the *OR* map, however instead of extracting and concatenating all the properties of the individual according to an *identityOR* rule (line 4), properties will be extracted and compared one by one.

For example, if we state that one gene is identified by the database **and** its gene id, for the *AND* map the value of the variable *resume* would be "database+gene_id" of the current individual. On the contrary, if we state that they are similar if the database **or** the gene id matches, then *resume* would contain "database" first, and if there are no coincidences for that certain database, then the algorithm would extract the next property "gene_id" and check if there is any duplicate.

Threading. This step provides a parallelization of the execution of SWIT and the largest speed up. The parallel design consists in setting one input file and one SWIT instance per core, so the parallelization only works when multiple files are established. When transforming a single large file, we need to split it in several smaller files to enable parallelization. We use the "gnu parallel" [14] software in order to carry out this task which allows us to parallelize at process level with a script.

3 Results

In this section we describe the results obtained by applying the renovated SWIT algorithm for the transformation of biomedical data. The results will be interpreted in terms of the acceleration obtained in comparison to the original algorithm. A demo version will be placed in a *git* repository[4], so reproducibility can be accomplished.

3.1 Experimental Setting

In this experiment we have used data from the orthology domain, which has been the domain in which SWIT has been recently used [6,9]. More concretely, we have used data from Inparanoid 8, which provides the data in XML[5]. Each XML file contains the pairwise orthologs between two given species identified by the InParanoid algorithm [11]. The Inparanoid files use the orthoXML format [12]. In this experiment we have used a total of 37436 orthoXML files corresponding to the pairwise orthology relations for two species and the complete InParanoid database:

- Hyaloperonospora arabidopsidis: 166 files, sizes between 167 KB and 3,1 MB
- Homo sapiens: 142 files, sizes between 168 KB and 5,6 MB
- The entire Inparanoid database: 37128 files (43G)

For the transformation of data we have used the Orthology Ontology[6] and the mapping file from orthoXML to the Orthology Ontology[7].

The experiments have been run in the following computers:
Personal computer (PC): It has an Intel® Core™ i5-6400 with 4 cores (4 threads, no hyper-threading) running up to 2,7 GHz and 8 GB RAM DDR4.
Mendel server (Mendel): It provides 2 chips of Intel® Xeon® E5-2698 v4 with 20 cores each (2 hyper-threading), making a total of 40 physical cores or 80 virtual cores, running at 2,2 GHz and 128 GB RAM DDR4.

[4] https://Neobernad@bitbucket.org/Neobernad/swit-test.git.
[5] http://inparanoid.sbc.su.se/download/8.0_current/Orthologs_OrthoXML/.
[6] http://purl.bioontology.org/ontology/ORTH.
[7] http://sele.inf.um.es/swit/ortho/mappingsOrthoXML.xml.

3.2 Results of the Experiment

The results of the transformation of the *H.arabidopsidis collection* are shown in Tables 1 and 2. The times are measured using the two different architectures described in Sect. 3.1. 'Java' stands for the original SWIT implementation, 'C' stands for the C version of SWIT, and 'Parallel' stands for the parallelization of the C version of SWIT. The column 'N.I' stands for the time without identity rules, that is to say, SWIT executions do not apply the identification of equivalent individuals shown in Sect. 2.1. On the contrary, the column 'W.I' (with identities) shows time executions making use of the detection of equivalent individuals. The most important column in these experiments is 'W.I', because it shows the optimization of the bottleneck of the SWIT Java version, and how it was reduced in the new version. The column 'N.I' also shows an execution time boost due to the general optimizations performed (i.e. code re-implementation from Java to C/C++, greedy algorithm/search).

In this collection the parallelized and optimized SWIT enhances the speed up to 4000x, from an execution time of 11626,296 s (≈3 h and 10 min) down to 2,78 s.

Table 1. Time table

Time (seconds)	N.I	W.I
Java	138, 238	11626, 296
C	78, 683	78, 9
C, Mendel	31, 446	31, 946
Parallel C, Mendel	2, 705	2, 786

Table 2. Speed up table

Speed up from Java, PC	N.I	W.I
C	1, 756x	147, 354x
C, Mendel	4, 396x	363, 935x
Parallel C, Mendel	51, 1x	4173, 114x

Next, the results for the *H.sapiens collection* are shown in Tables 3 and 4. This collection is one of the largest datasets found in InParanoid database. In order to process its 142 files, the personal computer required of 30111,894 s (≈ 8 h and 20 min) to finish the execution of SWIT Java. Running on the same architecture, PC, a speed up of 254x was achieved by using SWIT C/C++ (without parallel execution). The acceleration was improved even further when the optimized SWIT was processed in the server, reaching an increase of 7862 times faster than the original version, and decreasing the time from 30111,894 to just 3,8 s.

Table 3. Time table

Time (seconds)	N.I	W.I
Java	175, 378	30111, 894
C	116, 486	118, 271
C, Mendel	41, 415	42, 951
Parallel C, Mendel	3, 721	3, 830

Table 4. Speed up table

Speed up from Java, PC	N.I	W.I
C	1, 505x	254, 6x
C, Mendel	4, 234x	701, 07x
Parallel C, Mendel	47, 13x	7862, 114x

Finally, we tested SWIT with the whole InParanoid database which holds up to 274 species and a total of 37128 files. The executions were processed only in the server, using the detection of duplicated individuals (identity rules) for both runs. The results are displayed in Tables 5 and 6.

