Evaluating Grid Infrastructure for Natural Language Processing

Radovan Garabík[1], Jan Jona Javoršek[2], and Tomaž Erjavec[2]

[1] Ľ. Štúr Institute of Linguistics, Slovak Academy of Sciences, Bratislava, Slovakia
garabik@kassiopeia.juls.savba.sk
[2] Jožef Stefan Institute, Ljubljana, Slovenia
jan.javorsek@ijs.si tomaz.erjavec@ijs.si

Abstract. In this article we analyze common human language technology requirements and the possibility of implementing them using Grid infrastructure. Different possibilities for the setup of an execution environment are treated and the standard PKI based Grid security approach is explained, with an emphasis of securing data access in a potentially untrustworthy environment. Two examples of running unmodified NLP applications are presented.

1 Introduction

Increasing computing requirements for acquiring and processing large textual data-sets and working with larger and larger corpora in Natural Language Processing (NLP) and related disciplines together with ever increasing availability of computing resources allow us to work on NLP algorithms and tasks that were impractical just a few years ago. The core of the problem shifted from obtaining access to enough computation power and from optimizing algorithms into developing efficient ways of allocating the computing resources to various tasks and into finding efficient ways of dealing with huge amounts of data. Since the most accessible computing environment moved from large centralized supercomputers into the vast number of available servers connected with the ubiquitous Internet, a new paradigm in computing emerged: massively parallelized algorithms running on widely distributed networks of interconnected computers.

One of the infrastructure approaches is the Grid network, which provides a complete environment for heavy-duty computational tasks, with working solutions for user authentication, data storage, distributing load over the available resources, access control and the whole infrastructure for user management. First used mostly for computing tasks in high energy physics, the Grid is nowadays used for tasks in several different research areas. The use of Grid infrastructure for NLP has been previously discussed with several proposals (cf. [19], [4], Neuroth et al. [14], [9], [10], [12], [11], [13], reviewed in [6]).

In this article, we explore the idea of utilizing the Grid infrastructure for NLP-related tasks. Not just the computational requirements, but also commonly useful features of shared data repositories and existing Grid security features are discussed. Since the whole Grid environment runs exclusively on the GNU/Linux operating system (although there are efforts underway to port the Grid software to other Unix-like platforms, such as the BSD family and Mac OS X), our point of view will deliberately be

rather 'GNU/Linux centric', and the discussed software environment and tools will be implicitly understood to be GNU/Linux specific (unless stated otherwise).

2 Legal issues

The actual deployment of Grid computing in the natural language processing area (especially relevant for corpus linguistics) faces specific legal issues – the data being processed are in majority of cases copyrighted, and the research institutions either have very strict legal agreements governing the use of the data, or are operating entirely on copyright law sections allowing scientific and research use of the data (*fair use* in the U.S.A. jurisdiction, citation and educational use in many of the EU countries' copyright laws). The situation is somewhat similar to the problems the users of Grid computing in health care systems – though in that case, metadata are the most sensitive and protected part of the data-set, while in corpus linguistics the data (i.e. texts in the corpora) are sensitive, but the metadata is usually freely accessible [17].

In any case, the research institution using the data for research most likely does not have the right to distribute the data *at all*. If the contractual obligations prevent the institution from physically copying the data beyond the premises of the institution, it might be still advantageous to use the Grid infrastructure for computing clusters of the institution itself, and use middleware functions to restrict data-replication to those processing nodes and data storage elements physically located in the organization. This way, the whole Grid can still be used for less sensitive tasks, or for post-processing the results of operations on sensitive data (when the post-processing does not include access to sensitive data), while at the same time the computing nodes will be available as part of the whole Grid computing pool when they would be left idle otherwise.

While the actual uploading of the data to Grid-enabled storage is not to be considered a form of 'distribution' as long as no other person or organization is allowed to get the data, it is nevertheless desirable to protect the data from casual snooping. For one thing, an administrator of the Grid node where the data physically reside can get access rather trivially; and while he or she is legally obliged not to misuse his access (usually by rather strict agreements, in the case of European Grid infrastructure), a measure of additional protection seems to be necessary – to avoid data leaking in case the computer hosting the Grid node is compromised, unbeknown to the administrators. We discuss security measures used in the Grid infrastructure in the following section.

