Published at Journal of Grid Computing March 2013, Volume 11, Issue 1, pp 149-166

# Searching for translated plagiarism with the help of desktop grids

**Máté Pataki, Attila Csaba Marosi**
Computer and Automation Research Institute
MTA SZTAKI
H-1518 Budapest, P.O. Box 63, Hungary
{mate.pataki, atisu}@sztaki.hu

## Abstract

Translated or cross-lingual plagiarism is defined as the translation of someone else's work or words without marking it as such or without giving credit to the original author. The existence of cross-lingual plagiarism is not new, but only in recent years, due to the rapid development of the natural language processing, appeared the first algorithms which tackled the difficult task of detecting it. Most of these algorithms utilize machine translation to compare texts written in different languages. We propose a different method, which can effectively detect translations between language-pairs where machine translations still produce low quality results. Our new algorithm presented in this paper is based on information retrieval (IR) and a dictionary based similarity metric. The preprocessing of the candidate documents for the IR is computationally intensive, but easily parallelizable. We propose a desktop grid solution for this task. As the application is time sensitive and the desktop grid peers are unreliable, a resubmission mechanism is used which assures that all jobs of a batch finish within a reasonable time period without dramatically increasing the load on the whole system.

## 1. Introduction

We encounter contents that are plagiarized or copied on a word-by-word basis more and more frequently, both in higher education and in scientific life. To find such contents several services and solutions have already been set up. However, the spread of the Internet and the fact that most students can speak at least one foreign language – mainly English – on a level which enables them to find materials relevant to a given topic on foreign websites and to translate them as well, generated a new type of plagiarism. Foreign language knowledge gives students the opportunity to translate such works and to use them, without giving their source, as an own idea, own piece of work instead of elaborating them. To solve this problem and detect translational plagiarism a research was conducted in 2011 to enable the KOPI Plagiarism Detection Portal [1] to detect not only copy-paste plagiarism cases but translated ones as well. As KOPI is a Hungarian service, and English is the most frequently spoken foreign language

among students; one of the design goals was that the new algorithm should work for English-Hungarian language pair as well.

The development of the KOPI Plagiarism Detection Portal started almost ten years ago in 2003 at the Department of Distributed Systems of the Computer and Automation Research Institute, Hungarian Academy of Sciences (MTA SZTAKI DSD). A year later KOPI was launched in Hungarian and English as a free public service, supporting monolingual plagiarism search for European languages. This service allows the user to upload documents and start a plagiarism search based on these. The uploaded written works are comparable with the documents of the given user and also with the full database of the system which includes tens of thousands of documents uploaded by all the other registered users. The number of registered users increased constantly and reached 10,000 in 2009, with several university faculties already using KOPI on a regular basis. By the end of 2011 it was the first plagiarism checker to be able to detect translated plagiarism, searching in the full content of Wikipedia.

When it comes to plagiarism, “Wikipedia is the most popular single source for both secondary and higher education students”, being responsible for more than 10% of the cases in higher education, getting almost 3 times as many hits as the next single source [2]. This means that it is crucial for any plagiarism detection engine to incorporate Wikipedia in its database to look for both copied and translated parts. The largest such source is the English Wikipedia with 3.9 million articles but the following two largest ones, German and French, have also 1.4 and 1.2 million articles respectively. Wikipedia is an always changing, constantly growing source. The number of English articles will reach 4 million probably before the end of the year [3].To keep the database up-to-date the new versions have to be incorporated as quickly as possible. The easiest way without constantly harvesting and stressing the servers of Wikimedia Foundation, is downloading the regularly published database dumps, chunk and convert them as fast as possible, then combine the results into a single dataset and finally update our database. We plan to convert much larger, never seen document collections harvested from the Internet, thus the periodical processing of Wikipedia data can be considered as a (very useful and important) test collection and as a first step.

The conversion and preprocessing was done on the KOPI production server at the first couple of times, but the time it took and the resources it needed were not acceptable, so a new solution had to be found. It would be possible to do the conversion with a larger dedicated resource, but the work is easily parallelizable and is going to largely increase in the future when other source documents are added. Therefore we have chosen a faster, more reliable and scalable method. With the help of desktop grid computing each Wikipedia database dump and other documents are converted in batches by volunteer resources. KOPI is deployed at the SZTAKI Desktop Grid (SZDG) [4] which is a BOINC [5] based desktop grid with currently more than 92,000 hosts connected, running since 2005. BOINC is an open source desktop grid middleware, being the most popular used by more than 70 public projects including the well-known SETI@home project, and with over 2,350,000 volunteers and 7,000,000 hosts combined [6]. However using volatile volunteer resources for computing batches of jobs raises some issues. The most important is that the roundtrip time of the job running on the most non-

reliable or slowest resource is going to determine the completion of the whole batch (i.e., the conversion of a Wikipedia database dump) and thus it affects the update frequency of our search database. There are different approaches for mitigating this, e.g., resubmission of jobs that are overdue. We have employed such a method since the start of the desktop grid version of the project in October 2011. Based on our measurements, this approach allowed us to achieve up to 61.25% improvement of mean roundtrip times for jobs within a batch with only 25.1% of overhead caused by the resubmission.

In this paper a comprehensive description is given about the process of database creation and querying for the multilingual plagiarism search engine, as follows. Section 2 gives an overview of the architecture and the main processes of the Wikipedia based plagiarism detection. Section 3 details the desktop grid related components, while Section 4 discusses the plagiarism search and detection. Section 5 details our results and experiences, while Section 6 discusses related work and the final section details future work and concludes the paper.

# 2. Overview

Figure 1 shows the two main processes involved in the Desktop Grid enabled KOPI system. The first one (see 1. on Figure 1) is the preprocessing of Wikipedia data with the help of SZTAKI Desktop Grid. Here, one at a time, a periodically published Wikipedia database dump is sliced into multiple jobs (also called as work units) by the Work Unit Generator.

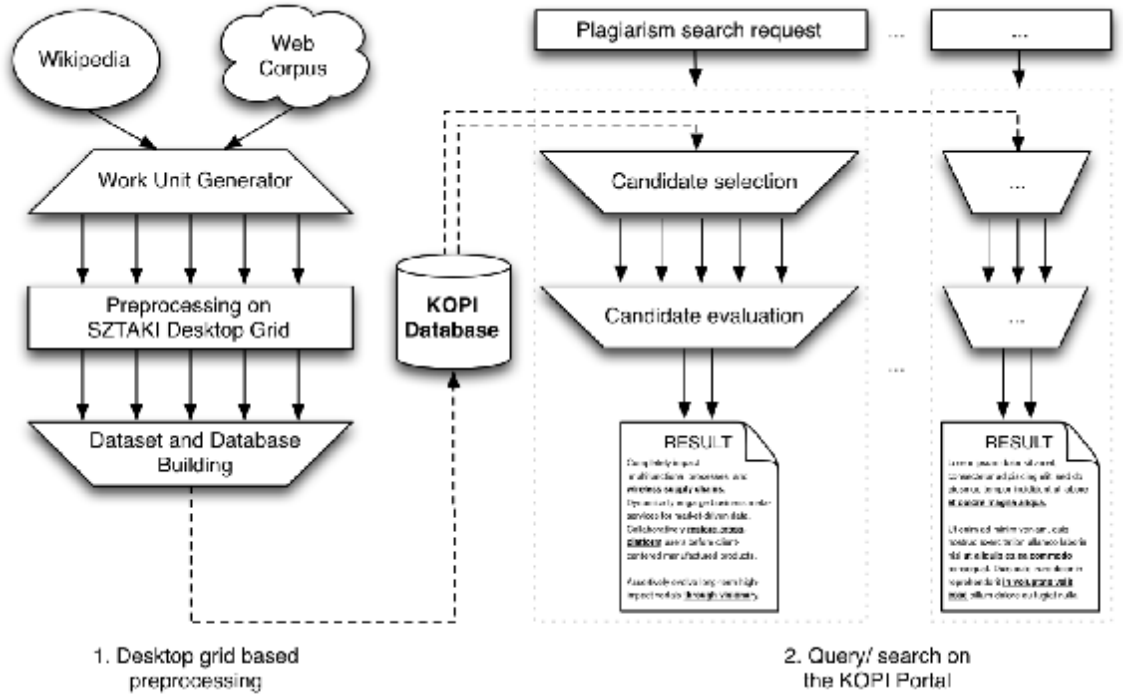


**Figure 1 The two main processes for the Desktop Grid enabled KOPI system: (1) the preprocessing of Wikipedia data on SZTAKI Desktop Grid, and (2) the plagiarism search at the KOPI Portal**

