ISSN 1346-5597

NII Technical Report Jan 2003

Evaluation of Enigma: an Open Mosix Cluster for Text Mining

Nigel Collier

NII-2003-001E
Jan 2003

Evaluation of Enigma: an Open Mosix Cluster for Text Mining

Nigel Collier
National Institute of Informatics
2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo, 101-8430 Japan
E-mail: collier@nii.ac.jp

Abstract

In this report we present the results of evaluating the speed of text mining experiments using the OpenMosix cluster software. Our primary purpose is to develop a computational environment where the time taken to perform compute intensive experiments over a large range of parameter settings is minimized. This will lead to clearer analysis of algorithms and ultimately to a better choice of models based on empirical results. Results are shown that characterize each of the cluster components and also show the speedup in performance over compute nodes.

1 Introduction

In this report we present the results of our experiments in adapting text mining algorithms to cluster computing using OpenMosix. Our primary purpose is to develop a computational environment where the time taken to perform compute intensive experiments over a large range of parameter settings is minimized. This will lead to clearer analysis of algorithms and ultimately to a better choice of models based on empirical results.

OpenMosix (OM) is based on Mosix [2] [3] [7] and provides a kernel-level solution for preemptive and transparent process (re-)migration among cluster nodes, i.e. without the knowledge or intervention of the user. Response is dynamic based on the current information a cluster node has about itself and a random list of other nodes. Overall, the user is presented with a collection of cooperating computers that look like a single system image where cluster resources (CPU cycles, memory and bandwidth) are shared among the nodes.

There are several other noteworthy features of OM clusters that make it suitable for our needs:

- good load balancing - i.e. an even distribution of computation among the cluster resources which maximizes usage of cluster-wide RAM and avoids the expense of disk paging [1]. This is achieved using a scheduling algorithm called the Commodity Market Model from economics research;
- a cluster-wide file system in which direct file system access (DFSA) is possible. DFSA allows for some I/O to be performed locally on the remote node;
- robustness - since control is decentralized, failure in one node does not bring the whole cluster down, it just reduces the resources available to the whole cluster;
- no requirement for explicit parallelism in the software code;
- support for process monitoring and management;

In particular OM seems well suited to our need to run mixed experiments in which some models may make very heavy use of I/O such as hidden Markov Models [12] using maximum likelihood estimates calculated directly from word list counts, to support vector machines [15]where periods of CPU intensive activity are much longer. OM also provides a balanced multi-user environment for our research team where user has a different level of expertise in parallelizing software.

There have been few empirical studies of OM since earlier tests were done on the parent Mosix system. Since OM developed as a separate project from MOSIX in 2001 there have been several performance enhancements such as an improved load balancing, reduced kernel latencies, node auto-discovery and a more sophisticated algorithm for process migration. We therefore feel that our investigation has two important aspects: (1) to show the application of cluster computing using OM for text mining, and (2) to provide empirical performance results for OM in general.

2 Task Description

The task we want to accomplish is the extraction of information from inside a free-text collection based on a small

In this report we demonstrate that in certain [CNS-derived cells]$_{source.ct}$ [Tat]$_{protein}$ is capable of activating [HIV-1]$_{source.vi}$ through a [TAR]$_{RNA}$-independent pathway. A [Tat-responsive element]$_{DNA}$ is found upstream within the viral promoter that in [glial- derived cell lines]$_{source.cl}$ allows transactivation in the absence of [TAR]$_{RNA}$. Deletion mapping and hybrid promoter constructs demonstrate that the newly identified [Tat-responsive element]$_{DNA}$ corresponds to a sequence within the viral [long terminal repeat]$_{DNA}$ ([LTR]$_{DNA}$) previously identified as the [HIV-1 enhancer]$_{DNA}$, or [NF-kappa B domain]$_{DNA}$. DNA band-shift analysis reveals [NF-kappa B]$_{protein}$ binding activity in [glial cells]$_{source.ct}$ that differs from that present in [T lymphoid cells]$_{source.ct}$. Further, we observe that [TAR]$_{RNA}$-deleted mutants of [HIV-1]$_{source.vi}$ demonstrate normal [late gene]$_{DNA}$ expression in [glial cells]$_{source.ct}$ as evidenced by syncytia formation and production of [viral p24 antigen]$_{protein}$.