3 Software environment in the Grid

The different implementations of Grid middleware all follow the same workflow: the user has to provide a way to parallelize the computing task in a number of jobs that can be run in parallel and encode the solution by providing a *job specification*, indicating the data files required (using URIs), suitable computing environment (ABI, API, execution environment), computing time and resources needed (wall-time, RAM, disk) and a way to store the results.

This can be done manually, and a job is then submitted with a dedicated command, or, alternatively, can be done with the help of a dedicated web application, usually domain or experiment specific.

The system then takes care of selecting the appropriate Grid site and free worker nodes, downloading the data files, making the pre-installed software (execution environment) available and starting the job script. A number of tools enable the user to monitor the progression of the task, including its standard input and standard/error output, working directory contents etc.

When the job finishes, it may upload the resulting data files to a Grid-enabled disk storage, or, alternatively, the user can use command-line tools to download the data from the working directory directly.[3]

To make the system work reliably and securely, a number of information, monitoring and accounting systems are part of the infrastructure. In addition, an advanced security model is used to ensure resource protection and data integrity. As we have to consider this system's suitability for protection of copyrighted data-sets in linguistic resources, a short overview is presented in the following subsection.

3.1 Security

Computing grids had to be very security-conscious from the very beginning, since the very premise of a Grid network is, from the point of view of the site administrator, to give external users access to the local computing infrastructure and, from the point of view of Grid users, to entrust data and applications to untrusted, foreign sites.

Moreover, the basic requirement for a viable, scalable and sustainable security infrastructure in the context of large Grid networks has to be a robust solution with as few single points of failure as possible to avoid failures of security services that could effect negatively the availability of the whole infrastructure.[4]

Grid security has several components: **(a)** Authentication, a method of confirming the identity of the user or organization behind an operation, is implemented on the basis of the Public Key Infrastructure (PKI) and standard x509 digital certificates (with a number of extensions to facilitate the use of PKI in the context of Grids). **(b)** Authorization is provided in the framework of virtual organizations (VOs), a mechanism enabling Grid users all over the world to organize themselves according to research topics and computing requirements, regardless of geographic constraints, and permitting sites to regulate the use of their resources according to user, discipline, software requirements etc. **(c)** Monitoring and ticketing permits users and administrators to keep track of infrastructure availability and to react to technical and security matters in a timely fashion. **(d)** Accounting reports on the use of the infrastructure and enables the community to regulate and enforce the use of the infrastructure.

Public Key Infrastructure. Public Key Infrastructure, first introduced to the general public in the context of securing the web and enabling on-line shopping and banking, has become the standard authentication model in many application domains. Defined

[3] See [5] for architecture overview of the NorduGrid ARC middleware.

[4] For an overview, see [8].

by a number of Internet Drafts, RFCs and standards, PKI is a widely deployed and evolving system.[5]

PKI is based on the property of asymmetric ciphers, where a different key is used for encryption and decryption. This property allows the encryption key to be always kept private and secret and the decryption key to be public, usually published with some information about the owner of secret key in the form of a x509 digital certificate.

In PKI, such a digital certificate is used as the token of identification: it is issued by a certification agency (CA) on the basis of an identification process (i.e. checking legally acceptable personal ID documents in person). But the certificate is coupled with a secret key that has been generated by the user requesting the certificate and is never exposed to the CA. To issue a certificate, the CA now sets up information about the entity (user, host or service) to be certified in accordance to the identification data provided in a standard form called a Distinguished Name (DN, following a LDAP-like name scheme: `CN = Joe User, OU = My Department, DO = Institute of Dispersive Linguistics, DC = San Marino`, and signs it with their own secret key from the CA certificate.

This scheme ensures that nobody, not even the CA, can use the certificate (since only the owner of the certificate possesses the secret key) and protects the information in the certificate with the signature, produced with the CA's own secret key.

To make the system work, CA certificates with public keys are published in a well advertised manner (or shipped with software, such as. web browsers, Grid middleware packages and GNU/Linux distributions). Recipient of a document or a connection that uses a client certificate and is encrypted or signed with such a certificate can therefore verify that the document or connection really was encrypted or signed by the said certificate by decrypting it with the public key included in the certificate, and it can verify the information in the certificate by checking the certificate with the CA public key in the same manner.

A number of additional security measures are used in the Grid: CA secret keys are kept in off-line systems or in dedicated certified hardware modules (hardware security modules or HSM) while end-entity certificates are re-issued with new keys yearly or kept in hardware security tokens. In addition, actual user certificates are never entrusted to non-trusted entities: for almost all operations in the Grid, short-lived proxy certificates are used instead (described bellow).

