Chapter 6

Conclusions and Future Work

6.1 Conclusions

Bearing in mind the need to coschedule distributed applications in Clusters or NOWs when they are executed jointly with local workload, two coscheduling techniques have been presented in this project. One of them follows an explicit-control trend and the other belongs to the implicit-control category. Their performance was measured and compared with other coscheduling techniques of the literature. Experiments were performed by simulation and in a real PVM-LINUX Cluster, the system where they were implemented. As a consequence of implementing all the studied techniques in the same Cluster, when comparing them very interesting conclusions arise.

The proposed coscheduling methods behave well for various kinds of message passing distributed applications. But as one may expect, coscheduling effects were mainly located in their respective synchronization phases and in the message receiving and sending exchange procedures.

6.1.1 Explicit-control Coscheduling

First of all, an Explicit coscheduling mechanism was designed and implemented in a real Cluster. As in gang scheduling, it allows guarantees to be made for the performance of distributed applications. This was accomplished by assigning two respective execution intervals to distributed (and local) applications. In doing so, the Cluster resources are entirely dedicated for regular an uninterrupted periods of time to distributed (local) tasks. This ensures the progress of distributed applications but of course, some performance penalties are introduced in the local tasks.

Two models were designed to achieve the speedup of message-passing intensive distributed applications (STATIC and BALANCED). They were based on synchronizing the parallel and local execution periods. Basically, performance improvements in communicating/synchronizing distributed applications exceeded the overhead introduced in the VM by the processes and mechanisms which implemented it. Furthermore, one model (BALANCED) adjusted these periods to the mean local workload. This also caused performance achievement of message-passing intensive applications.

For non-communicating distributed applications, that is, CPU bound distributed applications, no need to synchronize these periods was required. Synchronization only added unnecessary overhead to both distributed and local applications. In the DISTRIBUTED mode, the periods in each node were adjusted to its corresponding local workload.

With the proposed explicit-control scheme not only was it possible to increase performance of fine grained distributed applications but it was also possible to increase performance of the CPU-bound ones. The ability of such a mechanism to select the mode it will work in, is an advantage over the implicit-control ones. Also, by gathering information about the distributed applications it should be possible to set the most efficient coscheduling mode.

In explicit-control coscheduling, fault tolerance is a problem. Due to the centralized nature of this technique, the possibility of *master* crashes should be taken into account. If the environment were distributed, that is, if the master work were distributed between all the component nodes of the Cluster, then the controlling communication should increase, perhaps by an unacceptable amount. The synchronization would also be extremely complicated. Moreover, abnormal behavior of the explicit scheme must be considered; this will imply more extra communication and overhead.

However, explicit-control policies need not be discarded. In some situations where for example, the system is heavily loaded, Explicit coscheduling is very useful. Furthermore, the saturation of nodes (even only one particular node) in the Cluster can produce negative effects on the distributed applications. It may accordingly be beneficial to apply the Explicit model at the cost of introducing some additional penalties in the local workload in these situations.

6.1.2 Implicit-control Coscheduling

The above mentioned explicit-control problems pointed to the development of new implicit-control techniques.

The first coscheduling technique studied based on implicit-control, also named "Implicit", suspends a blocked receiving process until active waiting for messages is exceeded by the cost of doing a context switch. While the message waiting is performed, the CPU remains occupied doing empty loops; so local tasks (even distributed ones) cannot execute. Meanwhile, in time-sharing systems, opportunities for executing other tasks are wasted.

A variation of Implicit coscheduling, based on the spin-block technique was implemented and evaluated in a Linux Cluster. In general, an slightly gain was obtained when the system workload was low. The spinning gain in blocking receives did not justify the overhead introduced in both distributed and local applications. Also, the added penalties into local tasks degraded their performance excessively. The conclusion is that in time sharing systems, active waiting for an event to occur (in our case blocking receive) is not a good solution.

Special attention was paid to a new "Predictive" coscheduling technique. A model, some related performance metrics and a coscheduling algorithm for Predictive coscheduling in Cluster computing are also proposed in this thesis. This set of metrics provides a means for obtaining on time parallel performance. The coscheduling parameters can be set or tuned on time, while the distributed applications are executing.

When it was compared with other techniques, Predictive coscheduling gave the best performance values in almost all the situations and specially when the Cluster is lightly or medium loaded. This method reduced the synchronization phases and the communicating processes significantly. The Predictive model assigns more scheduling priority to processes with high communication frequency.

Starvation of local tasks (even distributed ones) is avoided by restricting the number of times a task can be delayed (overtaken) by another one.

The Predictive method was implemented in the system space inside the Linux kernel. In contrast, the previous commented coscheduling models were implemented in the user space, either by adding controlling processes (i.e. Explicit and HPDT) or by modifying the PVM communication libraries (Implicit). In doing so, it was first of all necessary to study the PVM-Linux communication mechanism and the Linux Scheduler. The location where the coscheduling mechanism must be incorporated depends basically on the features the resulting environment must provide. Implementation in the system space provides transparency to the final user and also the models do not depend on the DCE used (i.e. PVM or MPI). On the contrary, user level implementations do not require the reconfigurability of the underlying operating system.

A variation of predictive coscheduling, named Dynamic coscheduling was also presented. Unlike the Predictive model, Dynamic coscheduling is based only on receiving frequency. It was shown that this method did not identify the correspondent processes as well as in the Predictive model, and consequently the Dynamic mechanism worked worse.

The results obtained by executing various kernel benchmarks from the NAS suite (with different messaging characteristics) and three PARKBENCH Low-Level benchmarks (which measured performance in barriers and communication in one and two directions) demonstrated that Predictive model is applicable in a wide range of fine grained distributed applications.

The NAS kernel benchmarks allowed the different applications comprising it to be scaled, but some particular features, such as communication frequency or message length, cannot be tuned. Thus, it was necessary to develop various synthetic applications for doing this. Their use allowed the good behavior of the Predictive mechanism to be measured when the message length of a synthetic application was varied.

Also, performance slowdown on local workload produced by the coscheduling models was measured. Some local applications were developed for this purpose. The slowdown is obtained by comparing the overhead introduced in the local applications when the distributed applications are executed in the plain PVM-Linux environment and when they are executed in the different coscheduling models.

