*t*SEARCH

Leveraging the Automatic Analysis of Machine Translation Evaluation for Fast and Flexible Error Analysis

Laura Mascarell Espuny

Advisors: Lluís Màrquez and Meritxell Gonzàlez

Facultat d'Informàtica de Barcelona, Universitat Politècnica de Catalunya
Master in Information and Technology.

September 2013

Abstract

The development cycle of machine translation (MT) systems includes important stages such as the evaluation, the error analysis and the system refinement. These stages have promoted several research studies and tools focused on aiding the tasks of MT developers.

This master thesis is concerned with the error analysis stage. We have developed the *t*SEARCH tool, an open web-based application that aims to facilitate the qualitative analysis of the translation quality. So far, MT developers usually have to tackle the tedious and time-consuming task of inspecting a large set of translations almost manually. There exist few tools focused on the classification of the error types. In contrast, *t*SEARCH offers a new focus providing mechanisms for doing complex searches over a collection of translation cases evaluated with a large set of diverse translation evaluation measures, which makes easy the discovery of patterns that may help MT developers to improve the translation quality. Thus, the error analysis task can be carried out over a specific subset of translations examples, turning into a more efficient and feasible work.

To carry out this proposal, *t*SEARCH builds on ASIYA, an open toolkit for automatic machine translation evaluation, that provides a large set of evaluation measures. The results of the ASIYA evaluation are used by *t*SEARCH in order to extract a number of features and be able to process the user queries. The search engine offers a rich and flexible query language that allows to find translation examples matching a combination of numerical and structural features. Furthermore, its database design permits a fast response time for all queries supported on realistic-size testbeds. We also offer a friendly and easy to use web interface that makes possible an interactive access to the query results.

Resum

El cicle de desenvolupament dels sistemes de traducció automàtica (TA) consta de les següents etapes: avaluació, anàlisi dels errors i refinament del sistema. La importància d'aquestes etapes ha promogut molts estudis d'investigació i eines que tenen per objectiu ajudar en les tasques dels desenvolupadors de sistemes de TA.

Aquesta tesi de màster està enfocada en l'etapa d'anàlisi d'errors. Hem implementat l'eina *t*SEARCH, una aplicació web que té per objectiu facilitar l'anàlisi qualitatiu de la qualitat de la traducció. Fins ara, els desenvolupadors de sistemes de TA han d'enfrontar-se sovint amb la dura tasca d'examinar un gran conjunt de traduccions gairebé manualment. Algunes eines de recerca desenvolupades recentment es centren en la de classificació d'errors. Per contra, *t*SEARCH ofereix un nou enfoc proporcionant mecanismes per fer cerques complexes sobre un col·lecció de traduccions avaluades amb un conjunt heterogeni de mesures d'avaluació de la traducció, el qual permet detectar patrons que poden ajudar als desenvolupadors a millorar la qualitat de la traducció. Així, la tasca d'anàlisi d'errors es pot fer sobre un subconjunt específic de traduccions, convertint-se en un treball més eficient i factible.

Per dur a terme aquesta proposta, *t*SEARCH es desenvolupa sobre ASIYA, una eina lliure per la avaluació automàtica de la traducció, la qual proveeix d'un gran conjunt de mesures d'avaluació. Els resultats de l'avaluació d'ASIYA són utilitzats per *t*SEARCH per processar la consulta de l'usuari. El motor de cerca ofereix un llenguatge de consulta ric i flexible que permet trobar traduccions que compleixen una combinació de característiques numèriques i estructurals. A més, el seu disseny de base de dades permet un temps de resposta ràpid per totes aquelles consultes sobre un banc de proves d'un tamany raonable. També hem dissenyat i desenvolupat una interfície web fàcil d'utilitzar i amigable que fa possible l'accés interactiu als resultats de la consulta.

Acknowledgements

This Master thesis has been partially funded by the Spanish Ministry of Education and Science (OpenMT-2, TIN2009-14675-C03), the European Community's Seventh Framework Programme under grant agreement number 247762 (FAUST, FP7-ICT-2009-4-247762) and the EAMT Sponsorhip of Activities: Small research and development project, 2012.

Contents

Chapter 1

Introduction

1.1 Motivation

This master thesis has been developed in the area of natural language processing (NLP), specifically, within machine translation (MT) field. Machine translation is a technology used to support human language translation aiming to fully translate the input texts. Thus, MT is focused on producing fast and low cost translations, significantly reducing human effort, time and costs.

During the development of a machine translation system, MT developers follow several stages in order to guarantee that their improvements over the system increase the quality of the system output.

An MT system development cycle is shown in Figure 1.1. Basically, the cycle consists of indentifying and analysing system's weaknesses at *error analysis* stage; improve the system at *system refinement* stage according to the drawbacks detected; and, eventually, compare its performance with other systems at the *evaluation* stage.

The evaluation methods play an important role in the cycle. The characteristics of the measures used during the evaluation stage determine the nature of the errors addressed during the refinement. Note that these evaluation measures are authomatic, since the cost of doing it manually is too high for a development cycle.

Since the number of correct translations is not unique, a common MT evaluation setting uses several metrics to assess the quality of an automatic translation in comparison to a human translation. For instance, a sentence with high scores in most automatic evaluation measures has a strong likelihood of being a good translation. The MT community and

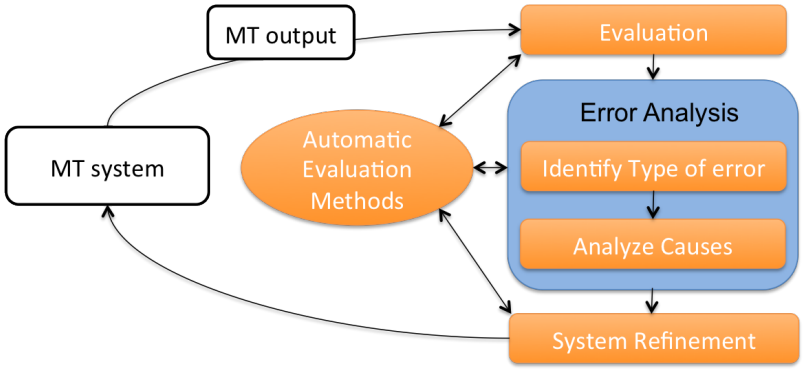

Figure 1.1: Development cycle of MT systems

related areas have developed dozens of measures and hundreds of metric variants that evaluate different quality aspects of translations. They rely on different similarity assumptions and some of them are continuously on development. `BLEU` [Papineni et al., 2002] is the most popular measure in the family of lexical measures, actually it has been widely accepted as the standard for years (See Section 2.2 for further details regarding MT evaluation). `BLEU` accounts for the number of word n-grams in common between the system translation (candidate) and the human translation (reference). Giving the following example,

Source: *Los que quieren pasar en Špindlerův Mlýn una Nochevieja según sus expectativas, no deberían dudar más en reservar la estancia.*

Reference: Anyone who wants to spend New Year's Eve in Špindlerův Mlýn just how they want, had better start thinking about booking soon.

Automatic Translation: Those who want to move in a according to their expectations, should not hesitate more in reserve the stay.

the lexical matching-type of metric for this automatic translation has a low score which denotes that the reference and the candidate translation are quite different and, thus, the translation is not good enough. Having the scores given by several measures, MT developers can identify specific weak points of their systems. This is mainly the purpose of the error analysis stage and it becomes a tedious task when a testbed of thousands of sentences and several systems are involved.

MT developers tackle the analysis of the translation quality by inspecting a large set of translations almost manually. Even though there exist some tools that help in this wearisome task, as is described in Section 2.3, they address the problem partially, and

most of the times they are designed for a specific target language or a limited set of metrics.

Considering the problem aforementioned and the importance of the error analysis of machine translation system's output, we focus our work on the development of a tool that, taking advantage of the evaluation methods and results provided by the Asiya toolkit[1] ([Giménez and Màrquez, 2010]), allows MT developers to analyse automatic translations efficiently.

1.2 Objectives

The main goal of this project is to develop and provide the community a tool and a web-based interface that facilitates the qualitative analysis of automatic translations. Thus, this tool should be able to:

1. analyse automatically an MT output,
2. search for a set of automatic translations that meet specific criteria expressed using a flexible query language,
3. present search results in a friendly manner along with useful additional information related to the query specified at the previous step,
4. let the user to export the search results in an structured and machine readable format, thus enabling its storage and further processing.
5. be easily extensible, since the query language needs to be adapted to future needs,
6. get results within a reasonable response time, taking into account that queries could be very complex and the datasets could be huge with a high number of measures applied at the evaluation stage.

1.3 How to Read this Document

This document is organized as follows. In *Chapter 2*, which details background information, we distinguish three different sections: a description of Asiya, a key ingredient in the development of this project; an introduction to machine translation evaluation; and finally, we outline previous works related to automatic error analysis.

[1] Asiya is an automatic MT evaluation suite available at `http://asiya.lsi.upc.edu/`.

Regarding *Chapter 3*, it gives a detailed description of the *t*SEARCH architecture and the main functionalities of the tool with special emphasis on the three main modules implemented: the data loader, the query parser and the search engine. The information related to the database, data representation and the reasons for the technical decision to choose a *NoSQL* solution are discussed in the last section of the chapter.

Concerning *Chapter 4*, it reports the experimentation done over different adjustments and concludes with the best configuration for our system in terms of performance.

Chapter 5 shows the planification of the project and discusses the deviations that has been produced.

The final remarks in *Chapter 6* present a discussion about our work followed by the conclusions and future work.

Chapter 2

Background

This documentation uses several concepts that need to be defined for the better understanding of the reader. First of all, *testbed* refers to all the files needed for the evaluation of automatic machine translations: the source file (i.e., the file containing all segments that need to be translated); the automatic translation of each one of the systems involved (also known as candidates); and one or several reference translations (i.e., the human translation of the given source text).

During the MT evaluation detailed in Section 2.2, several *measures*[1] are used in order to assess the quality of system translations. Thus, automatic metrics are able to estimate different aspects of the quality such as fluency, adequacy, the discourse structure and the complexity or simplicity of the syntactic structure. This proposal counts with Asiya, an automatic MT evaluation toolkit introduced in Section 2.1, for the evaluation of a testbed. Asiya uses several parsers (i.e., syntactic analysers) in order to generate all the information regarding linguistic elements such as lemmas, parts-of-speech, dependency relations, named entities, semantic roles and syntactic structures. For instance, giving the following sentence:

Not only those of Telesquíes in Krkonoie are concerned about the runways without snow.

the following are examples of linguistic elements: the *lemma* be (are), the *named entity* of type *location* (Krkonoie). Furthermore, different *parts-of-speech* are identified such as determinants (the), nouns (snow), verbs (are) and so; and also the conjunction phrase (Not only) and noun phrase (the runways) as a *syntactic structures*. Moreover, "the" has a *dependency relation* of determinant over "runways". Finally, regarding *semantic roles*, i.e., the relationship that a syntactic constituent has with a predicate, one of the arguments of the verb "concern", is "about the runways without snow".

[1] Along this document, we use *metric* as a synonymous of *measure*.

2.1 The Asiya toolkit

ASIYA[2] is an automatic MT evaluation suite [Giménez and Màrquez, 2010], which IQMT [Giménez and Amigó, 2006] predeces, that has been developed at the GPLN group which belongs to the TALP Research Center at Universitat Politècnica de Catalunya. ASIYA offers a rich and heterogeneous set of metrics and meta-metrics that rely on different similarity principles (such as precision, recall and overlap) and operate at different linguistic layers (from lexical to syntactic and semantic). The linguistic-pendendent metrics are also available for several language such as English, Spanish, Catalan, Arabic, Czech, French, German or Romanian. For instance, there are more than 800 different variants available to evaluate Spanish to English translations, which is the largest group.

ASIYA can be used through the command line[3] or via the ASIYA ONLINE INTERFACE[4] [Gonzàlez et al., 2012]. In both cases, the user must provide a complete set of files that constitute the testbed: the source file, having the original text in the original language; a set of system files, having the candidate translations given by each translation system, and a set of reference files, one for each gold translation. Each file should contain a list of segments that are extracted from one or several documents.

The ASIYA ONLINE INTERFACE offers the chance to upload a tesbed and to evaluate it remotely. It displays the evaluation results in a graphical interface that consists of three different views. The first one shows the evaluation output on an interactive table at three granularity levels: system, document and segment. The second view allows MT developers to generate interactive plots combining different metrics and systems. Finally, the last view displays the linguistic information related to the translations. It also lets to visualize the dependency and constituency trees corresponding to each translation.

The web-based tool presented in this work is build on top of ASIYA. It gathers the information calculated during the evaluation and fills a database for further analysis. These data includes the metric scores and the output of the linguistic processors such as syntactic parsing(e.g., Berkeley and MALT) or shallow annotations (i.e., word level part-of-speech and lemma labels).

[2]Asiya was the Israelite wife of the Pharaoh who adopted Moses after her maids found him floating in the Nile river (`http://en.wikipedia.org/wiki/Asiya`).

[3]The technical manual [Giménez and Gonzàlez, 2013] provides detailed information about installation and use of ASIYA.

[4]`http://asiya.lsi.upc.edu/`

2.2 MT Evaluation

The automatic evaluation of machine translation quality aims to compute the similarity between an automatic translation and a set of references (i.e., manually-produced translations). To do so, automatic machine translation evaluation uses automatic measures that assess the quality of an automatic translation in comparison to a human translation (or reference). They assess different quality aspects of translations (e.g., fluency, adequacy, grammaticality, and so), aiding the MT development and tuning. One of the main challenges of MT evaluation is to compare at the level of semantic equivalence, i.e., a system's output at a reference convey the same meaning.

There are two different approaches to automatic MT evaluation. The first one is based on lexical similarity and the second one suggest exploiting linguistic information beyond the lexical level to increase robustness.

Regarding lexical measures (also called n-gram-based or string-based measures), they compute the lexical similarity between automatic translations and a set of human translations. As aforementioned, `BLEU` [Papineni et al., 2002] is the most popular measure in this family of measures and has been widely accepted as the standard for years. `BLEU` accounts for the number of word n-grams in common between the translation and human reference. However, lexical similarity is not able to discriminate whether translation and reference texts convey the same meaning [Culy and Riehemann, 2003], [Coughlin, 2003], [Callison-Burch et al., 2006], since lexical-based metrics are not able to capture the syntax or semantic structure of sentences.

Concerning linguistically-based measures, they were suggested to cope with the issues regarding lexical similarity and, actually, they have shown higher correlation with human assessments. Nevertheless, they are not widely used by MT developers because of their high computational cost and its strongly dependency on parsers trained on specific corpora that may fail when applied to generally noisy text output. Some of the linguistically-based measures such as ROUGE, METEOR and TER have used additional linguistic knowledge to extend the reference lexicon [Snover et al., 2010, Denkowski and Lavie, 2010]. In contrast, HyTER, an edit-distance based metric, was proposed to avoid the problem of comparing to a very reduced set of references [Dreyer and Marcu, 2012].

On the other hand, there are several proposals that compare directly the syntactic and semantic structure of automatic translation and reference. So, for instance the work described in *The Naming of Things and the Confusion of Tongues: an MT Metric* [Reeder et al., 2001] introduces a similarity measure based on named entity overlap. Other works presented several measures based on edit distance over parts of speech [Popović and Ney, 2007] or defined several syntactic measures based on comparing head-word dependency chains and constituent subtrees [Liu and Gildea, 2005] .

2.3 Automatic Error Analysis

The research works in the field focused on the error analysis stage address (semi)-automatic error analysis from different perspectives and provide tools for graphical visualization of errors, yet having very limited functionalities and coverage of languages.

Currently, there do not exist other freely available automatic tools for aiding MT evaluation tasks comparable to the *t*SEARCH tool.

In the following, we describe some of these works that are partially related to this field of study.

Regarding the error analysis task, most of the works studied create a classification of the error types. The most recent are Hjerson [Popović, 2011] and *From Human to Automatic Error Classification for Machine Translation Output* [Popović and Burchardt, 2011]. Both address automatic error classification at word level such as morphological errors, reordering errors, missing words, extra words or lexical errors. The classification is obtained by focusing on the word error rate (`WER`) [Niessen et al., 2000] and the precision and recall error rate (`PER`) [Tillmann et al., 1997]. The open source toolkit Addicter [Zeman et al., 2011] automatically detects translation errors and label them based on the reference-hypothesis alignment. So that, it aids MT developers to better understand the reasons of the errors. Afterwards, Addicter was merged with Hjerson [Berka et al., 2012] becoming a web application that displays alignments and different types of errors colored. Last but not least, less recent works such as the one in [Vilar et al., 2006], considered as the inspiration of the previous ones, proposes a method for automatic identification of various error types. The methodology proposed is language independent and tackles lexical information.

For the project purpose of finding the translations that match a combination of structural features associated to translation quality measures, we have implemented a flexible and expressive query language. Currently, there is no language defined that focuses on machine translation.

To carry out this project, we count with the ASIYA evaluation tool that is highly used by the community. Hence, we can guarantee that its output and all information generated are priceless data for MT developers at the error analysis stage. The ASIYA interface and the *t*SEARCH tool together facilitate the qualitative analysis of the evaluation results yet providing a framework to obtain multiple evaluation metrics and linguistic analysis of the translations. They also provide the mechanisms to search and find relevant translation examples using a flexible query language, and to export the results.

Chapter 3

The *t*Search tool

*t*SEARCH is a free on-line tool built on top of ASIYA that has been designed to help MT developers during their error analysis tasks. To do so, all the information generated by ASIYA at the time to evaluate a testbed, such as the input data, evaluation results or even linguistic information, is loaded to a database in order to be properly used during the analysis.

The tool includes an on-line interface that allows the user to query the database in a friendly and interactive manner. The types of queries have been designed to address the principal traits of an evaluation scheme, such as the metric scores and linguistic information. The combination of all these queries allow users to obtain the sentences in a testbed matching the set of criteria specified. In the *t*SEARCH interface, the results of the search are displayed under several views and can be exported as XML files.

This chapter describes the *t*SEARCH architecture and provides details on the main modules implemented in Section 3.1. Section 3.2 gives a technical description of the data representation, storage and data access and Section 3.3 gives an overview of the *t*SEARCH ONLINE INTERFACE.

