Experiences with Parallelisation of an Existing NLP Pipeline: Tagging Hansard

Stephen Wattam[1], Paul Rayson[1], Marc Alexander[2] & Jean Anderson[2]

School of Computing and Communications[1]
Lancaster University
{s.wattam, p.rayson}@lancaster.ac.uk

English Language[2]
University of Glasgow
{marc.alexander, jean.anderson}@glasgow.ac.uk

Abstract

This poster describes experiences processing the two-billion-word Hansard corpus using a fairly standard NLP pipeline on a high performance cluster. Herein we report how we were able to parallelise and apply a "traditional" single-threaded batch-oriented application to a platform that differs greatly from that for which it was originally designed. We start by discussing the tagging toolchain, its specific requirements and properties, and its performance characteristics. This is contrasted with a description of the cluster on which it was to run, and specific limitations are discussed such as the overhead of using SAN-based storage. We then go on to discuss the nature of the Hansard corpus, and describe which properties of this corpus in particular prove challenging for use on the system architecture we used. The solution for tagging the corpus is then described, along with performance comparisons against a naïve run on commodity hardware. We discuss the gains and benefits of using high-performance machinery rather than relatively cheap commodity hardware. Our poster provides a valuable scenario for large scale NLP pipelines and lessons learnt from the experience.

1. Introduction

Requirements for NLP processing systems now typically include the need for extremely large scale processing of data derived either from vast online sources or increasing quantities of digitised archive material. Whether the scenario is to answer a particular set of research questions or for commercial text analytics, sources such as Twitter provide significantly more data than can be processed and ingested in anything like reasonable time. Parallel computation is typically used to address this bottleneck.

This poster presents the lessons learnt from our work which emerged from two serendipitous events: the need to support digital humanists in their analysis of the full Hansard data set, and a need for a real case study for Lancaster University's High Performance Cluster using textual rather than numerical data. Large infrastructure activities such as CLARIN (Váradi et al., 2008) and DARIAH (Constantopoulos et al., 2008) are providing distributed archives for language resources but NLP research teams still face local requirements during experimentation to process and reprocess very large resources through complex pipelines. Some toolkits, e.g. GATE, can now run in the cloud (Tablan et al., 2013) to support such activities. However, many universities have existing high performance clusters that may be under–exploited by language researchers.

The work described here is part of a larger project, Parliamentary Discourse[1], to include the 200-year corpus of Hansard texts in Enroller[2], an infrastructure to hold and cross-search large data sets. The wider aim was to enrich the Hansard corpus with linguistic annotations and named-entity markup, cross–link it to the resources within the Enroller portal and make the resource available for linguistic and historical research.

2. Toolchain

The toolchain being used to tag the corpus was a combination of tools comprising the tag wizard of the Wmatrix system[3]. This consists of a number of tagging and analysis tools, each communicating using intermediate files and managed using a series of shell scripts. The tag wizard shares a lot of commonalities with standard NLP processing pipelines as it consists of two annotation systems—CLAWS (Garside and Smith, 1997) and USAS (Rayson et al., 2004)—plus a frequency profiling and keyness comparison step.

3. High-Performance System

The system used for processing was the Lancaster University High-End Computing cluster (HEC)[4]. This consists of a number of compute nodes running Scientific Linux[5], connected using the Oracle Grid Engine[6]. In total the system comprises over 2,200 CPU cores, 11TB of memory, 32TB of high performance file storage, and a further 1PB of medium performance storage.

As shown in Figure 1, the system is reliant on network-attached shared storage in the form of a Panasas Activescale

[1] http://www.glasgow.ac.uk/hansard

[2] http://www.glasgow.ac.uk/enroller Enroller contains five corpora, one dictionary and one thesaurus. The data sets can be simultaneously searched and the 780,000-entry Historical Thesaurus of English enables searching of textual resources by concepts and across time.

[3] http://ucrel.lancs.ac.uk/wmatrix/

[4] The authors would like to thank Mike Pacey, HPC Manager, Information Systems Services, Lancaster University for his assistance and patience.