Similar penalties in local tasks were introduced by the Predictive and Dynamic models. Predictive introduced slightly more local overhead but not in the same proportion as the gains obtained for the distributed applications. Only a moderate overhead was introduced into local tasks. Moreover, it was shown in the experimentation that this overhead was proportional to the CPU requirements of the distributed applications. The communication hardly affected the performance of local applications. Thus, promoting distributed applications based on their communication frequency, as the Predictive model does, is an efficient coscheduling policy.

6.1.3 Additional Conclusions

Simulation vs. Implementation: it was not possible to find significant performance differences between the Predictive and the Dynamic model by simulation. In the contrary, in the real experimentation, the results obtained for these two coscheduling techniques demonstrated clearly the advantages of the Predictive over the Dynamic model. This means that simulation cannot totally model real environments. There are situations where the broad range of the different events and parameters that influence the behavior of the overall system cannot be efficiently simulated.

The motivation for making a simulator was to investigate the performance of the presented coscheduling policies separately, avoiding effects that do not influence the coscheduling schemes, such as, computing power, hardware resource features of each Cluster node (i.e. operating system latency), network latency, bandwidth and bottlenecks, and so on.

On the other hand, the experiments done in the real implementation cover the influence of the above commented effects. Also, there were many system processes and service daemons which influenced the measurements. Certainly, it would be possible to remove system processes or services to obtain more accurate measurements, but the interest was centered on commodity and non-dedicated Clusters. Usually, these kind of system deal with such kind of processes, so it was decided to keep them in the system.

CMC Metrics: to obtain traditional metrics, such as Speedup or Efficiency, two different executions must be performed: one in a serial (uniprocessor) and later in the parallel or distributed system (platform to evaluate these metrics). The Speedup (S) is defined as: $S = T_1/T_n$, where T_1 is the application execution time in the serial system and T_n, the execution time of the parallel version of the application in the parallel system. Efficiency (E) is defined as: $\mathrm{E} = S/n$, where n is the number of processors in the parallel system. Note that since the results are obtained in the serial system, there is no need to obtain them twice in the parallel one, but perhaps with different performance.

Unlike traditional metrics, CMC metrics can be obtained on time, so the distributed system parameters can be also tuned on time, while distributed applications are being executed.

By using CMC metrics (i.e. *CoDe metrics), there is no need to wait for the end of the distributed tasks. The problem is that these metrics do not give absolute performance. On the other hand for example, it should be possible to determine the good behavior of one model by comparing performance with HPDT. It is only necessary to have a command or system call (like the ones presented in Appendix A) which provide the ability to change between models dynamically by setting or modifying some system coscheduling variables. However, to compare performance of different models will only be advantageous for large enough distributed applications.

Tracing Mechanisms: PGPVM2, an environment which produces trace files to be visualized with ParaGraph, was used in the analysis of the message pattern of the distributed applications. Many problems or disadvantages in its use were detected: large tracing files, too much overhead, inefficiency (or impossibility) of tracing large distributed applications, and so on. Even the XPVM [76] tracing environment produced poorer results. It is necessary to develop more efficient trace system facilities with also more efficient methodological techniques.

6.2 Future Work

The performance of a good coscheduling policy can decrease drastically if memory requirements are not kept in mind. Various researchers ([80, 14, 78, 79]) have proposed different techniques to minimize the impact of job memory requirements on the performance of various scheduling policies. However, to our knowledge, there is an absence of research into minimizing the impact of the memory constraints in an implicit-control coscheduling environment.

We are interested in proposing implicit-control coscheduling techniques with memory considerations. That is, to coschedule distributed applications by considering dynamic allocation of memory resources due to the execution of local/distributed jobs by using implicit information.

In a non-dedicated system, the dynamic behavior of local applications (which consequently also varies the resident memory allocated) or a distributed job mapping policy without memory considerations cannot guarantee that parallel jobs have enough resident memory as would be desirable during their execution. In these conditions, the local scheduler must coexist with the demand-paged virtual memory mechanism. The paging mechanism improves memory and CPU utilization by allowing processes to run with only a subset of their code, and data to be resident in main memory. However, the traditional benefits that paging provides in uniprocessors, in distributed (Cluster or NOW) environments may be decreased depending on various factors, such as the interaction between the CPU scheduling discipline, the synchronization patterns within the application programs and the page reference patterns of these applications [73].

A new coscheduling environment over a non-dedicated Cluster system which also supports prevention of local-task starvation capabilities has already been proposed in [81]. Such a scheme is based on reducing the number of page faults in a non-dedicated Cluster system, giving more execution priority to the distributed tasks with lower page faulting probability, letting them finish as soon as possible. Thus, on their completion, the released memory will be available for the remaining (local or distributed) applications. Consequently, major opportunities arise for advancing execution of the remaining tasks. It was proved by simulation that under this coscheduling environment, the impact of demand-paged virtual memory is reduced.

Nowadays, the current technology makes it possible to acquire general purpose multiprocessor workstations at low cost. On the other hand, only Clusters made up by uniprocessor nodes where considered in the design of the CMC model and its associated Predictive algorithm. Despite the presence of multi-processor workstations, no consequences are produced in the CMC model, which is only applied in the uniprocessor nodes. However, the coscheduling mechanism can vary significantly if the model is provided with multi-processor capabilities. In its design, it would be necessary to answer questions such as: how efficient the assignment of more coscheduling priority to distributed processes residing in the same node should be; how must the workload between the processors be assigned and balanced to obtain distributed gains without damaging the local ones; and so one. The range of coscheduling possibilities increases when considering multiprocessor capabilities. Their resolution will also occupy our attention in the future.

