A practical experience concerning the parallel semantic annotation of a large-scale data collection

Javier Fabra, Sergio Hernández, Pedro Álvarez,
Aragón Institute of Engineering Research (I3A)
Computer Science and Systems Engineering
Universidad de Zaragoza, Spain
jfabra,shernandez,alvaper@unizar.es

Estefanía Otero, Juan Carlos Vidal, Manuel Lama
Centro Singular en Investigación en Tecnoloxías da Información (CITIUS)
Universidade de Santiago de Compostela, Spain
estefanianatalia.otero,juan.vidal@usc.es
manuel.lama@usc.es

ABSTRACT

From a computational point of view, the semantic annotation of large-scale data collections is an extremely expensive task. One possible way of dealing with this drawback is to distribute the execution of the annotation algorithm in several computing environments. In this paper, we show how the problem of semantically annotating a large-scale collection of learning objects has been conducted. The terms related to each learning object have been processed. The output was an RDF graph computed from the DBpedia database. According to an initial study, the use of a sequential implementation of the annotation algorithm would require more than 1600 CPU-years to deal with the whole set of learning objects (about 15 millions). For this reason, a framework able to integrate a set of heterogeneous computing infrastructures has been used to execute a new parallel version of the algorithm. As a result, the problem was solved in 178 days.

Categories and Subject Descriptors

C.2.4 [**Computer Systems Organization**]: COMPUTER-COMMUNICATION NETWORKS—*Distributed Systems* ; D.1.3 [**Software**]: PROGRAMMING TECHNIQUES—*Concurrent Programming, Parallel Programming* ; I.7 [**Computing Methodologies**]: DOCUMENT AND TEXT PROCESSING; K.3 [**Computing Milieux**]: COMPUTERS AND EDUCATION

General Terms

Performance, Experimentation

Keywords

Semantic annotation, Grid and cluster computing, Workflow technologies, DBpedia, Learning objects

ISEM '13, September 04 - 06 2013, Graz, Austria
 ACM 978-1-4503-1972-0/13/09 ... $15.00.
http://dx.doi.org/10.1145/2506182.2506192

1. INTRODUCTION

The Universia repository is a library of learning objects (LOs) whose aim is providing a single point for the access, diffusion, and reuse of millions of public educational resources that belong to collections of different institutions and countries. LOs provide the means to get these objectives, and, in fact, the provision of their metadata, usually in standardized formats, such as IEEE LOM [22] and Dublin Core [12], are the key of their portability and reusability. These metadata formats usually describe the potential contexts in which LOs may be used, and range from information about authorship, educational, to even technical properties. Nowadays, the Universia repository is composed of 213 collections of LOs in different languages and domains, totalling 15,750,979 academic resources.

All available resources of Universia have been automatically harvested from the source collections and are represented according to the IEEE LOM standard. Furthermore, resources are classified using the UNESCO nomenclature [28]. Although the aforementioned standards provide the structural means for a better access and reuse of LOs, some problems have been recently detected mainly motivated by (i) the data quality and (ii) the harvesting procedure, which is also in charge of indexing these data in the repository. In fact, we have identified that many metadata were not included after the harvesting process, difficulting thus the correct indexing of these resources. In most of these cases, the data were even not included in the source files, although in others, this information was lost simply by not being properly defined in the source file, or by not being correctly defined the harvesting process itself. As a consequence, many LOs were incorrectly categorized by the process in charge of creating the LOM instance, or even were never categorized because they did not fit any of the UNESCO classes. As result of this information loss, users cannot easily get the most suitable LOs through the classification or retrieval systems of Universia.

In order to overcome part of these difficulties, the aim of this paper is (i) to improve the categorization of the LOs and (ii) at the same time implement an infrastructure able to support this task. In a first stage, we defined an approach to annotate semantically the available LOs and use these annotations to provide a second classification mechanism, alternative to UNESCO classification, and based on

the instances contained in the DBpedia repository [4]. DBpedia selection was mainly motivated by two aspects: on the one hand, its semantic nature, DBpedia is an RDF-based repository, facilitating thus the use of semantic reasoning techniques to improve the annotation process; on the other hand, because its content is based on Wikipedia, a reliable and cross-domain source of information, whose categories perfectly fits features needed to classify Universia LOs. The proposal consisted of identifying the set of keywords of each LO and, then, annotating each keyword by means a graph of the DBpedia created from instances that were relevant in its domain. These resulting graphs are used to classify the LO (more detail about this approach are available in [21]). Although the results of this procedure were very promising in terms of precision and recall, the computational cost of annotating all LOs stored into the repository was unfeasible. A preliminary estimation concluded the use of a sequential implementation of the annotation algorithm required more than 1640 years of CPU to create the graphs (3 levels of depth, that is, the length of the path between the root node and any other node of the graph is 3) of Universia LOs. If 4 levels of exploration were computed, the estimated CPU time would see an increase of up to 25,000 years.