These work units are submitted as a single batch to the desktop grid where the volunteer resources process them. The returned results are continuously assimilated into a single dataset. Once all work units are finished the dataset is complete and the KOPI database is updated. This process must be repeated for each different language Wikipedia dump and every time a new dump is published by Wikipedia. The exact steps are detailed in the next section. The second process (see 2. on Figure 1) is the plagiarism search initiated at the KOPI Portal (c.f. Section 4.4). This is a request in the form of a document submitted by the user of the portal. First, during candidate selection the submitted text is compared to the KOPI database and suspicious sentences and fragments are selected. These candidates are then evaluated based on a similarity metric and the most promising ones are returned to the user via the portal. Section 5 discusses the details of the selection and evaluation.

# 3. Desktop grid based Wikipedia data preprocessing

Figure 2 shows the overview of the architecture that is used for preprocessing the Wikipedia content for the KOPI Plagiarism Search Portal using desktop grid resources. The architecture consists of four main components which are detailed in the following subsections. First the KOPI Server related parts: (i) the Master (Work Unit Generator and Dataset Builder) and the (ii) Scheduler used for improving batch completion times; then the (iii) SZTAKI Desktop Grid (SZDG) Server [7] components and volunteer resources; and finally (iv) the KOPI desktop grid application deployed at the SZDG Project.

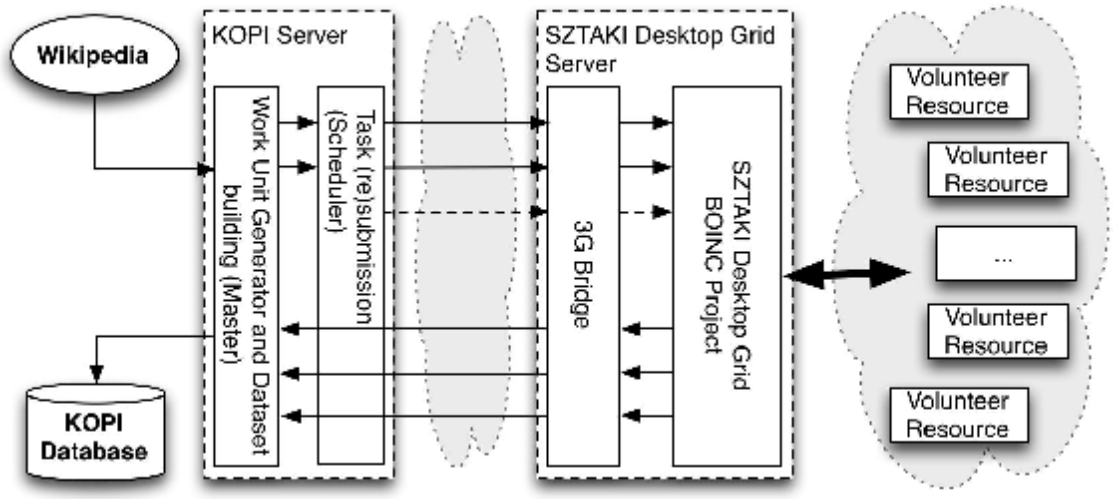


**Figure 2 The architecture used for Wikipedia data preprocessing**

## 3.1. KOPI Server

The KOPI Server (shown on Figure 2) is responsible for managing the desktop grid and contains two components: (i) the Master and (ii) the Scheduler. A single Wikipedia dump is

submitted to the desktop grid and handled as a batch. The Work Unit Generator (Master) is responsible for partitioning these XML dumps into smaller pieces for the desktop grid and combining the results into a single dataset as they return. Wikipedia publishes its content periodically in an XML format file for each different language. They have a size of several GBs (e.g. Hungarian - 1.7GiB, French - 8.7GiB, German - 10GiB and English - 36GiB) and are therefore too large and require too much CPU time to be processed sequentially. Fortunately the pre-processing procedure of these dumps can be easily parallelized by splitting them into smaller chunks (e.g., by splitting on article boundaries). The chunks can be sized arbitrarily, however we wanted to satisfy several constraints: (1) size of the input and output files and thus the time spent doing network I/O should be small compared to the time spent processing (e.g., inputs and outputs less than 10MiB); (2) runtime should be less than an hour, since we are going to run legacy code on volatile (volunteer) resources and have no possibility to implement any checkpointing mechanism. Based on our empirical studies a work unit with around a total of 3 MiB input files requires an hour of processing time on a current dual core computer (however the application is not multi-threaded) and will create around a 7 MiB compressed output file.

The Scheduler component is responsible for the submission and management of the work units of the batch. It is also responsible for minimizing the total time required to finish the batches ("makespan"). This is currently achieved by resubmission techniques which are discussed in detail in Section 3.4. The Dataset building part of the Master is responsible for processing the computed results returned from the desktop grid and combining them into a single dataset that will be used for updating the KOPI database.

## 3.2. SZTAKI Desktop Grid

SZTAKI Desktop Grid (SZDG) Project is a BOINC based public desktop grid (DG) project, or is also referred as volunteer computing (VC) project, running since 2005 at the Laboratory of Parallel and Distributed Systems (LPDS) of MTA SZTAKI. Currently SZDG has over 41,000 registered volunteers with more than 92,000 hosts total. Volunteers join the project via installing a small client application, the BOINC Client, which first downloads the deployed application(s) from the project. Second it fetches periodically new input files and parameters for the application in form of "work units". These compute intensive tasks are then processed in the background. Any application for the desktop grid must be first deployed and registered, thus it is not possible to submit and run arbitrary ones. Desktop grids are best suited for long term bag-of-task or parameter study type of applications where the application does not change frequently. There is no possibility of communication between running tasks either, thus so called "embarrassingly parallel" applications are required. These restrictions stem from many factors: (i) the volunteer resources are traditionally home desktop computers behind Network Address Translation (NAT) and firewall so they cannot be reached from outside; as a result (ii) the clients on the volunteer resources always pull tasks from the server, and not the server pushes them to the clients, thus there is no guarantee that two clients are processing work (two tasks are running on different hosts) at the same time. Contrary to traditional desktop grids, which run usually a single application, SZTAKI Desktop Grid is considered as an "umbrella" project since it hosts many applications, KOPI being one of them. Any volunteer has

the freedom to select which application they want to run from those deployed at the project. This means that applications usually compete against each other for a given set of resources and each of them can use a fraction of the available total for the project. Volunteer resources tend to be volatile and non-reliable; it is possible that an assigned work unit never finishes on a host. The middleware is prepared to handle this by setting a "deadline" for all in progress work units. Once this deadline passes the work unit is considered lost and a new instance is sent to another host.