Appendix A

DTS and Linux Command

DTS Console Commands

```
Console> help

GENERAL commands:

quit, exit:  quit Console
      help:  get this help
     hosts:  list readed hostfile
      ping:  list active hosts
   settout:  <sec> set timeout in receiving answers from hosts
  getloads:  get loads from all active hosts
   getqtms:  get PS and LS from all the active hosts
     stats:  get all the DTS parameters
      mode:  [static/balanced/distributed] set or print mode

CONTROL commands:
    setq:  <PS> <LS> set PS and LS (μsecs)
sethostq:  <host> <PS> <LS> set PS and LS in one host (μsecs)
 setlint:  <sec> set LI (BALANCED and DISTRIBUTED)
 setqtum:  <usec> set IP (μsecs)
syncsend:  send a synchronization message (STATIC and BALANCED)
```

Linux Command and System Call

Linux Command

```
#include<stdio.h>

#include<stdlib.h>

#include<linux/unistd.h>

_syscall3(int, cosched, int, policy, double, p);

main(int argc, char ** argv) {

   if (argc != 3) {

      perror("usage: $ crida policy p\n");

      perror("goodby");

      exit(1);

    }

   cosched(atoi(argv[1]),atoi(argv[2]));

   printf("policy:  %d, p %d \n",atoi(argv[1]),atoi(argv[2]));

}
```

System Call

The system call *cosched* is implemented in the following kernel files:

```
/* *************** entry.S *************** */

 .long SYMBOL_NAME(sys_cosched)      /* 191 */

/* *************** unistd.h *************** */

#define __NR_cosched 191

/* *************** sys.c *************** */

#include<linux/unistd.h>

asmlinkage int sys_cosched(int policy, int p) {

   int error= 0;

   struct task_struct * p;

   struct task_struct * q;

   POLICY = policy; P = p;

   q = &init_task;

   for (p = q->next_task; p != q; p=p->next_task)

      if (p->priority != 20) {

         p->priority = 20;

         p->counter = 0;

      }

   return error;

}
```

Appendix B

Additional Results

Simulation

The next figures show the obtained simulation results when the number of Cluster nodes is increased (*NSTATIONS*=8 and *NSTATIONS*=16).

As in chapter 4, SCODE figures correspond to the *SystemCoDe* (System Cos-cheduling Degree) comparison between the LIN, IMP, EXP, HPDT, DYN and PRE models. The TIMES figures show the *Dret* (Distributed return time), *Dwait* (Distributed waiting time), *Lret* (Local return time) and *Lwait* (Local waiting time) for the same executions.

Also a comparison of the *SystemCoDe* when the *MCO* is varied between $MRQL-2$ and $MRQL+2$ for the distributed and Predictive models is shown in the MCO figures.

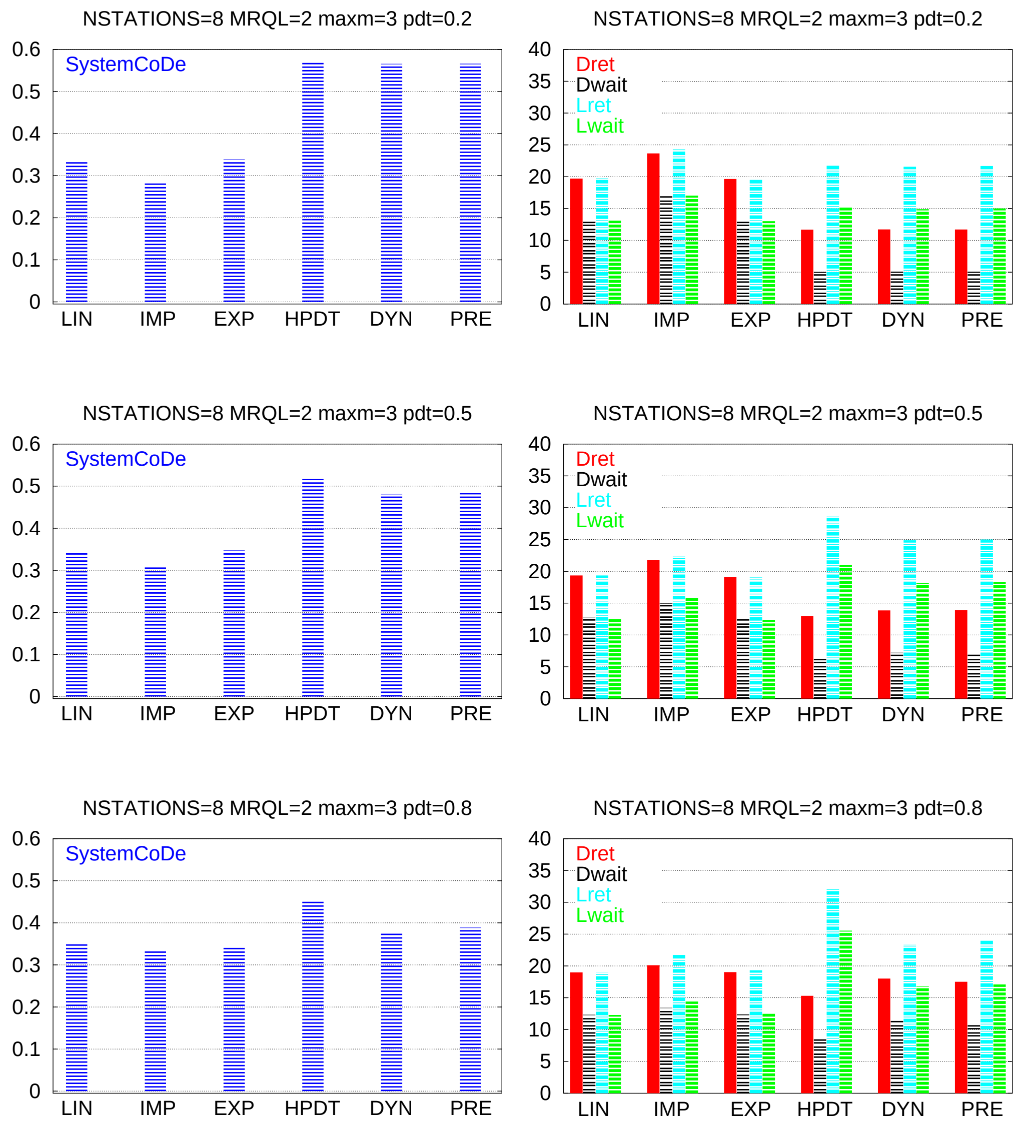

Figure B.1: *NSTATIONS*=8, *MRQL*=2. (left) SCODE (right) TIMES.