Also, we developed a resource management framework for the flexible deployment and execution of scientific workflows in a previous work [7]. This framework integrated a set of heterogeneous computing infrastructures at our disposal (more specifically, Grid, cluster and multi-cluster infrastructures) in order to create a more powerful execution environment. Its interface encapsulated the capabilities needed to manage the complexity and heterogeneity of the integrated computing resources and also the life-cycle of the applications that were executed over these resources. From a programming point of view, the framework was a powerful tool for the execution of applications with high computational and data storage requirements.

In this paper, we integrate our two previous works in order to generate efficiently semantic graphs of Universia LOs. For exploiting the computation capabilities of our resource management framework, a new parallel version of the annotation algorithm has been programmed and deployed on the framework. The programmed application makes optimum use of the available computing resources in order to solve the annotation problem in a reasonable time. Initially, three specific computing infrastructures were used: the HERMES cluster hosted by the Aragón Institute of Engineering Research (I3A) [1] and two research and production Grids hosted by the Institute for Biocomputation and Physics of Complex Systems (BIFI) [14], namely AraGrid [2] and PireGrid [26]. Integrating these resources, the annotation of 15 million LOs was solved in 178 days for three levels of exploration. Therefore, this paper recounts a successful practical experience of taking advantages of distributed computing to solve complex computational problems in the field of semantics.

The remainder of this paper is organized as follows. Section 2 presents the algorithm used to annotate the Universia repository semantically and an initial estimation of its computational cost. Section 3 describes the implementation of a new parallel version of the annotation algorithm and the resource management framework used for its deployment and execution. Results obtained when all computing infrastructures were integrated in order to solve the annotation process are shown in Section 4. Following this, the most relevant approaches for the integration of distributed computing resources and other practical experiences on the large-scale semantic annotation are revised in Section 5. Finally, Section 6 points out the conclusions and the future work of the paper.

2. SEMANTIC ANNOTATION OF THE UNIVERSIA REPOSITORY

The concept of LO represents an attempt to enhance the design of educational contents and offers a new conceptualization of the learning process that provides smaller, self-contained, and re-usable units of learning.

Universia is a repository that integrates many collections of LOs and provides the infrastructure to reuse these LOs. In order to facilitate the access to its contents, in theory, Universia categorizes LOs based on the UNESCO classification. In practice, this is not the case and LOs are in consequence difficult to retrieve.

The semantic annotation of LOs is a good solution to *(i)* complement the UNESCO classification with instances of the ontology and *(ii)* improve the retrieving of LOs thanks to the semantics included in these instances: each relation of an instance describes a specific property with semantics and whose target node can be either data, such as a string or a date, or another instance that is itself described by other properties. This configures a graph-based structure through which the user could easily navigate to get more information about a specific resource.

However, the success of this approach clearly depends on the selected ontology. If the ontology does not cover the topics of the LOs, they will not be classified. In this paper, we use DBpedia linked data to annotate the LOs. Firstly, because it is a general purpose ontology that covers most of the topics of Universia LOs. Secondly, because of its reliability and richness of content.

2.1 Graph-based annotation

The semantic annotation performed in this paper goes further than traditional approaches. Most semantic annotation solutions annotate the terms with *an instance* of the ontology [8, 23, 3, 18]. However, if the semantics are used to facilitate the retrieval, this approach has several disadvantages. On the one hand, although data and relations to other instances may help to get better results, the cost of exploration of large-sized ontologies requires an important computational effort each time a query is performed. On the other hand, general purpose ontologies need an additional filtering effort. For example, if the LO is about *ancient Egypt*, information about contemporary Egypt must be filtered so as not to introduce additional noise into the queries.

Taking this into account, new approaches start to annotate each term with a graph [24, 10]. These approaches are based on the working hypothesis that a graph of instances provides richer semantics than *only one instance*. Furthermore, they

may solve the two aforementioned drawbacks: *(i)* the subgraph is created during the annotation process and thus does not affect the search process; and *(ii)* the graph is filtered and thus only contains relevant information to the context of the annotated document. To sum up, an effort is made during the annotation to improve the search performance. Notice that this effort directly depends on the depth of exploration, that is, on the number of relations between the root node and the deepest node. As the exploration goes deeper, the description becomes better, but the computational cost also increases.

Our solution [21] is also based on this same principle; that a graph of instances provides richer semantics, but differs in how the instances of the solution graph are selected and in the extent of the exploration. On the one hand, our approach considers the context of the LO, that is the keywords of the LO, to filter which instances will belong to the solution. Therefore, only instances related to these keywords are included in the solution. On the other hand, our approach explores the taxonomic relations, such as class/subclass relations, included in the DBpedia. Through these relations the search space is expanded with instances that are siblings, that is, those that have the same type, or even with higher relationship, such as parents or grandparents.

Algorithm 1 (*ADEGA*) shows our filtering algorithm that annotates the categories of the LO with a graph of DBpedia instances. The algorithm starts from the root node of the graph (n_s) and expands the first child node of the RDF search tree going deeper and deeper until it hits a node that has no children or the depth limit λ_d has been reached. Then *ADEGA* backtracks, returning to the most recent node it has not finished exploring.

