Experiments with Data-Intensive NLP on a Computational Grid

Baden Hughes, Steven Bird
Dept of Computer Science
University of Melbourne
Victoria 3010, AUSTRALIA
{badenh, sb}@cs.mu.oz.au

Haejoong Lee
Linguistic Data Consortium
University of Pennsylvania
Philadelphia PA 19104, USA
haejoong@ldc.upenn.edu

Ewan Klein
School of Informatics
University of Edinburgh
Edinburgh EH8 9LW, UK
ewan@inf.ed.ac.uk

Abstract

Large databases of annotated text and speech are widely used for developing and testing language technologies. However, the size of these corpora and associated language models are outpacing the growth of processing power and network bandwidth available to most researchers. The solution, we believe, is to exploit four characteristics of language technology research: many large corpora are already held at most sites where the research is conducted; most data-intensive processing takes place in the development phase and not at run-time; most processing tasks can be construed as adding layers of annotation to immutable corpora; and many classes of language models can be approximated as the sum of smaller models. We report on a series of experiments with data-intensive language processing on a computational grid. Key features of the approach are its use of a scripting language for easy dissemination of control code to processing nodes, the use of a grid broker to manage the execution of tasks on remote nodes and collate their output, the use a data-decomposition approach by which parametric and parallel processing of individual language processing components occurs on segmented data, and the use of researchers' under-utilized commodity machines.

1 Introduction

For the past decade, the cost of constructing and curating large databases of annotated text and speech has been shared across the language technology community. Rarely does a single researcher or an institution have the resources to create the required corpora. Moreover, standard evaluation techniques require common datasets. This situation has given rise to language data centres (e.g. LDC, ELRA) and to community-wide collaborations in the construction of data (e.g. SENSEVAL). The scale of this data and associated analysis tasks are fast reaching the level where most researchers and institutions lack the resources to *process* the data. In response, the community needs to explore novel ways to collaborate on the processing of large datasets.

This paper reports a series of experiments with data-intensive natural language processing on a computational grid. A computational grid facilitates large-scale analysis using distributed resources. Given the prevalence of large data sources in natural language engineering and the need for raw computational power in the analysis and modelling of such data, the grid computing paradigm provides efficiencies and scalability otherwise unavailable to natural language engineering. A grid broker acts as a task-farming engine, employing the simplest form of distribution in which parallel tasks are fully independent. Thus we do not envisage transferring language engineering applications directly to the grid for real time execution. Rather, we view a computational grid as a promising framework for parallel and parametric processing in language engineering, running similar tasks a large number of times on different combinations of parameter settings. These tasks may generate new datasets subject to further processing: a large set of annotations of the original dataset; or a large set of parameter values and evaluation scores.

This paper is organized as follows. First we explore the motivation for adopting the grid model and discuss prior work on computational grids in NLP. Next, the research infrastructure and approach are described. Following this, the experiments and results are reported. We conclude by discussing our findings and identifying issues for further investigation.

2 Motivation

The growth in scale of NLP tasks and datasets is outpacing the growth in the processing power and network bandwidth available to most researchers. It is not uncommon to find reference to NLP tasks which exceed commonly available computational facilities, such as those described by Salomon et al. (2002), and it is relatively easy to conceive of similar tasks which require CPU time which is measured in days, weeks or months.

While we can always invest more effort in developing and optimizing NLP algorithms, for some tasks this has now reached the point of diminishing returns, and we need to consider alternative ways to achieve computational efficiency. The acquisition and management of enterprise scale computational resources is typically out of scope for many NLP research collaborations, which consist of small groups of people working on isolated tasks using small networks of commodity machines. The ability to deal with large-scale speech and language problems requires a wholesale shift from isolated computation to co-ordinated, distributed computation. To effect this change we must first understand how best to harness computational resources – both processors and data – which are distributed over many sites.

When we consider the nature and practice of language technology research, we observe four distinctive traits: many large corpora are already held at most sites where the research is conducted; most data-intensive processing takes place in the development phase and not at application runtime; most processing tasks can be construed as adding layers of annotation to immutable corpora; and many classes of language model can be approximated as the sum of smaller models. This state of affairs opens the way to a data-centric approach to distributed computation, in which large datasets that reside permanently on the grid nodes are processed in segments, and the results aggregated (cf Skillicorn (2002)). Unlike the processor-centric model, where the data and software are distributed as a single package to computational nodes, the distribution of tasks in a data-centric model is sensitive to the location of the data itself.