3.1 Architecture and Functionalities

Figure 3.1 shows the *t*SEARCH system integrated with ASIYA. The *t*SEARCH business model has been implemented in Perl[1], since it is widely used for NLP purposes. Perl allows defining very powerful regular expression and counts with a huge repository of

[1] `http://www.perl.org/`

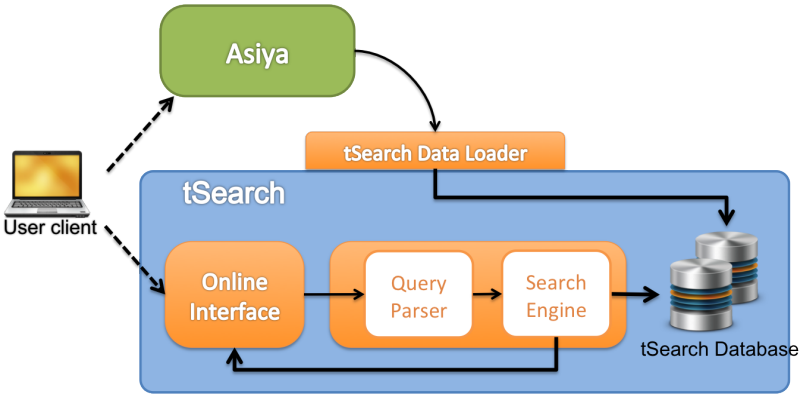

Figure 3.1: *t*SEARCH architecture

modules available for the community at the Comprehensive Perl Archive Network (CPAN)[2]. The *t*SEARCH business model consists of the parser (Section 3.1.2), the search engine (Section 3.1.4) and the database manager. The parser converts the query into a binary tree structure and then, the search engine computes the result of the query iterating over it. The results are retrieved from the *t*SEARCH database through the database manager which communicates with it. The Data Loader module (Section 3.1.1) is a key ingredient in the integration with the ASIYA evaluation toolkit, since ASIYA uses it to feed the *t*SEARCH database.

Concerning the *t*SEARCH design and implementation, it follows the model-view-controller (MVC) pattern (Figure 3.2), a software arquitecture pattern that allows the separation of concerns, thus simplifying the maintenence and future extensions of the tool. So that, it has been divided into:

- The *t*SEARCH view, it consists of three different views implemented into the *t*SEARCH ONLINE INTERFACE. The view details are given in Section 3.3.
- The controller, it is a PHP application that communicates between the view and the model. So that, it is in charge of updating the view after a user's query. It also implements functions to create the XML file needed for downloading the query results.
- The model, it is implemented in Perl and contains the *t*SEARCH data loader mentioned above, the parser and other classes such as *Metrics*, *LE* and *Utils* that define the functions of the *t*SEARCH search engine.

[2] `http://www.cpan.org/`

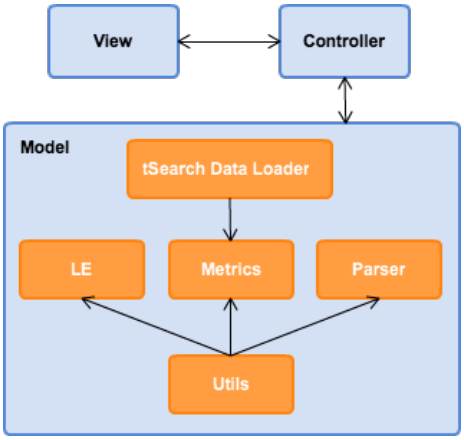

Figure 3.2: *t*SEARCH MVC model

3.1.1 tSearch Data Loader

The *t*SEARCH data loader is a Perl application programming interface (API) defined for ASIYA communication purposes. It is mainly in charge to feed the *t*SEARCH database from the evaluation results and other data generated by the application such as averages, thresholds, or percentiles values among others. Using the API functions, the insert task is made transparently to Asiya's users while a testbed is being evaluated. The insertion is done in parallel to the evaluation calculation. This way we achieve a notable reduction of the insertion run-time.

Moreover, the ASIYA ONLINE INTERFACE allows to upload the results of a previous evaluation. Thus, there is no need to evaluate the testbed again. Instead, the *t*SEARCH database is fed directly from the uploaded file.

On the other hand, the amount of data generated by ASIYA can be huge and hard to process (time consuming). For this reason, we decided to split the data according to its relevance and priorize the insertion according to the content of the user's query. In particular, we decided to defer the data related to linguistic elements and the metrics that evaluate specific types, since they are the less used. For instance, `NE-Oe(MONEY)` is not pre-calculated, since it just considers average lexical overlap between types named entities of types "MONEY"; in contrast, `NE-Oe(*)` is pre-calculated because it considers all named entities types. Thus, they are only computed on demand, i.e., when the user ask for a query with any of these elements not pre-calculated and accepts its computation. This *on-demand* task relies on the search engine described in Section 3.1.4 and improves the application performance making the user experience much satisfactory. The Section 4 details the *on-demand* experiment.

```
{
    'OR' => {
        'op2' => {
                'operator' => 'GT',
                'metric' => {
                        'value' => 'GTM-1'
                },
                'score' => '0.5'
        },
        'op1' => {
                'operator' => 'GT',
                'metric' => {
                        'value' => 'BLEU'
                },
                'score' => '0.3'
        }
    }
}
```

Figure 3.3: Giving the query `BLEU > 0.3 OR GTM-1 > 0.5`, this Figure shows the tree structure built by the parser.

3.1.2 The *t*Search Parser

This section describes the *t*Search Parser, providing details on how the query parser is implemented and describing the query language defined.

Regarding the query parser, it is responsible for interpreting the user's query, which is converted into a binary tree that is processed bottom-up. It defines rules not only for grammar validation with a context-free grammar, but also for semantic validation. For the grammar specification, which is detailed in the appendix B, we have used the Perl module *Parse-RecDescent* available at CPAN[3], since it provides powerful mechanisms to define the rules, the output and the errors that we want to be returned. If the query is well-defined according to grammar and semantic rules, the parser builds a binary tree structure where each leaf is the structure of a single part of an operation (e.g., `BLEU > 0.3` and `GTM-1 > 0.5` are singles parts of the query `BLEU > 0.3 OR GTM-1 > 0.5`) and each node is a logical connector (i.e, `AND` or `OR`). The structure of each leaf adapts to the kind of query and contains the information needed for the search engine to compute the results. Figure 3.3 displays the parser output for the query `BLEU > 0.3 OR GTM-1 > 0.5`.

Concerning the query language defined, there are available a wide range of types of queries

[3] `http://search.cpan.org/~jtbraun/Parse-RecDescent-1.967009/lib/Parse/RecDescent.pm`

such as arithmetic comparisons, statistical functions, range of values and linguistic elements[4]. These functions, which are sumarized in Tables 3.1 and 3.2, were defined taking into account the needs that some MT developers have conveyed to us, starting its development from the metric-based queries to the linguistic-based ones, which are more complex.

All query structures can be applied at segment (a full or part of sentence that is translated at once), system or document level and combined in the same query with the use of logical operators and parenthesis to denote priority. Furthermore, the user can define groups of metrics, systems and documents and even apply a combination of them in order to get more specific results. Further details are given in the following subsections.

Metric-based queries

The arithmetic comparison queries are those that use comparison operators to determine the criterion for the set of sentences that are requested. The common comparison operators `gt, lt, ge, le, eq` are supported and also the corresponding mathematical symbols `>, <, >=, <=, =`.

The most basic queries are those related to segment level scores, i.e., obtain all segments scored above/below a value for a concrete metric. For instance, `BLEU` $>$ `0.4` and `BLEU gt 0.4`, that are both correct and equivalent queries.

Statistical functions allow to use statistic variables as values, e.g., obtain the segments scored in the fourth quartile of BLEU. The maximum, minimum, average, median and percentile values of each metric are precalculated and saved into the `MAX`, `MIN`, `AVG`, `MEDIAN` and `PERC` variables, respectively. The thresholds and quartiles (TH,Q) are calculated at run-time based on percentiles. Moreover, the (`MIN`) and (`MAX`) functions may be used not only to know about the minimum and the maximum value for an specific criteria respectively, but also in order to obtain all segments in the testbed, e.g., `BLEU ge MIN` or `BLEU le MAX`.

Regarding the threshold function, the value specified at this function corresponds to a percentage value, i.e., `BLEU` $>$ `TH(20)`, which gets all segments with a `BLEU` score above the bottom 20%.

*t*SEARCH has also implemented a type of query that confines the result between two boundaries. For instance, the query `BLEU IN [0.3, 0.5)` matches examples with `BLEU` scores between 0.3 included and 0.5 excluded. Note that the use of parentheses excludes its boundary value from the range. Moreover, threshold functions can be used as a limit value, e.g., `BLEU IN [TH(20), TH(30)]`.

[4]Note that we use the `BLEU` measure in most of the examples, but it can be substituted by any other evaluation measure

Query Types	Examples	Comments
Arithmetic Comparison	(1) BLEU > 0.4 (2) BLEU > AVG (3) BLEU > TH(40) (4) BLEU ge MIN (5) BLEU le MAX (6) upc:BLEU > AVG (7) mydoc:BLEU > AVG (8) upc:mydoc:BLEU > AVG (9) BLEU > upc:AVG	Comparison operators: “>” (gt), “>=” (ge), “<” (lt), “<=” (le) and “=” (eq) (2) threshold parameter is a percentage (3) and (4) obtain all segments (6) Get segments from “upc” system (7) Get segments from document “mydoc”
Range of Values	(1) BLEU IN [0.2, 0.3) (2) BLEU IN Q(4) (3) BLEU IN PERC(2,10) (4) BLEU IN (TH(20),TH(40)) (5) upc:BLEU IN Q(1) (6) mydoc:BLEU IN Q(1) (7) upc:mydoc:BLEU IN Q(1) (8) BLEU IN upc:Q(1)	(2) Q(n), 1<=n<=4 (3) PERC(n,M) M>1, 1<=n<=M
Sistem comparison	(1) upc:BLEU > dfki:BLEU	

Table 3.1: *t*SEARCH metric-based queries

The semantic parser is in charge to check whether a query that is grammatically valid is also accepted semantically. Thus, `BLEU IN [0.8, 0.2)` is grammatically correct, but not semantically because the boundaries are in decreasing order. In this case, the parser returns an error warning against this incorrect order.

Concerning other functions that returns a range of values, the percentile function `PERC(n,M)`, where `M` $>$ 1 and `1` $<=$ `n` $<=$ `M`, splits the set of values in `M` parts and takes them all in the n^{th} part. Therefore, `BLEU IN PERC(20,100)` meets `BLEU` scores in the 20^{th} percentile. Quartiles are similar to percentiles, but it is assumed that the set of values is split in `4` parts. Hence, it accepts values from `1` up to `4`, e.g., `BLEU IN Q(3)`.

Thresholds and quartiles are calculated at run-time using the percentile values. In this way we avoid to insert redundant data in the database. For example, the first quartile corresponds to the 25^{th} percentile which had been already pre-calculated and stored in the database.

The queries defined above are applied at segment level. However, applying them at system and document level is as easy as specifying the system and/or document. For instance, the query `google:doc1:BLEU` $>$ `AVG` retrieves the results concerning only the segments from the `google` translations that belongs to the `doc1` document with a `BLEU` score above the (global) average. Furthermore, you may also ask for the query `BLEU` $>$ `google:doc1:AVG` which gets all segments (from all the translation systems) with a `BLEU` score above the average `BLEU` score of the translations that belong to the `doc1` document of the `google` system. This structure is the same used for groups detailed next. It is also worth pointing out that changing the order of the factors does not change the result interpretation, i.e., "`google:doc1:BLEU` $>$ `AVG`" and "`doc1:BLEU:google` $>$ `AVG`" are both correct and return the same result. As it can be noted, we use the punctuation mark `':'` in order to separate the system, document and metric identifiers.

We defined also a query that makes possible the system comparison. Thus, given an evaluation measure, it allows comparing its score between systems. For instance, the query `upc:BLEU` $>$ `dfki:BLEU` returns those translations from the `upc` system having a higher `BLEU` score than the translation of the same segment made by the `dfki` system.

Linguistic-based queries

Concerning linguistic-based queries (an schematic summary is shown in Figure 3.2, we have defined a powerful set of query structures to identify specific linguistic elements such as lemmas, parts-of-speech, dependency relations, named entities, semantic roles and even syntactic structures. They aim at aiding the analysis of translations errors of specific phrase structures, lemmas or grammatical categories. In contrast to the metric-based queries, this linguistic information can not be obtained from the scores' values, but from data generated

by taggers, lemmatizers, parsers and other analysers that ASIYA uses in order to compute the syntax-wise measures. ASIYA uses the SVMTool [Giménez and Màrquez, 2004], BIOS [Surdeanu et al., 2005], the Charniak-Johnson [Charniak and Johnson, 2005] and Berkeley constituency parsers [Petrov and Klein, 2007], and the MALT [Nivre et al., 2007] dependency parser, among others.

The simplest queries are those that request a single item of shallow parsing (`SP`), lemma (`lemma`), part-of-speech (`pos`), name entities (`NE`) or constituent parsing outputs (`CP`). These functions allow also to ask for n-grams, for instance, the query `LE[SP(NP, VP, PP), lemma(be), pos(NN, adj), NE(ORG)]` matches those segments that:

- have a structure of noun phrase (e.g., *who*), verb phrase (e.g., *want to move*) and prepositional phrase (e.g., *in*),
- have the sequence of a noun and an adjective,
- contain any of the forms of the lemma be (e.g., *is*, *was*, *be*) and
- contain a name of an organization (e.g., *Acme corp.*).

The dependency parsing function allows to specify a (syntactic) dependency relation as a criterion. Its simple structure is as follows `LE[DP(POS1, rel, POS2)]` where `POS1` refers to the part-of-speech of the modifier (or dependant), `rel` is the relation name identifier and `POS2` is the part-of-speech of the word which is related to (the *head*). In addition, it is possible to specify a chain of relations, e.g., `LE[DP(N,nsubj,V,dep,V)]`.

The use of the asterisk symbol is a key ingredient in the queries described above. MT developers can use it to allow any value, hence they obtain also information regarding all possibilities that can substitute the asterisk. For instance, the query `LE[SP(NP, *, PP)]` matches segments having the phrases secuences {`NP, VP, PP`}, {`NP, ADJP, PP`}, {`NP, ADVP, PP`}, {`NP, PP, PP`} or {`NP, NP, PP`}, among others, where `NP` refers to noun phrase, `VP` to verb phrase, `ADJP` to adjectival phrase, `ADVP` adverbial phrase and `PP` prepositional phrase. Furthermore, in the dependency parsing queries, the asterisk may help the user to know all possible arguments, e.g., `LE[DP(*,nsubj,V)]` in subject relation with the verb, or also to know about all possible relations of verb modifiers, e.g., `LE[DP(*,*,V)]`. These examples can also be applied for dependency parsing queries with a chain of relations, e.g, `LE[DP(*,nsubj,*,*,V)]`. When all arguments are asterisks (i.e., `LE[DP(*,*,*])`, the system returns all the existing dependency relations in the testbed.

Finally, the last function implemented is the one related to semantic roles. This function is useful to select the sentences that contain a specific verb and its list of arguments specified in the query. The comparison between the semantic roles of a segment and its references may help MT developers to detect flaws in their systems to translate specific arguments for

Query Types	Examples	Comments
***N*-grams**	(1) LE[SP(NP,VP,PP)] (2) LE[SP(NP,*,PP)]	(1),(2) Shallow Parsing function
	(3) LE[CP(NP,PP)] (4) LE[CP(*,PP)]	(3),(4) Constituent Parsing function
	(5) LE[lemma(be)] (6) LE[lemma(be,*,to)]	(5),(6) Lemma function
	(7) LE[pos(JJ)] (8) LE[pos(DT,JJ,*)]	(7),(8) Part-of-speech function
	(9) LE[NE(ORG)]	(9) Name entity function
Semantic Roles	(1) LE[SR(ask,A1,AM-TMP)] (2) LE[SR(*,A1,AM-TMP)] (3) LE[SR(**,A1,AM-TMP)]	(2) any verb with both args (3) all args in a sentence
Dependency Relationships	(1) LE[DP(N,nsubj,V)] (2) LE[DP(*,*,V)] (3) LE[DP(*,nsubj,*)] (4) LE[DP(N,nsubj,V,dep,V)] (5) LE[DP(*,nsubj,*,*,V)] (6) LE[DP(*,*,*)]	(4),(5) Chain of relations (6) Gets all relations
	(7) LE[DP(*,*,*),pos(DT,JJ)]	(7) conjuntion of two LE functions
	(8) upc:LE[DP(*,*,*)] (9) mydoc:LE[DP(*,*,*)] (10) upc:mydoc:LE[DP(*,*,*)]	

Table 3.2: *t*SEARCH linguistic-based queries

specific (groups of) verbs. The structure of this query is `LE[SR(`V`,` Arg_1`,` Arg_2`, ...,` Arg_n`)]` where the first item is a verb and the rest are the verb arguments. The verb can be replaced by `*` or `**`, e.g., `LE[SR(*, A1, A2, AM-TMP)]`. Here, the use of asterisk has different meaning than before. On the one hand, one asterisk specify that all arguments have to belong to the same verb in order to be matched. On the other hand, two asterisks in a row means that all arguments have to be in a sentence, regardless they belong to different verbs.

Linguistic-based queries can be combined together with metric-based queries. For instance, `LE[CP(ADJP)] AND CP-Op(*) < AVG` returns all segments that has adjectival phrases and the `CP-Op(*)` score obtained is below the average. Furthermore, as in the metric-based queries, one can also apply the linguistic elements-based queries at document and/or system level, e.g., `upc:doc1:LE[DP(*,nsubj,*)]`. Here, we use the punctuation mark `':'` not only to separate the system and document, but also to identify the linguistic query which is linked to.

Group Creation and Complex Queries

Groups of systems, documents and even metrics can be created at the on-line interface. The purpose of this feature is to facilitate the comparison between types of systems (e.g. statistical vs. rule-based), metrics (e.g., lexical vs. syntactic) or even groups of documents that are from different domains. We have to take into account that this comparison is a hard task if the user does not have any tool designed for this purpose.