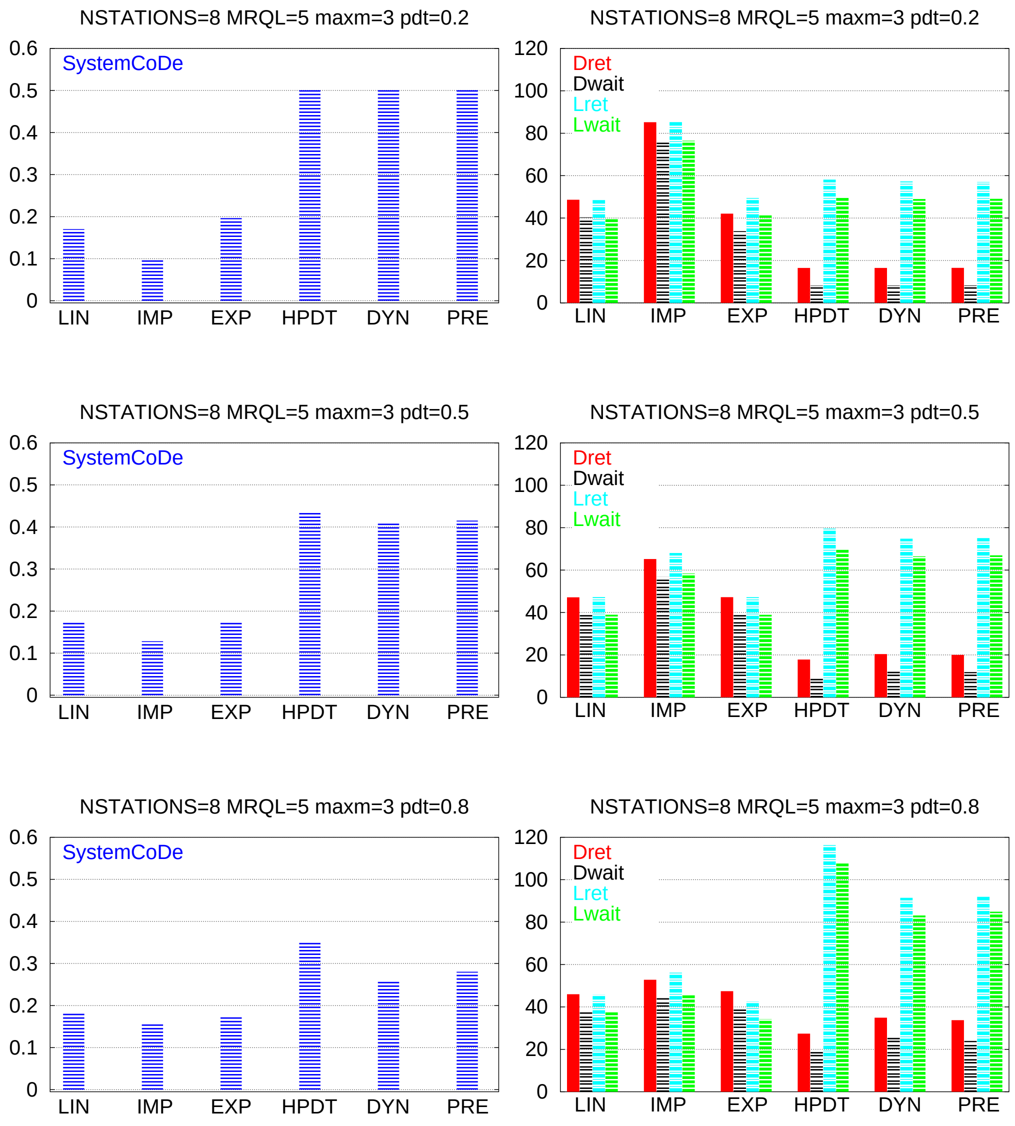

Figure B.2: *NSTATIONS*=8, *MRQL*=5. (left) SCODE (right) TIMES.