3 Previous Work

While the use of distributed computing services is well entrenched within other computational sciences such as physics and biology, the adoption of such an approach as a means to gain computational efficiency in NLP is still at an early stage. A small number of NLP researchers have investigated this area.

Curran (2003) focuses on the requirements for integration of freestanding applications into high performance computing environments, as motivated by the need for optimisation, rather than the general execution framework we focus on here.

Hughes and Bird (2003) designed an XML-based upper middleware layer to interface between distributed computing middleware and NLP software engineering frameworks. Here we build on that work, but move away from the integration issues to explore the computational economy which can be gained.

Tamburini (2004) reported experiments with the collation and derivation of corpora using a network of distributed high performance computing resources. The focus is on distributing the collection of corpora, rather than the analysis of the linguistic data itself.

All of these previous efforts have a common theme: the need for computational economy arising from the analysis of large data sets involving a number of repeated processes. This theme helps to define the environment into which the current paper is offered: namely that distributed computation provides a method by which larger scale language data sets can be efficiently processed.

4 Infrastructure

We have established grid infrastructure at three sites, namely Melbourne, Philadelphia and Edinburgh. Details of the hardware, software and network connectivity are reported in this section.

4.1 Hardware

The machines we use are commodity-level hardware running a variant of the Unix operating system. For performance profiling, we have included a single CPU and an SMP machine:

Hardware Infrastructure

Hosts	CPU	RAM	OS
Melbourne 1-CPU	P4-2Ghz	512Mb	Linux 2.4.20
Melbourne SMP	DualXeon-2Ghz	512Mb	Linux 2.4.8
Melbourne Node 1	Dual-P3-1Ghz	1Gb	Linux 2.4.20
Melbourne Node 2	P4-2Ghz	512Mb	Linux 2.4.21
LDC Node 1	Dual-P3-1.2Ghz	1Gb	FreeBSD 5.2
LDC Node 2	P4-2Ghz	512Mb	Linux 2.6.5
Edinburgh Node	P4-1.5Ghz	256Mb	Linux 2.4.7

4.2 Network

As with any experiment conducted over the Internet, ambient conditions may impact significantly on performance. Accordingly, we measured average network latencies between the three sites (TT=trip-time; RTT=round trip-time).

Network Performance

Link	TT	RTT
Melbourne-Philadelphia	268ms	536ms
Philadelphia-Melbourne	236ms	472ms
Melbourne-Edinburgh	238ms	476ms
Edinburgh-Melbourne	344ms	788ms
Philadelphia-Edinburgh	105ms	210ms
Edinburgh-Philadelphia	112ms	224ms

4.3 Software

While the hardware and operating systems differ between hosts, a common set of software is installed, providing the base on which applications can be executed.

The distributed computing middleware we use is the Globus Toolkit[1] (Foster and Kesselman, 1997). This includes software for security, resource management, communication, fault detection, and portability. The Globus Toolkit facilitates both distributed processing and local clustered processing.

For interaction with the distributed computing infrastructure we use the GridBus Broker (Venugopal et al., 2004). This manages interaction with the middleware layer, by generating, distributing, and monitoring jobs, and collating output. The broker acts as a task-farming engine, adopting the simplest form of distribution in which every parallel task is completely independent.

Our chosen implementation language is Python.[2] In these experiments we use the Natural Language Toolkit[3] (Loper and Bird, 2002; Loper, 2004). NLTK is a suite of Python libraries and programs for symbolic and statistical natural language processing. Since Python is an interpreted language, no compilation is required proior to execution and code can be distributed to nodes at runtime.

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

[2] http://www.python.org/

[3] http://nltk.sourceforge.net/

4.4 Data

The corpora used in these experiments are the Brown Corpus (one million words) and the English Gigaword Corpus (one billion words), as representatives of the two extremes of the large corpus scale.

The Brown Corpus is a collection of edited US English prose, drawn from a wide range of genres ranging from informative to imaginative prose styles (Francis and Kucera, 1964).