Finally, *ADEGA* returns the graph λ that contains the relevant nodes for a given context Δ, starting from a node n_s. In the first call, ϕ_n, ϕ_c, ϕ_t are empty lists. These three lists are used to reduce the exploration of nodes that have already been processed. Specifically, ϕ_n is used to store the visited nodes, and the latter two, ϕ_c and ϕ_t, for the visited categories and class types, respectively.

The algorithm has two parts. In the first one, relations r of the processed node n_s are visited, while in the second one n_s is evaluated:

- We only process relations that have a cardinality below a predefined threshold δ_i (line 3). If the cardinality is over that threshold, we consider that this relation is too generic to provide relevant information.
- Each instance of the relation r is recursively processed and the result graph $ADEGA(n_t)$ is added to the solution λ (line 5).
- If the relation links the node with a category or a class type (lines 6–16 and 17–22, respectively), we also explore brother, parent, and child instances of this category/class. The threshold δ_i also applies since too much information penalizes the exploration both in terms of the quality of results and in computation time.

ALGORITHM 1: Graph Filtering Algorithm (ADEGA)

Input: Node n_s, visited nodes ϕ_n, visited categories ϕ_c, visited types ϕ_t, search context Δ.
Output: The graph solution λ which root node is n_s.
Data: Depth limit δ_d, Instance exploration limit δ_i, Relevance threshold δ_r.

```
1  add node n_s to the visited node list φ_n;
2  for each relation r such that r(n_s, n_t) do
3      if |r(n_s, _)| < δ_i then
4          for each node n_t such that r(n_s, n_t) and n_t ∉ φ_n do
5              add ADEGA(n_t) to λ;
6              if r = dcterm:subject then
7                  add node n_t to λ;
8                  add category n_t to φ_c;
9                  if |dcterm:subject(_, n_t)| < δ_i then
10                     for each node n_i such that
                       dcterm:subject(n_i, n_t) and n_i ∉ φ_n do
11                         add ADEGA(n_i) to λ;
12                 for each category c such that skos:broader(c, n_t)
                   or skos:broader of(n_t, c) and c ∉ φ_c do
13                     add category c to φ_c;
14                     if |dcterm:subject(_, c)| < δ_i then
15                         for each node n_i such that
                           dcterm:subject(n_i, c) and n_i ∉ φ_n do
16                             add ADEGA(n_i) to λ;
17             if r = rdf:type then
18                 for each class t such that rdfs:subClassOf(t, n_t)
                   or rdfs:subClassOf(n_t, t) and t ∉ φ_t do
19                     add category t to φ_t;
20                     if |rdf:type(_, t)| < δ_i then
21                         for each node n_i such that rdf:type(n_i, t)
                           and n_i ∉ φ_n do
22                             add ADEGA(n_i) to λ;
23 if rel(n_s, Δ) ≥ δ_r then
24     add n_s to λ;
```

2.2 Computational issues

ADEGA obtains very promising results in terms of precision and recall independently of the max level of exploration. However, the precision is higher as much as more levels are explored. In fact, ADEGA improves significantly the quality of results obtained by other similar algorithms, such as *DBpediaRanker* [24] or *Relfinder* [10], and independently of the number of levels explored (more details of this comparison are available in [21]).

However, this improvement is obtained at the expense of the computational cost of the annotation process. Each classification implies the filtering of a set of (sub)graphs of DBpedia, one for each relevant term, and since this ontology has a high density of relations, the annotation process becomes both computational and time expensive. Each additional level of exploration in ADEGA also increments exponentially the number of visited nodes. For instance, if we suppose that each instance of DBpedia has at least n relations, the jump from the level m to the level $m+1$, with $m >= 0$, implies visiting n^{m+1} additional nodes. Taking into account that the instances of the DBpedia have more than 40 relations, the exploration of more than 3 levels implies visiting several millions of nodes. This is also the consequence of exploring both type- and skos-based taxonomic relations (lines 9 to 22 of the algorithm): instances with a very specific type do not have much siblings, but if the type is more abstract the number of relatives increases.

Table 1: Some indicators of the sequential version of ADEGA. Values show the average cost for obtaining the corresponding graph of each keyword of a LO.

Variable	Value
Average nodes visited	248,035.02
Average nodes discarded	199,461.73
Average nodes processed	48,573.29
Average literals processed	44,165.53
Average URL processed	4,407.76
Average number of SPARQL queries	22,882.64
Mean time per query in ms.	9.91
Mean time of ADEGA (depth=3) in ms.	376,414.17

Table 1 shows the performance of ADEGA when annotating 10,000 LOs selected randomly. During this process, 79,628 keywords where identified and each one these keywords was annotated with a graph limited to 3 levels of depth. As shown in this table, ADEGA visits an average of 248K nodes and discards most of these nodes, that is, 80% of the explored nodes are discarded. This is very indicative data that emphasizes the need for a filtering process. This is also indicative of the quality for future uses of the annotations associated to the LOs.