Figure 1. Example of part of a MEDLINE abstract marked up for NE expressions.

to medium collection of annotated examples. For example, company take-over announcements for stock analysts, missile launches for military planners, conference announcements for researchers. Text mining typically involves modelling the proto-typical events as a frame of slots to be filled and finding the individual facts in a text that fill the slots. At a lower level we need to identify names of things that fill the values of slots such as people, places, organizations as well as temporal and monetary values. This low level task is often referred to as the *named entity task* (NE) [10] and many of the best performing approaches use supervised machine learning algorithms on large annotated text collections. In our work we are actively investigating text mining in the molecular biology domain and the proto-typical events are interactions between genetic products such as proteins, DNA and RNA. The text collections that we investigate are MEDLINE abstracts [9] and EMBO Journal articles. An example of part of a MEDLINE abstract annotated with biological named entities is given in Figure 1.

The basic purpose of our cluster is to enable us to accurately evaluate text mining algorithms in a reasonable amount of time using large data sets. The basic procedure can be summarized in the following algorithm:

1 Data formatting

2 Experiment

- **2.1** create data partition
- **2.2** train model
- **2.3** test model

3 Group data

4 Evaluation

In the first stage we decide on the set of linguistic features that we want to investigate and then add them to the data set. Example features are lexical (orthographic, part of speech, lemma) and grammatical (head noun, predicate-argument position) within the window of context that we choose to explore.

In data partitioning we try to maximize accuracy by the traditional process of cross-validation, i.e. dividing the available data into P equal partitions ($P > 1$) and then training the model on $P - 1$ and evaluating it on the remaining partition. This is repeated P times so that each partition is left out in turn. Finally the results of all folds are collected together and evaluated.

The basic unit in the data collection is the sentence, and in the extreme case where P is equal to the number of sentences N this evaluation method becomes the *leaving-one-out* method [11]. We perform our partitioning deterministically, i.e. with no random picking of sentences. The size of the training and testing sets in relation to P and N is shown in Equations (2) and (4) respectively. The total amount of data that needs to be processed in the training stage grows with P whereas that in the testing stage is constant and independent of P. The situation is in practice more complex as we still need to consider time complexities of the algorithms that do the training and testing.

$$\begin{aligned} \sum_{1}^{P} \frac{N(P-1)}{P} &= \frac{NP^2 - NP}{P} && (1) \\ &= N(P-1) && (2) \end{aligned}$$

$$\begin{aligned} \sum_{1}^{P} \frac{N}{P} &= \frac{NP}{P} && (3) \\ &= N && (4) \end{aligned}$$

Besides partitioning data for cross-validation we are interested in seeing the effects of data set size on the results. In theory, more data should lead to better results and we are interested to see how the performance improves over total data size. For this reason we take M samples from N. Without considering the effects of cross-validation this would give us a simple arithmetic series as shown in Equation (7),

$$\sum_{i=1}^{M} \frac{iN}{M} = \frac{M}{2} \times \left[\frac{2N}{M} + \frac{(M-1)N}{M} \right] \qquad (5)$$

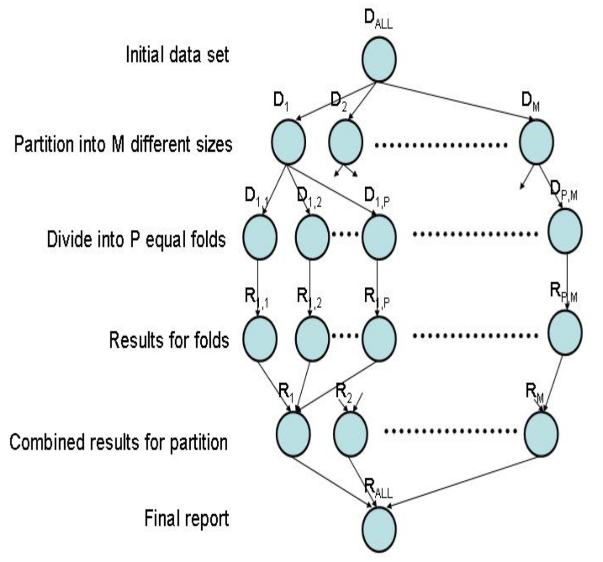

Figure 2. Embarrassingly parallel: data partitioning on the original data set.

$$= N + \frac{(M-1)N}{2} \tag{6}$$
$$= \frac{N(1+M)}{2} \tag{7}$$

Now if we substitute the Equations (2) and (4) into (7) we find that the effect of combining cross-validation and data size partitioning is

$$\frac{N(P-1)(M+1)}{2} \tag{8}$$

for the size of data in the training stage, and

$$\frac{N(1+M)}{2} \tag{9}$$

for the size of data in the testing stage.