The next example shows a comparison between lexic and syntactic metrics. Defining the lexical group `LEX` with the metrics `BLEU` ([Papineni et al., 2002]) and `NIST` ([Doddington, 2002]), and the syntactic group `SYN` with the metrics `DP-Or(*)`[5] and `SP-Op(*)`[6], the following query

```
(apertium:LEX > AVG AND apertium:SYN < AVG)
                        OR
(apertium:SYN > AVG AND apertium:LEX < AVG)
```

returns all segments belonging to the rule-based system `apertium` that have good scores for lexical-based metrics, bad scores for the syntactic ones, or viceversa.

Note that the structure defined at the previous point for systems, metrics and documents, is the same at group-level, e.g., `news:LEX:RBMT > AVG OR news:RBMT:LE[pos(DT,JJ)]`, where `RBMT` is a group of rule-based systems and `news` is the identifier for one of the documents in the testbed.

[5]Overlap between words ruled by grammatical relationships.

[6]Overlap over all parts of speech.

Query Types	Description	Error	Message
Percentile Function	PERC(n,M), M>1, 1<=n<=M	n < 1 n > M	Second argument should be > 0 Left value in PERC should be less or equal than right value
Range of Values	METRIC IN (x, y)	x > y	Left value in IN should be less or equal than right value
Quartile Function	Q(n), 1<=n<=4	n < 1 or n > 4	Quartile value must be between 1 and 4

Table 3.3: Semantic error checking. This Table shows the messages corresponding to the semantic errors defined.

3.1.3 Error Checking

The *t*SEARCH parser has also implemented a query error handling that aims to help the user to identify and solve the error in their query. Thus, it returns feedback not only for grammatical errors, but also for the semantic ones.

Regarding the semantic errors, there are some messages classified according to the type of query (Table 3.3). Concerning the grammatical errors, the parser returns some examples of correct queries replacing the error for what was expected. For instance, the query `BLEU > X` is grammatically incorrect, since `X` does not correspond to any of the rules defined. Thus, the parser returns examples of correct arithmetic comparison queries (Figure 3.4).

3.1.4 The Search Engine

The search engine aims to compute and return the result of a query. To do so, it recursively iterates over the binary tree generated by the parser bottom-up. During this process, the search engine gets the partial results that correspond to each leaf and prepares them for processing the tree nodes bottom-up until the root (the final result). Finally, the result structure is converted into a javascript object notation (JSON) in order to transmit easily the data to the controller.

As mentioned previously, the *t*SEARCH database stores not only information from the scores calculated by ASIYA, but also statistical information related to the average, median,

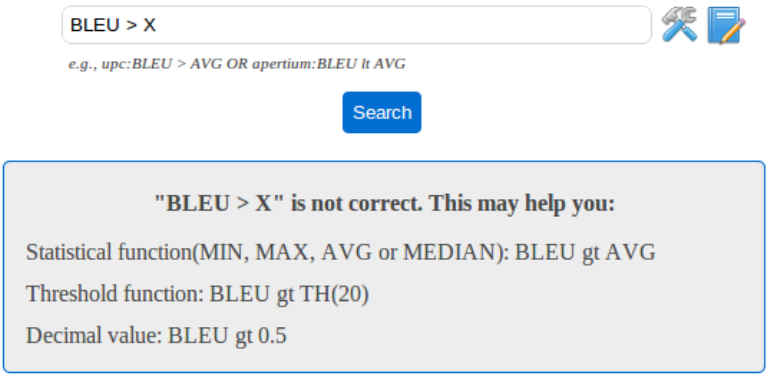

Figure 3.4: Grammatical error checking. This Figure shows the error message corresponding to the incorrect query `BLEU > X`.

minimum, maximum and percentiles values for each metric, system and document. This data storage decision is made to access to the data in constant time for many standard queries and is related to the database technology chosen which is further discussed in Section 3.2. However, on a query with a group defined (i.e., group of metrics, documents, systems or even a combination of them), if the information above mentioned is required, it has to be computed at run-time by the search engine.

As mentioned in Section 3.1.1 the data not pre-calculated by the *t*SEARCH Data Loader, i.e., related to linguistic elements and unusual metrics, e.g., `NE-Oe(LOC)`[7], is computed by the search engine. Once the information required is calculated, it is stored into the *t*SEARCH database and remains accessible for other queries (i.e., it's calculated only once). This is a decision made in order to improve the user experience and is justified in the experiments detailed in Section 4. For instance, let consider the following query: `LE[SP(NP,*,PP)] AND (BLEU > AVG OR NE-Oe(LOC) > AVG)`. The first time the search engine process such a query, it processes and inserts to the *t*SEARCH database all the information regarding the shallow parsing annotations and the metric `NE-Oe(LOC)`.

3.2 Data Representation, Storage and Access

The amount of data generated by ASIYA that has to be loaded to the *t*SEARCH database can be extremely large for those testbeds with thousands of sentences and especially when linguistic-wise measures have been calculated (along with their parsing results). Thus, we

[7]NE-Oe(LOC) reflects lexical overlap between name entities of type 'LOC' (i.e., location).

adopted a *NoSQL* database solution (also known as *not only SQL*). Concretely we use Apache Cassandra[8], which is developed in Java and deals successfully with huge amounts of data. This technology allows storing the data in the way that is needed by the query system, which guarantees a direct access to the desired results.

It is worth noting that there is no similarity between *NoSQL* databases and the traditional relational database management system model (`RDBMS`). The data representation and also the data to be stored is totally different in each of these technologies due to they focus on different kind of problems. We chose a *NoSQL* solution not only taking into account the current needs, but also considering the minor impact on *t*SEARCH due to future new requirements, e.g., adding information from new avaluation methods or new queries. Details on future work are given at Section 6.1. Furthermore, the complexity and variability of the data that has to be stored into the database would complicate considerably the design and maintenance of an `RDBMS` solution.

Cassandra implements a *column family* (CF) structure, where a unique key identifies each row and their values are a set of columns. The set of column families are created in a *key space* scope where settings such as the distribution or replication of the data are defined. The *t*SEARCH data model consists of three CFs: *scores*, *statistics* and *linguistic elements* as shown in Figures A.1.1, 3.5 and 3.6, respectively. The *scores* CF contains the data related to metrics and scores values which are needed for the more basic metric-base queries, e.g., `BLEU > 0.7`. Other metric-based queries require statistical information data such as the minimum, maximum, average, median and percentiles values for each metric evaluation, which is stored at the *statistics* CF. Finally, the linguistic information such as dependency relations, parts-of-speech, lemmas or grammatical categories required for linguistic-based queries are stored in the *linguistic_elements* CF.

The more data is pre-calculated, the better performance *t*SEARCH has at run time to compute the query results. Thus, all the above information is also calculated at segment, system, document level and even at system-document level. As aforementioned, having the data stored as is needed by the *t*SEARCH tool ensures that we can get it in constant time. For example, the data related to the segments that meet the query `BLEU > 0.8` are at the *scores* CF with the metric *BLEU* as a part of the key and the value *0.8* as a column.

3.2.1 Distributed Data Nodes

The Cassandra technology is based on a distributed architecture of data nodes. Despite one of the recommended configuration is a distributed multi-node cluster, currently the public version of *t*SEARCH has only one node because of our resources limitation. The Cassandra configuration applied for *t*SEARCH application considers that the task of adding

[8] `http://cassandra.apache.org`

CF keys	CF values			
TS key + BLEU	**0.0**	...	**0.8**	**1.0**
	$s_1\{d_2\{seg_2\}\}$	...	$s_1\{d_1\{seg_6\}\}, s_2\{d_1\{seg_7\}\}$	$s_1\{d_2\{seg_8\}\}$
TS key + BLEU+upc	**0.0**	...	**0.8**	**1.0**
	$upc\{d_1\{seg_6\}\}$	...	$upc\{d_1\{seg_6\}, d_2\{seg_23\}\}$	$upc\{d_1\{seg_1\}\}$
TS key + BLEU+news	**0.0**	...	**0.8**	**1.0**
	$s_1\{news\{seg_2\}\}$	...	$s_1\{news\{seg_6\}\}$	$s_3\{news\{seg_2\}\}$
TS key + BLEU + upc +news	**0.0**	...	**0.8**	**1.0**
	$upc\{news\{seg_1\}\}$	...	$upc\{news\{seg_4, seg_6\}\}$	$upc\{news\{seg_8\}\}$

Table 3.4: *scores* Column Family. The *TS* (testsuit) key is the concatenation of the user and the testbed identifier. Each cell, where $seg_1..seg_n$ are segments, $d_1..d_n$ are documents and $s_1..s_n$ are systems, contains information related to those segments matching a column family key and a score value.

CF keys	CF values						
TS key + BLEU	**MIN**	**MAX**	**AVG**	**MEDIAN**	**PERC(1)**	...	**PERC(100)**
	0.0	1.0	0.83	0.87	0.0-0.2	...	1.0-1.0
TS key + BLEU+upc	**MIN**	**MAX**	**AVG**	**MEDIAN**	**PERC(1)**	...	**PERC(100)**
	0.1	1.0	0.6	0.55	0.1-0.33	...	0.9-0.95
TS key + BLEU+news	**MIN**	**MAX**	**AVG**	**MEDIAN**	**PERC(1)**	...	**PERC(100)**
	0.3	1.0	0.8	0.76	0.3-0.5	...	1.0-1.0
TS key + BLEU + upc +news	**MIN**	**MAX**	**AVG**	**MEDIAN**	**PERC(1)**	...	**PERC(100)**
	0.3	1.0	0.6	0.7	0.3-0.35	...	1.0-1.0

Table 3.5: *statistics* Column Family. The *TS* (testsuit) key is the concatenation of the user and the testbed identifier. Each cell contains the value(s) matching a column family key and a statistical function, e.g., the `AVG BLEU` value in the current testbed evaluated is `0.83`.

CF keys	CF values			
TS key + SP	**NP**	**VP**	**PP**	...
	$s_1\{d_2\{seg_2\}\}$	$s_1\{d_1\{seg_6\}\}, s_2\{d_1\{seg_7\}\}$	$s_1\{d_2\{seg_8\}\}$	...
TS key + CP	**ADJP**	**ADVP**	**NP**	...
	$s_2\{d_3\{seg_1\}\}$	$s_3\{d_1\{seg_4\}\}, s_4\{d_1\{seg_6\}\}$	$s_3\{d_1\{seg_1\}\}$	...
TS key + DP	**D_nsubj_V**	**C_cc_V**	**N_pobj_I**	...
	$s_1\{d_1\{seg_3\}\}$	$s_2\{d_1\{seg_6\}\}$	$s_3\{d_3\{seg_1\}\}$	...
TS key + SR	**A0**	**AM-TMP**	**AM-LOC**	...
	$s_5\{d_1\{seg_4\}\}$	$s_6\{d_3\{seg_1\}\}, s_2\{d_1\{seg_5\}\}$	$s_6\{d_1\{seg_4\}\}$	...
TS key + lemma	**be**	**have**	**would**	...
	$s_3\{d_3\{seg_9\}\}$	$s_2\{d_1\{seg_23\}\}$	$s_5\{d_3\{seg_7\}\}$	...
TS key + pos	**DT**	**NN**	**VBZ**	...
	$s_6\{d_1\{seg_1\}\}$	$s_4\{d_5\{seg_3\}\}$	$s_1\{d_7\{seg_2\}\}$	...
TS key + NE	**ORG**	**MISC**	**PER**	...
	$s_1\{d_1\{seg_2\}\}$	$s_4\{d_9\{seg_3\}\}$	$s_6\{d_6\{seg_3\}\}$	...
TS key + NE + upc	**ORG**	**MISC**	**PER**	...
	$upc\{d_3\{seg_7\}\}$	$upc\{d_5\{seg_7, seg_23\}\}$	$upc\{d_7\{seg_6\}\}$	...
TS key + NE + news	**ORG**	**MISC**	**PER**	...
	$s_3\{news\{seg_1\}\}$	$s_5\{news\{seg_2, seg_5\}\}$	$s_1\{news\{seg_8\}\}$	...
TS key + NE + upc +news	**ORG**	**MISC**	**PER**	...
	$upc\{news\{seg_1\}\}$	$upc\{news\{seg_4, seg_6\}\}$	$upc\{news\{seg_8\}\}$	...

Table 3.6: *linguistic_elements* Column Family. The *TS* (testsuit) key is the concatenation of the user and the testbed identifier. The column family values are linguistic elements that belong to the type of processor, e.g., shallow (SP), constituency (CP), dependency (DP), semantic (SR). Thus, each cell, where $seg_1..seg_n$ are segments, $d_1..d_n$ are documents and $s_1..s_n$ are systems, contains information related to those segments matching a column family key and a linguistic element.

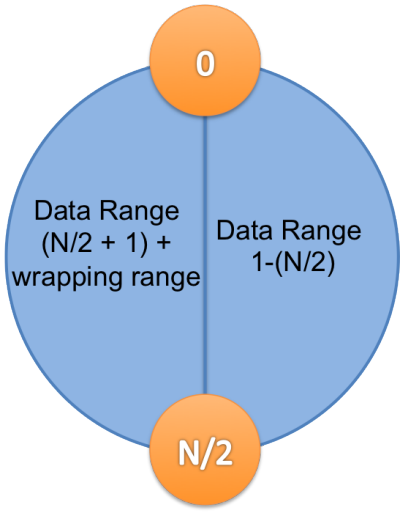

Figure 3.5: Cassandra data distribution in a two-node cluster. The data is distributed among two nodes, thus one of them contains all rows having a hash value from *0* up to *N/2* and the rest is stored in the other node.

new nodes to a cluster in the future could be accomplished smoothly. In Section 4 there is an analysis and comparison between the results obtained executing a test in both single-node and two-node cluster. Next, we discuss the main configuration points such as data partitioning or replication strategy.

Regarding data partitioning, we have set the *RandomPartitioner* strategy which uses a hash algorithm that computes an MD5 hash value for each row key. Then, having assigned an upper limit token to each node, it stores a range of rows with their hash value between the token assigned to the previous node and its own. The example in Figure 3.5 shows this data distribution in a two-node cluster. This strategy ensures that in case we had the proper Cassandra configuration of a cluster with more than one node, the data stored would be distributed evenly among them.

Concerning data replication, *Cassandra* allows this functionality in order to ensure reliability and fault-tolerance. The number of replicas and its distribution is defined with the replica strategy chosen at the time to create a *key space*. For the *t*SEARCH *key space* we set the *SimpleStrategy*. This strategy is as simple as placing the replica on its node according to its hash value. If there are more replicas, they are placed on the next nodes clockwise. This number of replicas is the *replication factor* parameter set also at the time to create a key space. It can take a value between 1 and the number of nodes running in the cluster (values above the number of nodes are not recommended because writes are

rejected). *t*SEARCH is currently limited by one node, so it has a replication factor of 1, i.e., one copy of each row. This value can be easily increased if is needded and the cluster counts with more nodes. However, taking into account the current *t*SEARCH requirements data loss is not a big deal. Note that *t*SEARCH neither allow creating profiles nor access to the testbed data after its session finishes, i.e., the data are erased after each user session.

Figure 3.7 summarizes the Cassandra data load regarding the insertion of two testbed with corpora of `100` (*corpora100*) and `1,000` sentences (*corpora1000*) . Both testbeds have results for 3 systems and are evaluated with `40` metrics.

CFs	Parameter	corpora100	corpora1000
Scores	**Columns Count**	43,730	239,084
	Data Size	∼10MB	∼53MB
Statistics	**Columns Count**	58,800	58,800
	Data Size	∼13MB	∼13MB
Linguistic elements	**Columns Count**	40,994	258,106
	Data Size	∼55MB	∼275MB

Table 3.7: Cassandra data load. For each column family, this table shows the number of columns and the data size stored resulting from the insertion of a testbed with corpora of 100 and 1,000 sentences.

As can be noted, the *linguistic_elements* CF is the one that stores more data, e.g., in the *corpora100* testbed, roughly 55MB in front of the 9MB from *scores* CF.

3.3 The On-line Interface

The *t*SEARCH ONLINE INTERFACE[9] is available via the ASIYA ONLINE INTERFACE[10] where we can found two ways to access it. The first one involves running ASIYA, hence, for a given testbed, the user uploads the source, the references, the systems translations and the selects metrics that they want to calculate (among other information). During the evaluation, ASIYA is in charge of feeding the *t*SEARCH database, so that, once the evaluation is done, the *t*SEARCH ONLINE INTERFACE appears as one of the tools that the user can run. In contrast, the second option is useful for those users who have their testsuit already evaluated. Thus,

[9]The *t*SEARCH ONLINE INTERFACE user manual is in appendix A
[10]`http://asiya.lsi.upc.edu/demo`

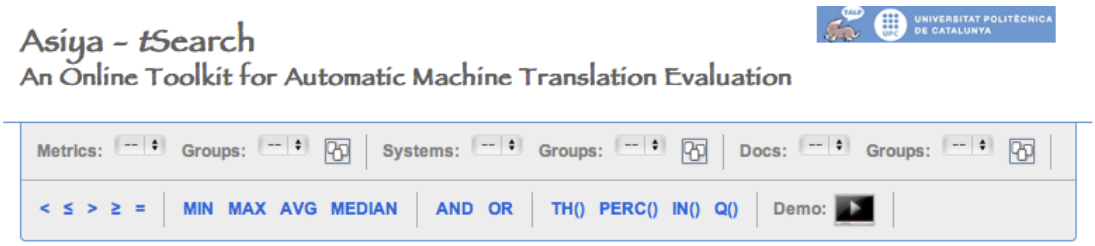

Figure 3.6: *t*SEARCH ONLINE INTERFACE toolbar. It contains all metrics, functions and operations and allow to create groups of metrics, systems and/or documents.

they can upload the compressed file that contains the ASIYA evaluation output and wait until the data is loaded to the *t*SEARCH database. Section 4 offers an extensive analysis of the loading time depending on the testbed size (i.e., number of metrics evaluated, systems and number of segments for each system). During the *t*SEARCH implementation, usability was one of the non-functional requirements, so that, we make a special effort to minimize this loading time.

The *t*SEARCH ONLINE INTERFACE counts with a helpful toolbar (Figure 3.6) that aids users to type queries. It contains the list of systems, documents and the metrics calculated. Furthermore, it contains the mechanisms to create the groups of systems, document and/or metrics to be used in the queries. There are also the symbols for all comparison operators, statistical functions and logical connectors available along with a simple example.