Table 1 also shows the distribution of the processed nodes to be included in the solution. In this case, 91% of the nodes are literals, which are the most costly nodes to process because they are analysed using natural language processing (NLP) and semantic techniques (text-based relations go through a morphology, syntactic, and semantic analysis to determine the relevant words, named entities, synonyms, etc.). The remaining processed nodes are URIs, and require a SPARQL query to retrieve their child nodes. If we multiply the number of SPARQL queries performed by their mean time, we can deduce that 67% of the computational time of ADEGA is directly related to the accessing and querying the DBpedia repository. Summarizing, each graphs is filtered in an average of 376,414.17 ms. which means that each LO requires around 3 million ms., i.e. nearly 50 minutes, supposing a mean of 10 terms to annotate per LO. If we extrapolate this number to the 15 million LOs, more than a millennium would be required to annotate the Universia repository.

3. A SOLUTION FOR LARGE-SCALE COLLECTIONS

The sequential ADEGA algorithm reported execution times that are not suitable for annotating large-scale collections of LOs. For this type of collection we propose a solution based on the integration of different distributed computing paradigms (Grid, cluster and Cloud) and the use of parallelization techniques. On the one hand, a resource management framework for the flexible deployment and execution of scientific workflows is used in order to provide a powerful execution environment. This framework integrates a set of heterogeneous computing infrastructures at our disposal. On the other hand, a software application based on a new and more efficient version of the ADEGA algorithm has been programmed to be executed over the previous framework. Intuitively, the aim of this proposal is to make the best use of the available computing resources.

Let us now introduce an overview of the framework and a high-level description of the programmed application.

3.1 A framework to integrate heterogeneous computing resources

In [7, 11] we proposed a framework for the flexible deployment and execution of scientific workflows. Flexibility has been achieved at different levels: from a computing point of view, the framework is able to integrate heterogeneous computing infrastructures to create more powerful execution environments; from a programming point of view, workflows can be programmed independently of the computing infrastructures where related jobs will be executed using different high-level languages widely accepted by the scientific community; and, finally, from a configuration point of view, new functionalities can be dynamically added/removed to/from the framework in order to meet the different needs of each application and user.

An integration model based on a *message bus* is the key to achieving the flexibility of the proposed solution. More specifically, the cornerstone of the proposal is a bus inspired by the Linda coordination model [5]. This component provides an application interface (API) for sending and receiving messages in an asynchronous way, coding them as Linda tuples. The other system components offer their capabilities through the common bus, and collaborate by exchanging messages using it as the communication channel. This integration model has several advantages compared to other more traditional approaches: (1) a bus reduces the coupling between system components (they are connected by making use of an asynchronous message passing mechanism); (2) components can be dynamically added or removed without disturbing the execution of other existing components (to adopt new functionalities, for example); (3) a bus favours the scalability and distribution of the solution; and, finally, (4) a bus supports complex message exchange patterns (publish and subscribe mechanism, content-based message routing, etc.) that facilitate more flexible integration strategies.

In this communication model, framework components are not aware of other components connected to the message bus. Each message has assigned an exclusive tag to identify the receiver and each component identifies the messages addressed to it with that tag. Thus, components can be easily replicated to improve framework performance and reliability (they compete for the same message and the bus decides which component gets each message), or replaced to adopt new functionalities, change them or fix bugs.

Figure 1 shows the high-level system architecture which is composed of three different layers. At the top, the *User interface layer* is composed of the different programming tools that can be used to program scientific workflows (Taverna, Kepler, Pegasus, etc.). Resulting workflows are submitted to the framework for their execution. The components of the *Execution layer* are responsible for managing the workflow execution life-cycle. Internally, this layer is composed of the *message bus* and the components that provide the core functionalities. In order to provide this functionality, two types of components have been connected through the bus: *management components* and *mediators*. The first ones offer extra functionalities to enhance work-

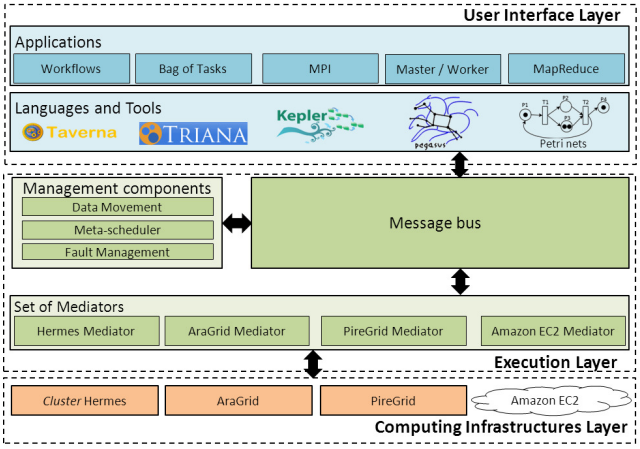

Figure 1: Architecture of the proposed framework for the execution of scientific workflows in multiple heterogeneous computing infrastructures.

flows life-cycle and framework capabilities (meta-scheduling, fault-tolerance, etc.). On the other hand, mediators encapsulate the heterogeneity of each specific computing infrastructure to facilitate its integration (in more detail, a mediator must interact with a specific infrastructure to submit jobs, move input/output data, monitor jobs execution, detect undesirable states, etc.). Finally, at the bottom of the architecture, the *Computing infrastructures layer* is formed by several heterogeneous computing infrastructures. At the beginning, three computing environments were integrated: the HERMES cluster hosted by the Aragón Institute of Engineering Research (I3A) [1], which is managed by the HTCondor middleware [13]; and two research and production grids managed by the gLite middleware [9] and hosted by the Institute for Biocomputation and Physics of Complex Systems (BIFI) [14], namely AraGrid [2] and PireGrid [26].