The 3G Bridge [8] (in Figure 2) running on the SZDG Server is a generic grid-grid bridge, which provides interoperability between different distributed computing infrastructures (DCIs) on a job based level. It is able to fetch jobs from different DCIs, queue them internally and submit them to different middleware using its modular plug-in architecture. Its "WSSubmitter" component provides a generic SOAP based web service interface for accessing infrastructures and remote job management: it allows submitting jobs directly to the different queues of 3G Bridge and also managing these jobs. 3G Bridge provides command line tools as well, which can access the web service interface, thus allowing (a) remote job submission to SZDG (or any BOINC based project with 3G Bridge deployed); (b) querying the status of running jobs; (c) cancelling jobs and (d) retrieving their outputs. In our case this service and the remote access feature are used by the KOPI Server components to interact with SZTAKI Desktop Grid.

## 3.3. KOPI Desktop Grid Application

Generally developing and porting applications to desktop grids are not a straightforward process. BOINC has several constraints and requirements for application development: (a) binaries have to be linked with the BOINC libraries; (b) special BOINC API function calls for initialization and termination must be called from the application; (c) file access functions should be replaced by the ones provided by the BOINC API; (d) various optional functions (e.g., reporting completion ratio of the currently running work unit) should be implemented and (e) the BOINC API is only available for a set of programming languages (C/ C++, Fortran and Python). There are some additional requirements if the desktop grid is open to the public and volunteer resources are used: (i) application binaries should be available for the major operating systems (e.g., Windows, Linux); (ii) 32/ 64 bit application binaries should be provided as well for the supported operating systems; and (iii) from the optional functions (see (d)) the progress reporting should be implemented to keep the volunteers informed about the work they are currently processing. These requirements and constraints stem from many factors, an extensive list detailing them and the API description can be found on the BOINC wiki [9].

In the case of KOPI, chunking and natural language processing are used at two distinct places in the whole system. First, for the data used for dataset building and database update (see process 1 in Figure 1), second, when the suspected document is compared to the index and the candidate chunks (see process 2 in Figure 1). It is necessary to use exactly the same method at these two points to ensure that even if there are some bugs or errors (like a stemming error) they are the same on both sides. The easiest way to ensure this is to have only one implementation of the functions. The functionality was originally implemented in PHP at the KOPI Portal, so we decided to use this PHP code for the desktop grid application as well.

The resulting code is possibly slightly slower than it would be in case of using compiled code (e.g., Java). However, it requires less memory and the stemming, which is one of the slowest parts, is done by an external program, namely Hunspell [10]. This functionality (the preprocessing) is resource-intensive and embarrassingly parallel, therefore it can be ported to desktop grids. It performs five distinct steps: (i) language identification; (ii) MediaWiki XML to plain text conversion; (iii) text cleansing and chunking; (iv) stemming; and (v) output creation.

Language identification (see (i) above) at first seems to be superfluous as the language of the different Wikipedia from which these chunks are extracted is known by the system. However, as Wikipedia is only one of the many possible inputs, the system is designed so that the same software can be used to chunk text files with unknown content. The input does not even need to be one single file either; it can be hundreds of smaller files differing in language and size. Our current scenario is Wikipedia, however later the same system are to be used to convert document collections harvested from other sources (e.g., from the web). The system uses the n-gram method for language identification [11] [12], but the algorithm was modified to be able to quickly recognize if the document contains parts written in other languages as well, even if those parts are scattered all over the document [13]. The latter is important when, for example, documents harvested from the Internet are used. There are currently 15 languages supported, including English, Hungarian, French, German, Spanish, Bulgarian and Dutch. The list can be easily extended with the use of some sample text written in the desired language. The client can detect the file format of the input document; in case MediaWiki is detected it will be converted to plain text (see (ii) above). There are several possibilities to convert wiki format to plain text. The most obvious would be to install a MediaWiki instance and harvest the pages. This, however, raises two obstacles: first, on the desktop grid client this would be unfeasible; secondly, this would generate high load even on a server (which the clients could somehow access) as the XML to HTML conversion is compute-intensive. The other option is to convert it with some stand-alone software. Most of the software available freely are either operating system dependant or need installed software, which makes them unsuitable to be used in desktop grid environments. Others make errors when converting special pages, or truncated long ones. For these reasons the conversion is done by a self-written component with the following features:

- the names and boundaries of the Wikipedia pages are kept,
- within this only the textual information is necessary,
- “info boxes” – as they are duplicated information – are filtered out,
- comments, templates and math tags are dismissed,
- other pieces of information, like tables, are converted to text.

For the purposes of multilingual plagiarism search having text versions is only half of the work. Each XML file contains thousands of Wikipedia articles which have to be split to determine where the copied content can be found at the plagiarism search level. Articles are then split into smaller chunks (see (iii) above), according to the algorithm used by the search engine and all non-alphanumeric characters are disregarded, as they do not carry any useful information. After chunking the chunks are stemmed word-by-word (see (iv) above) and unlike by automatic

translation engines, all the possible stems (lemmas) are kept (e.g., divers → divers (assorted), diver). In English this is not so important, but for agglutinative languages like Hungarian this step is essential as word-forms have more often multiple lemmas. The free Hunspell program is used for stemming, as it is available for both Linux and Windows with dictionary files for more than 100 languages and dialects. Dictionary files are 1-2 MiB large per language so only English, German, French and Hungarian are currently included in the client. The last step of the process is the creation of the output result file (see (v) above), which is used as an input for the database upload and indexing step at the server. This file contains all the necessary information for the search process: the fast indexed candidate selection and the linear similarity metric.

Our goal was to reuse the existing implementation from the KOPI Portal code – written in PHP – and to be able to update the desktop grid application with little effort if the portal code changes over time. This meant that we could not use the BOINC API directly, thus alternative methods for application development had to be considered. GenWrapper [14] – a previous work of ours – is aimed at specially solving these types of problems. It acts as a generic wrapper between the application and the middleware, in this case BOINC. It allows running applications without modification ("legacy applications") on different DCIs. It provides a POSIX like scripting environment (shell) for bootstrapping the environment for the application, executing the application and post-processing the results on the clients before the upload. GenWrapper is available for different operating systems (for 32 and 64 bit versions of Mac OS X, Linux and Windows). In the case of KOPI this meant that no changes to the portal code were necessary. Only the GenWrapper script had to be written, which performs the following steps: (1) bootstraps the environment by deploying the bundled PHP and Hunspell binaries in the working directory, (2) prepares the input files; (3) executes the portal code via PHP which then processes the input files and finally (4) collects and compresses output files to be uploaded to the desktop grid server. The resulting desktop grid application bundle including all binaries is only 8-13 MiB, depending on the operating system.

## 3.4. Improving batch completion times on the desktop grid

Desktop grids provide access to a considerable amount of computing power; however reliability is always a problem when using volatile volunteer resources. Batch makespans are greatly affected even by a single faulty (or slow) resource. Machine availability in wide-area distributed computing follow the Weibull or Pareto distributions [15], which are power law probability distributions, thus resulting in the so called "long tail effect" for batch completion times. This effect can be mitigated, thus batch completion times improved e.g., by simple resubmission mechanisms. In this section our "black box" approach is detailed. We refer to a Wikipedia dump as a "batch", tasks belonging to a batch and submitted to 3G Bridge as "jobs", and finally tasks in the SZDG middleware as "work units". Depending on the application settings of the desktop grid a single submitted job can be executed by multiple volunteers. In our approach the middleware is considered as a black box, namely we have (a) no influence on the work unit settings (e.g., replication factor, required quorum), (b) no influence on the scheduling policies. However, we assume that (c) the middleware is not overcommitted and (d) its load (non KOPI

related in our case) does not cause interference (or can be considered constant) for our application. We also do not handle failures explicitly, we assume that (e) in the first place the middleware is responsible for failure tolerance and (f) there is no permanent failure in the system and (g) if we still encounter a failed job from the middleware it can be resubmitted safely. We chose the black box approach (see (a), (b) and (c)) since we wanted to make a generic approach independent of the capabilities of the underlying middleware (in this case BOINC). However we acknowledge that in some cases this approach may result in increased load at the middleware, e.g., BOINC uses replication and a resubmission at a higher level will result in a new work unit which is then replicated to multiple instances instead of a single new instance of the previous work unit. Constraints (c) and (d) can be considered heavy restrictions however for example in our case public desktop grid projects usually run a single application which in turn can use all the resources available.