Virtual organizations. While PKI provides authentication, a different system is needed to provide authorization, i.e. to help decide if a given user, host or service is to be allowed to carry out a specific task: use a specific resource or access specific data. In the context of Grid computing infrastructure, this role is implemented in the framework of virtual organizations (VOs).

A Virtual Organization serves two purposes: **(a)** As an organizational form, a VO permits a number of researches from different organizations, usually geographically dispersed, to collaborate and share tools, data and resources. **(b)** In the Grid security infrastructure, a VO provides means of regulating access to resources, i.e., a VO provides authorization after authentication is provided by PKI.

[5] See [2] for an extensive up-to-date overview of the relevant documents.

With this combination of roles, Virtual Organizations have proven themselves to be most efficient in enabling a higher level of international collaboration and have permitted the European Grid network to foster new, faster development in many disciplines by providing an unprecedented framework for international collaboration.

In practice, members of a research project or a discipline can set up a VO and decide on its modes of operations and access to resources quite independently. They have to decide what kind of tools the VO members will be using in the Grid, define the data formats, prepare data repositories, develop execution environments with the tools installed and set up a Virtual Organization Membership Service server (VOMS server) to store authorization credentials.

Then some resources have to be made available to the community of VO members. In practice, that means obtaining support of a number of Grid sites (organizations owning computing clusters partaking in the Grid) that have to configure their Grid middleware installations to include the new VOMS server in its authorization procedures and to either install the execution environment (or, more realistically, environments) for the VO or give access to some members of the VO so that they can perform the installation and maintenance if the execution environment on the site themselves. Additionally, a number of Grid storage elements (SE) has to be configured to allow the VO members to access and store the data on their disk space.

Proxy certificates. With the VO and VO supporting Grid sites, a VO member can submit Grid jobs and access VO-owned data using his certificate. This is implemented in an indirect manner by means of Grid proxy certificates, as mentioned previously in the discussion of PKI infrastructure.

Grid proxy certificates are **(a)** primarily used to permit a job to authenticate in the name of the user spawning the job, without the requirement of direct user interactions during the course of the job. This means that the proxy certificate must have the same DN as the users' certificate, but it has a different secret key which is not protected with a pass-phrase that would require user interaction on the keyboard. Proxy certificates are generated with a tool that uses the users' certificate to sign the proxy (as if it were a CA), thus confirming that the proxy was indeed generated by the user. In addition, gird proxy certificates are protected with file permissions and are always short-lived (from several hours to a few weeks) to mitigate the risk of the unprotected secret key.

To interact with the VO authorization system, the user generates a VOMS Grid proxy certificate that obtains special certificate extensions from the VOMS server and incorporate them in the proxy certificate. These extensions encode VO group and role attributes of the user and are themselves signed by the VOMS server with its service certificate, using the PKI infrastructure's authentication facilities to implement an authorization layer.

In this manner, a job can obtain authorization to use computing resources and data simply by providing a suitable VOMS proxy certificate. Its attributes are recognized by the Grid manager servers that provide it with to data storage (storage resource managers, SRM) and other resources.

As and additional level of security, Grid managers assign each job a temporarily unique user ID in the underlying operating system mapped from its active VO role in

such a way that no jobs with different roles (and therefore potentially different access permissions) can share access on the underlying implementation.

In this way the system implements fine-grained control over the use of Grid resources and data without any reliance on the availability of authentication and authorization servers, thus avoiding a single point of failure that would have a significant impact on the scalability of the system.

Data Protection. Using these security components, additional measures of data protection can be implemented when necessary. In the context of NLP, such a measure is of critical importance, since most of the data-sets in corpus linguistics contain copyrighted texts that need to be protected.

To solve this problem, the corpus data has to be suitably protected where it is permanently stored. Therefore we propose to store the corpus data in encrypted form in a dedicated storage element and set up the access authorization in such a way that access is restricted to VO users who belong in a VO group of users who signed the necessary legal agreements to access the data. Furthermore, we propose that the data is transferred to the untrusted environment of Grid worker nodes, where jobs perform their computations, in the encrypted form and that the decryption keys are issued to the jobs protected with asymmetric encryption decryptable only by the job's Grid proxy keys so that only the jobs can access the keys and decrypt the data.

In this manner, access and decryption is regulated with the authorization of embedded VOMS attributes in the proxy certificate without any additional authorization steps, while the data is never shipped or stored in unencrypted form.