We have also implemented three different views: the first one shows an interactive table with all segments and the metric scores involved in the query (Figure 3.7); the second view (Figure 3.8) shows the output grouped by system and, for each system, grouped by document; and finally, the last view displays a grouping by segment where MT developers can easily compare translations not only with the source and the references, but also among them (Figure 3.9).

In addition, the interface provides a useful mechanism that displays an information box just moving the the mouse over them. This shows the source sentence, the references translations, the document to which the segment belongs to, the system that has produced the translation and also the metric scores (in the case the query includes a metric-based constrain). Concerning the linguistic-based queries, the information box appends a table for dependency parsing or/and semantic role results. Furthermore, *t*SEARCH ONLINE INTERFACE lists all linguistic elements related to the query and offers the chance to colour and highlight its occurrences in the segments in order to easily identify and compare them (Figure 3.10).

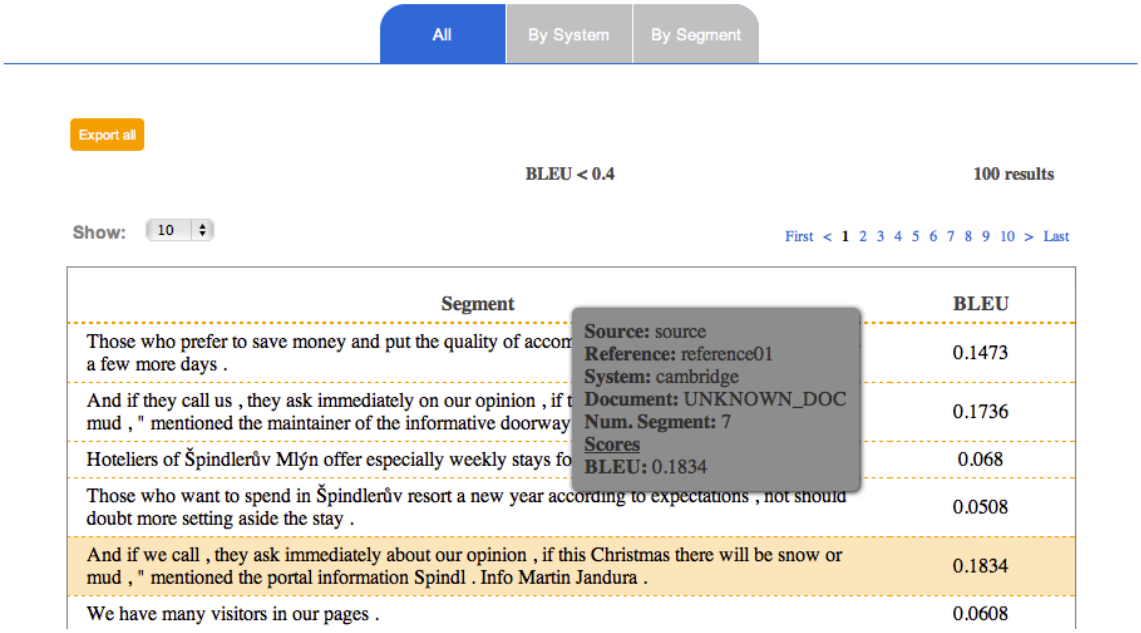

Figure 3.7: *All* view. This view displays all segments and metrics mentioned in the query. The floating box shows additional information regarding a segment, which appears moving the mouse over the text.

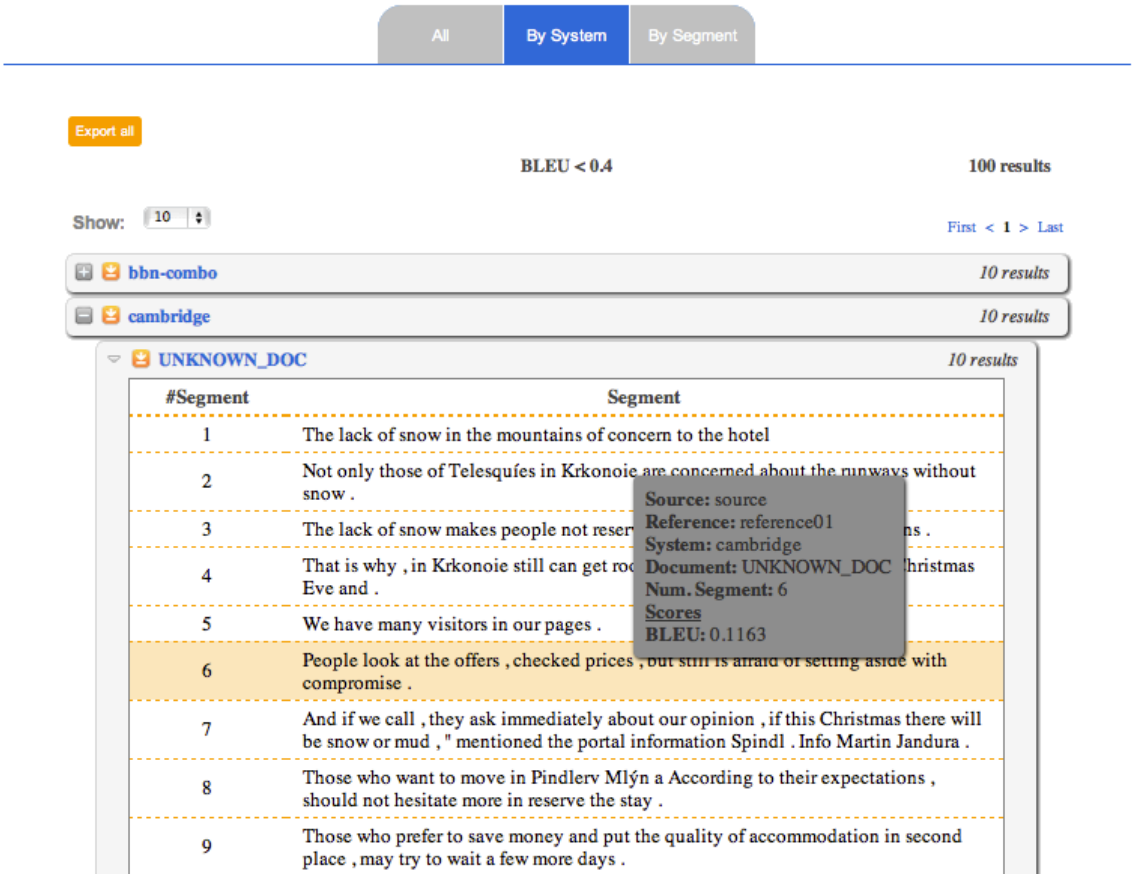

Figure 3.8: *By System* view. This view displays all segments grouped by system and document. The floating box shows additional information regarding a segment, which appears moving the mouse over the text.

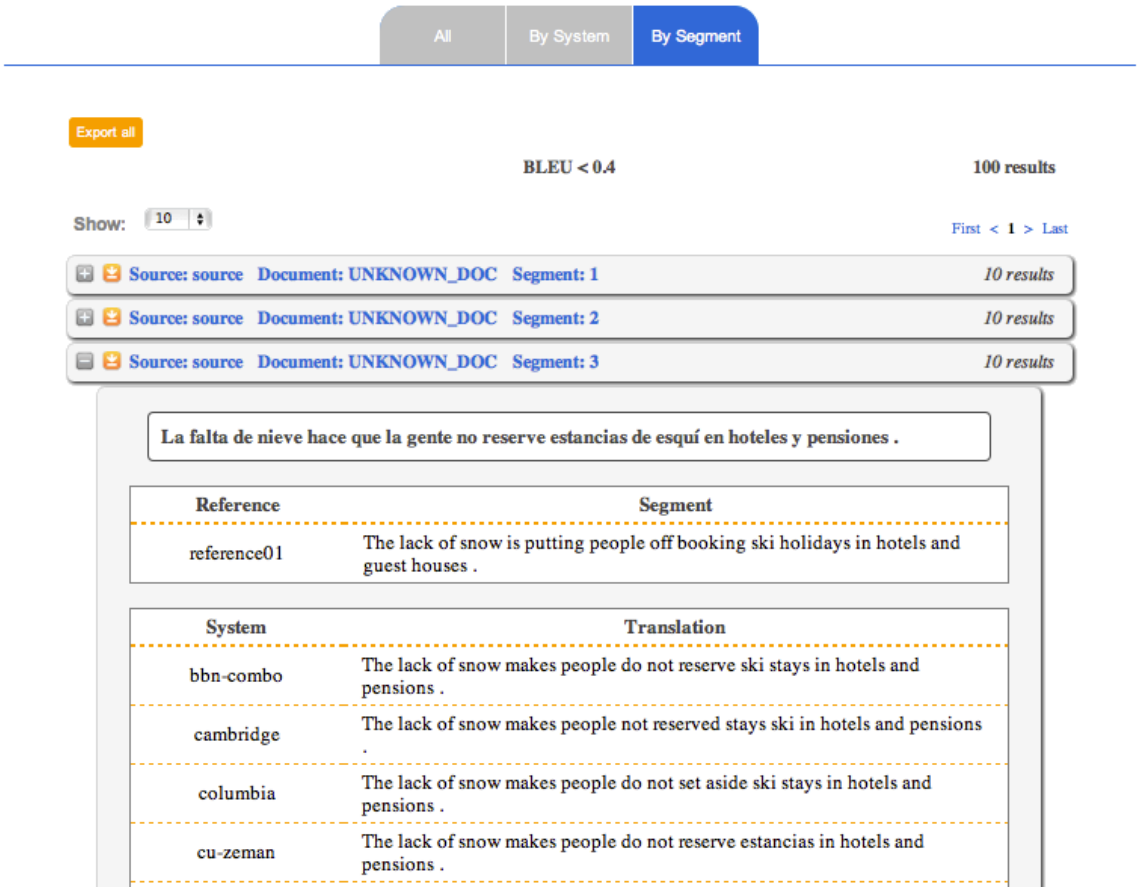

Figure 3.9: *By segment* view. This view displays the segment organization, which allows an easy comparison between several translations. Each group contains all the information related to a segment number, such as the source and the reference sentences along with the candidate translations that matched the query.

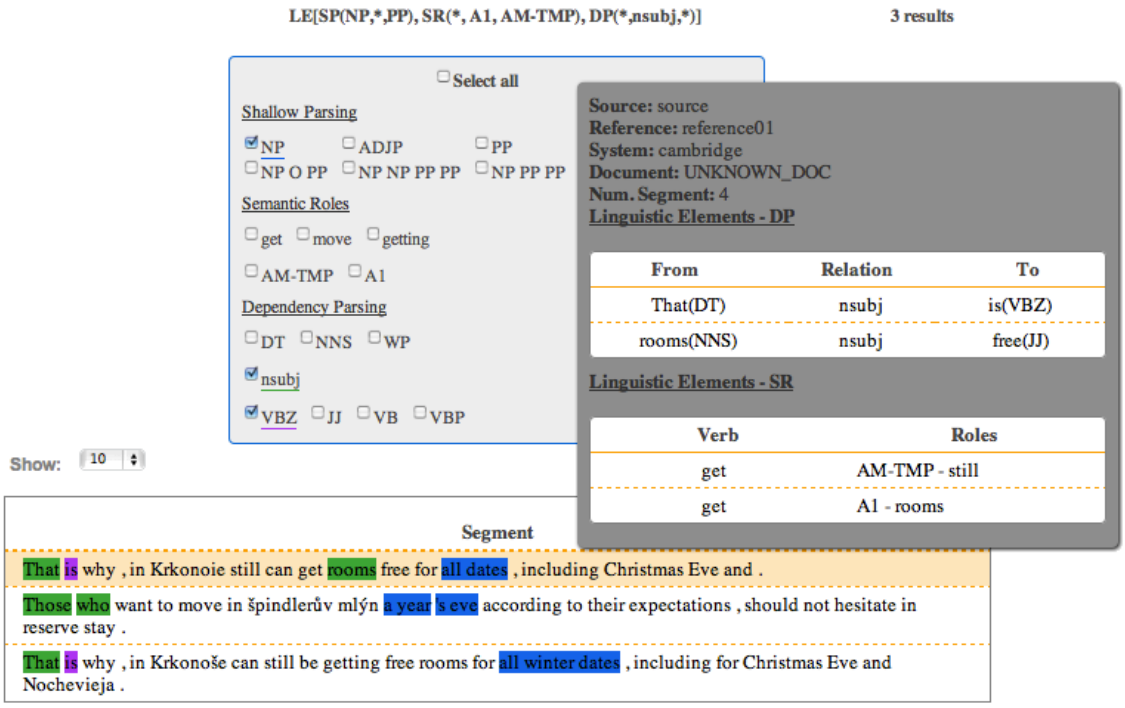

Figure 3.10: Linguistic-based query's output. The linguistic-based queries show a box at the top containing all linguistic processors and the linguistc elements mentioned at the query. It allows to highlight their occurrences in the segments to identify them. Furhtermore, for those queries regarding dependency relation and semantic role, the floating box not only shows additional information regarding a segment, but also a table that sumarizes the relations and roles, respectively. This box appears moving the mouse over the text.

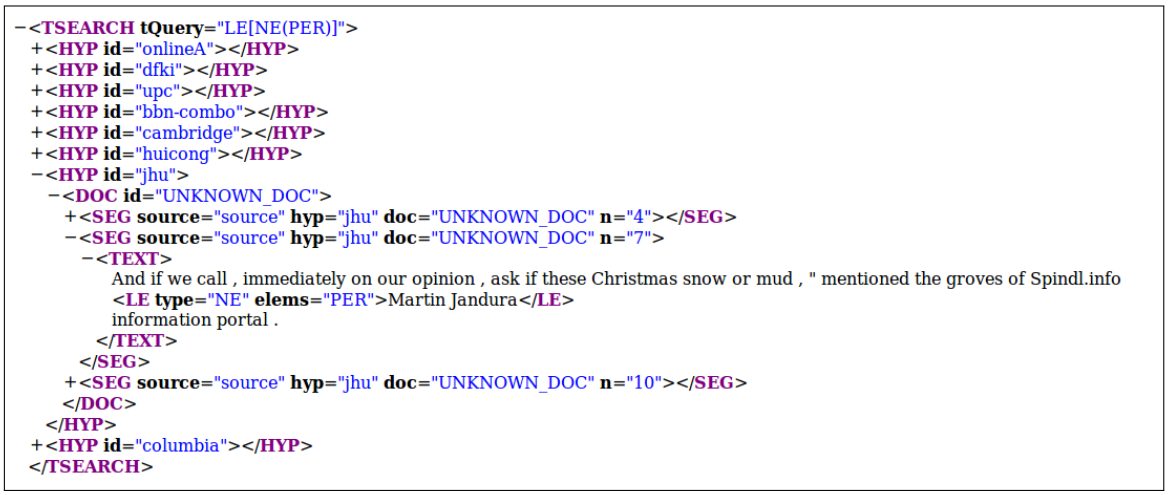

Figure 3.11: XML output of `LE[NE(PER)]` at *By system* view. The XML, at this view, is grouped by system, and for each system, by document. For each segment, it identifies the chunks (i.e., a set of words) of type "PER" (i.e., person).

The presence of statistical functions and groups in the query requires the use of pre-calculated data and, in some cases, run-time calculation. This is a priceless data that *t*Search Online Interface also displays along with the output.

Finally, the search output can be exported as an XML file, thus the results of an specific query can be stored for further analysis and processing. Each view offers the chance to do it at any of the granularity levels displayed, i.e., it allows exporting the results grouped by segment, by system, by document and even all segments without being grouped at the main view. Figure 3.11 shows the XML output of the query `LE[NE(PER)]`, which matches those segments having a name entities of type 'PER' (i.e., person)

Chapter 4

Experimental Results

This chapter details several experiments that we have ran to evaluate the performance of the *t*SEARCH tool under different testbed conditions (Section 4.1) and system configuration (Section 4.2). Although these experiments are performance oriented, we have also prepared an experiment to be done that evaluates the usability, the user friendless and user satisfaction. This experiment is currently available at `asiya.lsi.upc.edu/demo/HCItest/` and will be carried out along the next weeks.

Our purpose is, on the one hand, to analyze the convenience of calculating some data on demand when the user really requires it, either because the time required to calculate these data is too large, or because is unlikely to be used in most evaluation schemas. On the other hand, we also want to compare the perfomance of the system for different testbed sizes, being our variables:

- the number of segments,
- the number of systems,
- the query type.

The reason is that the size of the testbed depends directly on the number of systems and the number of corpora segments, i.e., a testbed with corpora of `1,000` segments and `3` systems, is analysing a total of `3,000` segments. Our experimental results correspond to testsuits with corpora of `100`, `200`, `500`, `1,000` and `2,000` segments and `1`, `2`, `3`, `5`, `10` and `14` systems. We define a reference testbed as the one regarding a corpora of `1,000` segments and `3` systems, since `1,000` segments is the most common corpora size and MT developers usually need to compare the evaluation resuls of their system against a better one and also a baseline. Note that the amount of data generated that needs to be stored

in the *t*SEARCH database is huge, even when inserting the reference testbed. Table 3.7 (in previous chapter) shows the data load regarding the insertion of two testbed with corpora of `100` and `1,000` sentences. Although the task of inspecting `3,000` segments manually is practically unfeasible, *t*SEARCH makes it possible reducing significantly human effort, time and costs.

These experiments were run on a two-node cluster where each Cassandra node is configured in two different servers: *slifer* and *nlp2*. Table 4.1 details their main features.

	slifer	nlp2
MEM	∼8GB	∼4GB
Disk	∼150GB	∼60GB
CPU(s)	4	3
CPU speed	2,833 MHz	2,792 MHz

Table 4.1: *slifer* and *nlp2* servers features

4.1 Pre-calculated vs. Not Pre-calculated

We assess the performance of different aspects of the system at the time to insert a testbed, and also the time response to different types of queries.

One of the main aspects we are worried about is the time that a user has to wait until the testbed is completely inserted into the *t*SEARCH database. Here we can distinguish two different approaches. In the first one, the pre-calulated (P) approach, all the information related to the evaluation scheme at the time to load a tesbed is pre-calculated. This approach increases the elapsd inserting time, but then almost all information is already calculated and the query results are quickly computed. The second approach, not pre-calculated (NP), consists on not to pre-calculate the most expensive computations. As can be seen in Section 4.1.1, these correspond to the operations regarding linguistic elements. In this approach, the user decides to calculate these data exactly when it is needed. Moreover, other metrics that evaluate specific elements are not pre-calculated, since they are unlikely needed in most evaluation scheme, i.e., `NE-Oe(MONEY)`[1]. In our final evaluation setting, we distinguish two different scenarios: the not pre-calculated NP scenario that does not process the files related to linguistic elements, neither the less used metrics; and the pre-calculated P scenario that pre-calculates everything.