First let *t* denote a job, and *B* a batch consisting of *n* jobs:

$$B = t_1, t_2, t_3, ..., t_n$$

$i_{mj}$ is the *j*-th running instance of the *m*-th job, $r_c$ is a resend constant which defines how many jobs can be resubmitted for a smaller batch, $r_r$ is a relative value which defines the percentage of jobs that can be resubmitted, the function *unfinished(B)* returns the unfinished *t*-s from a batch *B,* the function *useful*($i_{mj}$) returns 1 if $t_m \in$ *unfinished*(B) and 0 else. All the unfinished jobs are resubmitted if the following criterion is met:

$$\sum_{m=1}^{n}\sum_{j=1}^{c} \text{useful}(i_{mj}) < \max(r_c, r_r \cdot |B|)$$

If at all resubmissions all unfinished jobs are resubmitted it can be rewritten: let *c* denote the number of times jobs have been resubmitted:

$$\sum_{m=1}^{n}\sum_{j=1}^{c} \text{useful}(i_{mj}) = (c+1)\,|unfinished(B)|$$

and with that the new criterion for resubmission is:

$$(c+1)\,|unfinished(B)| < \max(r_c, r_r \cdot |B|)$$

For our experiments we chose empirically the following constants:

$$r_c = 100$$
$$r_r = 0.1$$

These parameters mean that batches with less than 1000 jobss are resubmitted after having less than 100 unfinished ones running. For larger batches this occurs after having fewer potentially useful running jobs than 10 percent of the size of the batch.

# 4. The plagiarism search

The novelty of the presented multilingual plagiarism search algorithm used by the KOPI Plagiarism Search Portal is threefold; in contrast to other similar approaches it does not use automatic translation engines, it uses the information retrieval method for much smaller chunks than usual (therefore it is more computationally intensive), and double-checks the results with a novel metric to evaluate the similarity between chunks written in different languages. The process consists of four steps: (1) an index database for candidate selection is built from the preprocessed data from the grid, (2) the index is used to select candidates for the search, (3) for a much more accurate result the similarity metric evaluates the returned sources against the suspicious chunks, (4) the final results are calculated and a report is assembled for the user. These steps are explained in detail in the next four chapters.

## 4.1. Candidate Selection

The algorithm uses a special similarity metric which is explained in section 4.2 below. This metric can precisely determine what the chances are that two sentences are the translations of each other. Comparing a suspicious sentence pair wise to all the source sentences in the database results in a linear runtime, which even for smaller databases would be impossible to calculate. To overcome this hurdle a candidate sentence selection precedes the use of the similarity metric which returns the most likely candidates from the database. Our algorithm uses information retrieval for candidate sentence retrieval. Depending on the number of requested results the algorithm can be tuned for speed or for precision: with more candidate documents the precision increases slowly but runtime decreases linearly. Nawab et al. [16] use a similar method to score and select source documents for their monolingual plagiarism search algorithm.

## 4.2. Similarity metric

The Hungarian language has three main obstacles when being compared to other (European) languages: (a) loose word order; (b) conjugation and (c) having a significantly different grammar. Most recently researched translational plagiarism checker algorithms are based on machine translation [17] [18] [19], but this cannot be used for Hungarian as statistical machine translation algorithms, at the time of this writing, and for the above-mentioned reasons, are almost useless from and to Hungarian. As Grozea et al. [19] writes “The idea to translate first, followed by the same language plagiarism detection is neither a scientific contribution, nor a distinguishing feature for a plagiarism detection system. […] mismatch between this (human) translation and the fairly poor automatic translation will resemble a heavy obfuscation, making the actual plagiarism detection task only more difficult. We have tested several translation engines for the same language, and their (very frequent) translation mistakes rarely matched.”,

and they are speaking mostly of German-English translation which has one of the best quality among all language pairs.

The new algorithm developed does not use automatic translation; instead it employs a unique method based on a similarity metric between sentences. The similarity metric is capable of deciding how much two sentences are similar to each other, even if the sentences are written in different languages. With the help of this metric each sentence of the suspicious document is compared against tens of suspicious sentences. If the similarity metric exceeds a certain value, the sentences are regarded as translations of each other. If two documents have a certain number of common sentences, they are indicated as possible translations of each other.
In most language pairs (and especially in Hungarian and English) the word order differs largely so it can be disregarded. If the words of two sentences are compared to each other without taking into account the order, it results in a bag of words algorithm. It is used in most cases for document categorization, SPAM filtering, but some authors use it to recognize emotions as well. For this research purpose the bag of words algorithm will be used in a much smaller context, to compare sentences to each other.
An *n* word long sentence (*S*) should be represented by its words (*w*). Of course this is a simplification as in theory one can construct different sentences from the same words as well, but in most cases it is not ambivalent.

$$S_x = w_{x1}, w_{x2}, w_{x3}, ..., w_{xn}$$

The similarity between two sentences is dependent on the number of common words between the sentences minus the number of mismatching words between them. This research shows that a better result can be achieved if weights are assigned to the common words ($\alpha$) and the missing words ($\beta$), as the common ones are more important than the missing ones (the latter ones are often indications of a poor dictionary). With all these considerations the following formula is used for the similarity measure:

$$Sim(S_x, S_y) = \min(\alpha \cdot |S_x \cap S_y| - \beta \cdot |S_x \setminus S_y|, \alpha \cdot |S_y \cap S_x| - \beta \cdot |S_y \setminus S_x|)$$

For example, if $\alpha=2$ and $\beta=1$ then common words are considered twice as important as the missing ones. This is about the right value, but it depends on the dictionary and the languages used as well. Using the minimum of two values ensures that a much longer sentence is not considered an appropriate translation of a smaller one just because it is a part of it, also ensuring that the definition is symmetric, meaning that

$$Sim(S_x, S_y) = Sim(S_y, S_x)$$

which is a design goal, as if a Hungarian sentence is a translation of an English one, then this English sentence also has to be a possible translation of the Hungarian one.

The equivalence between words also needs to be defined as in this case $w_x$ and $w_y$ are in different languages. First *trans*($w_x$) is defined as the set of words which are the translations of $w_x$. As translation is symmetric

$$if\ w_x \in trans(w_y)\ then\ w_y \in trans(w_x)$$

so the definition is

$$if\ w_y \in trans(w_x)\ then\ w_x \equiv w_y$$

similarly

$$if\ w_y \notin trans(w_x)\ then\ w_x \neq w_y$$

Contrary to other multilingual plagiarism search algorithms which use machine translation in the first step and then a dictionary of synonyms to compare the texts in the next, this above metric eliminates the necessity to use a word disambiguation first and then – when comparing texts written in the same language to each other – synonyms. The *trans* function solves this problem by including all the possible translations of the word.

## 4.3. Final result

The last step of the multilingual plagiarism search – exactly like in monolingual cases – is the post processing of the raw results. This is done by counting the number and distance of possible translations for each source document and the score for each one. If they exceed a given threshold, the matching parts of the candidate and source documents are considered being similar or cases of plagiarism. The method differs from the one used by most other systems [20] where two chunks either match or not, that here the matching chunks also have a score which represents the similarity between them: a higher score meaning higher similarity.
Let $Sim_i$ and $Sim_j$ denote the similarity score of the $i^{th}$ and the $j^{th}$ chunk, the parameters can be changed but the current system displays a match if in the suspicious document:

- $Sim_i > 8$ or
- $Sim_i > 0$ and $Sim_j > 0$ and $|i - j| < 10$

The source documents are sorted by the number of matching chunks and the result is trimmed if there are more than 50. As of now adjacent detection for merging chunks is not used but will be considered in the future versions. The achieved results of this new approach are described in more detail in another paper [21].