[5] https://www.scientificlinux.org/

[6] http://www.oracle.com/us/products/tools/oracle-grid-engine-075549.html

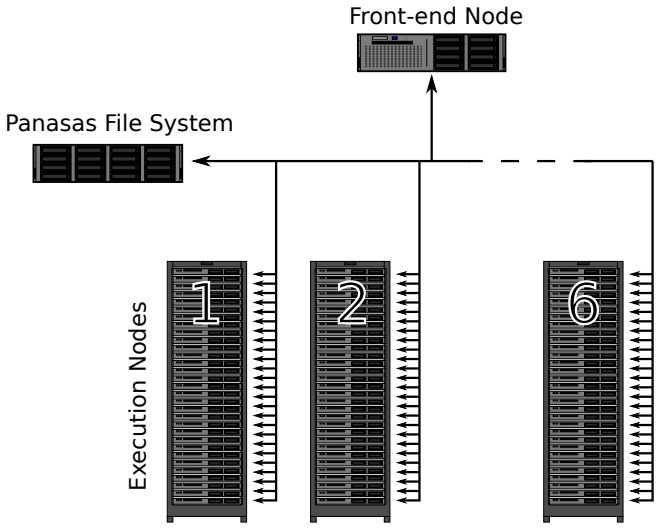


Figure 1: HEC architecture

Series 8 storage cluster[7](Nagle et al., 2004). Further storage is available upon the compute nodes themselves. There are 262 compute nodes in total, each with two four- or six-core Intel E5520 CPUs each. Most of these compute nodes have 24GB of RAM available locally (those that do not have 96GB). The Oracle Grid Engine scheduling framework works in a batch processing manner, with active jobs competing for access to compute nodes. This scheduler can be used to queue up job arrays in large batches, which will then be distributed to each compute node individually as they become free. This was the primary mechanism used to distribute jobs across nodes.

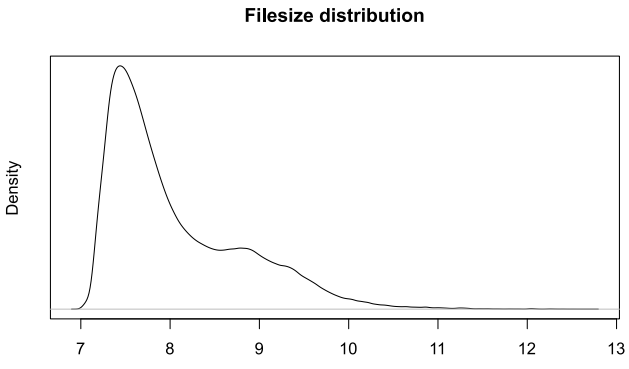
4. The Hansard Corpus

Hansard is the official published report of both oral and written UK Parliamentary proceedings. It is an edited verbatim report of speeeches in both the House of Commons and the House of Lords. Members' words are recorded by Hansard reporters and then edited to remove repetitions and obvious mistakes but without changing the meaning. Reports of the latest proceedings are published online and updated during the day and can be read online back to 1988[8]. These records are of importance for scholars of the development of the English language, of British politics and history, and are also used by community-focused organisations. Historic records back to 1803 have been digitized in XML format and are publically available on the Historic Hansard web site created by the Commons and Lords Libraries and Millbank Systems. There one can search by word or phrase then filter results by speaker or House (Lords or Commons). This is freely available online under an Office of Public Sector Information licence. The Hansard corpus is one of the biggest humanities data sets in the UK but was limited in its use by being tagged only by speaker and date. To provide greater utility we wanted the speeches to be searchable by speaker, date, parts of speech (POS) and most interestingly, by topic. This would enhance this significant public material and expose it to new audiences within Higher Education, including linguists, discourse analysts, historians, digital humanists, and cultural scholars. We downloaded the XML corpus from the Historic Hansard site, ran routines to standardise the XML coding in order to prepare it for the NLP toolchain. As well as POS tagging, we applied the USAS semantic tagger. Once these annotation steps were complete, we needed to produce word, POS and semantic frequency lists for each speech as per the standard Wmatrix tag wizard pipeline. Finally, in order to expose the key topics of each speech we compared each semantic frequency list to a standard reference corpus, the British National Corpus spoken sampler (one million words) using the log-likelihood statistic to find a set of key semantic tags.

The full 200-year collection is 2,271,985,142 words and 32.7GiB of data, including mark–up. The corpus is split into 7,545,103 XML files, each representing a speech made in either house of the UK Parliament. These files are organised into a hierarchical directory structure by house and date, comprising 48,482 folders in total. A sample of this structure is shown in Figure 2.

```
Hansard
+-- Commons
| +-- commons 1803-1820
| | \-- commons
| |     +-- 1803
| |     | +-- dec
| |     | | +-- 01
| |     | | +-- 02
| |     | | +-- 03
| |     | | +-- 05
| |     | | +-- ...
```

Figure 2: Sample directory layout



Percentile	5	25	50	75	95
Words	40	83	147	308	1400

Figure 3: Distribution of log filesizes for all corpus data, and word counts (without markup) for each file.