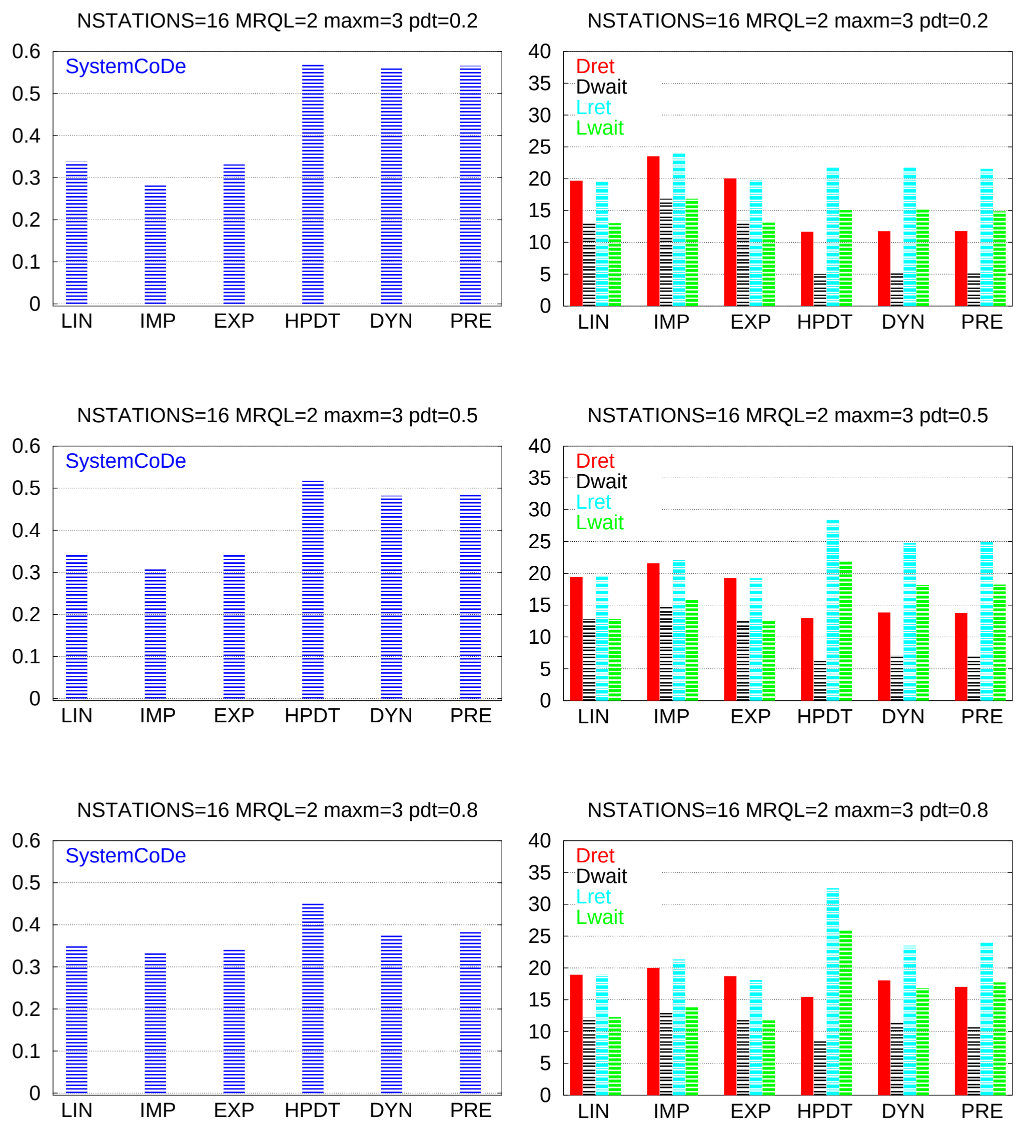

Figure B.3: *NSTATIONS*=16, *MRQL*=2. (left) SCODE (right) TIMES.

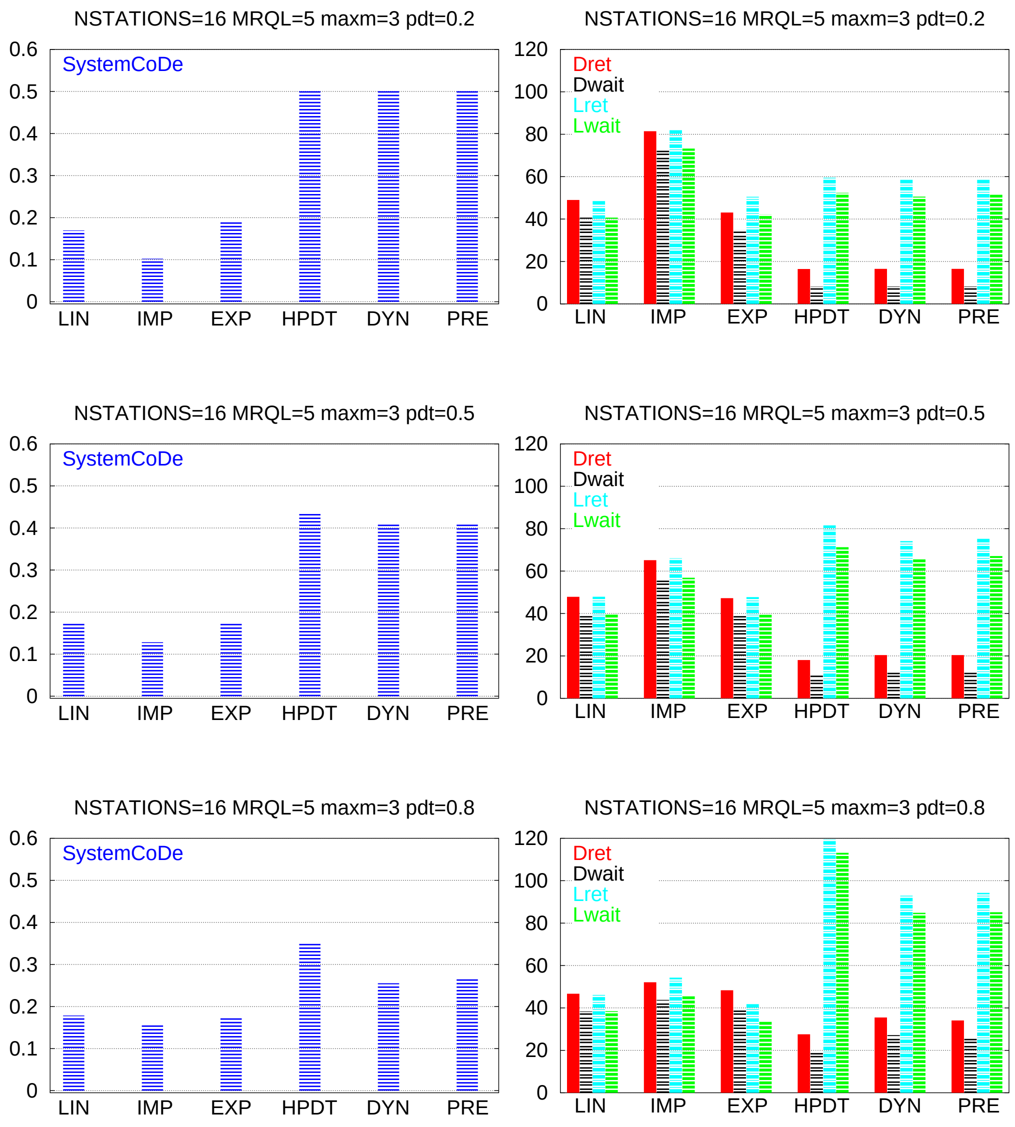

Figure B.4: *NSTATIONS*=16, *MRQL*=5. (left) SCODE (right) TIMES.