## 4.4. KOPI Portal

The KOPI Online Plagiarism Detection Portal is a unique, open service for Hungarian and English speaking web users that enables them to check for identical or similar contents between their own documents and the files uploaded by other authors. The monolingual plagiarism search function works in any European language, due to the language-independent algorithm. At the time of this writing Hungarian, English and German texts can be compared to the English and Hungarian Wikipedia, but the translational detection system is being continuously improved to support other languages as well. On the web interface (see Figure 3) the user can upload documents and start a plagiarism search using these documents and the options defined above. When the search is finished, the result is emailed to the user and can be seen on the portal interface as well. If the document was compared to the Wikipedia, a list of possible Wikipedia articles is returned (see Figure 4), with the title of the article, the original sentences, and the suspicious ones from the document. Thus the user can hand-check the result given by the system, which is necessary as the system does not decide in place of the user: it does not distinguish between citation and plagiarism, the last word is that of the users.


**Figure 3 The interface of the KOPI Portal**

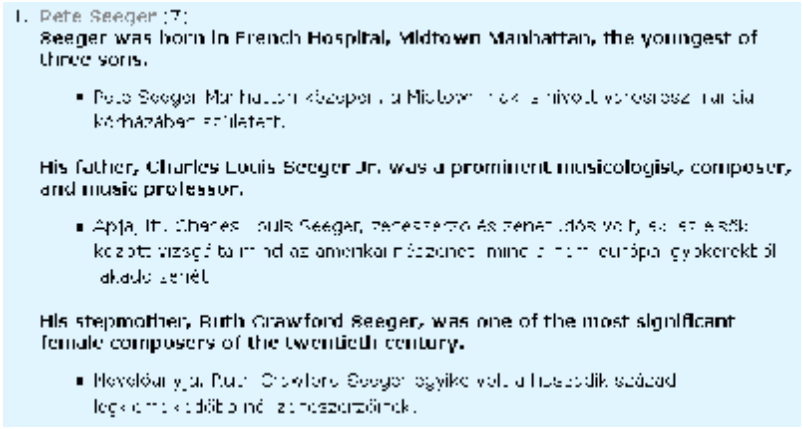


**Figure 4 Results of a plagiarism search presented by the KOPI Portal. The user can decide what is considered as citation or plagiarism.**

# 5. Results

Until June 2008 static HTML dumps from all Wikipedia wikis were available from Wikimedia Foundation [22], but this project has discontinued since then. As these text versions can be used for several other purposes as well, we decided to share them and make them available for everybody [23]. Currently the English (5.5 GiB), German (2.1 GiB), French (1.4 GiB) and Hungarian (311 MiB) versions can be downloaded, other languages will follow shortly.

The KOPI application is deployed and running on SZTAKI Desktop Grid permanently. SZDG is an umbrella project, but donors can select which application(s) they want to run. This leads to a great number of donors who support KOPI exclusively. KOPI work units have higher priority set on the server, so if volunteers allow multiple applications from SZDG, still KOPI will be processed first. These steps ensure that there is computing capacity available for the KOPI application (see criterion (c) and (d) in Section 3.4) whenever required. For evaluation we included the measurements of six representative Wikipedia batch conversions: two English, one French, one German and two Hungarian batches.

Table 1 shows the results, namely (i) the number of jobs in the batch; (ii) the total number of jobs executed (including resubmissions) for the batch; (iii) the mean round trip time (RTT) for the initial submission of the batch; (iv) the standard deviation of the RTT's; (v) the mean and (vi) standard deviation for all jobs; finally (vii) the mean and (viii) standard deviation for the "useful" jobs. The dates given in Table 1 denote the date of the Wikipedia dump used by the conversion and not the date when experiments started, as all experiments were executed independently. For discussion we group the six conversions (batches) into three groups based on their number of jobs: (1) the English ones, (2) the French and German ones and (3) the Hungarian conversions.

| | jobs in batch | jobs executed | first submission: mean | first submission: deviation | all jobs: mean | all jobs: deviation | useful jobs: mean | useful jobs: deviation |
|---|---|---|---|---|---|---|---|---|
| **English - 16/11/2011** | 3326 | 4277 | 120464 | 157630 | 117847 | 152527 | 76744 | 53313 |
| **English - 02/12/2011** | 3348 | 4189 | 151052 | 182702 | 130929 | 172591 | 94140 | 69613 |
| **French - 17/01/2012** | 823 | 1093 | 67114 | 104129 | 71312 | 111911 | 43767 | 22632 |
| **German - 17/01/2012** | 946 | 1209 | 96231 | 100432 | 100080 | 123244 | 72002 | 26847 |
| **Hungarian - 06/01/2012** | 162 | 409 | 62483 | 91627 | 72987 | 119442 | 24285 | 16354 |
| **Hungarian - 15/01/2012** | 162 | 381 | 75307 | 120757 | 61917 | 114965 | 22181 | 14100 |

**Table 1 Statistics for round trip times of batches in seconds**

For the first group our algorithm introduced 28.5% (951 jobs) and 25.1% (841 jobs) overheads respectively for the total job numbers (including resubmissions), while resulting in 36.29% and 37.28% improvements in mean round trip times and 66.18% and 61.9% in standard deviation considering the useful jobs and first submission round trip times. For the second group the algorithm caused 33.2% (270 jobs) overhead for the French and 27.8% (263 jobs) overhead for the German conversion in terms of job numbers. However the mean round trip times were improved by 38.63% and 28.06% while the standard deviations reduced by 79.78% and 78.22%. The algorithm caused the largest overhead for the last group (for the Hungarian conversions) in terms of percent, namely 152.47% and 135.19%, however these mean only 247 and 219 additional jobs. In this case the mean round trip time was improved by 66.73% and 64.18%, while the standard deviation by 86.31% and 87.74%. This algorithm proved to be effective, it only resulted in additional 25-28% of jobs at the largest, the English Wikipedia.

As can be seen in Table 1 smaller languages like Hungarian (which has around 20 times less content and thus fewer jobs in the conversion than the English) results in relatively the highest resend rate (152.4%), but even so in numbers it is less than for the English Wikipedia (245 vs. 841 and 219 vs. 951). The overall resend rate for the 4 languages is 31%. By resending all the unfinished jobs regularly the probability that all copies of a given one will fail decreases exponentially (considering criteria (e), (f) and (g) in Section 3.4). This algorithm calculates the time of the resubmission based on the number of returned jobs. As our results show, this is more efficient for larger batches, where the resend rate is around 25-38%. As the size of the batch decreases the resend rate (in percentage) grows.