[7] http://www.panasas.com/products/activestor-14

[8] For more information on Hansard and its history see http://parldisc.jiscinvolve.org/wp/ and http://www.parliament.uk/about/how/publications/hansard/

Figure 3 shows that filesizes are generally small, exhibiting a largely Zipfian distribution beyond the modal size of 1.8KB. The median size is just 2.4KB, and the 95th percentile is 1.4MB.

5. Method

The Wmatrix tag wizard toolchain was originally designed and developed to run on commodity computing hardware in a batch processing fashion. The toolchain thus respects a number of common limits that apply to desktop computers (e.g. low memory limits) and attempts to exploit other resources that are available in abundance (fast sequential file I/O). Many of these properties do not transfer to larger clusters, and a number of changes had to be made in order to align the processing stages with the resources available on the HEC system.

File Access The Wmatrix toolchain makes extensive use of small output and intermediate files during operation. This incurs a significant performance overhead when used with filesystems that are typical in high-end distributed systems, which often use large block sizes, carry metadata for versioning and redundancy, or are accessible only over network links. For this particular process, each input file causes creation of eleven intermediate and output files. When parallelised, the number of concurrent I/O operations performed, along with the organisation of the corpus (stored as many small files), soon overwhelmed the bandwidth limitations of the shared filesystem.

The solution to this effected changes to the whole toolchain—first, the corpus was restructured to move each day's files into a single `tar` archive. This could then be copied with a single operation from the shared drive. This input file was then extracted into a temporary directory on the local disk of the compute node. Following execution, output was again archived using `tar` and copied back to the shared drive. This process necessitated a further post–processing stage to extract and validate the output, which was performed after downloading data from the HEC.

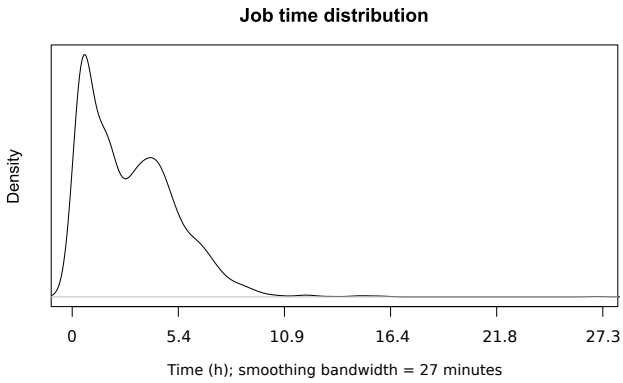
Memory Size Memory provisions on systems have grown considerably since the toolchain was designed, and as such it fails to exploit even modern desktop systems. Without significant re-engineering we were unable to exploit the relatively high 24GB memory space on each of the compute nodes, and this remains the most under-utilised resource.

Indexing and Co-ordination As work was completed away from the shared disk, a system was necessary to co-ordinate compute nodes' access to data. This was accomplished by using a flatfile, centrally stored and accessed once per batch, that contained a list of input archives and their corresponding output directories. The numeric 'job ID' environment variable provided by the scheduler was used to compute an offset, which each job used to split the input file by line, taking a number of input files for a single job.

The number of input archives tagged per job was chosen to balance scheduling overheads with the need to 'smooth out' file access and ensure that any failures did not affect large areas of the corpus. Ultimately, each job was run using 20 input archives, chosen to take 30 minutes using average input files.

Scheduling Overhead The overhead incurred by a shared scheduling system is significant compared to a typical process creation task (exacerbated by the need to copy and extract archives). In practice, running 2230 jobs with a predicted execution time of 30 minutes each, this did not prove to be a limiting factor.

6. Performance



Percentile	5	25	50	75	95
Duration	20m	53m	2.5h	4.5h	7.3h

Figure 4: Distribution of job execution times on the HEC

All jobs were complete in 3 days, meaning that the system tagged at a rate of 31.5 million words per hour. Because of the heterogeneity in task length, this rate was not constant, decaying towards the end (the final 231) as the queued tasks ran out and were not replaced. Had we used larger batches, this effect would have proven more severe as the variance in job length was liable to increase. Were the corpus particularly large (in the range of hundreds of billions of words), it would be prudent to model and control for this effect ahead of time. Jobs used a maximum of between 80 and 120MB of memory—our greatest underutilised resource.