[1]NE-Oe(MONEY) reflects lexical overlap between name entities of type 'MONEY'.

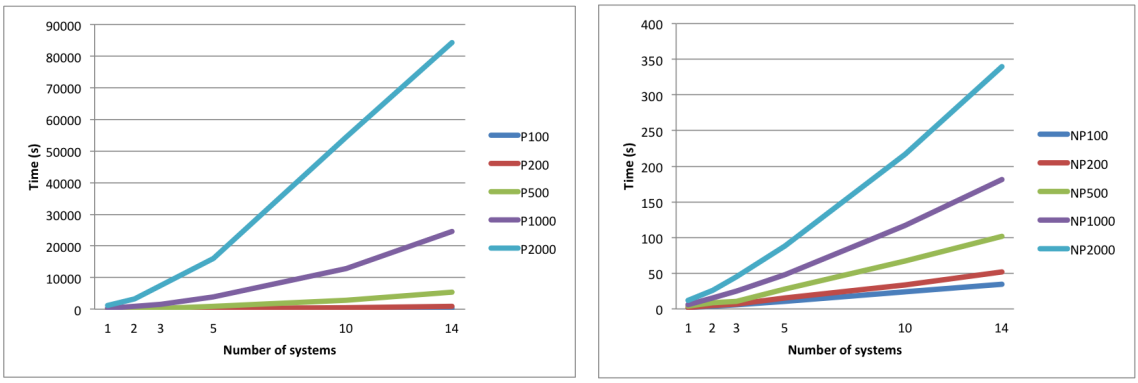

Figure 4.1: Time in seconds of running the *initialize* opreration over several testsuits with corpora of `100`, `200`, `500`, `1,000` and `2,000` segments and `1`, `2`, `3`, `5`, `10` and `14` systems; all them evaluated over 40 different metrics. The Figure on the left shows the results of initializing, i.e., generating and preparing the data for its insertion to the database, the different testbeds in the scenario *NP*, i.e., pre-calculating everything. The Figure on the right, on the other hand, displays the results of initializing the same testbeds but without processing the files related to linguistic elements, neither the less used metrics.

The following sections describe the experimental results regarding the insertion of a testbed (Section 4.1.1) and some queries worth to detail (Section 4.1.2).

4.1.1 Loading the Testbed

The *t*SEARCH Data Loader has two main operations *initialize* and *insert* that are called at the time to load a testbed. On the one hand, *initialize* computes all data regarding the metrics evaluated and also its statistical information, i.e., average, median, minimum, maximum and percentiles values. Furthermore, computes all the linguistic elements information at the *P* scenario. On the other hand, *insert* is in charge to send all the data previously calculated to the database connector in order to be inserted.

The *initialize* operation

The ASIYA output consists of a collection of zipped XML files having the evaluation results for a single metric. The *t*SEARCH data loader has to unzip each metric file to be able to process it. The cost of unzipping files is high and proportional to the number of metrics.

For a fixed number of metrics (40), Figure 4.1 shows the computation time required to obtain the results having a testbed with different number of systems and segments.

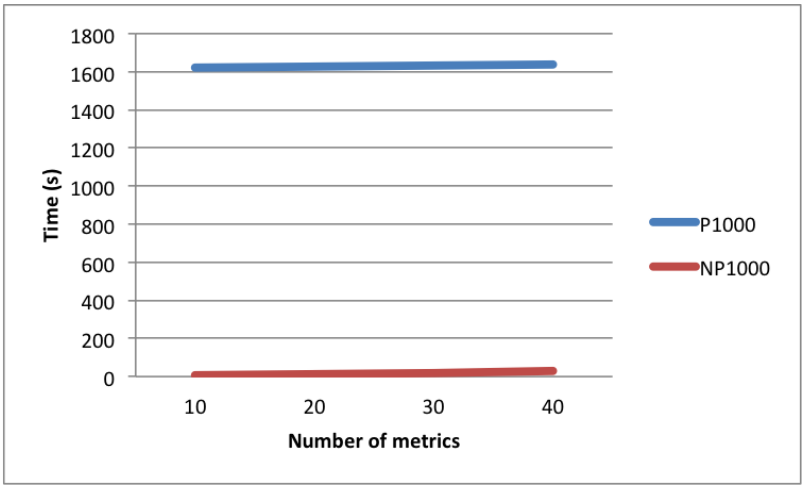

Figure 4.2: Time in seconds of running the *initialize* opreration over a testsuit with corpora of `1,000` segments and `3` systems in the two scenarios P, i.e., pre-calculating everything; and NP, i.e., without processing the files related to linguistic elements, neither the less used metrics.

In comparison, Figure 4.2 summarizes the computation time required in relation to the number of metrics, for a fixed number of segments (`1,000`), a fixed number of systems (`3`) and the two scenarios described above (P and NP). Note that the cost of computing the linguistic elements data (NP) is by far higher than only computing the metric information (P).

The *insert* operation

The experimental results obtained from the different testbeds evaluated over `40` different metrics are shown in Figure 4.3. At the NP scenario, the computational time tends to increase slightly along with the number of systems. Considering the number of corpora segments, again the time barely increases as we increase the number of segments. However, at P scenario, the angle of slop increases excessively and mostly for those testbeds with a corpora of 1,000 segments or more.

Figure 4.4 summarizes the computation time regarding the number of metrics processed in the reference testbed. As it can be seen, the required time is almost constant despite of increasing the number of systems. There is a gap of `1,000` seconds between P and NP approaches. This is the time that tSEARCH needs to insert the data regarding linguistic elements.

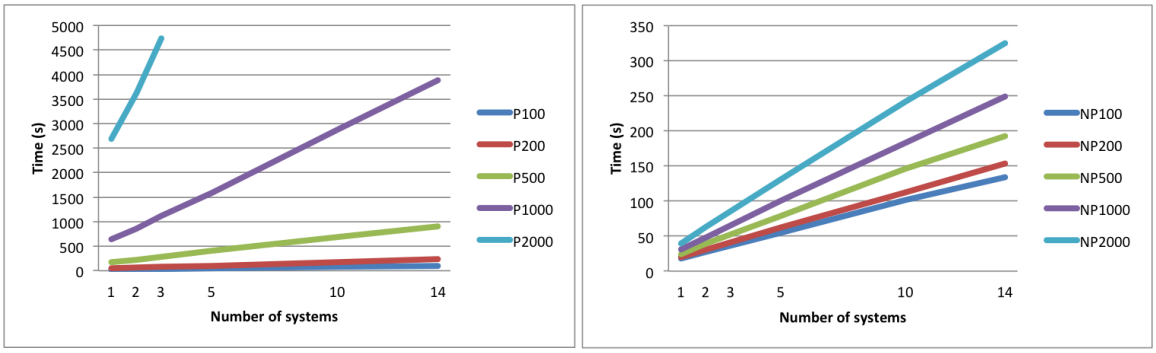

Figure 4.3: Time in seconds of running the *insert* operation over several testsuits with corpora of `100`, `200`, `500`, `1,000` and `2,000` segments and `1`, `2`, `3`, `5`, `10` and `14` systems; all them evaluated over 40 different metrics. The Figure on the left shows the results of inserting the different testbeds in the scenario *NP*, i.e., pre-calculating everything. The Figure on the right, on the other hand, displays the results of inserting the same testbeds but without processing the files related to linguistic elements, neither the less used metrics.

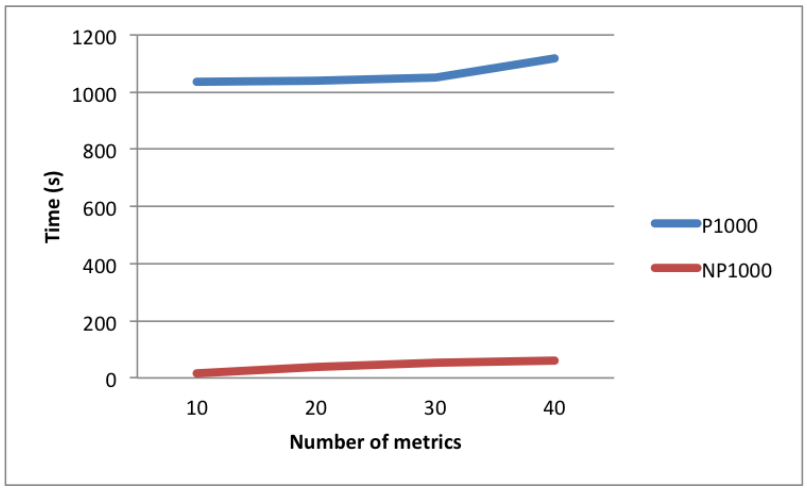

Figure 4.4: Time in seconds of running the *insert* opreration over a testsuit with corpora of `1,000` segments and `3` systems in the two scenarios *P*, i.e., pre-calculating everything; and *NP*, i.e., without processing the files related to linguistic elements, neither the less used metrics.

4.1.2 Query Analysis

Once the testbed is loaded, the system is ready to answer queries. We assess the performance analyzing the response time required to produce the output for the following queries:

- `BLEU ge MIN` gets all testbed segments.
- The query `BLEU ge TH(20)`, that needs to compute the threshold value at run-time from the percentiles pre-calculated and then, gets the 80% of the segments.
- Some of the main Linguistic-based queries: `LE[SP(NP, *, VP)]`, `LE[SR(*, A0, AM-TMP)]`, `LE[DP(*, *, V)]`, `LE[lemma(be)]`, `LE[pos(JJ)]`. Furthermore, the query `LE[SP(NP, *, VP)]` also give us the chance to evaluate the n-gram computation.
- `upc[LEX]` $>$ `AVG OR BLEU ge TH(20)`, where `LEX` is a group of two metrics, is interesting to be analysed because is a composed query.
- The queries `upc[G2] IN upc[PERC(50, 100)]` and `upc[G5] IN upc[PERC(50, 100)]`, where `G2` is a group of 2 metrics and `G5` a group of 5. Here, we can compare the experimental results between them, taking into account that the only difference is the number of elements of each metric group. Moreover, we can obtain the computational cost of `upc[PERC(50, 100)]` which is also calculated at run-time.

We start first with the experiments on those queries that do not distinguish between scenarios *P* and *NP* because linguistic elements are not involved. However, the three last experiments are related to linguistic-based queries, so that we not only show the computational time to get the output, but also the time to calculate on demand all data related to the query. This test and also the results concerning the insertion of a testbed as described in previous Section 4.1.1, are **key ingredients on the decision of whether *t*Search should pre-calculate linguistic elements or not**.

Getting all segments

In this experiment we want to know the time required by *t*Search to prepare an output that contains all segments. For this purpose, we asked for the query `BLEU ge MIN`.

The Figure 4.5 presents the results obtained. Note that the maximum time value obtained is `61` seconds with a test suite of `14` systems and a corpora of `2,000` segments, i.e., `61` seconds to get an output of `28,000` segments. However, *t*Search takes `5` seconds on the reference testbed, i.e., a total of `3,000 segments`.

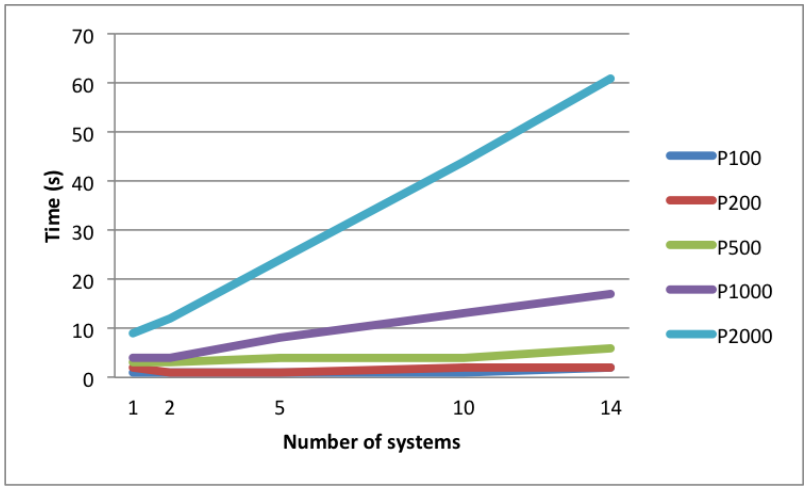

Figure 4.5: Time in seconds of executing the query `BLEU ge MIN` over several testsuits inserted with corpora of `100`, `200`, `500`, `1,000` and `2,000` segments and `1`, `2`, `3`, `5`, `10` and `14` systems; all them evaluated over 40 different metrics.

The TH() function

The main goal asking for the query `BLEU ge TH(20)` is to compute the time to obtain the 80% of the segments and observe if the computation of a funcion, here the value of threshold of the 20%, makes the computational time increases significantly. We have to take into account that thresholds, as quartiles, are calculated on run-time from the percentiles values.

As it can be seen in Figure 4.6 the time to get the 80% of the segments is sightly lower than getting all segments (see Figure 4.5).

Composed query

The query `upc[LEX] > AVG OR BLEU ge TH(20)` is worth being analysed because is a composed query. Thus the *t*SEARCH search engine has to compute the results for `upc[LEX] > AVG` and `BLEU ge TH(20)` separately and then, compute the union of both result sets.

The Figure 4.7 displays the results obtained and curiously are almost the same results obtained in Figure 4.6 that reprents the query `BLEU ge TH(20)`. This is because the `AVG` value here is higher than the threshold value, hence the results obtained are the ones get from `BLEU ge TH(20)`. Note that this is due to the logical connector `OR`, if it was an `AND`,

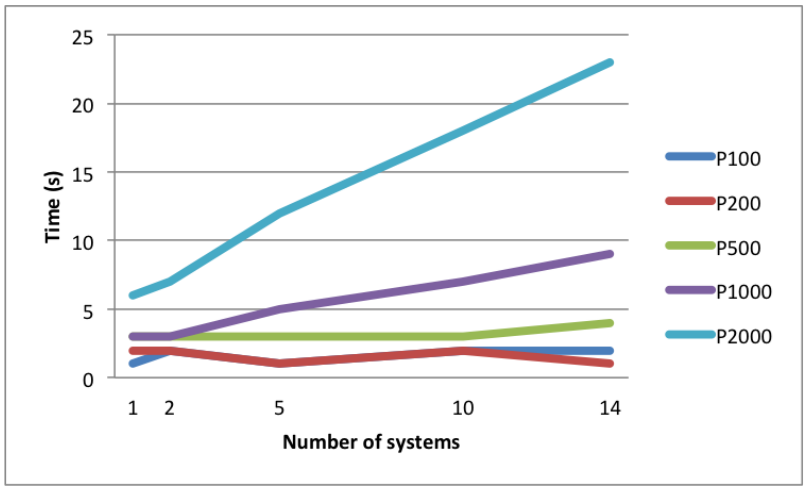

Figure 4.6: Time in seconds of executing the query `BLEU ge TH(20)` over several testsuits inserted with corpora of `100`, `200`, `500`, `1,000` and `2,000` segments and `1`, `2`, `3`, `5`, `10` and `14` systems; all them evaluated over 40 different metrics.

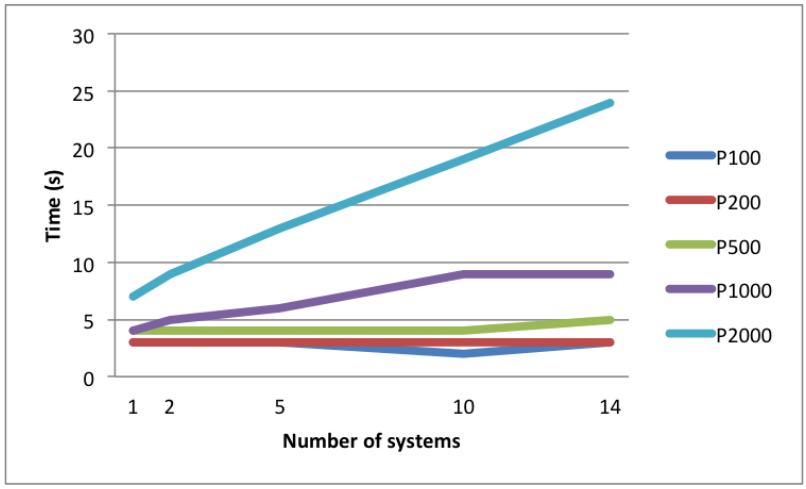

Figure 4.7: Time in seconds of executing the query `upc[G2] > AVG OR BLEU ge TH(20)`, where `G2` is a group of two metrics, over several testsuits inserted with corpora of `100`, `200`, `500`, `1,000` and `2,000` segments and `1`, `2`, `3`, `5`, `10` and `14` systems; all them evaluated over 40 different metrics.

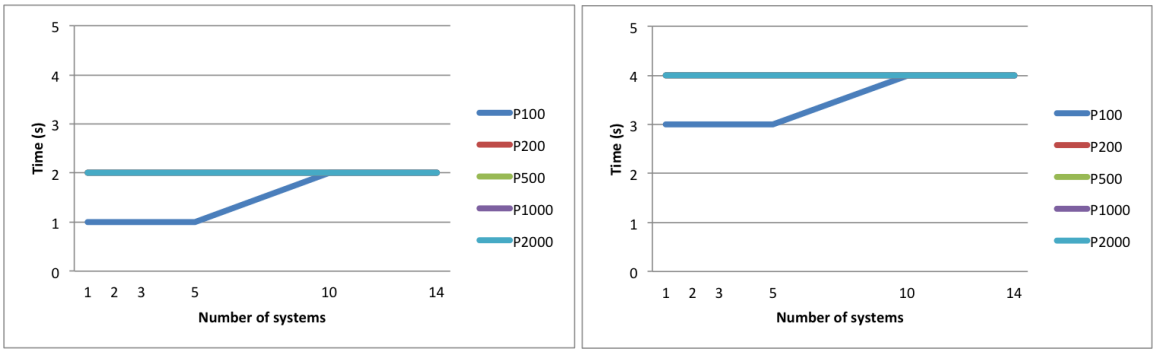

Figure 4.8: Time in seconds of executing the query `upc[G2] IN upc[PERC(50, 100)]` (Figure on the left) and `upc[G5] IN upc[PERC(50, 100)]` (Figure on the right), where `G2` is a group of two metrics, and `G5` is a group of five metrics. The results showed corresponds to several testsuits inserted with corpora of `100`, `200`, `500`, `1,000` and `2,000` segments and `1`, `2`, `3`, `5`, `10` and `14` systems; all them evaluated over 40 different metrics.

it would be an intersection and the results were the ones corresponding to `upc[LEX] > AVG`. Thus, the increment of time is mostly due the postprocess and not to the DB access.