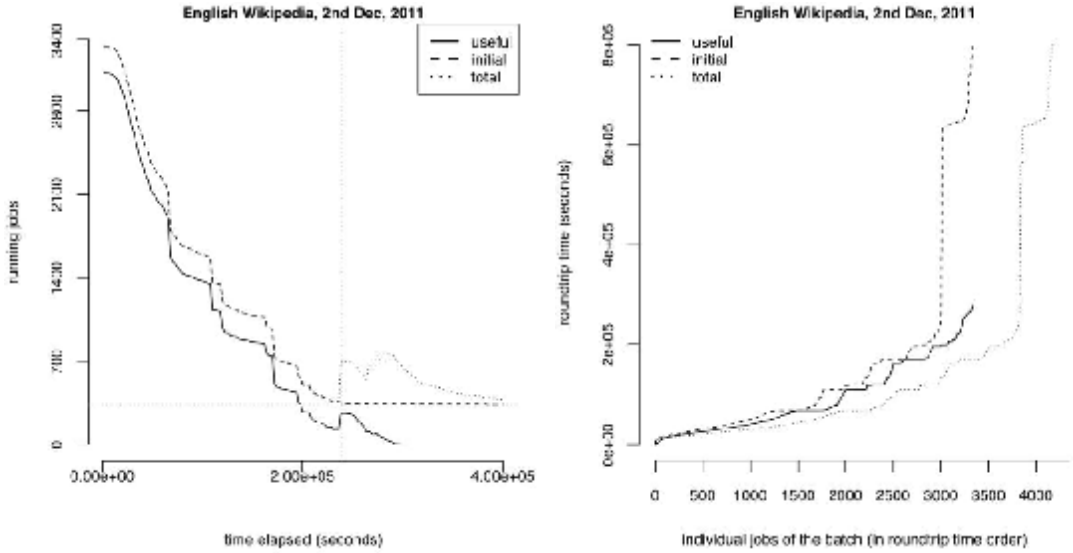


**Figure 5 Time lapse of a batch (left side) and its jobs in round trip time order (right side) for an English Wikipedia dataset**

Figure 5 - 7 show the time lapse (left side) and jobs in round trip time order (right side) for three selected batches from the previous six. The time lapse measurements are displayed up to 4.00e+05 seconds (~4.6 days) from start. Similarly round trip time measurements are limited in time, but show times up to 8.00e+05 seconds (~9.25 days) and any job reaching this threshold can be considered as unfinished and highly responsible for the “tail effect”. Each chart shows the details of (i) the initial submission, (ii) total jobs and (iii) the useful jobs, which can be considered as three individual batches with own time lapse and round trip measurements on the same chart. The vertical and horizontal dashed lines on the time lapse charts represent the time and number of initial jobs when the first resubmission occurred.

We can see that the time lapses on the three charts have similar characteristics and the round trip time measurements as well. We can see that the number of initial and total jobs is the same obviously until the first resubmission; however the difference between the initial and useful jobs shows from the beginning that some jobs were considered from the first resubmission. Also the first size difference of the job number increase at the first resubmission (denoted with dashed horizontal and vertical lines) shows that a single resubmission was not enough, not all new jobs are considered useful. Similarly the round trip measurements show that there are jobs from the initial submission which are considered unfinished and resubmitted instances were useful jobs. We can also observe that although some jobs have lower round trip times they were still not considered as useful jobs, e.g., the chart of the French conversion shows this: solid line (total jobs) deviates upwards from the dashed line (total). This means that a job was resubmitted, but a previous running instance finished before the new one. In such case our algorithm currently does not cancel the remaining instances, so they remain running.

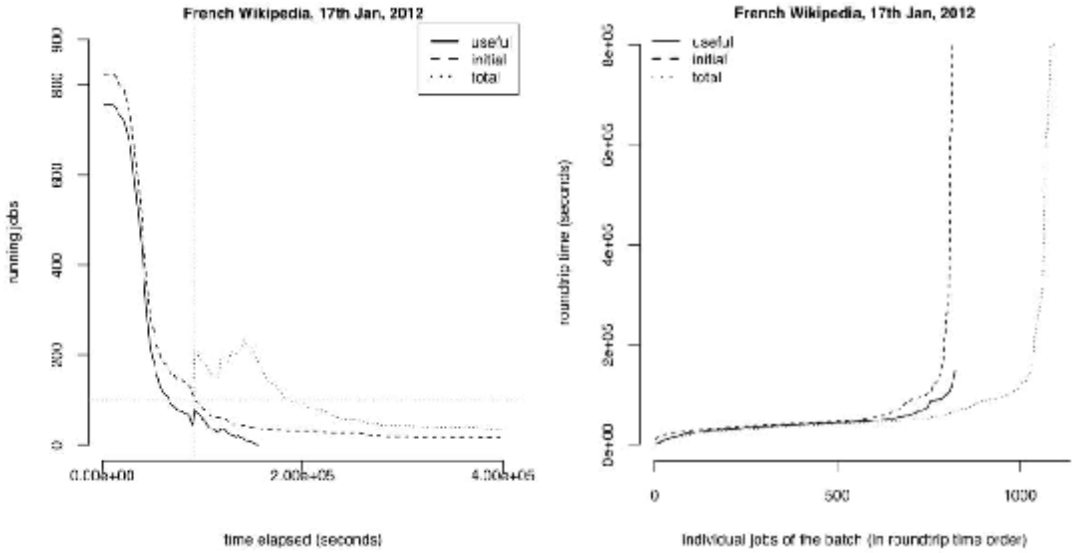


**Figure 6 Time lapse of a batch (left side) and its jobs in round trip time order (right side) for a French Wikipedia dataset**

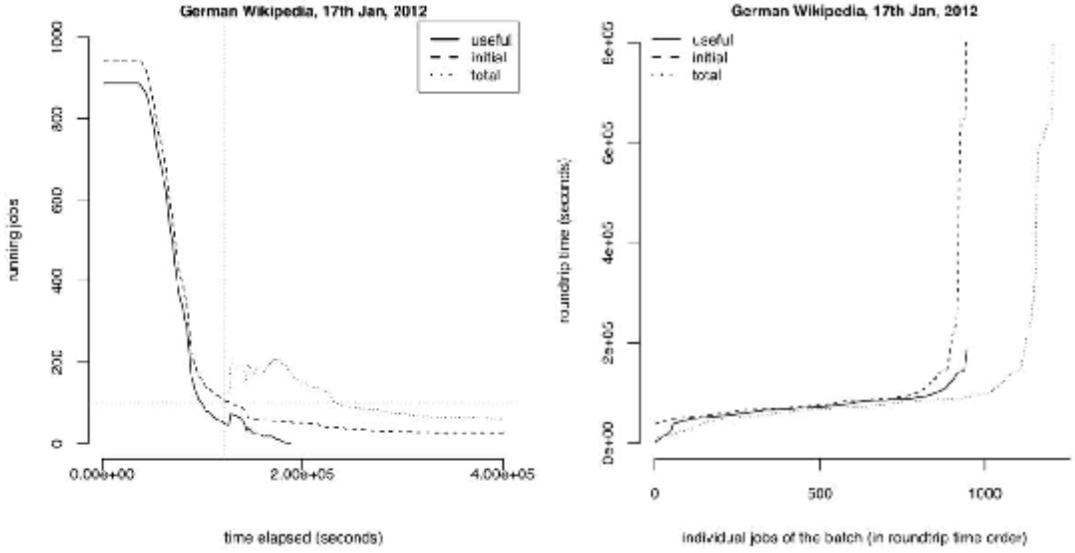


**Figure 7 Time lapse of a batch (left side) and its jobs in round trip time order (right side) for a French Wikipedia dataset**

If we look at the useful "batch" completion times we see that the English conversion finished after 3.1e+05 seconds, the French one after 1.76e+05 seconds while the German one after 1.97e+05 seconds. However, if we look at the initial batch we can see that in each case jobs were running at the 8e+05 cutoff thresholds of the measurements. Considering this threshold we can state that the English conversion took 61.25% less time with 25.1% (841 jobs) overhead, the German conversion took 75.38% less time with 27.8% (263 jobs) overhead and the French conversion took 78.01% less time with 33.2% (270 jobs) overhead total.

# 6. Related work

The whole process detailed in this paper can be thought of as four distinct problems: (i) the conversion of the Wikipedia XML dumps to plain text; (ii) the natural language processing (NLP); (iii) the use of desktop grids and the applied techniques for reducing makespan; and finally (iv) the multilingual plagiarism search. However (ii) and (iv) can be considered as two distinct steps of the same process and both are usually covered within a single (related) work.