3.2 Overview of the available computing infrastructures

Let us now provide a brief description of the capabilities of each computing infrastructure integrated into our framework. Different computing paradigms (cluster, multi-cluster, and Grid computing) have been combined and managed in a manner which is transparent from the end user perspective.

HERMES is a cluster that consists of 126 heterogeneous computing nodes with a total of 1,308 cores and 2.92 TB of RAM. Additionally, computing nodes in HERMES are connected by Gigabit links, allowing high-speed data transfers. The cluster is used by a very large variety of researchers. Because of this diversity, jobs submitted to HERMES have very different characteristics, in terms of execution time, size of the input data, memory consumption, etc. System utilization is mainly CPU-intensive rather than memory intensive, and data inputs are usually small. Analysis of workloads shows that users are unaware of load peaks or advanced configuration issues, which normally means that experiments last an extremely long time or are queued excessively.

On the other hand, AraGrid consists of four homogeneous sites located at four different faculties in different geolocated cities in Spain (two sites in Zaragoza, one in Teruel and another one in Huesca). Each site is composed of 36 computing nodes with two 6-core 2.67 GHz processors and 24GB of RAM per node, making up an overall of 1,728 cores and 3.375 TB of RAM. Both nodes and sites are interconnected by Gigabit links. The Grid is managed by the gLite middleware. AraGrid infrastructure is oriented to long-term experiment in the fields of physics, biochemistry, social behaviour analysis, astronomy, etc. Users are more conscious of loads and resource usage, although they deploy experiments similarly to the HERMES case, with long waiting times.

And, finally, PireGrid is composed of four heterogeneous sites located at four different locations (two in Zaragoza, Spain, and another two in Toulouse and Pau, France). It consists of 600 cores and 2TB of RAM. Both sites and nodes are interconnected by 10 Gigabit links. The software configuration used in PireGrid is the same as in AraGrid, as both initiatives are managed by the BIFI.

3.3 A parallel application to annotate LOs

Intuitively, the model of software application that is executed by our framework consists of a large number of *jobs* (or units of computation) whose execution requires a high computation intensity and complex data management. Once the application has been deployed over the framework and its execution has begun, each job is mapped onto the specific computing resource to be executed. This mapping can be decided by the application programmer or by the scheduling components integrated into the execution framework.

The ADEGA computation consists of creating the semantic graph of a term from the DBpedia repository. Therefore, a job should manage the semantic annotation of one or more terms. The number of terms to be processed in a job is what is known as *granularity*. From the point of view of the available computing infrastructures, the granularity of the jobs depends on the computing resources where a specific problem will be executed. Some infrastructures are configured to efficiently execute small jobs, for instance. The execution of large-sized jobs may involve a great number of evictions and, consequently, high queue delays. As a part of this work we have experimentally defined the most suitable granularity for each available computing infrastructure. In HERMES, the optimum size of job was 100 terms per job. On the other hand, both ARAGrid and PireGrid showed better performance results when using 400 terms per job.

On the other hand, some relevant decisions regarding data management have been taken in order to improve performances of the parallel application. Firstly, a copy of the DBpedia database has been stored in each computing resource. This decision is possible owing to the fact that the application requires read-only access to the database. Secondly, an off-line process copied input data (more specifically, the terms to be semantically annotated) in the involved computing infrastructures before the beginning of the application execution (the size of these input data is more than 30 GBs). And, finally, once the application execution has finished, another off-line process is responsible for moving output data (the resulting RDF graphs and statistical results) from the computing infrastructures to an external data storage (their size is more than 120 GBs). Currently,

both off-line processes use the data movement capabilities of our framework to complete the copy of input and output data. Therefore, they could be programmed as a part of the application (an initial and final task, respectively).

Figure 2 shows the application workflow programmed to carry out the annotation process. We have implemented the workflow using the Reference nets and the Renew tools. Nevertheless, for the sake of simplicity, we show the programmed workflow using the graphical notation of Taverna.

Intuitively, the application receives the following input parameters: the host where the terms that must be annotated are stored (*Host* and *Context_dir*), the access credentials to access the host (*Access_credentials*) and, finally, the granularity of the jobs for each computing infrastructures (*Granularity*). The first task, *Get_input_data*, accesses the host specified as input parameter, and obtains a list of files that contain the terms and contexts to be annotated and the number of contexts (*File_list* and *Num_contexts* output parameters, respectively). Then, the *Get_number_jobs* task receives as input data the number of terms to annotate together with the desired granularity, it then processes them and generates as an output the number of jobs to execute (*Num_jobs* parameter). Both tasks have been programmed in Java and invoked by means of Java-based scripts. Once the *Get_number_jobs* task finished, the subworkflows that send these jobs to the framework (and get the results from their execution) are executed in parallel.