Relation between different number of metrics groups

At this section, we make a comparison of the query `upc[G2] IN upc[PERC(50, 100)]` and `upc[G5] IN upc[PERC(50, 100)]`, where `G2` is a group of two metrics, and `G5` is a group of five metrics. Here, we want to check how does to increase the computational time in relation to the number of metrics taking part in the result. In addition, we want to ensure if the run-time operation `upc[PERC(50, 100)]` has a high cost of computation, since here the 50% corresponds only to the `upc` system. Figure 4.8 shows the results regarding `upc[G2] IN upc[PERC(50, 100)]` and `upc[G5] IN upc[PERC(50, 100)]`.

The results display how an increase from `2` to `5` metrics implies a time increase twice the run-time. Nevertheless, the total time of `4` seconds in the worst case is completely acceptable.

So far in this section, we have executed only metric-based queries that makes no differences between the two scenarios *P* and *NP* described previously. However, the remaining three subsections are related to linguistic-based queries, therefore, it has been also calculated for each one the time to compute the data requested on demand, i.e., if a user asks for a dependency relationship (`DP`), all `DP` data is calculated. Therefore, for the nexts requests regarding `DP`s, the data will be already computed.

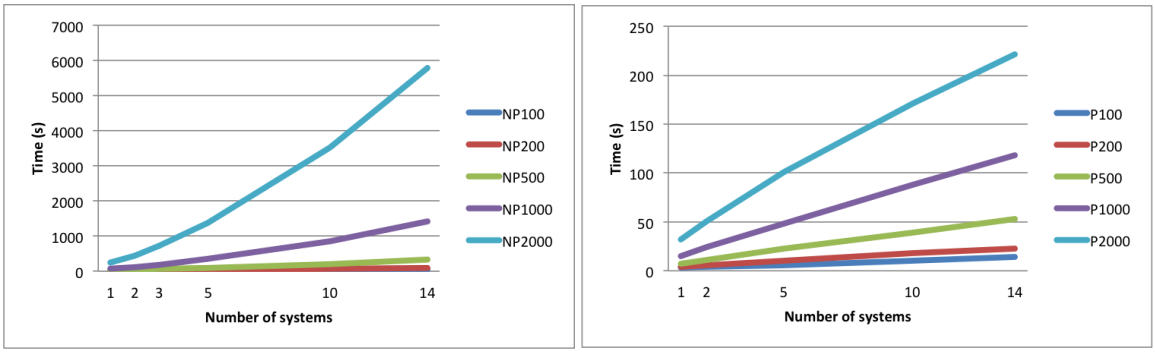

Figure 4.9: The Figure on the left displays the time in seconds of computing the data needed to execute the shallow parsing (`SP`) function, i.e., process and insert the data from shallow processor. In contrast, the Figure on the right shows the time in seconds to retrieve the results concerning the query `LE[SP(NP, *, VP)]` when all the data needed was already computed.

N-grams

The n-gram queries are the most time consuming operations. This is because, we do not pre-calculate all possible n-grams in a sentence because we strongly believe that this is defenitely worse and thus, not an option. So that, in order to compute the results of `LE[SP(NP, *, VP)]` we obtain for each sentence all `NP` and `VP` separately and then, we check whether the distance of the words is the right (e.g., here a distance of 2 is required) in order to include them as a part of the solution. This makes the computational time considerably higher than the others seen so far. The comparison made at Figure 4.9 shows the impact of having the data already calculated.

Dependency Parsing Query

The Dependency Parsing Query is, by far, the most expensive computation on demand as it can be seen in Figure 4.10. When *t*SEARCH computes the data regarding dependency relationships, it processes the corresponding files and computes groups of three elements, where the first one is the part-of-speech of a head, e.g., `N` (noun); the second one is the dependency relation, e.g., `nsubj` (nominal subject); and the last one is the part-of-speech of the dependent, e.g., `V` (verb). Moreover, we also count for each word with the coarse-grained part-of-speech, e.g., `NN` (noun, singular or mass) or `VBZ` (verb, third person singular present). Thus, for each word there are four different possibilities, e.g., {`N, nsubj, V`}, {`N, nsubj, VBZ`}, {`NN, nsubj, V`}, {`NN, nsubj, VBZ`}. Furthermore, we also save the *any* option represented by the asterisk, e.g., {`N, *, V`}, {`*, nsubj, V`}. {`N, nsubj, *`}.

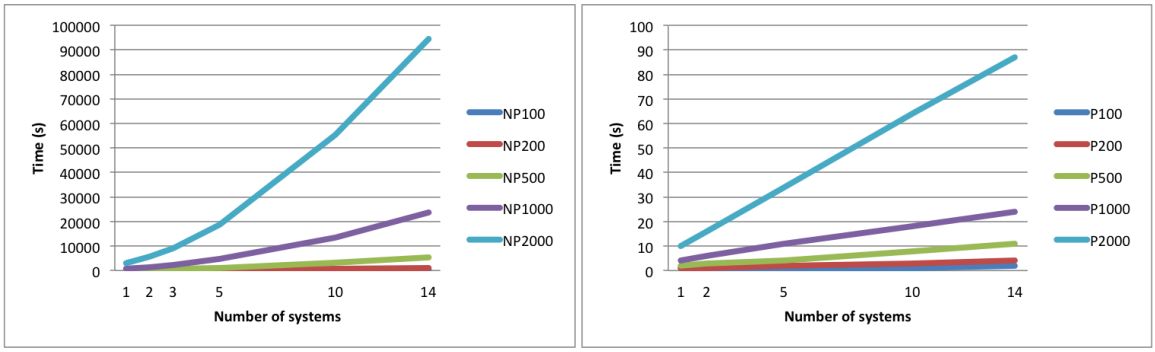

Figure 4.10: The left Figure displays the time in seconds of computing the data needed to execute the dependency parsing (`DP`) function, i.e., process and insert the data from dependency processor. The Figure on the right shows the time in seconds to retrieve the results concerning the query `LE[DP(*,*,V)]` when all the data needed was already computed.

As a result, for each word we compute `18` different variants which is the reason of the high cost of its computation. However, as far as we know, this is the best way to compute all the Dependency Parsing data. Fortunatelly, once the data is computed, the time to obtain the results is completely reasonable as is shown in Figure 4.10.

Other linguistic-based queries

The remaining queries evaluated are `LE[SR(*, A0, AM-TMP)]`, `LE[lemma(be)]`, `LE[pos(JJ)]`. Fortunatelly, both, the cost to calculate their data on demand and the cost to obtain the results are low (Figures 4.11, 4.12 and 4.13).

4.2 Single Node vs. Two-Nodes Cluster

The next experiment is related to the *t*SEARCH database. As detailed in Section 3.2, we implemented the Cassandra solution and we want to check whether the performance of the *t*SEARCH improves along with the number of nodes in the cluster. As aforementioned, due to the lack of resources we can just run the experiments on a single node and a two-nodes cluster (Figure 4.14).

As can be noted, the reads are improved, and so the computational time. Actually, the improvement is about nearly the 20%. Nevertheless, we cannot affirm that the more nodes we add, the best performance we obtain, since the Cassandra performance depends on selecting the right balance of the following resources: memory, CPU, disks, number of

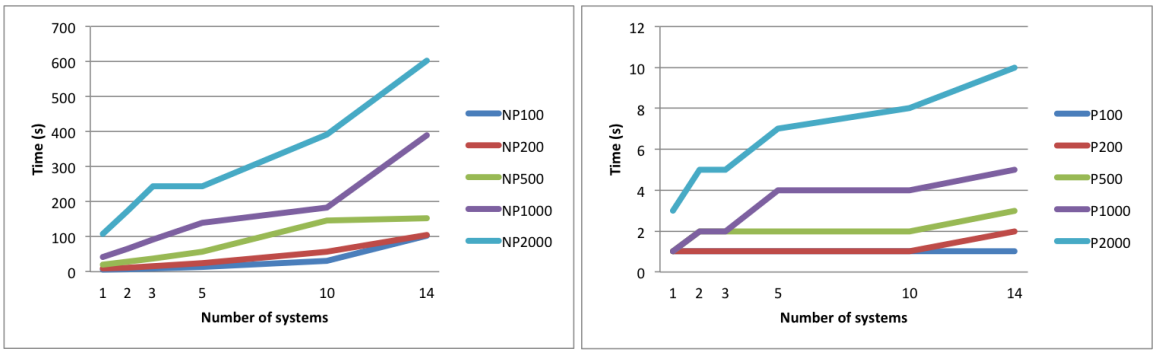

Figure 4.11: The Figure on the left displays time in seconds of computing the data needed to execute the semantic role (`SR`) function, i.e., process and insert the data from semantic processor. The Figure on the right shows the time in seconds to retrieve the results concerning the query `LE[SR(*, A0, AM-TMP)]` when all the data needed was already computed.

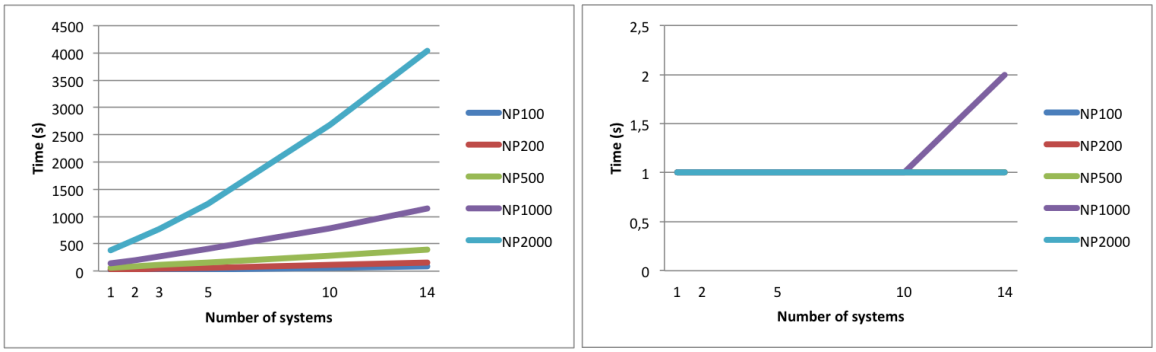

Figure 4.12: The Figure on the left displays the time in seconds of computing the data needed to execute the lemma function. The Figure on the right shows the time in seconds to retrieve the results concerning the query `LE[lemma(be)]` when all the data needed was already computed.

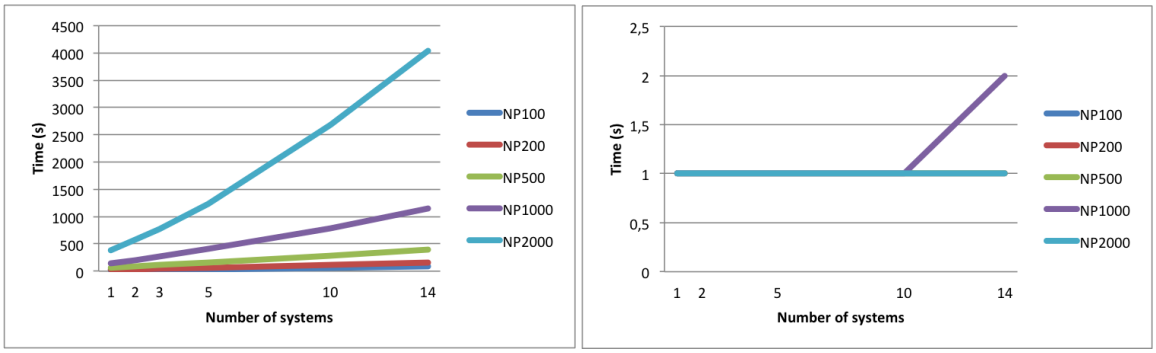

Figure 4.13: The Figure on the left displays the time in seconds of computing the data needed to execute the part-of-speech (`pos`) function. The Figure on the right shows the time in seconds to retrieve the results concerning the query `LE[pos(JJ)]` when all the data needed was already computed.

nodes and network. Unfortunatelly, the servers are not only dedicated to this project, but also provides other services, hence we could not adjust, either test, other configurations adjusting the other resources.

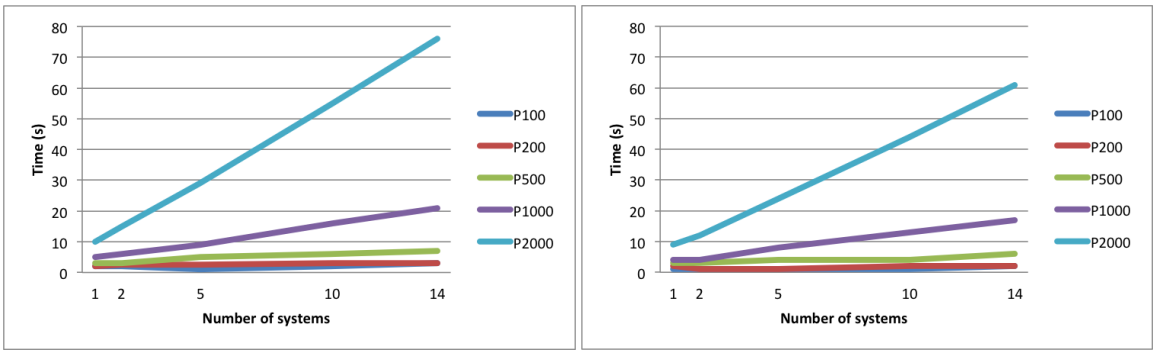

Figure 4.14: Computational time in seconds of executing the query `BLEU ge MIN`, i.e, getting all segments, over several testsuits inserted with corpora of `100`, `200`, `500`, `1,000` and `2,000` segments and `1`, `2`, `3`, `5`, `10` and `14` systems; all them evaluated over 40 different metrics. The Figure on the left shows the results obtained in a single node configuration. The Figure on the right displays the results obtained in a two-nodes cluster configuration.

4.3 Summary

We summarize below the conclusions drawn from the experiments carried out in this chapter.

Regarding the first experiment, pre-calculated vs. not pre-calculated, the best option is to not compute the information regarding linguistic elements. This decision reduces considerably the time to load a testbed which improves the user experience. Despite some of the linguistic-based queries require a long time in order to be computed, e.g., dependency relationships, this is only required the first time and, at best, the user does not need to ask for this kind of queries. As can be noted, some of the experiments, e.g., those testbeds with a corpora of `2,000` sentences lasts few hours to finish its insertion. One of the future work (Section 6.1) tasks is the development of a command line interface. This will allow users to download the *t*Search code and install in their servers, making feasible this long waiting time. In this direction, we could let the user to decide whether they want to pre-calculate everything.

Concerning the Cassandra configuration experiment, i.e., single node vs. two-node cluster, we observed an improvement of the computational time when executing queries, since reads are more efficient in the two-node cluster where data is properly distributed along the two servers. However, we can not assert that the more nodes we add, the best performance Cassandra obtains, because, as mentioned above, its performance depends directly on the right balance of memory, CPU, disks, number of nodes and network.

Chapter 5

Planning

This Chapter presents the planning of this project, making a comparison between the initial and the final schedulings and discussing the deviations produced.

We start with the list of tasks defined in both plannings (Figure 5.1 and Figure 5.2). Basically, the decision of applying the Cassandra database, added some tasks to the final scheduling, such as the study, design and implementation of the database. Furthermore, we added two experiments performance oriented: single node vs. two-node cluster and pre-calculate vs. not pre-calculate. The tasks are dividend through several groups. We specify the roles involed in each one of them:

- The "project management" group basically lists all the tasks regarding the scheduling, analyses and organization of this proposal. The project manager is in charge of this group of tasks.
- "*t*Search database" groups all tasks related to the Cassandra database. Here we count with the system and database administrators.
- The "*t*Search implementation" group contains all tasks concerning the definition and implementation of the queries, the *t*Search Online Interface and the *t*Search data loader. Here, the analyst and the developer play an important role.
- The roles involved in the "integration with Asiya" were the developer, the system and database administrators.
- Testing is one of the most important parts of the development of a project, since it guarantees that the system works properly. The tester is responsible of this task.
- The project manager has specified a group of three "milestones". The first one requires a simple version of the system that works with a single fixed query; the

second one implies having implemented at least two types of metric-based queries; and finally, the last milestone marks the completion of the *t*SEARCH development.

Finally, Figure 5.3 and Figure 5.4 show the initial and the final schedulings, respectively. As can be noted, the final planning finishes almost four months later than the initial one. The deviation is due to different issues. First of all, the decision of applying a NoSQL solution was made after having started the project, so it was not contemplated at the initial planning. This decision affected the scheduling since we spend roughly a month learning and setting up the Cassandra database. We also prepared and ran two different experiments in order to evaluate the performance of *t*SEARCH. The first one assesses the performance of different Cassandra configurations, i.e., single node versus two-node cluster; the second one evaluates the cost of pre-calculating everything or not (See Section 4), which required the development of both scenarios delaying the scheduling. Furthermore, we spent approximately two weeks writting and preparing the paper for the system demonstration track at the Annual Meeting of the Association for Computational Linguistics (ACL) ([Gonzàlez et al., 2013b]). Compared to the initial planning, we have also increased the extension of the query language, since during the development of *t*SEARCH we realized useful queries that were not considered at the beginning of the project, e.g., the system comparison. Finally, there is a three weeks gap of no development from June to July that corresponds to a holiday period.