The first and most obvious solution to process the Wikipedia (see (i)) was to download the dump in a plain text or HTML format. Unfortunately the last available static dump is almost 4 years old dating back to 29 June 2008 [24]. There are numerous MediaWiki XML to plain text and HTML converters available on the Internet [25] [26], but most of them are outdated, not maintained, and each of the tested had some major flows. Some were too slow, run only under Windows, and a lot of them even failed to convert the XML e.g., trimmed long articles without any error message. We also experimented with the installation of an own MediaWiki/ Wikipedia instance and the crawling of the pages. The problem was that as all pages are accessed only once and all the pages are accessed so, no caching can be used, and the process was very slow. Even worse, the process could not be easily parallelized as the MediaWiki server was the bottleneck which requires a lot of disk and CPU resources. These were the main reasons why we decided to write our own converter. For those who will face the same problem that we did, we regularly publish the Wikipedia as plain text with the page boundaries kept [23]. The large computing capacity offered by the desktop grid for the Wikipedia pre-processing opened new opportunities to include some additional compute intensive tasks like heavy natural language processing (see (ii)) along with the conversion (see (i)), however this part uses widely known, freely available external tools.

In the following we discuss some related works for topics (ii) and (iv). Potthast et al. [27] performed an analysis of the current plagiarism search algorithms and came to the conclusion: “After analyzing all 17 reports, certain algorithmic patterns became apparent to which many participants followed independently. [...] In order to simplify the detection of cross-language plagiarism, non-English documents in D are translated to English using machine translation (services).” Grman et al. [17] use an anti-plagiarism system that can be divided into three main parts: (a) pre-processing of input data (plain-text pre-processing), (b) detection of passage pairs (plagiarism candidates) and (c) post-processing (removal of overlapping passages, merging of passages, and exclusion of uncertain passage pairs). In the first phase they do three sub-steps from which the first is text translation into English with the Google Translator API. Grozea et al. [19] propose a new similarity measure and a new ranking method (Encoplot). They use n-grams and compare those pairwise in documents; the outcome of this procedure is a set of pair of indices, the first in each pair being an index in the first document and the second an index in the other document, such that the N-gram starting at the positions indicated by the two indices coincide. The results can be visualized in a graph, and the matching parts are clearly visible as lines. For multilingual plagiarism search they use automatic translation, but it was “prohibitively slow” so they did not use it at the end. They conclude therefore that “automatic translation is a rather separate problem in NLP. Of course,

there could exist an approach to cross-language plagiarism detection that doesn't bring first all texts to the same language by automatic translation, but we are not aware of any such approach with good performance either in the competitions so far, or in the existing literature." Ceska et al. [28] describe an approach to multilingual plagiarism detection called MLPlag which is based on analysis of word positions, it utilizes the EuroWordNet [29] thesaurus to transform words into language independent form. This allows them to identify documents plagiarized from sources written in different languages. The method could be used for detecting translated plagiarism, the main limitation of this method is that a large parallel Wordnet [30] is needed for every language pair which has to be supported.

Makespan (or batch completion time) improvement on desktop grids (see (iii)) through the use of (a) different direct scheduling strategies (e.g., periodic resubmission) [31] [32], (b) statistical modeling of availability and reliability [33], (c) checkpointing [34], or (d) elasticity via on-demand dedicated resources ("cloud-bursting") [35] [36], is a very active field of research. Reynolds et al. [35] investigate how to reduce the makespan of scientific workflows running on desktop grids by augmenting them with dedicated resources from Infrastructure as a Service Clouds. They are addressing the "tail problem" of volatile resources where a small percentage of a batch or workflow is assigned to resources which will never return the results (e.g., the resource or the desktop grid software running on them is shut down permanently; they have some hardware failure, etc.). In these cases the task will "time out" after a period and the desktop grid will resend it to a different resource. However this radically increases the makespan of the whole batch or workflow. To overcome this problem the authors use dedicated cloud resources ("cloud-bursting") to compute the remainder of the batch after a fixed percentage is completed by the desktop grid. The authors assume that the start of the tail can be characterized by the completion of a fixed percent of the total number of tasks in a batch, and their results show that they were able to reduce the makespan by up to 40% this way. This work is partially based on a previous work of ours [37] in which we discuss the possibilities of creating and extending existing computing infrastructures with on-demand resources from infrastructure-as-a-service clouds. SpeQuloS [36] aims to provide Quality of Service (probabilistic guarantee for makespan) for best-effort distributed computing services like desktop grids by re-allocating jobs to more reliable resources and allocating dedicated resources when needed (cloud-bursting). Kondo et al. [31] propose resource selection techniques to improve performance and reduce makespan on desktop grids for applications that require rapid turnaround times. Three resource selection techniques are proposed: (i) resource prioritization, (ii) resource exclusion and (iii) different task replication strategies. They evaluated these via simulation running on data based on collected traces from real desktop grid configurations. Their main conclusions are the following: (a) in resource prioritization using static clock speed information delivered the best results; (b) using dynamic information like historical host availability does not improve application performance much more, but (c) by using different task replication techniques performance can dramatically increase. Bouguerra et al. [34] investigate strategies for scheduling checkpoints of sequential jobs on desktop grids. Their model is based on "lost computation time" and "re-execution ratio" and introduces the "cumulative checkpointing overhead". Their results are achieved using a simulation framework (SimGrid) with parameters from real-world systems. They conclude that (i) their proposed

model outperforms periodic checkpointing and standard execution and (ii) using fault tolerance blindly (both checkpointing and replication) can lead to dramatic system performance deterioration. Iglesias et al. [33] investigate long-term availability for groups of volatile volunteer hosts to broaden the set of applications usable on desktop grids. They consider collective availability as a key factor for enabling parallel applications and workflows for volunteer computing. They evaluated their methods using availability traces from an existing volunteer computing project (SETI@home). They represent the availability of each host as a binary vector where each bit denotes an hour of a week. They use the binarization threshold for deciding whether a host is considered online during a given hourly period. The authors used the k-means algorithm to cluster the hosts, and they found that the largest clusters are made of the always-on and always-off hosts. They used the first weeks of life of a host as training data and they use this to predict the behavior of a host, and define different metrics for measuring the quality of prediction. Their results show that service deployment based on the binarization approach can achieve high availability and small redundancy.

# 7. Conclusions and future work

In this paper we presented a novel method for detecting translations between language pairs where the traditional machine translation methods still produce low quality results. Our method is based on information retrieval and a dictionary-based similarity metric. We provide a free online portal (KOPI) which utilizes this method for detecting translated material from Wikipedia content or any uploaded material. Wikipedia evolves continuously and keeping the KOPI database up-to-date requires the periodic re-preprocessing of its data (i.e., the periodic dumps of content from the different language Wikipedia). We use volatile volunteer computing resources from SZTAKI Desktop Grid for these periodic batches of jobs. The volatility of the used volunteer resources increases the completion time of the batches dramatically. This is not acceptable for us since we want to use the new datasets as soon as possible. To mitigate this "long tail" effect we currently apply a task resubmission algorithm. The results of our real world measurements show that the ~30% overhead caused by the resubmission results in (i) ~4.5 times faster and (ii) less diverse (mean) batch completion times. Currently we do not take into consideration the replication factor of jobs on the desktop grid and the number of jobs in the batches which can result in high replication numbers in some cases. To deal with the increasing load and to use our resources more effectively we plan to investigate alternative scheduling algorithms and use on-demand dedicated resources (i.e., cloud-bursting) when needed as well. The next logical step in terms of extending the KOPI database will be to use the same system for documents crawled from the web, which are many orders of magnitude larger than Wikipedia. However the English web is a very important source of plagiarism in non-English speaking countries, and can not be left out. We strongly believe that SZDG can cope with processing much more data, and most of the pages (e.g., articles and scientific papers) on the web are not updated regularly like the Wikipedia, so the resources can be used much more effectively. Nonetheless, it is very likely that a selective search has to be used in order to ignore pages with useless, short or non-scientific content. Selecting which domains and pages to include could be the subject of future work. The plagiarism search algorithm offers plenty of opportunities for research. The similarity metric can be improved by recognizing

word-groups and compounds. The search can be made more resilient to heavy editing by dealing with sentence splitting and combining. Using more and better quality dictionaries would also increase the precision, and incorporating other language pairs would benefit users coming from other countries.