If the tools used by the job have to store temporary files on disk, these are protected from other processes (with the exception of system administrators, who are already bound by strong agreements pertaining to data security on the Grid) and are in addition of short-lived nature.

There exist different implementations of the system described. The simplest form involves the use of a decryption filter in the job script and is rather simple to deploy. A more flexible solution, based on CryptoSRM (cryptographic storage resource manager) and Hydra Key Storage (a distributed fragmented encryption key storage system) is described in [17].

4 General requirements of NLP related tasks

Contemporary NLP tasks are rather varied; some of them require a lot of "pure" computing power, but many tasks, especially in the area of corpus linguistics, merely process large data files. From the software point of view, the tools used cannot be more diverse – they are often programmed in typical computer languages, like C or C++, but a lot of data processing is done in scripting languages, such as Perl or Python, and Java is increasingly popular, and more often than not, one specific task uses several different tools bound by short programs written in a shell script. The use of (high level) scripting languages even for the computing intensive tasks means that the analysis is less effective than it could be, but the ease of creating and maintaining the tools more than outweights this particular disadvantage. From this follows than the tools are often

fragile and require a specific environment, which sometimes means that even using a different GNU/Linux distribution that the one the software has been developed on can be a major problem.

The Grid environment, due to its initial connection with the use in High Energy Physics, predominately uses Scientific Linux CERN distribution (SLC) version 4 for the job computing environment (with a changeover to version 5 currently in progress). The ideal solution would be of course to put all the necessary NLP software into the execution environment (which is available at each of the computing nodes) and use the standard distribution. It is, however, sometimes much more convenient to use an operating system environment more suitable for the users and their tools. There are two possible solutions: to run under a chroot environment or to use virtualization. Both options are discussed below.

4.1 Userspace and full virtualization

Chroot is a UNIX system that changes the effective root of the filesystem for the process and its children. The basic usage for chroot is twofold: it can be used to restrict untrusted (or potentially dangerous) processes from accessing the rest of the filesystem, or it can be used to run processes in a different filesystem environment (different filesystem layout with different system executables and dynamic libraries). It should be noted that chroot does not offer true virtualization since isolation from the host system is not complete – in particular, system kernel, networking subsystem and process management are shared with the host system, so that the processes in the chroot environment cannot bind to sockets that are used on the host system (and vice versa), and if process management is to be possible in a chroot environment, the `proc` filesystem has to be mounted inside chroot environment, enabling the guest to access the information about host processes[6].

On the other side of the spectrum, there are complete virtualization solutions, emulating the guest system. These can emulate the CPU completely in software (approach commonly used in emulating vintage computers on modern operating systems, or when a computer platform switches the architecture), or run the guest machine natively, trapping and emulating only privileged or unimplemented instructions. Modern computer architectures usually offer dedicated hardware features to facilitate the implementation of virtual machines[7], some mainframe architectures even offer complete, seamless virtualization in hardware.

Then there are several different approaches that lie somewhere in between those two extremes, ranging from **(a)** paravirtualization, which requires cooperation from the guest operating system kernel (in order to achieve negligible performance loss due to the virtualization), used e.g. by the `XEN` virtualization solution; to **(b)** compartmentalization (i.e Linux virtual servers and `OpenVZ`), which divides the host operating system into different compartments with completely separated processes, network access and

[6] This does not matter as much as it seems as long as the chrooted processed run under different PID from the host ones, because a non-root user cannot affect other processes, and a chrooted superuser can break out of the chroot anyway.

[7] Until rather recently (before the introduction of VT-x and AMD-V), such features were not available in common Intel-compatible off-the-shelf computers.

filesystems but sharing the same kernel; to vanilla kernel namespace support, which only separates user and process management (slightly extending chroot separation).

The virtualization techniques mentioned differ on performance impact [15] – ranging from none at all in case of a simple chroot or chroot with namespaces, over very little for `OpenVZ`-like compartmentalization to a more significant one for full virtualization. The specific areas of impact vary, too – while the raw CPU performance rarely decreases by more than a few percent (with the exception of complete software emulation of the guest architecture), I/O penalties are sometimes severe.

From our point of view, the best way to use the specific software is to install it inside a runtime environment which is made available to the jobs when submitted to the Grid. This is directly supported by the Grid infrastructure and requires no additional steps or privileges. However, at this time this requires a significant effort, since all the tools and their dependencies have to be compiled (or installed in a non-standard location inside the runtime environemnt) on the standard SLC distribution, which can be a problematic if the software has many external dependencies.