ID	Task Name	Start	Finish
1	**Project Management**	**Sat 15/09/12**	**Fri 31/05/13**
2	Goal Definition	Sat 15/09/12	Mon 24/09/12
3	**Analysis**	**Mon 17/09/12**	**Mon 15/10/12**
4	Related Work	Mon 17/09/12	Mon 01/10/12
5	Viability	Mon 24/09/12	Mon 01/10/12
6	Requirements	Mon 01/10/12	Mon 15/10/12
7	Study of Asiya	Sat 15/09/12	Mon 26/11/12
8	Design	Tue 25/09/12	Mon 29/10/12
9	Planning	Mon 08/10/12	Mon 29/10/12
10	Validation of the Planning	Mon 29/10/12	Mon 29/10/12
11	tSearch Documentation	Mon 15/10/12	Fri 31/05/13
12	**tSearch implementation**	**Wed 26/09/12**	**Mon 25/02/13**
13	**Query Implementation**	**Wed 26/09/12**	**Mon 25/02/13**
14	Query Language Definition	Wed 26/09/12	Fri 25/01/13
15	Parser Implementation	Mon 01/10/12	Mon 04/02/13
16	Search Engine Implementation	Mon 15/10/12	Mon 25/02/13
17	**tSearch Interface**	**Mon 08/10/12**	**Mon 25/02/13**
18	View Definition	Mon 08/10/12	Mon 14/01/13
19	Implementation	Mon 15/10/12	Mon 25/02/13
20	**API tSearch Data Loader**	**Sat 08/12/12**	**Mon 14/01/13**
21	Definition	Sat 08/12/12	Mon 17/12/12
22	Implementation	Tue 18/12/12	Mon 14/01/13
23	Integration with Asiya	Mon 14/01/13	Mon 11/03/13
24	**Tests**	**Mon 04/02/13**	**Mon 11/03/13**
25	Integration test	Mon 04/02/13	Mon 25/03/13
26	Testing system	Mon 12/11/12	Mon 11/03/13
27	**Milestones**		
28	Simple system working	Wed 31/10/12	
29	Interface + 2 queries	Fri 21/12/12	
30	All queries developed	Mon 25/03/13	

Figure 5.1: List of *initial* tasks

ID	Task Name	Start	Finish
1	**Project Management**	**Sat 15/09/12**	**Mon 23/09/13**
2	Goal Definition	Sat 15/09/12	Mon 24/09/12
3	**Analysis**	**Mon 17/09/12**	**Mon 15/10/12**
4	Related Work	Mon 17/09/12	Mon 01/10/12
5	Viability	Mon 24/09/12	Mon 01/10/12
6	Requirements	Mon 01/10/12	Mon 15/10/12
7	Study of Asiya	Sat 15/09/12	Mon 26/11/12
8	Design	Tue 25/09/12	Mon 29/10/12
9	Planning	Mon 08/10/12	Mon 29/10/12
10	Validation of the Planning	Mon 29/10/12	Mon 29/10/12
11	tSearch Documentation	Mon 15/10/12	Mon 23/09/13
12	**tSearch Database**	**Thu 01/11/12**	**Fri 14/12/12**
13	Study of Cassandra	Thu 01/11/12	Fri 30/11/12
14	Design Database	Mon 26/11/12	Fri 07/12/12
15	Connector Implementation	Mon 10/12/12	Fri 14/12/12
16	**tSearch implementation**	**Wed 26/09/12**	**Mon 27/05/13**
17	**Query Implementation**	**Wed 26/09/12**	**Mon 27/05/13**
18	Query Language Definition	Wed 26/09/12	Mon 22/04/13
19	Parser Implementation	Mon 01/10/12	Mon 06/05/13
20	Search Engine Implementation	Mon 15/10/12	Mon 27/05/13
21	**tSearch Interface**	**Mon 08/10/12**	**Mon 25/02/13**
22	View Definition	Mon 08/10/12	Mon 14/01/13
23	Implementation	Mon 15/10/12	Mon 25/02/13
24	**API tSearch Data Loader**	**Sat 08/12/12**	**Mon 14/01/13**
25	Definition	Sat 08/12/12	Mon 17/12/12
26	Implementation	Tue 18/12/12	Mon 14/01/13
27	Integration with Asiya	Mon 11/02/13	Fri 05/04/13
28	**Experiments**		
29	Single Node vs. Two-node cluster	Mon 15/04/13	Mon 10/06/13
30	Pre-calculated vs. Not Pre-calculated	Mon 08/07/13	Thu 22/08/13
31	**Tests**		
32	Integration test	Mon 04/02/13	Mon 25/03/13
33	Testing system	Mon 12/11/12	Mon 11/03/13
34	**Milestones**		
35	Simple system working	Wed 31/10/12	
36	Interface + 2 queries	Mon 31/12/12	
37	All queries developed	Mon 10/06/13	

Figure 5.2: List of *final* tasks

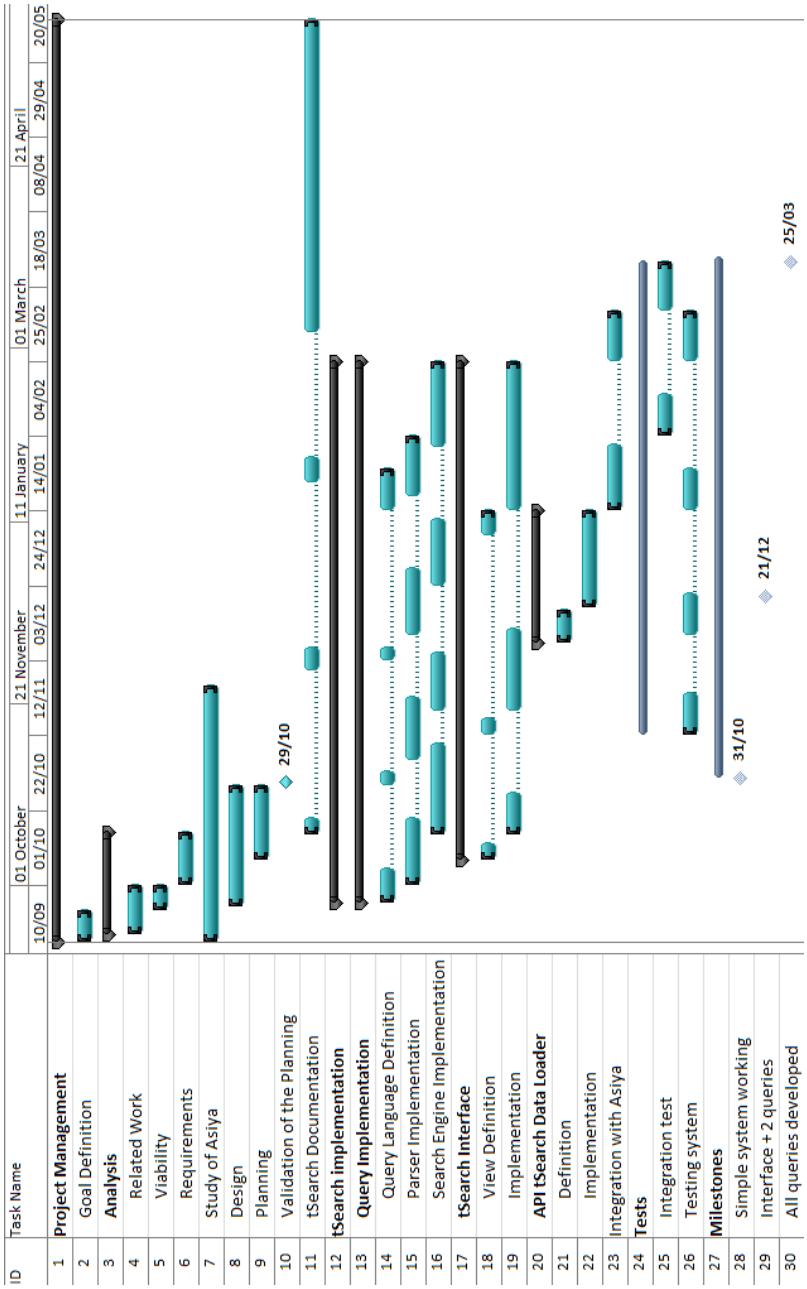

Figure 5.3: Gannt chart showing the *initial* *t*SEARCH planning

Figure 5.4: Gannt chart showing the *final* *t*SEARCH planning

Chapter 6

Conclusions

The result of this work is a free web-based tool that automatically processess the results of an MT evaluation (calculated with Asiya) that supports MT developers during the MT error analysis stage (Objective 1). To make it happen, we have also designed and implemented a rich and flexible query language (Objective 2) that can be easily extensible (Objective 5). Most of the queries were obtained from the feedback of some MT developers who expressed their needs when they do the error analysis of their MT systems. The wide range of queries implemented and the possibility to combine them allow for searching for a specific collection of translations that meet some specific criteria, which notably facilitates the detection of errors.

Regarding the *t*Search Online Interface, we put a great deal of effort into its development in order to address usability goals, i.e., we provide a friendly online interface easy to use. The search results are presented with additional information, organized in three different views allowing to perform the Error Analysis task efficiently (Objective 3). Moreover, the results can be exported as an XML file for further processing (Objective 4).

We have also addressed the *t*Search performance (Objective 6). In this direction, we made some implementation decisions regarding the computation time based on the result of the experiments. So that, Against the initial idea of pre-calculating everything, we concluded that is better to calculate only on demand the most time consuming information (i.e., the information related to linguistic elements) and also the least used information (i.e., the metrics that evaluate specific elements). This decision not only decreases the *t*Search load time, but also avoids computing data that would not be required, which increases user satisfaction and reduces the need for system storage resources.

Concerning the database, we chose the NoSQL solution offered by the Apache Cassandra database because it deals with huge amounts of data. Ideally, we should test the *t*Search

performance in different multi-node configurations in order to choose the one that achieves the best results. Nevertheless we just could test it at the most in a two-nodes cluster because of the resources limitation we have. This was the main drawback during the development, but at least we can ensure that the current configuration fulfil the requirements. It is also important to point out that the use of Cassandra bring us the opportunity to experiment and learn such new technology.

In summary, all of the master's thesis goals, which are listed in Section 1.2, have been successfully accomplished. Furthermore, we should point out that this Master Thesis:

- has produced a research paper that was presented in the system demonstration track at the Annual Meeting of the Association for Computational Linguistics (ACL) ([Gonzàlez et al., 2013b]),
- was awarded with a grant from the European Association for Machine Translation (EAMT)[1],
- provides the MT community a free tool available at `http://asiya.lsi.upc.edu/demo/`.

Finally, this project was partially supported by the European Community's Seventh Framework Programme under grant agreement number 247762 (FAUST, FP7-ICT-2009-4-247762)[2] ([Gonzàlez et al., 2013a]).

6.1 Ongoing and Future Work

In the following, we sketch the main goals that we are intended to develop as a continuation of this research work.

1. Evaluation of user satisfaction.
   We have developed a test to assess the usability of the Asiya Online Interface and the *t*Search tool. It consists of three easy steps: first, a simple training; second, the resolution of test scenario; and third, the completion of a short questionnaire. The experiment is already prepared and available at `asiya.lsi.upc.edu/demo/HCItest/`. Along the next weeks, the test will be carried out with real users for a later analysis and publication of an article.

2. Creation of profiles.
   The implementation of user profiles in the Asiya and *t*Search online interfaces is

[1] `http://www.eamt.org/` Sponsorship of projects research 2012
[2] `http://cordis.europa.eu/fp7/home_en.html`

planned for a near future development, which will allow MT developers to save the analysis of their testbeds. It would require a distributed architecture and, as far as we know, the open-source software Apache Hadoop[3] is a good choice for dealing with this kind of architecture. One of the Hadoop related projects is Cassandra and this is one of the main reasons why we implemented this solution.

3. Development of a command line interface.
   Another line of development is the command line interface, which would allow users to install *t*SEARCH on their servers.

4. Adding new query structures.
   Although we worked hard to implement a query language that suits the MT developers needs, we look forward to receive suggestions of new types of queries once the community starts using the *t*SEARCH tool.

5. Other extensions.
   The ASIYA framework is continuosly being extended with new taggers, lemmatizers, parsers or other analysers that would require the update and extension of *t*SEARCH in order to cope with new ASIYA's outputs.

[3] `http://hadoop.apache.org/`

References

[Berka et al., 2012] Berka, J., Bojar, O., Fishel, M., Popović, M., and Zeman, D. (2012). Automatic MT Error Analysis: Hjerson Helping Addicter. In *Proceedings of the 8th International Conference on Language Resources and Evaluation. International Conference on Language Resources and Evaluation (LREC), Istanbul, Turkey*. European Language Resources Association (ELRA).

[Callison-Burch et al., 2006] Callison-Burch, C., Osborne, M., and Koehn, P. (2006). Re-evaluating the Role of BLEU in Machine Translation Research. In *Proceedings of 11th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*.

[Charniak and Johnson, 2005] Charniak, E. and Johnson, M. (2005). Coarse-to-Fine n-Best Parsing and MaxEnt Discriminative Reranking. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 173–180, Ann Arbor, Michigan. Association for Computational Linguistics.

[Coughlin, 2003] Coughlin, D. (2003). Correlating Automated and Human Assessments of Machine Translation Quality. In *Proceedings of Machine Translation Summit IX*, pages 23–27.

[Culy and Riehemann, 2003] Culy, C. and Riehemann, S. Z. (2003). The Limits of N-gram Translation Evaluation Metrics. In *Proceedings of MT-SUMMIT IX*, pages 1–8.

[Denkowski and Lavie, 2010] Denkowski, M. and Lavie, A. (2010). METEOR-NEXT and the METEOR Paraphrase Tables: Improved Evaluation Support for Five Target Languages. In *Proceedings of the Joint Fifth Workshop on Statistical Machine Translation and MetricsMATR*, pages 339–342, Uppsala, Sweden. Association for Computational Linguistics.

[Doddington, 2002] Doddington, G. (2002). Automatic Evaluation of Machine Translation Quality Using N-gram Co-occurrence Statistics. In *Proceedings of the second international conference on Human Language Technology Research*, HLT, pages 138–145, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

[Dreyer and Marcu, 2012] Dreyer, M. and Marcu, D. (2012). HyTER: Meaning-Equivalent Semantics for Translation Evaluation. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 162–171, Montréal, Canada. Association for Computational Linguistics.

[Giménez and Gonzàlez, 2013] Giménez, J. and Gonzàlez, M. (2013). An Open Toolkit for Automatic Machine Translation (Meta-)Evaluation (Technical Manual).

[Giménez and Màrquez, 2004] Giménez, J. and Màrquez, L. (2004). SVMTool: A general POS tagger generator based on Support Vector Machines. In *Proceedings of the 4th International Conference on Language Resources and Evaluation (LREC)*, pages 43–46.

[Giménez and Màrquez, 2010] Giménez, J. and Màrquez, L. (2010). Asiya: An Open Toolkit for Automatic Machine Translation (Meta-)Evaluation. *The Prague Bulletin of Mathematical Linguistics*, (94):77–86.

[Giménez and Amigó, 2006] Giménez, J. and Amigó, E. (2006). IQmt: A Framework for Automatic Machine Translation Evaluation. In *Proceedings of the 5th International Conference on Language Resources and Evaluation (LREC)*, pages 685–690.

[Gonzàlez et al., 2012] Gonzàlez, M., Giménez, J., and Màrquez, L. (2012). A Graphical Interface for MT Evaluation and Error Analysis. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (ACL). System Demonstration*, pages 139–144, Jeju Island, Korea.

[Gonzàlez et al., 2013a] Gonzàlez, M., Màrquez, L., and Mascarell, L. (2013a). Open Source Release of the Final Asiya Suite. In *Deliverable 4.7 of FAUST Project (FAUST D4.7) of the Seventh Framework Programme of the European Community.*

[Gonzàlez et al., 2013b] Gonzàlez, M., Mascarell, L., and Màrquez, L. (2013b). *t*SEARCH: Flexible and Fast Search over Automatic Translations for Improved Quality/Error Analysis. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 181–186, Sofia, Bulgaria. Association for Computational Linguistics.

[Liu and Gildea, 2005] Liu, D. and Gildea, D. (2005). Syntactic Features for Evaluation of Machine Translation. In *Proceedings of ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for MT and/or Summarization*, pages 25–32.

[Niessen et al., 2000] Niessen, S., Och, F. J., Leusch, G., and Ney, H. (2000). An evaluation tool for machine translation: Fast evaluation for MT research. In *International Conference on Language Resources and Evaluation (LREC)*, pages 39–45, Athens, Greece.

[Nivre et al., 2007] Nivre, J., Hall, J., Nilsson, J., Chanev, A., Eryigit, G., Kübler, S., Marinov, S., and Marsi, E. (2007). MaltParser: A Language-Independent System for Data-Driven Dependency Parsing. *Natural Language Engineering*, 13(2):95–135.

[Papineni et al., 2002] Papineni, K., Roukos, S., Ward, T., and Zhu, W.-J. (2002). BLEU: a Method for Automatic Evaluation of Machine Translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL, pages 311–318, Stroudsburg, PA, USA. Association for Computational Linguistics.

[Petrov and Klein, 2007] Petrov, S. and Klein, D. (2007). Improved Inference for Unlexicalized Parsing. In *Proceeding of the Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, pages 404–411, Rochester, New York.

[Popović, 2011] Popović, M. (2011). Hjerson: An Open Source Tool for Automatic Error Classification of Machine Translation Output. *The Prague Bulletin of Mathematical Linguistics*, 96.

[Popović and Burchardt, 2011] Popović, M. and Burchardt, A. (2011). From Human to Automatic Error Classification for Machine Translation Output. In *15th International Conference of the European Association for Machine Translation (EAMT), Leuven, Belgium.*

[Popović and Ney, 2007] Popović, M. and Ney, H. (2007). Word Error Rates: Decomposition over POS classes and Applications for Error Analysis. In *Proceedings of the Second Workshop on Statistical Machine Translation*, pages 48–55, Prague, Czech Republic. Association for Computational Linguistics.

[Reeder et al., 2001] Reeder, F., Miller, K., Doyon, J., and White, J. (2001). The Naming of Things and the Confusion of Tongues: an MT Metric. In *Proceedings of the Workshop on MT Evaluation "Who did what to whom?" at Machine Translation Summit VIII*, pages 55–59.

[Snover et al., 2010] Snover, M. G., Madnani, N., Dorr, B., and Schwartz, R. (2010). TER-Plus: Paraphrase, Semantic, and Alignment Enhancements to Translation Edit Rate. *Machine Translation*, 23(2-3):209–240.