The process of data partitioning is shown in Figure 2. Clearly this lends itself to a single process multiple data model of processing where we can in theory run the same training and testing algorithm on the partitions of N in parallel, collect the results together and evaluate. In practice experimental runs need to be repeated several times with different subsets of features that we want to explore, but this is ignored in this discussion.

The analysis of the other stages in the task is as follows: data formatting is essentially linear on N, and evaluation is done only over the sum of the test data given in Equation (9).

Table 2. Summary of dedicated LAN

Network	Fast Ethernet (100 Mbps)
Topology	Switched ethernet, single switch
Interconnect	Twisted-pair (RJ45)
Network switch	HP ProCurve 2324
NIC	Intel 82550 10/100

In theory if we had available $P \times M$ CPUs we could do the whole training and testing stages linearly with respect to the data size. In practice we only have a finite number R of CPUs and we have implemented a queuing system to pool the list of $P \times M$ unique partitions that need to be processed. In the tests reported here we used the Perl Parallel ForkManager[1] with a maximum limit of R on the number of processes that are run concurrently. These processes are then automatically load balanced by Open Mosix across the cluster nodes.

3 Experimental Context

3.1 Compute Nodes

The Enigma cluster is a small cluster of Appro dual Pentium Xeon 2.4 GHz computers on a 400MHz front-side bus and one IBM xSeries 330 with dual Pentium III CPUs. This is summarized in Table 1

G01 has a special role as it holds the cluster file system with access through the OM file system. It also provides natural language parsing service to other nodes using the FDG dependency parser from Conexor [14].

G01 is different to the other compute nodes in that it has 2 Pentium III CPUs and a 133MHz front-side bus. It is well known [4] that this architecture has a memory bottleneck when both CPUs are used in symmetric multiprocessing (SMP). In other words the CPU remains idle while it waits for a response to its memory request. The Intel Xeon was designed to overcome this limitation.

3.2 LAN and Switch

All nodes are connected over standard 100Base-T Fast Ethernet using a Hewlitt-Packard Procurve 2324 unmanaged layer 2 switch with 24 10/100Base-TX autosensing ports, 9.6 Gbps switch fabric, 6.6 million pps (64-byte packet) throughput, and a latency of less than $10\mu s$ (LIFO). The LAN‘s characteristics are summarized in Table 2.

[1] Available from http://hacks.dlux.hu/Parallel-ForkManager/

Table 1. Description of cluster nodes. Note that Open Mosix was installed on RedHat Linux 7.3 on all cluster nodes. †In L1 figures are given for D=data cache, I=instruction cache, or a unified cache.

Host Name	System Description	Clock (MHz) & CPU name	RAM (GB)	L1 (KB)†	L2 (KB)	Operating System
G01	IBM xSeries 330	2 × 1200 Pentium III	0.76	16	512	Linux OM 2.4.19.4smp
E01	Appro 1224Xs	2 × 2400 Xeon	1	8(D)/12(I)	512	Linux OM 2.4.19.4smp
E02	Appro 1224Xi	2 × 2400 Xeon	1	8(D)/12(I)	512	Linux OM 2.4.19.4smp
E03	Appro 1224Xi	2 × 2400 Xeon	1	8(D)/12(I)	512	Linux OM 2.4.19.4smp

4 Metrics

We describe below general and specific metrics of comparison that characterize the overall performance of the Enigma cluster using the node set {G01,E01,E02,E03}. When evaluating the system performance it is important to select the most appropriate benchmarks. In this case we have run tests using three standard benchmarks: Netpipe to measure the node interconnection speed, HPL to measure overall cluster floating-point performance on a SPMD task, and LMbench. The best overall benchmark though for our purpose are our own Text Mining scripts which involve heavy bursts of I/O for formatting data and CPU for finding the best named entity assignments with the HMM using maximum likelihood estimates in the Viterbi algorithm [16]. This is described later.