Brown Corpus

Source	Files	Mb-gzip	Mb	Words
CA	44	.31	.88	100,554
CB	27	.19	.54	61,604
CC	17	.13	.35	40,704
CD	17	.11	.347	39,399
CE	36	.24	.71	82,345
CF	48	.34	.958	110,299
CG	75	.52	1.5	173,096
CH	30	.19	.64	70,117
CJ	80	.51	1.6	181,888
CK	29	.20	.56	68,488
CL	24	.16	.46	57,169
CM	6	.42	.12	14,470
CN	29	.19	.56	69,342
CP	29	.20	.57	70,022
CR	9	.65	.18	21,695
TOTAL	**500**	**3.48**	**9.96**	**1,161,192**

The second corpus used is the English Gigaword Corpus (Graff, 2002). This corpus is a collection of four international sources of English newswire, from Agence France Press English Service (AFE), Associated Press Worldstream English Service (APW), The New York Times Newswire Service (NYT), and The Xinhua News Agency English Service (XIE).

English Gigaword Corpus

Source	Files	Mb-gzip	Mb	M-words	M-docs
AFE	44	417	1,216	171	.656
APW	91	1,213	3,647	540	1.48
NYT	96	2,104	5,906	914	1.30
XIE	83	320	940	132	.679
TOTAL	**314**	**4,054**	**11,709**	**1,757**	**4.11**

5 Approach

Running an experiment on a computational grid involves five phases: *experiment pre-processing*, when data is set up for the entire experiment; *execution pre-processing*, when data is prepared for each particular task; *execution*, when the task is executed for a given set of parameter values; *execution post-processing*, when data from a particular task is reduced; and finally *experiment post-processing*, when results are processed. Each of these five phases has the potential to be optimised for a given distributed computation.

A plan file specifies the jobs to be distributed over the network and executed. Plans contain a description of parameters together with instructions on how to execute user programs and perform file transfer. Plan files consist of two main groups of commands: parameter statements and tasks.

5.1 Parameter Statements

There are two kinds of parameter: *static parameters* are variables which are well defined either as a range of values, a single value, or one among a set of values; *dynamic parameters* have either undefined or unbounded boundary conditions which are established at runtime. Common parameter types supported by the dominant model for grid brokering services include `single value`, `range` (values between a lower and upper bound), `select anyof` (values in a value list), `select oneof` (values in a value list), `random` (generated between a lower and an upper bound), and `compute` (determined by some computation on the values of other parameters) (TurboLinux Inc, 2000).

5.2 Task Decomposition

Task decomposition takes three different forms: trivial (or parametric) decomposition, functional decomposition and data decomposition.

Trivial (or parametric) parallelism is the simplest task decomposition type. It does not involve parallelising code. Instead, sequential code is run on several nodes in parallel, with each node having its own set of parameters. Different instances of the sequential code must be independent; one version job cannot depend on the results of another job which is run at the same time, and there is no communication between nodes.

Functional decomposition divides an application into a number of different tasks, each performed on different nodes. Each node is given a unique task, and a pipeline architecture is adopted. Hence the amount of parallelism is based on the number of segmentation possibilities. Functional decomposition lends itself naturally to workflow-based implementations such as those proposed by Hoheisel (2004) and Ludaescher (2004).

Data decomposition, the approach we adopt here, divides the data between nodes. The analysis task is then performed at each node in parallel, before the results are collated. There is some overhead in distributing and balancing the workload across the nodes.

5.3 Types of Execution Plan

In the decomposition phase, a distributed computation environment allows a great deal of flexibility in the type of execution plan adopted. We can construe the possible execution plans (the complexity and number of tasks given to the broker) as being one of three types — derived by the application itself; derived by a combination of the application and the broker's subsequent derivation of parameters from a plan; and derived by the broker alone. These three different plan types correspond to different levels of iteration: at the root node it is the number of times the broker itself is instantiated; at the broker level, it is number of jobs executed; while at the node level, it is the number of nested loops within the task itself.

6 Experiments

In this section we discuss three experiments: word frequency counting, indexing, and training a part-of-speech tagger. We describe each task briefly, then report performance metrics for executing them on a grid.[4] In order to establish a baseline, the experiments are also executed on a single CPU system and an SMP system.

We do not claim exclusive use of the computational nodes during the execution of

[4] *The grid consisted of just two nodes; we will report the result of a larger experiment for the final version of the paper.*

these experiments, and ambient system conditions impact overall performance. The subsequent need for an accurate instrumentation framework is discussed later.

In all cases, the words per second figure is derived from the total size of corpora in words divided by elapsed system plus user time as reported by `time()`.