Table 5. Time table

Time (seconds)	W.I	Time (hours)
Java, Mendel	3310920	919
Parallel C, Mendel	3450	0, 95

Table 6. Speed up table

Speed up from Java, Mendel	W.I
Parallel C, Mendel	959, 68x

The original version of SWIT (Java) required of 919 h and 43 min (38 days) to transform the entire InParanoid database. Nevertheless, after the optimization and parallelization step, we have increased the speed up to 959x, taking less than 1 h to process the same amount of data.

4 Discussion and Conclusions

HPC techniques can be useful for accelerating semantic applications such as SWIT, which uses OWL ontologies and reasoning for generating RDF/OWL datasets. The methods developed in this work have obtained accelerations of SWIT executions up between 1000x and 7000x. This means that transformation processes which could take even one day with the original version would only need seconds to complete with the new method.

SWIT not only supports the generation of semantic repositories but is also able to generate integrated repositories by exploiting identity conditions which permit to identify equivalent individuals in different resources, which can then be merged or linked through *owl:sameAs* axioms. The application of the identity rules has also been optimized as a result of this work, so we are also optimizing the semantic integration of datasets, although further tests with an integration goal are needed.

It should also be noted that we have focused on the optimization of the SWIT kernel, which means that some post-processing activities such as the generation of "pure" RDF from OWL content has not been studied so far. We are also in the process of migrating the online version of SWIT and creating the web services that will permit third-party applications to use the new version of the algorithm.

These results are a clear justification for applying HPC to support semantic web tools, especially in biomedicine, which require the processing of large volumes of data. Speeding-up the generation of logically consistent, ontology-driven semantic datasets could also ensure that the latest version of the resources are rapidly available for the community. For example, we believe that this will help to increase the productivity of normalization efforts such as the one pursued in the Quest for Orthologs consortium.

Acknowledgements. This work has been partially funded by to the Spanish Ministry of Economy, Industry and Competitiveness, the European Regional Development Fund (ERDF) Programme and by the Fundación Séneca through grants TIN2014-53749-C2-2-R and 19371/PI/14.

References

1. Belleau, F., Nolin, M.A., Tourigny, N., Rigault, P., Morissette, J.: Bio2RDF: towards a mashup to build bioinformatics knowledge systems. J. Biomed. Inform. **41**(5), 706–716 (2008)
2. Bizer, C.: The emerging web of linked data. Intell. Syst. IEEE **24**(5), 87–92 (2009)
3. Bodenreider, O., Stevens, R.: Bio-ontologies: current trends and future directions. Brief. Bioinf. **7**, 256–274 (2006)
4. Bourne, P.E., et al.: Biomedicine as a data driven science. In: National Data Integrity Conference-2015, Colorado State University. Libraries (2015)
5. Carriero, N., Osier, M.V., Cheung, K.H., Miller, P.L., Gerstein, M., Zhao, H., Wu, B., Rifkin, S., Chang, J., Zhang, H., White, K., Williams, K., Schultz, M.: A high productivity/low maintenance approach to high-performance computation for biomedicine: four case studies. J. Am. Med. Inf. Assoc. **12**(1), 90–98 (2005)
6. Fernández-Breis, J.T., Chiba, H., Legaz-García, M.D.C., Uchiyama, I.: The orthology ontology: development and applications. J. Biomed. Semant. **7**, 34 (2016)
7. Galperin, M.Y., Fernndez, X.M., Rigden, D.J.: The 24th annual nucleic acids research database issue: a look back and upcoming changes. Nucleic Acids Res. **45**(D1), D1–D11 (2017)
8. Hautaniemi, S., Laakso, M.: High-performance computing in biomedicine. In: 2013 International Conference on High Performance Computing and Simulation (HPCS), p. 233. IEEE (2013)
9. Legaz-García, M.D.C., Miñarro-Giménez, J.A., Tortosa, M.M., Fernández-Breis, J.T.: Generation of open biomedical datasets through ontology-driven transformation and integration processes. J. Biomed. Semant. **7**, 32 (2016)
10. Magalhães, G.G., Sartor, A.L., Lorenzon, A.F., Navaux, P.O.A., Beck, A.C.S.: How programming languages and paradigms affect performance and energy in multithreaded applications. In: 2016 VI Brazilian Symposium on Computing Systems Engineering (SBESC), pp. 71–78. IEEE (2016)
11. O'brien, K.P., Remm, M., Sonnhammer, E.L.: Inparanoid: a comprehensive database of eukaryotic orthologs. Nucleic Acids Res. **33**(suppl-1), D476–D480 (2005)
12. Schmitt, T., Messina, D.N., Schreiber, F., Sonnhammer, E.L.: Letter to the editor: Seqxml and orthoxml: standards for sequence and orthology information. Brief. Bioinf. **12**(5), 485–488 (2011)
13. Sonnhammer, E.L., Gabaldón, T., da Silva, A.W.S., Martin, M., Robinson-Rechavi, M., Boeckmann, B., Thomas, P.D., Dessimoz, C., et al.: Big data and other challenges in the quest for orthologs. Bioinformatics (2014) btu492
14. Tange, O.: GNU parallel - the command-line power tool. The USENIX Mag. **36**(1), 42–47 (2011)