4.1 NetPIPE

We measured the point-to-point node interconnect performance using NetPIPE [13], a ping-pong LAN speed measurement over block size and obtained results shown in Figures 3. In the *throughput graph* of Figure 3(a) it can clearly be seen that at just under 90Mbps we are obtaining close to the theoretical maximum from the switch for large packet sizes. (b) shows the *network signature graph* which highlights the latency in cluster communication, i.e. the first plotted point on the graph. (c) shows the saturation point by plotting block size versus the transfer time on logarithmic scales. The saturation point is the knee of the curve and shows the point at which network throughput cannot be increased by an increase in block size. Finally (d) shows results for local TCP/IP traffic, i.e. without using the network, to show the influence of the interconnect.

Table 3. Time in microseconds to execute common system commands on E01

	Linux-2.4.19-openmosix4smp	
	μs	σ_{n-1}
null call	0.84	0.00
null I/O	1.40	0.00
stat	4.83	0.00
open/close	6.63	0.00
select	25.75	0.06
signal install	1.23	0.01
signal catch	4.51	0.02
fork proc	217.25	1.26
exec proc	563.75	2.22
shell proc	2955.75	2.50

4.2 LMbench 2.0 Benchmark

LMbench is a series of micro-benchmarks that measure OS and hardware performance. The results reported in Tables 3 to 6 are for four independent runs of LMbench 2.0 [8] on the E01 node since this is representative of the other E-series compute nodes.

4.3 HPL

The High Performance Linpack benchmark (HPL) [5] was used to provide an overall level of performance for the whole cluster. HPL basically solves a series of random dense linear systems and is the standard used to evaluate the world's top performing cluster computers and the results are therefore highly transparent. Results are given in GFlops (billions of floating point operations per second).

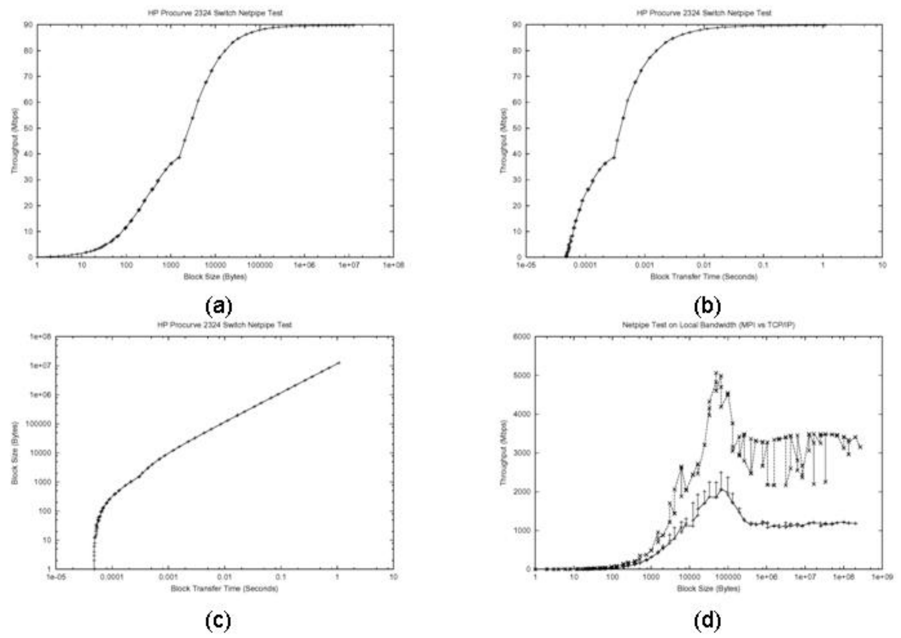

Figure 3. NetPIPE measurements for the HP Procurve interconnect: (a) LAN throughput, (b) LAN signature, (c) LAN saturation, (d) local traffic for MPI and TCP/IP

Table 4. Context switching: the time in microseconds it takes for n processes of size s to switch context on E01.