**Acknowledgments**

We would like to thank Gabor Kecskemeti for the pre-review and the many valuable suggestions. Special thanks to Éva Virág for the proofreading of the text. The research has been partially supported by the Hungarian National Office for Research and Technology (NKTH) under grant No TECH_08-A2/2-2008-0097 (WEB2GRID). The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement No. 261556 (EDGI) and 283686 (GLOBAL excursion).

# References

[1] KOPI Online Plagiarism Search Portal. [Online] http://kopi.sztaki.hu/index.php?language=eng.

[2]**Huang, R.** Plagiarism and the Web, A Comparison of Internet Sources for Secondary and Higher Education Students. *http://pages.turnitin.com/rs/iparadigms/images/Turnitin_WhitePaper_SourcesSECvsHE.pdf.* [Online] 11 2011.

[3] Size of Wikipedia. [Online] http://en.wikipedia.org/wiki/Wikipedia:Size_of_Wikipedia.

[4]SZTAKI Desktop Grid. [Online] http://szdg.lpds.sztaki.hu/szdg/.

[5]**Anderson, D. P.** BOINC: A System for Public-Resource Computing and Storage. *5th IEEE/ACM International Workshop on Grid Computing.* November 8, 2004.

[6] BOINC statistics. [Online] http://boincstats.com.

[7]*SZTAKI Desktop Grid (SZDG): A Flexible and Scalable Desktop Grid System.* **Kacsuk, P., et al., et al.** 4, s.l. : Springer Netherlands, 2009, Journal of Grid Computing, Vol. 7, pp. 439-461. ISSN: 1570-7873.

[8]**Kacsuk, P., Farkas, Z. and Fedak, G.** Towards Making BOINC and EGEE Interoperable. *IEEE Fourth International Conference on eScience, 2008. eScience '08.* 2008, pp.478-484.

[9] The BOINC application programming interface. [Online] http://boinc.berkeley.edu/trac/wiki/BasicApi.

[10]BME MOKK: Hunspell stemmer. [Online] http://hunspell.sourceforge.net/.

[11]**Cavnar, W., B. and Trenkle, J., M.** N-Gram-Based Text Categorization. *Proceedings of Third Annual Symposium on Document Analysis and Information Retrieval.* 1994, Las Vegas, NV, UNLV Publications/Reprographics (1994), pp. 161-175.

[12]**Rehurek, R. and Kolkus, M.** Language Identification on the Web: Extending the Dictionary Method. *10th International Conference on Intelligent Text Processing and Computational Linguistics.* 2009.

[13]**Pataki, M. and Vajna, M.** Detecting the language of document written in multiple languages ("Többnyelvű dokumentum nyelvének megállapítása"). *VIII. Hungarian Computer Linguistics Conference (MSZNY 2011.).* 2011.
[14]**Marosi, A. Cs., Balaton, Z. and Kacsuk, P.** GenWrapper: A generic wrapper for running legacy applications on desktop grids. *IEEE Parallel and Distributed Processing Symposium, International.* pp. 1-6, 2009.
[15]**Nurmi, D., Brevik, J. and Wolski, R.** Modeling Machine Availability in Enterprise and Wide-area Distributed Computing Environments. *In proceedings of the Euro-Par'05 Conference.* 2003, pp. 432-441.
[16]**Nawab, R., M., A., Stevenson, M. and Clough, P.** External Plagiarism Detection using Information Retrieval and Sequence Alignment. *Notebook for PAN at CLEF.* 2011.
[17]**Grman, J. and Ravas, R.** Improved Implementation for Finding Text Similarities in Large Collections of Data. *Notebook for PAN at CLEF.* 2011.
[18]**Sameer, R., et al., et al.** External & Intrinsic Plagiarism Detection VSM & Discourse Markers based Approach. *Notebook for PAN at CLEF.* 2011.
[19]**Grozea, C. and Popescu, M.** The Encoplot Similarity Measure for Automatic Detection of Plagiarism. [Online] 08 2011. http://brainsignals.de/encsimTR.pdf.
[20]**Kasprzak, J. and Brandejs, M.** mproving the Reliability of the Plagiarism Detection System. *Lab Report for PAN at CLEF.* 2010.
[21]**Pataki, M.** A new approach for searching translated plagiarism. *In proceedings of the Fifth International Plagiarism Conference.* Newcastle, UK, 2012.
[22] A copy of all pages from all Wikipedia wikis, in HTML form. [Online] 6 2008. http://dumps.wikimedia.org/other/static_html_dumps/.
[23]Wikipedia text dumps. [Online] http://kopiwiki.dsd.sztaki.hu/.
[24]Wikipedia Static HTML Dumps. [Online] http://dumps.wikimedia.org/other/static_html_dumps/.
[25]WikiTaxi. [Online] http://www.wikitaxi.org/delphi/doku.php/products/wikitaxi/index.
[26]WP2TXT: Wikipedia to Text Converter. [Online] http://wp2txt.rubyforge.org/.
[27]**Potthast, M., et al., et al.** Overview of the 2nd International Competition on Plagiarism Detection. [Online] http://www.clef2010.org/resources/proceedings/clef2010labs_submission_125.pdf.
[28]**Ceska, Z., Toman, M. and Jezek, K.** Multilingual Plagiarism Detection. *In: Proceedings of the 13th International Conference on Artificial Intelligence.* 2008, pp. 83–92. Springer Verlag, Berlin Heidelberg.
[29]**Vossen, P.** EuroWordNet: a multilingual database for information retrieval. *Proceedings of the DELOS workshop on Cross-language Information Retrieval.* 1997.
[30]**Fellbaum, C. (ed.).** *WordNet: An Electronic Lexical Database.* Cambridge, MA : MIT Press, 1998.
[31]**Kondo, D., et al.** Scheduling Task Parallel Applications for Rapid Application Turnaround on Enterprise Desktop Grids. *Journal of Grid Computing.* 2007, Vols. 5(4), pp. 379-405.
[32]*Computing Low Latency Batches with Unreliable Workers in Volunteer Computing Environments.* **Heien, E., Anderson, D. and Hagihara, K.** 4, s.l. : Springer Netherlands, 2009, Journal of Grid Computing, Vol. 7, pp. 501-518.

[33]**Iglesias, D. L., Kondo, D. and Piug, J. M.I M.** Long-term Availability Prediction for Groups of Volunteer Resources. *Journal of Parallel and Distributed Computing.* 2012, Vols. 72(2), pp. 281-296.
[34]**Bouguerra, S., Kondo, D. and Trystam, D.** On the Scheduling of Checkpoints on Desktop Grids. *11th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing (CCGrid 2011).* 2011.
[35]**Reynolds, C., et al., et al.** Scientific Workflow Makespan Reduction through Cloud Augmented Desktop Grids. *3rd IEEE International Conference on Cloud Computing technology and Science, CloudCom 2011.* Athens, Greece. DOI: 10.1109/CloudCom.2011.13, 2011.
[36]**Delamare, S., et al., et al.** SpeQuloS: A QoS Service for BoT Applications Using Best Effort Distributed Computing Infrastructures. *International Symposium on High Performance Distributed Computing (HPDC'2012).* 2012, Delft, Nederlands.
[37]**Marosi, A., Cs. and Kacsuk, P.** Workers in the clouds. *In Parallel, distributed and network-based processing (pdp), 2011 19th euromicro international conference on.* 2012, pp. 519-526.