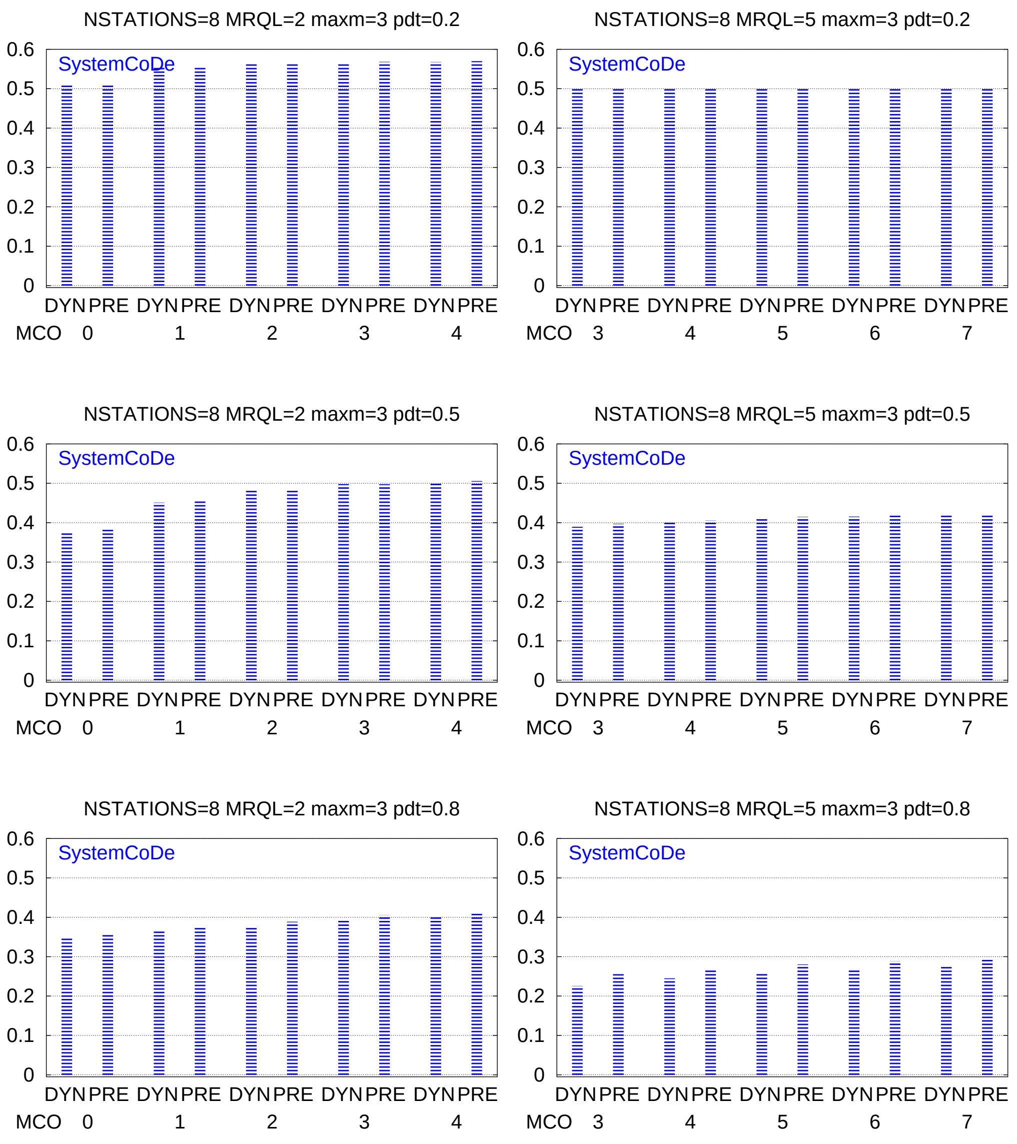

Figure B.5: MCO. *NSTATIONS*=8. (left) *MRQL*=2 (right) *MRQL*=5.

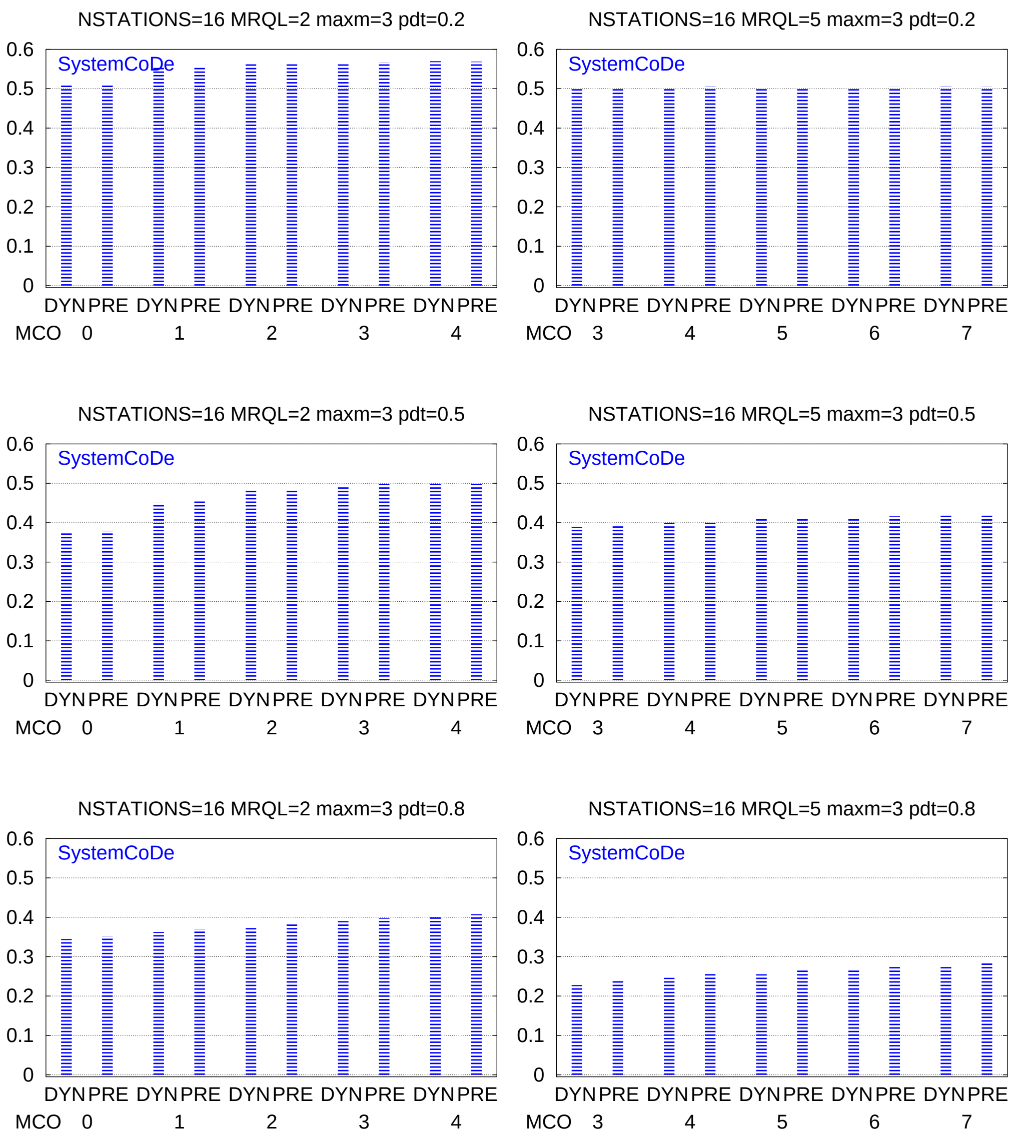

Figure B.6: MCO, *NSTATIONS*=16. (left) *MRQL*=2 (right) *MRQL*=5.