	Linux-2.4.19-openmosix4smp	
	μs	σ_{n-1}
2p/0K	3.93	0.13
2p/16K	4.89	0.14
2p/64K	6.48	0.78
8p/16K	5.33	0.25
8p/64K	13.33	0.71
16p/16K	6.26	0.22
16p/64K	36.1	1.28

Table 5. Interprocess communication latencies in microseconds between two processes on E01

	Linux-2.4.19-openmosix4smp	
	μs	σ_{n-1}
2p/OK	3.93	0.13
pipe	14.15	0.60
AF UNIX	23.23	1.00
UDP	22.7	0.42
RPC/UDP	32.58	0.57
TCP	28.4	2.11
RPC/TCP	42.18	0.53
TCP conn	48.55	0.08

Table 6. Memory latencies in nanonseconds for E01

	Linux-2.4.19-openmosix4smp	
	μs	σ_{n-1}
L1 cache	0.834	0.00
L2 cache	7.70	0.01
Main memory	149.9	0.26

HPL was installed on one of the Xeon nodes (E01) of the cluster. HPL used the MPICH (version 1.2.4) [6] implementation of the Message Passing Interface (version 1.1 compliant), installed on all nodes and configured to allow a maximum of 4 processes for E01 to E03 and 2 processes for G01. The Vector Signal Image Processing Library (VSIPL) implementation of the Tactical Advanced Signal Processing (TASP) Common Operating Environment Working Group[2] was also installed on E01.

We performed two sets of experiments, the first using MPICH and OM together. Processes were created on remote nodes using *rsh* with the xinet.d files modified so that processes created in this way would explicitly run using OM using the *mosrun* command. In the second set of experiments OM was disabled. Despite our hope that OM would load balance MPICH processes we could not find a way to make them migrate and results for both runs were not significantly different. We therefore report below the results for our second set of experiments average over 10 runs in Table 7 below. The reason for migration failure may well be connected with MPI processes being socket-bound and OM inability to migrate sockets (although there are currently plans to enable this in the future).

Since the HPL tests require message passing for interprocess communication they are very sensitive to bandwidth and latency. The overall results are very poor considering the theoretical maximum CPU power of the overall cluster and indicate the limitation imposed by the network interconnect. MPICH with SMP generally performs better than MPICH without SMP.

5 HMM Text Mining Task

The Text Mining Task is the most important performance measure we wish to evaluate. In these tests we evaluate a hidden Markov model that uses linear interpolated bigrams. The main cost during training is imposed by finding counts of words and bigrams from the annotated texts which requires heavy use of sorting and other I/O intensive tasks. In testing the main cost is in the $O(n^2)$ Viterbi algorithm used to find a near-optimal path through the path of class labels assigned to words from the named entity class list. The task is therefore characterized by mixed bursts of input/output bound processes and CPU bound processes.

[2]Available from: http://www.vsipl.org/software

Class	**#**	**Description**
PROTEIN	2125	proteins, protein groups,families, complexes and substructures
DNA	358	DNAs, DNA groups, regions and genes
RNA	30	RNAs, RNA groups, regions and genes
SOURCE.cl	93	cell line
SOURCE.ct	417	cell type
SOURCE.mo	21	mono-organism
SOURCE.mu	64	multi-celled organism
SOURCE.vi	90	viruses
SOURCE.sl	77	sublocation
SOURCE.ti	37	tissue

Table 8. Markup classes used in Bio1 with the number of word tokens for each class.

After performing experiments in the previous section we expanded the cluster to include one more compute node called E04 with a similar specification to E03 except that it has 2GB RAM. We also expanded E01 to have 2GB RAM.

In early experiments we used E01 as the combined file server and Unique Home Node (UHN) for all processes. For reasons that we comment on in the Conclusion this caused E01 to regularly crash and we decided to put all file servicing on G01 through the OM file system and to use E04 as the UHN. G01 was configured to forbid accepting migrated processes from other nodes to reduce its processing load.

Using this setup we performed tests on a molecular biology data set with characteristics of named entities shown in Table 8. The total size of the data including annotations is quite small, only 232Kb. Each test ran 10 times, used 10 sample sizes (M=10) and performed 10-fold cross validation (P=10) on each sample. The total number of experiments performed is therefore 1000 for each timed result.

Results are shown in Table 9 for absolute wallclock times and Speedup, calculated as,

$$Speedup(n) = \frac{\text{Processing time of 1 fork}}{\text{Processing time of n parallel forks}} \tag{10}$$

We also performed two types of experiments in which the queue of jobs was given linearly (in order of M and P)

Table 7. Results of running the High Performance Linpack Benchmark (HPL) on the Enigma cluster using only MPICH on all nodes. All results are given in GFlops.