6.1 Task 1: Word Frequency Counting

The word frequency task builds an associative array of all the word tokens in the corpora, mapping from words to a frequency count. The execution of this application was across two computational nodes.

Word Frequency Counting

	Words/Sec 1-CPU	Words/Sec SMP	Words/Sec Distrib
Brown			
CA	6,703	11,172	7,182
CB	6,160	8,800	5,133
CC	6,784	8,140	4,070
CD	6,566	7,879	3,939
CE	6,862	10,293	3,939
CF	6,893	10,027	7,353
CG	6,923	9,616	7,868
CH	7,011	8,764	5,393
CJ	6,995	9,573	8,267
CK	6,848	9,784	5,707
CL	6,352	9,528	5,197
CM	7,235	7,235	1,447
CN	6,934	8,667	5,334
CP	6,365	10,003	5,835
CR	5,423	7,231	1,972
Average	6,670	9,114	5,402
Gigaword			
AFE	1,361	1,793	2,494
APW	1,320	1,741	2,300
NYT	1,342	1,773	2,215
XIE	1,328	1,874	2,263
Average	1,337	1,795	2,318

For the Brown corpus, we observe a performance gain of 27% in moving from a single CPU to an SMP system, but a performance *degradation* of 20% in moving to a two node grid. For the Gigaword corpus, we observe a performance gain of 26% in moving from a single CPU to an SMP system, and a gain of 38% in moving to a two node computational grid.

6.2 Task 2: Indexing

The process of constructing an inverted index is broadly representative of a large family of NLP tasks. In this experiment, we build an inverted index of all noun phrases occurring in the headline of a newswire article, against the document number in which they occur.

Indexing

	Words/Sec 1-CPU	Words/Sec SMP	Words/Sec Distrib
Gigaword			
AFE	1,279	2,066	3,160
APW	1,770	1,836	2,785
NYT	607	828	1,137
XIE	1,244	2,343	3,876
Average	1,252	1,561	2,739

We observe a performance gain of 24% in words per second over the single CPU baseline in moving to an SMP machine, and a gain of 55% in moving to a two node computational grid.

6.3 Task 3: Training a Part-of-speech Tagger

In this experiment, we train an n-gram part-of-speech tagger on a segment of the Brown corpus. Each node uses a different segment and ends up with a slightly different tagger.

Training a Part of Speech Tagger

	Words/Sec 1-CPU	Words/Sec SMP	Words/Sec Distrib
Brown			
CA	1,596	1,933	2,513
CB	1,540	1,811	2,369
CC	1,565	2,142	2,907
CD	1,575	2,073	2,814
CE	1,583	1,871	2,287
CF	1,598	2,042	2,690
CG	1,588	2,136	2,663
CH	1,630	1,947	2,596
CJ	1,765	2,362	2,887
CK	1,630	1,902	2,853
CL	1,588	2,041	2,722
CM	1,607	2,411	1,033
CN	1,612	1,981	2,667
CP	1,628	2,000	2,693
CR	1,549	2,711	1,446
Average	1,603	2,090	2,476

There is performance gain of 24% in words per second over the single CPU baseline in moving to an SMP machine, and a gain of 36% in moving to a grid.

7 Discussion

The experimental results demonstrated that NLP tasks can be subjected to data-decomposition and executed efficiently on a grid. However, we observed a performance degradation when

the task was too small (i.e. for word frequency counting on the Brown corpus), and so we conclude that tasks must be sufficiently large to warrant execution in grid mode.

The performance comparison with SMP machines is striking: multiple CPUs are more effective when they have their own disk and memory. We attribute this to the fact that the tasks were I/O-bound, and that large result sets were not aggregated.

It is also of interest that the words per second measurement is the size of the corpora divided by the time taken from instantiation at the central node (in other words, transit time between nodes is included in these figures). In the event that data transfer was significant (eg for language modelling task), the relative performance of the distributed task against the SMP executed task is impacted by the large resultant data transfer.

While it may appear that the relative performance of the English Gigaword word frequency counting experiment is significantly lower than the Brown corpus for the same task, it is important to note that a key difference between the corpora is their disk storage state - Gigaword is compressed, and hence the words per second measurement is lower than expected owing to the need to decompress the corpus prior to processing. Assuming that disk I/O could be accurately measured, we would expect to find that in fact the words per second metric for processing Gigaword would be increased, possibly substantially.