Internally, the *Send_jobs* subworkflow processes the list of the files with the terms and contexts to annotate, and performs a guided iteration in order to create the new JDSL job descriptions with the number of terms specified in the granularity parameter. It then invokes the Web service that sends jobs to our framework using the generated JDSL. The execution of a job consists of computing the graph of each of the input terms using a new parallel version of the ADEGA algorithm. Besides the parallelization work, the original algorithm has been slightly modified. Now, firstly, the root node of the semantic graph is directly calculated by the execution of a SPARQL query against the DBpedia datastore. Then, the original (sequential) ADEGA algorithm is used to compute in parallel the graph branches of a term. Obviously, the graphs of a list of terms (the terms of a concrete job, for instance) are also processed in parallel.

On the other hand, the *Get_job_results* subworkflow is in charge of retrieving the results of the different job submissions. Similarly to the *Send_jobs* workflow, this performs an iterative process in order to retrieve and verify the results of every job. Finally, the output parameters generated by both subworkflows (*Num_jobs_verified*, *Verification_results* and *Num_jobs_submitted*) contain the results of the verification of jobs and log data about their executions.

4. FINAL RESULTS: PROCESSING THE UNIVERSIA REPOSITORY

As was previously stated, all the infrastructures were integrated using the framework in order to solve the annotation problem. In this section, the results from the experimentation are presented. The aim was to annotate the nearly 150 million terms corresponding to the 15 million Universia LOs (each LO has about 10 terms). The whole set of terms was taken as an input for the programmed application, and some measures were analysed: the number of jobs executed in each individual computing infrastructure, the execution time per term in each infrastructure, fault management statistics, etc.

The annotation of the 149,427,907 terms was carried out in 4,260.83 hours (178 days). This computation required the three available computing infrastructures to be constantly processing jobs. As was previously stated, 400-terms per job were deployed in both AraGrid and PireGrid infrastructures, and 100-terms per job were used in HERMES. Specifically, 352,745 jobs were executed in HERMES (35,274,419 terms), 186,210 jobs were executed in AraGrid (74,483,721 terms) and, finally, 99,175 jobs were executed in PireGrid (39,669,767 terms).

In terms of speedup, from the user's perspective the integration approach improves the execution time of AraGrid, PireGrid, and HERMES 2.0, 3.6, and 3.8 times, respectively. Regarding job parallelization, HERMES got an average of 164 jobs processed in parallel, 116 for AraGrid and, finally, 68 jobs for PireGrid.

On the other hand, the time required to compute the graph corresponding to a term was also measured from both the system and the user's perspective. To get this, the clock was started at the moment a job was submitted to a specific computing resource by the middleware, and stopped when the job finished. Queue waits and other related times are not considered. The mean time required to compute a term in HERMES was 71.18 seconds (222.8 seconds from the user perspective). In the case of AraGrid, times are much better: 23.84 seconds and 41.56 seconds, respectively. PireGrid results were quite similar to those obtained by AraGrid, computing a term in 26.09 seconds and the average user's perspective time being 43.70 seconds per term. Regardless of the computing infrastructure, the mean time need to compute a term has been reduced with respect to the time obtained by the sequential ADEGA algorithm. This is due to the change introduced in the original version of the algorithm which was described above.

Finally, we found that 87.61% of the initial jobs finished at the first attempt (i.e., as they were deployed in a specific computing infrastructure). From the remaining jobs (failing jobs), the 6.97% finished after a first retry (using the same computing infrastructure in order to execute the failed job). From the remainder, 5.11% finished after a second retry (using an alternative computing infrastructure). These re-scheduling policies are internally implemented in the framework and, therefore, the user is not aware of job faults and their corresponding retries.

5. RELATED WORK

In this section, the most relevant and well-known approaches to integrating distributed computing resources are presented. To the best of our knowledge, these solutions have not been applied in the field of semantic annotation. Therefore, we also examine the use of distributed solutions to annotate large-scale data semantically.

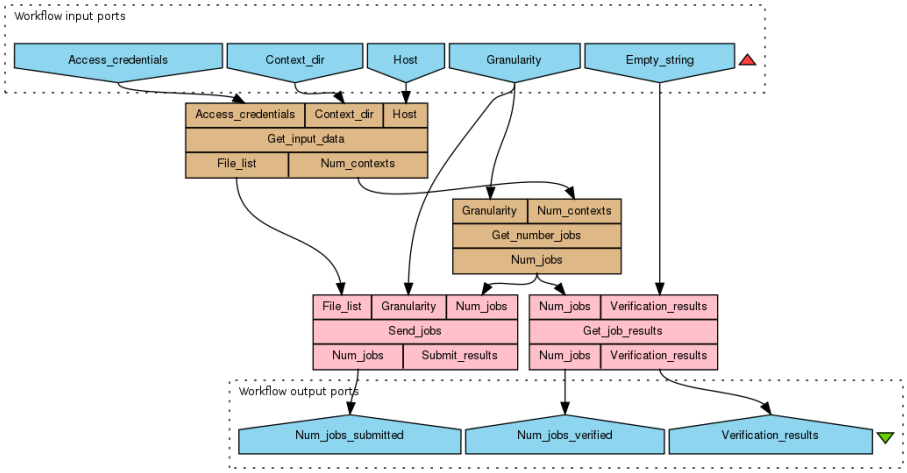

Figure 2: Implementation of the workflow using Taverna.