Implementation

NAS Benchmarks

This appendix shows additional experimentation performed with the NAS parallel benchmarks CG, FT, BT and SP.

The FT class A memory requirements overlapped the total available main memory, so the obtained results were discarded.

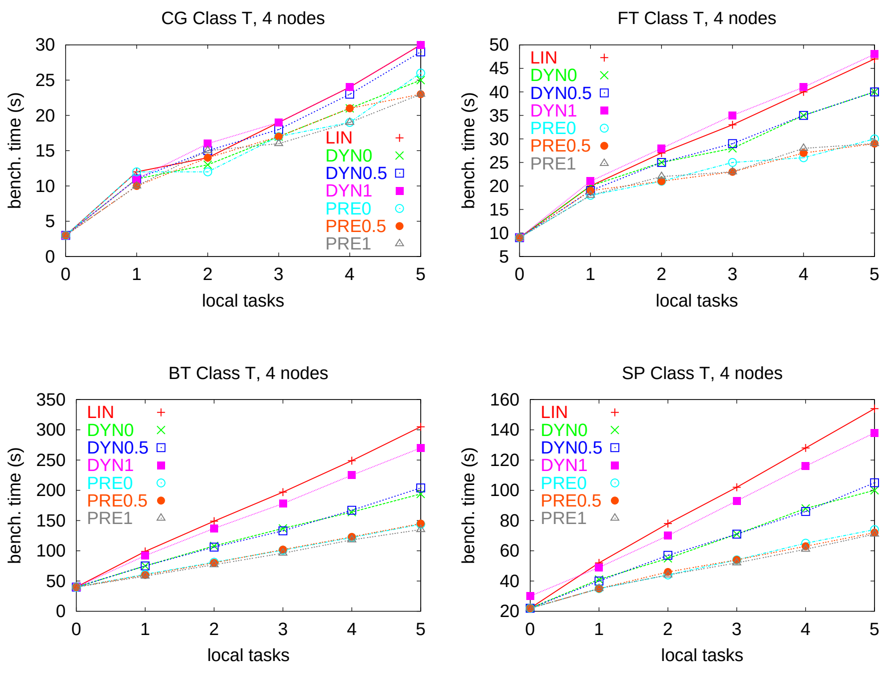

Figure B.7: CG, FT, BT and SP. Class T, 4 Nodes.

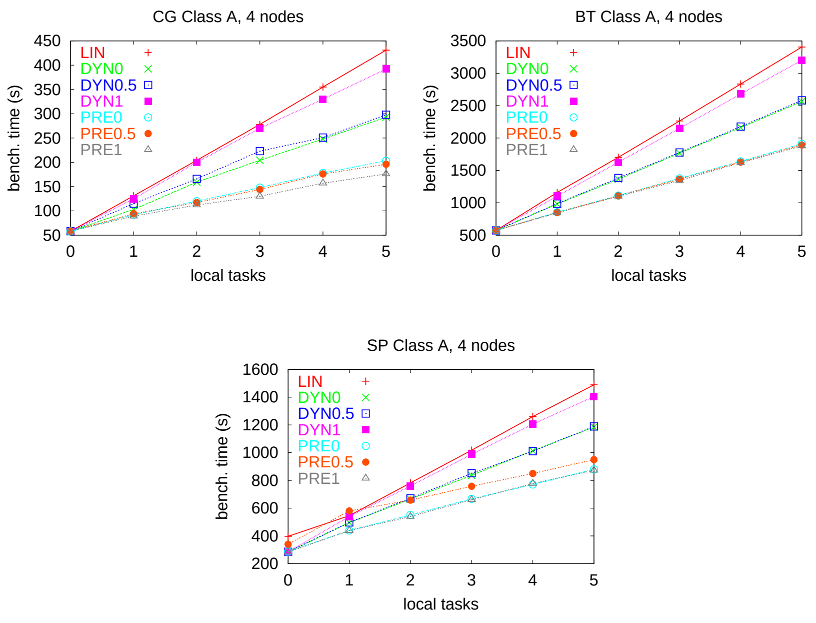

Figure B.8: CG, BT and SP. Class A, 4 Nodes.

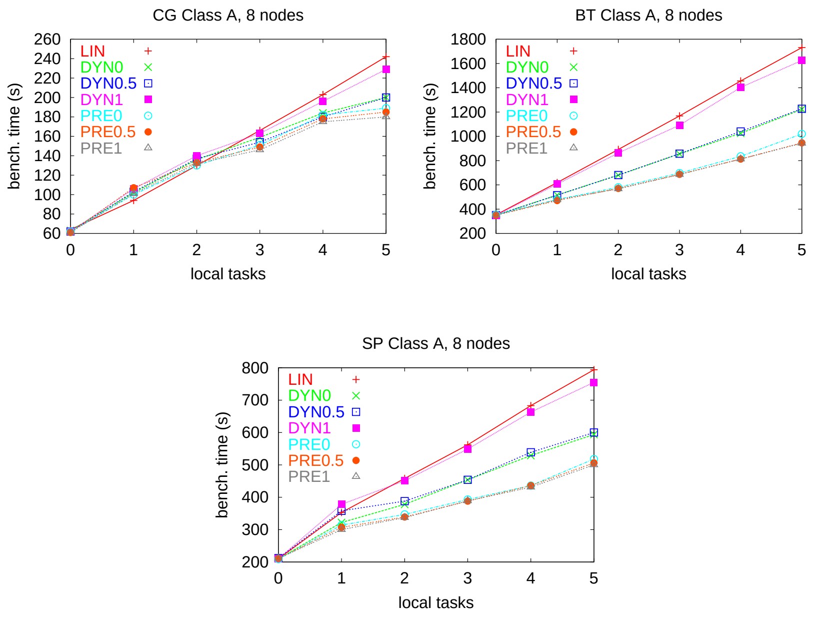

Figure B.9: CG, BT and SP. Class A, 8 Nodes.