Grid	**2000**	**5000**	**8000**	**10000**
4 nodes (G01,E01,E02,E03) and MPI with SMP				
1 × 14	0.5097 (± 0.0117)	1.0540 (± 0.0170)	1.0735 (± 0.0285)	0.9899 (± 0.0210)
1 × 12	0.5569 (± 0.0098)	1.0672 (± 0.0158)	0.9739 (± 0.0310)	0.8758 (± 0.0178)
1 × 8	0.7180 (± 0.0093)	0.8884 (± 0.1001)	0.7019 (± 0.0247)	0.6510 (± 0.0278)
2 × 4	0.7872 (± 0.0942)	0.5589 (± 0.0423)	0.4519 (± 0.0086)	0.3975 (± 0.0066)
1 × 4	0.7790 (± 0.0026)	0.4848 (± 0.0014)	0.3796 (± 0.0006)	0.3662 (± 0.0006)
4 nodes (G01,E01,E02,E03) and MPI without SMP				
1 × 14	1.0229 (± 0.0275)	1.2155 (± 0.0283)	1.0562 (± 0.0198)	0.9640 (± 0.0207)
1 × 12	0.9718 (± 0.0311)	1.0726 (± 0.0195)	0.8447 (± 0.0175)	0.7820 (± 0.0237)
1 × 8	0.7823 (± 0.0209)	0.6459 (± 0.0271)	0.4837 (± 0.0078)	0.4467 (± 0.0085)
2 × 4	0.4670 (± 0.0145)	0.3221 (± 0.0028)	0.2910 (± 0.0014)	0.2747 (± 0.0020)
1 × 4	0.3717 (± 0.0190)	0.2319 (± 0.0028)	0.2062 (± 0.0015)	0.1975 (± 0.0020)

and in which it is randomly ordered. The intuition behind random ordering is to spread the I/O bursts randomly among the forks which is governed by the size of the data set used in the experiment.

We clearly see that Speedup is far below linear and is in fact converging quickly. Even though we did not use the maximum number of CPUs in the cluster we can see that it would not be greatly beneficial to add more given the current setup. It is clear though that the bottleneck is in the slow speed of the network as both CPU and memory appear under utilized in our analysis of execution profiles.

6 Conclusion

In general OM was shown to provide an excellent mechanism for running SPMD experiments without the need to incorporate explicit parallelism into the software code. In two key areas though our experience of using OM under HPC conditions showed that bottlenecks exist, at least with regard to the types of processes we want to run.

Firstly too much communication between CPUs and a low speed network in practice caused the cluster to perform at well below optimal performance levels given the cluster-wide CPU and memory.

Secondly, we saw a heavy burden on the Unique Home Node (UHN), i.e. the node where the processes originated. This is probably due to the way in which OM migrates processes in which processes that perform moderate to heavy I/O are encouraged to migrate to the node where the files are located in order to save network time. If this is the UHN then this means that there will be a heavy CPU and I/O burden on the node leading to a building cycle of slower job completion times. In our earlier experiments with a combined central UHN and fileserver node this caused the node (E01) to crash. An alternative explanation may be due to the scheduling algorithm which may have tried to adjust processing for the low interconnect speed. It is unclear at this point whether this was a contributing factor pending further analysis.

One solution (without changing the network) that seems to be slightly more robust is to dedicate one node as a fileserver and to forbid process migration to this node from others which are used as compute nodes. In practice we have found that this reduced the number of node crashes considerably, but at a cost of speed in processing time. This should be overcome with a new faster interconnect.

Since we are aiming to achieve high performance we plan now on upgrading to a Myrinet-2000 network very soon. We understand that plans exist to make Myrinet drivers for OM and when this is done we fully expect performance to increase significantly.

A further point to note is that OM tries its best to balance the load of jobs across the cluster given the available resources, but it cannot take a job off the cluster if the cluster is overloaded. Monitoring job completion and putting new jobs onto the cluster is a function performed by a job scheduler in traditional batch queuing systems. In our system we used Parallel Fork Manager to control scheduling but since the task of the software is only to maintain a fixed number of running processes, it has no way to interact with the load information on the system to add more processes or delay adding processes. We therefore feel that a fruitful line of investigation will be to combine intelligent job queuing with OM to get the best out of our cluster.