In the field of distributed computing, the challenge of integrating heterogeneous computing resources still remains open. The latest solutions are based on the use of resource management frameworks able to manage the execution of applications over a pool of distributed resources. The most relevant frameworks are GJMF [25], P-GRADE [15, 16], SWAMP [29], and GMBS [17]. In general, these solutions consist of a Web portal or a service-based interface that offers a transparent access to several Grid infrastructures. Internally, in order to manage the life-cycle of the applications, they are composed of a set of services for scheduling, fault-tolerance support or data movement, for instance. The main contribution of our framework with respect these solutions is the flexibility to integrate different types of resources (clusters, Clouds, etc.), to improve its functional framework capabilities, and to program applications independently of the available resources. A more detailed discussion about the differences between these works can be found in [7].

The previous approaches have usually been exploited in the field of scientific computing. Nevertheless, there are some practical experiences in the large-scale semantic annotation area based on the distributed computing paradigm, and focused on improving the performance of semantic annotations. In [20, 6], collections of Web documents were annotated using pattern-based annotation systems that run on a distributed architecture. In [20], Google's MapReduce architecture and a Hadoop-based implementation were used (this work is an evolution of a previous solution deployed over a Grid infrastructure and evaluated experimentally only on 5,000 documents [19]); meanwhile, in [6], the Seeker system and its data storages were deployed over a dedicated cluster. On the other hand, in [27], the GATE annotation and search tool was migrated to the Amazon Elastic Compute Cloud. GATECloud.net provides a service for annotating document collections using a cloud, being the execution of the annotation process internally managed. As a relevant conclusion, all these systems were programmed to execute in a specific execution environment and, therefore, they are highly coupled to the available computing resources. This is an important difference with respect to our approach, where applications are programmed independently of the available computing resources, and the framework optimizes their use.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we have conducted a specific problem in the field of large-scale semantic annotation. Workflow technologies were used to program a parallel application able to annotate learning objects from DBpedia instances. Unlike other existing approaches, this application has been implemented independently of the computing resources responsible for its execution, and then executed over a set of heterogeneous and distributed computing infrastructures. We have also proved the possibility of exploiting our resource management framework to execute semantic processes with high computational and data storage requirements. The service-based interface provides researchers with transparent access to a powerful and scalable execution environment. Our future work will also address the integration of cloud-based infrastructures (such as the Amazon EC2 cloud computing platform or a private cloud managed by OpenStack and hosted by the BIFI institute, for instance) into the framework, and will then evaluate its impact on the annotation process from a computational point of view.

Taking into account that Universia continuously increases and updates its resources, we also plan to improve the performance of the algorithm through some mechanism for indexing the information of DBpedia. In this sense, and although the current infrastructure has proven to significantly reduce the time needed to annotate LOs, we will study the impact of indexing the relations (particularly, text-based relations) or even directly indexing subgraphs of DBpedia nodes, reducing in this way significantly the SPARQL queries needed to access to the information, which is one of the flaws of the present version of the algorithm.

Acknowledgments

This work has been supported by the research projects TIN 2010-17905 and TIN2011-22935, granted by the Spanish Ministry of Science and Innovation, and the regional project DGAFSE 287-191/3, granted by the European Regional Development Fund (ERDF).

7. REFERENCES

[1] Aragón Institute of Engineering Research (I3A). http://i3a.unizar.es, 2013. Accessed 24 June 2013.

[2] AraGrid. http://www.aragrid.es/, 2013. Accessed 24 June 2013.

[3] S. Araújo, G.-J. Houben, and D. Schwabe. Linkator: Enriching web pages by automatically adding dereferenceable semantic annotations. In *the 10th International Conference on Web Engineering (ICWE 2010)*, volume 6189 of *Lecture Notes in Computer Science*, pages 355–369. Springer, July 2010.

[4] C. Bizer, J. Lehmann, G. Kobilarov, S. Auer, C. Becker, R. Cyganiak, and S. Hellmann. DBpedia: A crystallization point for the Web of Data. *Journal of Web Semantics*, 7(3):154–165, 2009.

[5] N. Carriero and D. Gelernter. Linda in context. *Commun. ACM*, 32(4):444–458, Apr. 1989.

[6] S. Dill, N. Eiron, D. Gibson, D. Gruhl, R. Guha, A. Jhingran, T. Kanungo, K. S. McCurley, S. Rajagopalan, A. Tomkins, J. A. Tomlin, and J. Y. Zien. A case for automated large-scale semantic annotation. *Web Semantics: Science, Services and Agents on the World Wide Web*, 1(1):115 – 132, 2003.