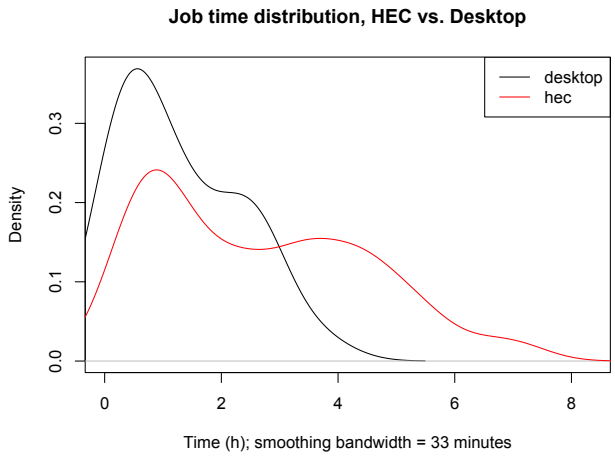
6.1. Commodity Hardware

In many cases, the alternative to deployment on a HEC cluster will be use of one or many commodity desktop machines. In order to compare the performance of the toolchain, a random sample of 50 jobs was run through the toolchain using the scheduling scripts described above.

The hardware used was a desktop machine with a single Intel i5 processor, 15GB of memory, and two 7200rpm mechanical hard disks in a RAID-0 configuration. The system was running Arch Linux[9], and, as the system is the first author's office machine, work continued on other projects during the benchmarks. We believe this constitutes a system equivalent to many found across offices today.

As can be seen in Figure 5, the desktop system runs individual jobs significantly faster. Fitting a linear model indicates that the desktop is able to run jobs approximately 2.1 times faster than a single HEC core. The same model indicates

[9] `http://archlinux.org`



Percentile	5	25	50	75	95
HEC	22m	57m	2.4h	4.1h	5.6h
Desktop	9m	22m	1h	2.3h	2.8h

Figure 5: HEC and Desktop job execution times for a sample of 50 jobs

a 100 second job startup overhead on the HEC (including copying of files). Had we continued to tag the corpus in this manner, it would have taken 98 days on the desktop system, and thus required at least 33 equivalent machines to achieve the HEC's performance. It seems unlikely that manually splitting the data across that many systems would save time compared to the development overheads incurred for the HEC tagger.

7. Discussion and conclusion

Two weeks was spent developing and testing the HEC deployment of the toolchain. Of this, a significant portion was spent adapting the toolchain to run on the scheduler without restriction from the shared filesystem. The problem is embarrassingly parallel, and the design of many HEC facilities is well suited to exploit that. It is certainly the case that, even if we had manually split the data and run it on (faster) commodity hardware, we would still have incurred significant overhead in doing so. Assuming development in that case were simpler and took just a week, we would still require ten desktop systems in order to achieve the same overall completion time.

It is worth noting that we did not fully exploit the parallel capacity of the cluster, and modifications such as parallelising upon each compute node, or pipelining, would yield vastly improved execution times[10]. Though these incur further development effort, they may be worthwhile for many projects, or be used to extend the lifetime of existing toolchains.

In contrast to manually deploying jobs, where scheduling is automated and fast there is a benefit to having smaller jobs, as they may compete more efficiently for resources (though this must be balanced with the overheads of any scheduling itself). For problems where time constraints are less severe, the toolchain used is particularly specialised, or the data is less inherently parallel, there may be benefits to the simpler approach. For our purposes, where the task was extremely parallel, the toolchain easily ported, and the timescale short, the greater concurrency offered by a high performance cluster proved invaluable.

In future work, we intend to test the HEC facilities with other corpora with different filesize distribution characteristics. We will also experiment with the MapReduce programming model through implementations such as Apache Hadoop. A key future requirement is to expose such parallel processing facilities through a programming-lite interface so that the benefits can be exploited by corpus linguists and digital humanists alike.

[10]Running once per core would have reduced execution times 8-fold to just 9 hours.

8. References

Panos Constantopoulos, Costis Dallas, Petern Doorn, Dimitris Gavrilis, Andreas Gros, and Georgios Stylianou. 2008. Preparing dariah. In *Proceedings of the International Conference on Virtual Systems and MultiMedia (VSMM08)*.

Roger Garside and Nicholas Smith. 1997. A hybrid grammatical tagger: Claws4. *Corpus annotation: Linguistic information from computer text corpora*, pages 102–121.

David Nagle, Denis Serenyi, and Abbie Matthews. 2004. The panasas activescale storage cluster: Delivering scalable high bandwidth storage. In *Proceedings of the 2004 ACM/IEEE conference on Supercomputing*, SC '04, pages 53–, Washington, DC, USA. IEEE Computer Society.

Paul Rayson, Dawn Archer, Scott Piao, and Tony McEnery. 2004. The ucrel semantic analysis system.

Valentin Tablan, Ian Roberts, Hamish Cunningham, and Kalina Bontcheva. 2013. Gatecloud. net: a platform for large-scale, open-source text processing on the cloud. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 371(1983).

Tamás Váradi, Steven Krauwer, Peter Wittenburg, Martin Wynne, and Kimmo Koskenniemi. 2008. Clarin: Common language resources and technology infrastructure. In *LREC*.