Installing a chroot environment, on the other hand, enables us to avoid porting the software to the SLC distribution – inside the chroot, we can install any reasonably standard GNU/Linux distribution and any necessary software packages. In addition, many of the commonly used distributions already have support for (at least partial) installation inside a chroot environment built in. But in the context of Grid infrastructure this solution has a significant disadvantage, since it requires support from the cluster administrator since chroot environments are not a standard feature of the Grid environment.

Using a complete virtual machine allows us to run a complete GNU/Linux distribution, with completely separate networking support and user management, including the ability to run processes with superuser privileges, and the ability to use filesystems otherwise not supported by the host system[8]. But the main advantage is the possibility to run completely different operating system[9]. However, installing and using virtual machines requires not just administrator cooperation, but often also nonstandard host operating system extensions (such as special kernel modules). One of the more interesting virtualization systems in this context is User Mode Linux, which does *not* require any special host support, runs as an ordinary user process and provides a complete guest Linux kernel environment. Unfortunately, guest environment in this case suffers from a big I/O performance degradation, which can be a noticeable problem when dealing with very large corpus data.

While there is significant research in the use of different kinds of virtualization in the context of Grid technologies, this is not a wide spread feature at this time. We have been able to use clusters with full support for chroot environemnts, but we realize that for quick adoption and widespread use of Grid computing in NLP, porting of tools to the most often supported environment, i.e. SLC, will be necessary.

[8] Such as encrypted filesystems.

[9] Therefore we can use e.g. the tools available only for Microsoft® Windows® family of operating system, if we can get around their mostly point-and-click nature and run them noninteractively.

5 Proof of concept

This section presents and experiment in using the Grid to execute two NLP tasks for the Slovak language. The first subsection introduces TectoMT, a machine translation system, the second morphosyntactic tagging with morče, and the third gives the experimental usage of the two systems on the Grid.

5.1 TectoMT

TectoMT [20] is a software framework aimed at machine translation at the tectogrammatical level of analysis. The system is modular – the framework itself consists of many independent modules (*blocks* in TectoMT terminology), each implementing one specific, independent NLP-related task. Each of the blocks is a Perl module that interacts with the system using a single, uniform interface. However, sometimes the module serves only as a wrapper for the underlying implementation in another programming language. The tectogrammatical annotation and consequently the TectoMT framework primarily stores linguistic data in its own format, called TMT. TMT is an XML-based format, designed as a schema of the Prague Markup Language (PML)[10] [16]. Nevertheless, its blocks are by no means obliged to use this format.

TectoMT has been developed with modern Linux systems in mind, and as such its installation requirements are easily met by any contemporary Linux distribution. It should be noted that TectoMT, being written mostly in Perl, depends on many external Perl modules and its installation scripts are intelligent enough to automatically download and install any missing dependencies; this, however, circumvents standard distribution packaging systems, therefore it is better to install all the necessary packages with the packaging system tools before attempting to install TectoMT. There are also some C language modules that are not compiled by default, but have to be compiled separately inside the TectoMT installation source tree.

TectoMT also has some built-in capabilities for parallelization of its tasks, using the Sun Grid Engine – it is possible to adapt the Sun Grid Engine batch software to various Grid middlewares [3], but TectoMT can be run on the Grid system directly without relying on its internal parallelization possibilities, if the user takes care of splitting the input data into appropriate chunks for parallel processing.

5.2 Tagging a corpus

Morphosyntactic tagging of the Slovak National Corpus consists of two steps. The first performs morphosyntactic analysis, where each word in the input texts is assigned a set of possible morphosyntactic tags. This step essentially consists of looking up the possibilities of lemma/tag combinations in a constant database table using the wordform as a key, with an additional step for unknown words, where the list of possible tags is derived from the similarities of word endings to the ones present in the database tables. The software is implemented in the Python programming language and is actually quite fast, since the core of the task consists simply of a look-up in the possibilities in the

[10] Not to be confused with the Physical Markup Language.

tables, and most of the CPU work is spent on I/O operations, parsing the input file and assembling the output. On a reasonably recent hardware (Intel Xeon 2.33 GHz CPU) it is able to process over 10 000 words per second. It can also parallelize easily, since the words can be analyzed independently of each other.