[7] J. Fabra, S. Hernández, P. Álvarez, and J. Ezpeleta. A framework for the flexible deployment of scientific workflows in grid environments. In *the Third International Conference on Cloud Computing, GRIDs, and Virtualization*, CLOUD COMPUTING '12, pages 1–8, 2012.

[8] A. Garcia, M. Szomszor, H. Alani, and Ó. Corcho. Preliminary results in tag disambiguation using DBpedia. In *the 1st International Workshop on Collective Knowledge Capturing and Representation (CKCaR 2009)*, September 2009.

[9] gLite Middleware. http://glite.cern.ch/, 2013. Accessed 24 June 2013.

[10] P. Heim, S. Hellmann, J. Lehmann, S. Lohmann, and T. Stegemann. RelFinder: Revealing relationships in RDF knowledge bases. In *Semantic Multimedia*, volume 5887 of *Lecture Notes in Computer Science*, pages 182–187. Springer, 2009.

[11] S. Hernández, J. Fabra, P. Álvarez, and J. Ezpeleta. A Simulation-based Scheduling Strategy for Scientific Workflows. In *the 2nd International Conference on Simulation and Modeling Methodologies, Technologies and Applications*, SIMULTECH '12, pages 61–70, 2012.

[12] D. Hillmann. Using dublin core. Technical report, Dublin Core Metadata Initiative, Mar. 2005. DCMI Recommended Resource.

[13] HTCondor Middleware. http://research.cs.wisc.edu/htcondor/, 2013. Accessed 24 June 2013.

[14] Institute for Biocomputation and Physics of Complex Systems (BIFI). http://bifi.es/en/, 2013. Accessed 24 June 2013.

[15] P. Kacsuk, G. Dózsa, J. Kovács, R. Lovas, N. Podhorszki, Z. Balaton, and G. Gombás. P-grade: A grid programming environment. *J. Grid Comput.*, 1:171–197, 2003.

[16] P. Kacsuk, T. Kiss, and G. Sipos. Solving the grid interoperability problem by P-GRADE portal at workflow level. *Futur. Gener. Comp. Syst.*, 24(7):744–751, 2008.

[17] A. Kertész and P. Kacsuk. GMBS: A new middleware service for making grids interoperable. *Futur. Gener. Comp. Syst.*, 26(4):542–553, 2010.

[18] G. Kobilarov, T. Scott, Y. Raimond, S. Oliver, C. Sizemore, M. Smethurst, C. Bizer, and R. Lee. Media meets semantic web: How the BBC uses DBpedia and linked data to make connections. In *the 6th European Semantic Web Conference (ESWC 2009)*, volume 5554 of *Lecture Notes in Computer Science*, pages 723–737. Springer, 2009.

[19] M. Laclavík, M. Ciglan, M. Šeleng, and L. Hluchý. Empowering automatic semantic annotation in grid. In *the 7th international conference on Parallel processing and applied mathematics*, PPAM'07, pages 302–311. Springer-Verlag, 2008.

[20] M. Laclavík, M. Šeleng, and L. Hluchý. Towards large scale semantic annotation built on mapreduce architecture. In *the 8th international conference on Computational Science, Part III*, ICCS '08, pages 331–338. Springer-Verlag, 2008.

[21] M. Lama, J. C. Vidal, E. Otero-García, A. Bugarín, and S. Barro. Semantic Linking of Learning Object Repositories to DBpedia. *Educational Technology & Society*, 15(4):47–61, 2012.

[22] Learning Technology Standards Committee. Draft standard for learning object metadata. Technical Report IEEE Standard 1484.12.1-2002, Institute of Electrical and Electronics Engineers, July 2002. Final Draft Standard.

[23] P. Mendes, M. Jakob, A. Garcia-Silva, and C. Bizer. DBpedia Spotlight: Shedding light on the Web of Documents. In *the 7th International Conference on Semantic Systems (I-SEMANTICS 2011)*, September 2011.

[24] R. Mirizzi, A. Ragone, T. D. Noia, and E. D. Sciascio. Semantic tag cloud generation via DBpedia. In *the 11th International Conference on E-Commerce and Web Technologies (EC-Web 2010)*, volume 61 of *Lecture Notes in Business Information Processing*, pages 36–48. Springer, 2010.

[25] P.-O. Östberg and E. Elmroth. GJMF - a composable service-oriented grid job management framework. *Futur. Gener. Comp. Syst.*, 29(1):144–157, 2013.

[26] PireGrid. http://www.piregrid.eu/, 2013. Accessed 24 June 2013.

[27] V. Tablan, I. Roberts, H. Cunningham, and K. Bontchev. Gatecloud.net: a platform for large-scale, open-source text processing on the cloud. *Philosophical Transactions of the Royal Society A*, 371(1983), 2013.

[28] United Nations Educational, Scientific and Cultural Organization (UNESCO). *Proposed International Standard Nomenclature for Fields of Science and Technology*, Mar. 1988. Accessed 24 June 2013.

[29] Q. Wu, M. Zhu, Y. Gu, P. Brown, X. Lu, W. Lin, and Y. Liu. A distributed workflow management system with case study of real-life scientific applications on grids. *J. Grid Comput.*, 10:367–393, 2012.