It is well accepted that the number of CPUs and their speed are not the only factor in assessing relative performance. Many operations are not bounded by CPU but by disk I/O or memory, and we were unable to include these factors in the study without a comprehensive systems instrumentation suite as discussed in the following section.

8 Future Work

While we regard the experiments and methodology reported here as exploratory, there are a number of natural extensions to this work which we envisage implementing to facilitate subsequent investigations. These are an NLTK API for the execution of NLP tasks on computational grids; a framework for modeling and evaluating computational efficiency for NLP tasks in a distributed environment; integration of workflow-based aggregation models; integration of a systems instrumentation and profiling suite; and larger scale experiments.

8.1 API for HPC Execution of NLP Tasks

For ease of execution, a common framework such as NLTK would provide methods for broker invocation. This module would define a Grid class which would store the location of the config directory (containing a list of hosts, certificates, dependencies, and identifying available brokers).

The execution plan would be set up by the `plan()` member function, including information about the command to be run at each client, the parameters

The plan member function would define the NLTK command and its command line arguments, and whether these were to be passed in from the calling program (static) or generated by the broker (dynamic).

```
>>> grid = Grid(config_dir)
>>> for group in brown.groups():
...   for order in range(4): # order of n-gram model
...     outfile = "brown.%s.%s" % (group, order)
...     plan = grid.plan(
...       code="brown_tagger.py",
...       command="python ./brown_tagger.py",
...       args="-g $group -o $order -s $smoothing",
...       static=[group, order],
...       dynamic=[(smoothing,"[0.1,1.0] step 0.1")]
...       outfile=outfile)
...     plan.exec()
...     process(plan.outfile())
```

A feature of this approach is that we can easily change the granularity of the grid processing. E.g., by putting `order` inside the plan. (Compare the boldface lines in both code fragments.)

```
>>> grid = Grid(config_dir)
>>> for group in brown.groups():
...   outfile = "brown.%s.%s" % (group, order)
...   plan = grid.plan(
...     code="brown_tagger.py"
...     command="python ./brown_tagger.py"
...     args="-g $group -o $order -s $smoothing",
...     static=[group, order],
...     dynamic=[(smoothing,"[0.1,1.0] step 0.1"),
                 (order,"[0,3] step 1")]
...    outfile=outfile
...  plan.exec()
...  process(plan.outfile())
```

We could also move the iteration over `order` down inside `brown_tagger.py`, so that it is handled locally on the node. Conveniently, this three-way choice of root vs broker vs node are

controlled by the programmer writing code at a single location in a single language.

8.2 Economic Models of Computation and Optimal Task Decomposition

The requirement to decompose a processing task into a set of discrete jobs poses a number of questions about the optimal set of jobs which are executed in a distributed computation environment. Since each task is decomposed further into a series of smaller jobs, it is in fact the performance on a per job basis which requires profiling rather than the task itself. Each job therefore must be assessed as to the resource impact of instantiation, execution, and conclusion.

We can conceive of this task as the generation of a cost curve on which predictions can be based for future executions. The incline of the curve may vary between types of tasks. The objective is to generate a costing on a per item basis using an n+1 approach to determine the cost of eg tokenising an additional word.

One technique which could be investigated is to determine computational economy of particular types of tasks. We can envisage profiling each host (using a tool such as linpack[5] to determine machine independent CPU performance); profiling network infrastructure (using a tool such as NWS[6]) to determine ambient bandwidth conditions); and gathering metrics for task execution (using time or similar). Using these metrics, CPU and bandwidth act as reference points for determining the computational economy of executing any particular task. Retention of the decompostions of different types of tasks and their performance is a requirement in learning about more efficient processing in this mode.

8.3 Workflow

The data decomposition approach that we have employed will generate slightly different models at each node. We would like to aggregate these and redistribute them back to the nodes for evaluation on test data, before aggregating the scores. Aggregating scores requires minimal bandwidth, but aggregating the models could involve the transfer of a substantial quantity of data. Modest savings may be found by assuming that frequent events have been adequately sampled on each node and that it is only necessary to exchange data pertaining to rare events. Further research is needed on efficient ways to share model fragments between nodes.

8.4 Integrated Instrumentation

In the absence of dedicated systems for NLP tasks, there are possible performance impacts owing to shared system activity. In order to discover a robust set of metrics for any particular task, a systems instrumentation framework is required so that experimental results can be evaluated in light of competing tasks which impact system performance through aspects such as disk I/O, swapping and page file access, memory management, CPU partitioning and allocation. Such a framework would allow an objective evaluation of the performance of a given NLP task against the aggregate resource requirements of all competing processes on the same computational node.