The second step is disambiguation, where each word is assigned a unique lemma and a morphosyntactic tag out of the possibilities assigned in the first step. For disambiguation, we use *morče*, an averaged perceptron model originally used for the Czech language tagging [18], re-trained on the Slovak manually annotated corpus. Disambiguation is much slower that the morphology analysis, its average speed reaches only about 300 words per second. Parallelization at the application level is also not possible without some redesign of the *morče* itself, but the nature of tagging makes it easy to split the input data into as many chunks as we want and run *morče* instantiations in parallel.

Given the speed differences between morphology analysis and disambiguation, we can safely consider the morphology analysis execution time negligible and design the whole tagging to be done in one step, without the need to parallelize the morphology analysis process while the disambiguation is to be run in parallel.

5.3 Installation and usage

As our GNU/Linux distribution of choice is Debian[11], we did all the testing on the Squeeze (testing) Debian distribution, which is a "moving target" distribution, meant for users that want newer version of the distribution and included packages, but do not want to deal with (potentially) broken bleeding edge packages from the *unstable* Debian repositories. To summarize, a package will get into *testing* if it has no release-critical bugs, has spent several days in the *unstable* repository and its inclusion in *testing* will not break other packages. We used *testing* deliberately, because it is advantageous to use new versions of the required packages which will not become obsolete in near future, even if the packages in *testing* repositories will be rather quickly replaced by still newer versions.

Debian has a standard method for installing the base system into a chroot environment, implemented by a tool, called `debootstrap`[1].

Once the chroot environment was created, we proceeded and installed TectoMT Perl dependencies into the chroot system. Finally, we ran the TectoMT installer. Similarly we installed the Slovak version of the *morče* software from within the chroot. The installation process was unremarkable comparing to a regular installation.

After the chroot environment had been prepared and installed on the Grid servers, we were able to submit batch jobs with either TectoMT or the disambiguation tasks. We used the Grid infrastructure to morphosyntactically tag and analyse the spoken Slovak language corpus [7]. We divided the corpus (434 676 words) into 11 approximately equally sized chunks and submitted them to be tagged in 11 independent batch jobs, then joined the results. The run times ranged from 103 to 236 seconds, with the average time around 141 seconds per job[12]. Even including the overhead spent on submitting

[11] `http://www.debian.org`

[12] Times mentioned are *wall times*, i.e. total time elapsed from the beginning to the end of the task, not including the time spent waiting in the job queue or downloading the data files.

and waiting for the job to start, this is a significant reduction of the total time needed to tag the corpus. In fact, as far as the typical Grid tasks are concerned, our jobs were very short, and we could achieve less overhead by running longer jobs (a typical Grid jobs takes from 2 hours to more than a day). Such a s setup will actually be typical when tagging a bigger, representative text corpus.

6 Conclusion

We demonstrated the possibility to use the Grid infrastructure for NLP related task. From user point of view, a Grid computer behaves like any ordinary workstation running Scientific Linux CERN distribution; however, in this article we discussed different methods of using a custom GNU/Linux environment (or even another operating system) to better support the tools needed by the user. We set up and tested a chroot environment running Debian GNU/Linux *Squeeze* distribution. Installing and running TectoMT framework inside the chrooted environment was straightforward; similarly we experienced no obstacles in installing and using the Slovak language morphology tagger – we therefore do not expect any problems in deploying all kinds of "well behaved" Unix (Linux) based software.

However, in order to truly exploit the Grid potential, we envisage a scheme where the linguistic data (especially text corpora) are stored on the Grid infrastructure as well, and the existing Grid access control infrastructure is extended in order to be provide secure access to the data to third parties interested in accessing the data in such a way that all the limitations and conditions arising from the copyright law and other binding agreements are met.

In this way, we hope that the Grid infrastructure will soon become available to researches in computational linguistics and, by multiplying the computing resources available, will speed up linguistic research tasks and enable us to develop new algorithms, research methods and tools.

References

[1] Installing new Debian systems with debootstrap. `http://www.debian-administration.org/articles/426`. Retrieved 2009-10-05.

[2] Public-Key Infrastructure (X.509) (pkix). `http://www.ietf.org/dyn/wg/charter/pkix-charter.html`. Retrieved 2009-10-10.

[3] Borges, G., David, M., Gomes, J., Fernandez, C., Lopez Cacheiro, J., Rey Mayo, P., Simon Garcia, A., Kant, D., & Sephton, K. (2007). Sun Grid Engine, a new scheduler for EGEE middleware. In *IBERGRID – Iberian Grid Infrastructure Conference*.