Finally, the approach we have outlined here to performing experiments naturally extends to other types of machine

Table 9. Results of running the NEHMM text mining on the Enigma cluster using OpenMosix on all nodes. All results are given in wall clock seconds timed using the Perl Benchmark module. The number of Forks is the maximum number of parallel forks given to the Perl Parallel::ForkManager module.

Number of	**Wallclock execution times**			
Forks (n)	Non-random	Speedup(n)	Random	Speedup(n)
1	27123	1	27113	1
2	16249	1.669	16029	1.691
3	12063	2.248	12092	2.242
4	10756	2.522	10343	2.621
5	9167	2.959	8805	3.079
6	8296	3.269	7934	3.417
7	7585	3.576	7483	3.623
8	7187	3.774	6990	3.879
9	6775	4.003	6654	4.075
10	6676	4.063	6427	4.219
11	6292	4.311	6183	4.385
12	6222	4.359	6118	4.432

learning tasks such as speech processing which also makes heavy use of HMMs during training.

Acknowledgements

This work was supported in part by the Japanese Ministry of Education and Science (grant no. 14701020).

References

[1] Y. Amir, B. Awerbuch, A. Barak, A. S. Borgstrom, and A. Keren. An opportunity cost approach for job assignment in a scalable computing cluster. *IEEE Trans. Parallel and Distributed Systems*, 11(7), July 2000.

[2] A. Barack, O. La'adan, and A. Shiloh. Scalable cluster computing with MOSIX for LINUX. In *Proceedings of Linux Expo '99, Raleigh, N.C.*, pages 95–100, May 1999.

[3] A. Barak and O. La'adan. The MOSIX multicomputer operating system for high performance cluster computing. *Journal of Future Generation Computer Systems*, 13(4–5):361–372, March 1998.

[4] Cornell Theory Center. Benchmarking intel systems and understanding the results. *High-Performance Computing*, 4:53–55, 2001.

[5] J. Dongarra. Performance of various computers using standard linear equations software. Computer Science Technical Report CS-89-85, University of Tennessee, Knoxville, TN, 37996, 1989. Available from http://www.netlib.org/benchmark/performance.ps.

[6] W. Gropp, E. Lusk, N. Doss, and A. Skjellum. A high-performance, portable implementation of the MPI message passing interface standard. *Parallel Computing*, 22(6):789–828, Sept. 1996.

[7] S. McClure and R. Wheeler. MOSIX: How linux clusters solve real world problems. In *Proceedings of USENIX 2000 Annual Technical Conference, San Diego, CA.*, pages 49–56, June 2000.

[8] L. McVoy and C. Staelin. lmbench: Portable tools for performance analysis. Technical report, Silicon Graphics, Inc. and Hewlett-Packard Laboratories, January 1996. Available at http://www.bitmover.com/lmbench/.

[9] MEDLINE. The PubMed database can be found at:, 1999. http://www.ncbi.nlm.nih.gov/PubMed/.

[10] Nancy Chinchor, editor. *MUC-7 Named Entity Task Definition, Version 3.5*, This document should be available from Nancy Chinchor chinchor@gso.saic.com, September 17th 1997. DARPA.

[11] H. Ney, S. Martin, and F. Wessel. Statistical language modeling using leaving-one-out. In S. Young and G. Bloothooft, editors, *Corpus-Based Methods in Language and Speech Processing*, pages 174–207. Dordrecht: Kluwer Academic, 1997.

[12] L. Rabiner and B. Juang. An introduction to hidden Markov models. *IEEE ASSP Magazine*, pages 4–16, January 1986.

[13] Q. O. Snell, A. Mikler, and J. L. Gustafson. NetPIPE: A network protocol independent performance evaluator. In *Proceedings of IASTED International Conference on Intelligent Information Management and Systems*, June 1996.

[14] P. Tapanainen and T. Järvinen. A non-projective dependency parser. In *Proceedings of the 5th Conference on Applied Natural Language Processing, Washington D.C., Association of Computational Linguistics*, pages 64–71, 1997.

[15] V. N. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag, New York, 1995.

[16] A. J. Viterbi. Error bounds for convolutions codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, IT-13(2):260–269, 1967.