8.5 Scale-Up

We would like to run further experiments on a much larger grid. In the ideal case we would like to carefully scale the size and configuration of the grid so that we can build predictive models of NLP task execution across a given number of nodes. This in turn will assist with developing optimised models for task decomposition and an overall model of computational economy for large scale NLP tasks.

We plan to make our software available from our project website at `nle-grid.sourceforge.net`, along with detailed instructions, and invite other researchers to participate in the experiments.

9 Conclusion

This paper has described a data-centric model for distributing language processing tasks on a computational grid, and demonstrated it for a range of tasks involving large corpora. Processing in distributed mode delivers the near-linear speed up that we expected, demonstrating that grid infrastructure has significant potential for coordinating the distributed processing of large corpora.

[5] http://www.netlib.org/linpack/

[6] http://nws.cs.ucsb.edu/

Acknowledgements

The research reported in this paper is supported by the National Science Foundation Grant Number 0317826 (Querying Linguistic Databases), Victorian Partnership for Advanced Computing Expertise Grant EPPNME092.2003, and the University of Melbourne Advanced Research Computing Centre.

References

James R. Curran. 2003. Blueprint for a high performance NLP infrastructure. In *Proceedings of the Workshop on The Software Engineering and Architecture of Language Technology Systems*, pages 39–44. Association for Computational Linguistics.

Ian Foster and Carl Kesselman. 1997. Globus: A metacomputing infrastructure toolkit. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(2):115–128. http://citeseer.ist.psu.edu/foster96globu.html.

W. N. Francis and H. Kucera. 1964. *A Standard Corpus of Present-Day Edited American English, for use with Digital Computers*. Department of Linguistics, Brown University. Providence, Rhode Island, USA. `http://www.hit.uib.no/icame/brown/bcm.html`.

David Graff, editor. 2002. *English Gigaword*. Linguistic Data Consortium. `http://www.ldc.upenn.edu/Catalog/CatalogEntry.jsp?catalogId=LDC2003T05`.

Andreas Hoheisel. 2004. User tools and languages for graph-based grid workflows. In *Proceedings of the Workflow in Grid Systems Workshop, Global Grid Forum 10*. `http://www.extreme.indiana.edu/groc/ggf10-ww/index.html`.

Baden Hughes and Steven Bird. 2003. Grid-enabling natural language engineering by stealth. In *Proceedings of the Workshop on The Software Engineering and Architecture of Language Technology Systems*, pages 31–38. Association for Computational Linguistics. East Stroudsburg, USA.

Edward Loper and Steven Bird. 2002. NLTK: The natural language toolkit. In *Proceedings of the ACL Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics*, pages 62–69. Somerset, NJ: Association for Computational Linguistics. `http://arXiv.org/abs/cs/0205028`.

Edward Loper. 2004. NLTK: Building a pedagogical toolkit in Python. In *Proceedings of PyCon Washington DC 2004*. `http://python.tpnet.pl/pycon/dc2004/papers/35/nltk/nltk.pdf`.

Bertram Ludaescher. 2004. Kepler: Towards a grid-enabled system for scientific workflows. In *Proceedings of the Workflow in Grid Systems Workshop, Global Grid Forum 10*. `http://www.extreme.indiana.edu/groc/ggf10-ww/index.html`.

Jesper Salomon, Simon King, and Miles Osborne. 2002. Framewise phone classification using support vector machines. In *Proceedings of the 7th International Conference on Spoken Language Processing*. `http://www.icslp2002.org`.

David Skillicorn. 2002. Motivating computational grids. In *Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid*. IEEE Computer Society Press.

Fabio Tamburini. 2004. Building distributed language resources by grid computing. In *Proceedings of the 4th International Language Resources and Evaluation Conference*. European Language Resources Association: Paris.

TurboLinux Inc. 2000. Enfuzion 6.0 user guide. `http://www.csse.monash.edu.au/cluster/enFuzion/plans.htm`.

Srikumar Venugopal, Rajkumar Buyya, and Lyle Winton. 2004. A grid service broker for scheduling distributed data-oriented applications on global grids. `http://www.gridbus.org/papers/gridbusbroker.pdf`.