[Surdeanu et al., 2005] Surdeanu, M., Turmo, J., and Comelles, E. (2005). Named Entity Recognition from Spontaneous Open-Domain Speech. In *Proceeding of the 9th International Conference on Speech Communication and Technology (Interspeech)*, pages 3433–3436. ISCA.

[Tillmann et al., 1997] Tillmann, C., Vogel, S., Ney, H., Zubiaga, A., and Sawaf, H. (1997). Accelerated Dp Based Search For Statistical Translation. In *European Conference on Speech Communication and Technology (EUROSPEECH)*, pages 2667–2670.

[Vilar et al., 2006] Vilar, D., Xu, J., D'Haro, L. F., and Ney, H. (2006). Error Analysis of Machine Translation Output. In *Proceeding of the 5th International Conference on Language Resources and Evaluation (LREC)*, pages 697–702, Genoa, Italy.

[Zeman et al., 2011] Zeman, D., Fishel, M., Berka, J., and Bojar, O. (2011). Addicter: What Is Wrong with My Translations? *The Prague Bulletin of Mathematical Linguistics*, 96:79–88.

Appendix A

*t*Search User Manual

A.1 Getting started

Let us introduce the user manual of *t*Search[1], a web-based application that aids the error analysis stage of machine translation development facilitating the qualitative analysis of translation quality. The *t*Search Online Interface is accessible at `http://asiya.lsi.upc.edu/demo/` where you can find to ways to access it. The first one consists of evaluating a testbed with Asiya and once the evaluation is completed, *t*Search appears as one of the tools that the user can run. However, if you have the data already evaluated by Asiya, the second option allows you to upload the compressed folder that contains the Asiya evaluation output and start using *t*Search.

A.1.1 Getting to know *t*Search

The Figure A.1 describes some of the features available in *t*Search Online Interface:

1. **Toolbar**: use the toolbar to find all metrics, systems and documents, operate with groups, and view examples and select the functions and operations available.
2. **Output area**: this area displays the results of your query.
3. **Query input**: use this input box to write yout query.
4. **View tabs**: navigate through the different organization views: all segments, by system or by segment.

[1]There is also a video tutorial available at `www.youtube.com/watch?v=GBEnmOsKmT4&vq=hd720`.

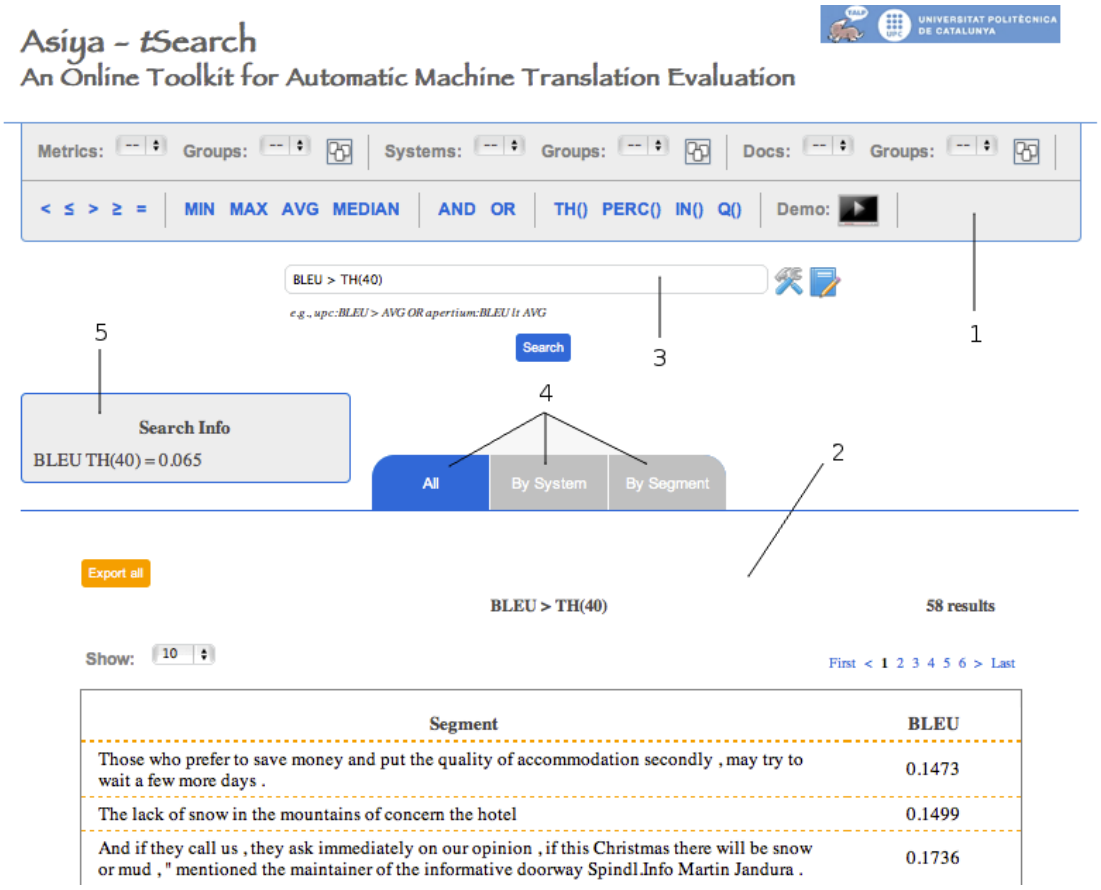

Figure A.1: Getting to know *t*SEARCH

5. **Info panel**: gives you additional information related to the query such as groups of metrics, systems and documents, and the actual values used for the statistical functions such as the `MIN`, `MAX`, `AVG`, `MEDIAN`, `TH()`, `PERC()` or `Q()`.

Click the following icons to...

Toolbar

 Show and hide the toolbar.

 Create or edit a group of metrics, systems or documents.

 See the video manual.

Other

 Export as an XML file the partial results depending on the current view.

 Show and hide the examples window.

A.1.2 Views

*t*SEARCH lets you navigate the results of the search accross all the automatic translations selected and their evaluations. Three different views organize the segments according to the user preferences:

All: this view shows all segments and the scores for the metrics involved in the query.

By system: it groups the segments by system name and, for each system, by document name.

By segment: this view offers the segment organization, which facilitates the comparison between several translations, the reference and the source for each segment.

A.2 Create and Edit Groups

The interface allows to create groups of systems, documents and/or metrics. The purpose of this feature is to facilitate the comparison between types of systems (e.g., stadistical vs.

rule-based) or metrics (e.g., lexical vs. syntactic) or even, groups of documents that belong to different domains.

The following steps describe how to create a new group:

1. From the toolbar, click the Groups button  . The button is in three different blocks in order to distinguish between metrics, systems and documents.
2. Write the name of your new group at the *Group name* field.
3. Chose form the left panel what do you want to include in your group passing them to the right panel.
4. Click the create button.

Later on, if you want to edit an existing group...

1. From the toolbar, click the Groups button  . The button is in three different blocks in order to distinguish between metrics, systems and documents.
2. Select from the *Groups* list the one you want to edit and then, its name and elements are displayed.
3. Edit the values you want to change, i.e., the name of the group or the elements, passing to the left panel the ones you want to eliminate from the group or passing to the right panel the ones you want to include.
4. Click the update button.

A.3 Let's query

There are several types of queries, depending on the operations used: arithmetic comparisons, statistical functions (e.g., average, quartiles), range of values, linguistic elements and logical operators. Table A.1 lists some of the most representative queries of each group.

Regarding metric-based queries, the arithmetic comparison queries let you obtain all segments scored above/below a value for a concrete metric. Such value can be a real number or also a statistical variable such as minimum `MIN`, maximum `MAX`, median `MEDIAN`, average `AVG` or the threshold function `TH`(). We have also implemented statistical functions such as the quartile function `Q`() or the percentile `PERC`(`n`, `M`), which returns all the segments with a

score in the n^{th} part, when the range of scores is divided in M parts of equal size. The last query in this group refers to the system comparison. Thus, given an evaluation measure, it allows comparing its score between systems.

Concerning linguistic-based queries, we have implemented queries that match N-grams of lemmas `lemma`, parts-of-speech `pos` and items of shallow `SP` or constituent parsing `CP`, dependency relations `DP`, semantic roles `SR` and named entities `NE`. The `DP` function allows specifying a structure composition criterion (i.e., the categories of two words and their dependency relationship) and even a chain of relations. The `SR` function obtains the segments that match a verb and its list of arguments. The use of the asterisk symbol substitutes any value, e.g., $\texttt{LE}[\texttt{SP}(\texttt{NP}, *, \texttt{PP}), \texttt{DP}(*, *, \texttt{V})]$. However, when combined with semantic roles, one asterisk substitutes any verb that has all the arguments specified, e.g., $\texttt{LE}[\texttt{SR}(*, \texttt{A0}, \texttt{A1})]$, whereas two asterisks in a row allow arguments to belong to different verbs in the same sentence.

The above queries are applied at segment level. However, applying them at system and document-level is as easy as specifying the system and/or document names, e.g., `(upc:BLEU > AVG) AND (upc:LE[DP(*,nsubj,*)])`. In addition, there is also the possiblity to use a group of metrics, systems and/or documents instead, e.g., `(LEX:RBMT > AVG) AND (RBMT:LE[DP(*,nsubj,*)])`, where `RBMT` is a group of rule-based systems and `LEX` is a group of lexical metrics defined and created by the user.

Metric-based Queries	**Arithmetic Comparison**	BLEU > 0.4 BLEU > TH(40) BLEU le MEDIAN
	Range of Values	BLEU IN [0.2, 0.3) BLEU IN Q(4) BLEU IN PERC(2,10) BLEU IN (TH(20),TH(40))
	Sistem comparison	upc:BLEU > dfki:BLEU
LE-based Queries	**N-grams**	LE[SP(NP,*,PP)] LE[CP(NP,PP)] LE[lemma(be),CP(VP,PP)] LE[pos(DT,JJ,*)] LE[NE(ORG)]
	Semantic Roles	LE[SR(ask,A1,AM-TMP)] LE[SR(*,A1,AM-TMP)] LE[SR(**,A1,AM-TMP)]
	Dependency Relationships	LE[DP(N,nsubj,V)] LE[DP(N,nsubj,V,dep,V)] LE[DP(*,nsubj,*)]
Group Creation and Complex Queries	**Logical Composition**	BLEU > AVG AND LE[DP(N,nsubj,V)] LEX = {BLEU,NIST} SYN = {DP-Or(*),SP-Op(*)} SMT = {bing,google} (SMT:LEX > AVG OR apertium:LEX < AVG) AND (SMT:SYN < AVG OR apertium:SYN > AVG)

Table A.1: *t*SEARCH query examples

Appendix B

Query Language Grammar

We present de grammar defined for the *t*SEARCH query language. The code is written in Perl[1] and uses the package *Parse-RecDescent*, which is available at CPAN[2] [3]. Note that this is a simplified version, since we have eliminated the code regarding the generation of the output for the better understanding of the reader.

```
parse : expr eofile { $item[1] }

expr  : query {$item[1]}

query   : term query_[ $item[1] ]
query_  : logic term query_
          [[{$item[1] => {op1 => $arg[0], op2 => $item[2]}}]]
        | { $arg[0] }

term  : OPENPAR expr CLOSEPAR { $item[2] }
      | simple_query { $item[1] }

simple_query  : system_comp op system_comp
              | op_metric func
```

[1] `http://www.perl.org/`

[2] Comprehensive Perl Archive Network `http://www.cpan.org/`, a repository of Perl modules available for the community

[3] `http://search.cpan.org/~jtbraun/Parse-RecDescent-1.967009/lib/Parse/RecDescent.pm`

```
              | 'LE' OPENCLAU LEs CLOSECLAU { $item[3] }
              | op_metric_LE 'LE' OPENCLAU LEs CLOSECLAU


# Linguistic-based Queries

LEs    : X LEs_[ $item[1] ]

LEs_   : COMMA X LEs_
         [[{AND => {op1 => $arg[0], op2 => $item[2]}}]]
       | { $arg[0] }

X      : TERMX X_[$item[1]]
X_     : COMMA TERMX X_
         [[{AND => {op1 => $arg[0], op2 => $item[2]}}]]
       | { $arg[0] }

TERMX  : (ngram_term | DPterm | SRterm)


CPterm : 'CP' OPENPAR IDENT CLOSEPAR


# n-grams Queries


ngram_term  : ngram_term2 ngram_term_[ $item[1] ]
ngram_term_ : COMMA ngram_term2 ngram_term_
              [[{AND => {op1 => $arg[0], op2 => $item[2]}}]]
            | { $arg[0] }

ngram_term2 : WHAT OPENPAR ngram_X[$item[1]] CLOSEPAR {++$q; $counter = 0;}

ngram_X     : ngram_TERMX[$arg[0]] ngram_X_[$item[1], $arg[0]]
ngram_X_    : COMMA ngram_TERMX[$arg[1]] ngram_X_
              [[{AND => {op1 => $arg[0], op2 => $item[2]}}], $arg[1]]
            | { $arg[0] }

ngram_TERMX : (IDENT|'*')

WHAT        : ('SP' | 'CP' | 'pos' | 'lemma' | 'NE') {$item[1]}
```

```
# Dependency Relation Query

DPterm  : 'DP' OPENPAR DP_elems CLOSEPAR
        | DPterm2 DPterm_[ $item[1] ]

DPterm_ : COMMA DPterm2 DPterm_
          [[{AND => {op1 => $arg[0], op2 => $item[2]}}]]
        | { $arg[0] }

DPterm2 : 'DP' OPENPAR DP_X[$item[1]] CLOSEPAR

DP_X    : (DP_elems|'*') COMMA (DP_elems|'*') COMMA (DP_elems|'*') DP_X_
          [{rel => {SP => $item[1], DP => $item[3]},
            SP => $item[5]}, $arg[0], $item[5]]

DP_X_   : COMMA (DP_elems|'*') COMMA (DP_elems|'*') DP_X_
          [[{AND => {op1 => $arg[0],
                     op2 => {rel => {SP => $arg[2], DP => $item[2]},
                     SP => $item[4]}}}],
            $arg[1], $item[4]]
        | { $arg[0] }


# Semantic Role Query

SRterm : 'SR' OPENPAR SR_X CLOSEPAR

SR_X   : (IDENT|'**'|'*') COMMA SR_TERMX
         [$item[1]] SR_X_[$item[3], $item[1]]
SR_X_  : COMMA SR_IDENT SR_X_
         [[{AND => {op1 => $arg[0],
                    op2 => {SR => $item[2], verb => $arg[1]}}}],
            $arg[1]]
       | { $arg[0] }

SR_TERMX : SR_IDENT

SR_IDENT : /[a-zA-Z][a-zA-Z0-9_]*\-[a-zA-Z][a-zA-Z0-9_]*/ | IDENT
```

```
op_metric_LE : doc ':' system ':'
             | system ':' doc ':'
             | m{$docs} ':'
             | m{$systems} ':'
             | m{$system_groups} ':'
             | m{$doc_groups} ':'



# Metric-based Queries

op_metric : system ':' doc ':' metric
          | system ':' metric ':' doc
          | metric ':' doc ':' system
          | metric ':' system ':' doc
          | doc ':' metric ':' system
          | doc ':' system ':' metric
          | system ':' metric
          | metric ':' system
          | doc ':' metric
          | metric ':' doc
          | metric

func : op op_functions
     | op system ':' op_functions
     | op doc ':' op_functions
     | op system ':' doc ':' op_functions
     | op doc ':' system ':' op_functions
     | IN in_functions
     | IN system ':' in_functions
     | IN doc ':' in_functions
     | IN system ':' doc ':' in_functions
     | IN doc ':' system ':' in_functions

op_functions : threshold | function | decimal

in_functions : percentile | quartile | range

# In functions
```

```
percentile : PERC OPENPAR INT COMMA INT CLOSEPAR

quartile : Q OPENPAR INT CLOSEPAR

range : left_part COMMA right_part

left_part : (OPENPAR | OPENCLAU) (DECIMAL | threshold)

right_part : (threshold|DECIMAL) (CLOSEPAR | CLOSECLAU)


# Statistics functions

threshold : TH OPENPAR INT CLOSEPAR

function : (MIN | MAX | AVG | MEDIAN)


logic : (AND | OR)
op : (GE | LE | EQ | GT | LT)


system : m{$systems} | m{$system_groups}
doc : m{$docs} | m{$doc_groups}
metric : m{$metrics} | m{$metric_groups}
decimal : DECIMAL



# keywords

AND : "AND" | "&&"
OR : "OR" | "||"
IN : "IN"
EQ : "eq" | "="
GT : "gt" | ">"
LT : "lt" | "<"
GE : "ge" | ">="
LE : "le" | "<="
OPENPAR : "("
CLOSEPAR : ")"
```

```
OPENCLAU : "["
CLOSECLAU : "]"
PERC : "PERC"
TH : "TH"
Q : "Q"
MIN : "MIN"
MAX : "MAX"
AVG : "AVG"
MEDIAN : "MEDIAN"
COMMA : ","
IDENT : /[a-zA-Z][a-zA-Z0-9_]*/
INT : /\d+/
DECIMAL : /^-{0,1}\d+((\.|,)\d+)?/
DP_elems : (EQUAL|PLUS|MINUS|ARROBA|PERCENT|
EXCLAMATION_D|EXCLAMATION_U|DOLAR|ADVERSAND|
DP_OPENPAR|DP_CLOSEPAR|DP_OPENCLAU|DP_CLOSECLAU|
DOUBLE_QUOTES|QUOTES|DOUBLE_SINGLE|SINGLE_QUOTE|
DP_COMMA|STOP|SEMICOLON|COLON|IDENT) {$item[1]}
SINGLE_QUOTE : "\\'"
DOUBLE_SINGLE : "\\'\\'"
QUOTES : '"'
DOUBLE_QUOTES : '""'
STOP : "."
SEMICOLON : ";"
COLON : ":"
DP_COMMA  : ","
DP_OPENPAR : "("
DP_CLOSEPAR : ")"
DP_OPENCLAU : "["
DP_CLOSECLAU : "]"
ADVERSAND : "&"
DOLAR : "\$"
EXCLAMATION_D : "\\!"
EXCLAMATION_U : "¡"
PERCENT : "%"
ARROBA : "@"
PLUS : "+"
MINUS : "-"
EQUAL : "="
eofile : /^\Z/
```