[4] Carroll, J., Evans, R., & Klein, E. (2005). Supporting Text Mining for e-Science: the challenges for Grid-enabled Natural Language Processing. In *Proceedings of the UK e-Science All Hands Meeting*.

[5] Ellert, M., Gronager, M., Konstantinov, A., Konya, B., Lindemann, J., Livenson, I., Nielsen, J., Niinimaki, M., Smirnova, O., & Waananen, A. (2007). Advanced Resource Connector middleware for lightweight computational Grids. *Future Generation Computer Systems*, *23*(2), 219–240.

[6] Erjavec, T. & Javoršek, J. J. (2008). Grid Infrastructure Requirements for Supporting Research Activities in Digital Lexicography. In *Mondilex: Lexicographic Tools and Techniques*, (pp. 5–13). IITP RAS.

[7] Garabík, R. & Rusko, M. (2007). Corpus of Spoken Slovak Language. In Levická, J. & Garabík, R. (Eds.), *Computer Treatment of Slavic and East European Languages*, Brno. Tribun.

[8] Laccetti, G. & Schmid, G. (2007). A framework model for grid security. *Future Generation Computer Systems*, *23*(5), 702–713.

[9] Luís, T., Martins de Matos, D., Paulo, S., & Ribeiro, R. D. (2008). Natural Language Engineering on a Computational Grid (NLE-GRID) T5 – Performance Experiments. Technical Report 35 / 2008, INESC-ID, Lisboa.

[10] Martins de Matos, D., Luís, T., & Ribeiro, R. D. (2008). Natural Language Engineering on a Computational Grid (NLE-GRID) T1 – Architectural Model. Technical Report 38 / 2008, INESC-ID, Lisboa.

[11] Martins de Matos, D. & Ribeiro, R. D. (2008). Natural Language Engineering on a Computational Grid (NLE-GRID) T2h – Encapsulation of Reusable Components: Lexicon Repository and Server, January 2008. Technical Report 32 / 2008, INESC-ID, Lisboa.

[12] Martins de Matos, D., Ribeiro, R. D., Paulo, S., Batista, F., Coheur, L., & Pardal, J. P. (2008). Natural Language Engineering on a Computational Grid (NLE-GRID) T2 – Encapsulation of Reusable Components. Technical Report 31 / 2008, INESC-ID, Lisboa.

[13] Marujo, L., Lin, W., & Martins de Matos, D. (2008). Natural Language Engineering on a Computational Grid (NLE-GRID) T3 – Multi-Component Application Builder. Technical Report 33 / 2008, INESC-ID, Lisboa.

[14] Neuroth, H., Kerzel, M., & Gentzsch, W. (Eds.). (2007). *German Grid Initiative D-Grid*. Universitätsverlag Göttingen.

[15] Padala, P., Zhu, X., Wang, Z., Singhal, S., & Shin, K. G. (2007). Performance evaluation of virtualization technologies for server consolidation. Technical report, HP Laboratories.

[16] Pajas, P. & Štěpánek, J. (2006). XML-based representation of multi-layered annotation in the PDT 2.0. In Hinrichs, R. E., Ide, N., Palmer, M., & Pustejovsky, J. (Eds.), *Proceedings of the LREC Workshop on Merging and Layering Linguistic Information (LREC 2006)*, (pp. 40–47)., Genova, Italy.

[17] Santos, N. & Koblitz, B. (2008). Security in distributed metadata catalogues. *Concurrency and Computation: Practice and Experience*, *20*(17), 1995–2007.

[18] Spoustová, D., Hajič, J., Raab, J., & Spousta, M. (2009). Semi-supervised training for the averaged perceptron POS tagger. In *EACL '09: Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics*, (pp. 763–771)., Morristown, NJ, USA. Association for Computational Linguistics.

[19] Tamburini, F. (2004). Building distributed language resources by grid computing. In *Proc. of the 4th International Language Resources and Evaluation Conference*, (pp. 1217–1220).

[20] Žabokrtský, Z., Ptáček, J., & Pajas, P. (2008). TectoMT: Highly modular MT system with tectogrammatics used as transfer layer. In *ACL 2008 WMT: Proceedings of the Third Workshop on Statistical Machine Translation*, (pp. 167–170)., Columbus, OH, USA. Association for Computational Linguistics.