Executing Together Various NAS Benchmarks

Results obtained by executing simultaneously IS, CG, BT and SP.

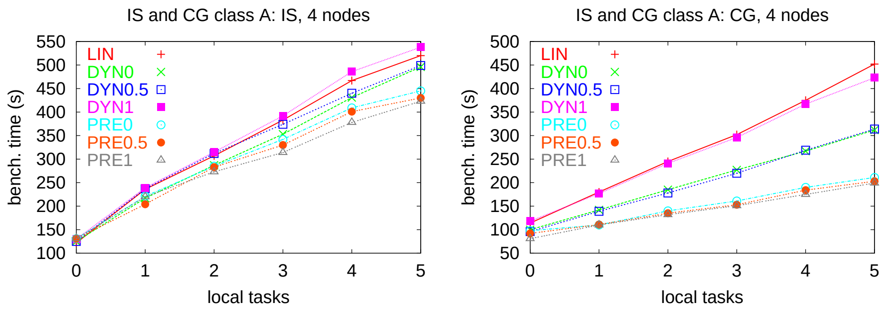

Figure B.10: IS and CG. (left) IS (right) CG.

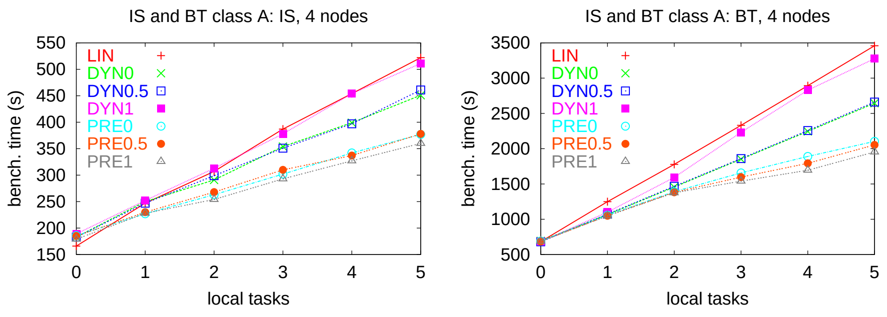

Figure B.11: IS and BT. (left) IS (right) BT.

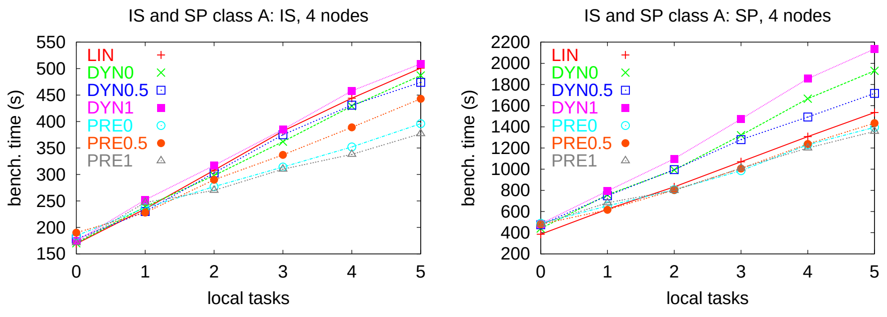

Figure B.12: IS and SP. (left) IS (right) SP.

Bibliography

[1] Casavant, T.L. and Khul, J. G.: A Taxonomy of Scheduling in General-Purpose Distributed Computing Systems. IEEE Transactions on Software Engineering, vol. 14 (2), pp. 141–154. 1988.

[2] Ousterhout, J.K.: Scheduling Techniques for Concurrent Systems. In Proceedings of the 3rd International Conference on Distributed Computing Systems, pp. 22–30. 1982.

[3] Ousterhout, J.K., Cherenson, A.R., Douglis, F., Nelson, M.N. and Welch, B.B.: The Sprite Network Operating System. IEEE Computer, vol. 21(2), pp. 23–26. 1988.

[4] Anderson, T., Culler, D., Patterson, D. and the Now team: A case for NOW (Networks of Workstations). IEEE Micro, pp. 54–64. 1995.

[5] Litzkow, M., Livny, M. and Mutka, M.: Condor - A Hunter of Idle Workstations. In Proceedings of the 8th International Conference of Distributed Computing Systems, pp. 104–111. 1988.

[6] Hagman, R.: Process Server: Sharing Processing Power in a Workstation Environment. In Proceedings of the 6th International Conference on Distributed Computing Systems, pp. 260–267. 1986.

[7] Russ, S., Robinson, J., Flachs, B. and Heckel, B.: The Hector Distributed Run-Time Environment. IEEE Transactions. on Parallel and Distributed Systems, vol. 9 (11). 1988.

[8] Crovella, M. et al.: Multiprogramming on Multiprocessors. In Proceedings of the 3rd IEEE Symposium on Parallel and Distributed Processing, pp. 590–597. 1994.

[9] Gupta, A., Tucker, A. and Urushibara, S.: The impact of operating system scheduling policies and synchronization methods on the performance of parallel applications. In ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems, pp. 120–132. 1991. Available from http://xenon.stanford.edu/~tucker/papers/sigmetrics.ps.

[10] Chandra, R., Scott, D., Verghese, B., Gupta, A. and Rosenblum., M.: Scheduling and page migration for multiprocessor compute servers. In Proceedings of the 6th International Conference on Architectural Support for Programming Languages and Operating Systems, pp. 12–24. 1994.

[11] Feitelson, D.G. and Rudolph, L.: Gang Scheduling performance benefits for fine-grain Synchronization. Journal of Parallel and Distributed Computing, vol. 16 (4), pp. 306–318. 1992.

[12] Feitelson, D.G. and Rudolph, L.: Coscheduling Based on Runtime Identification of Activity Working Sets". International Journal on Parallel Programming, vol. 23 (2). 1995.

[13] Feitelson, D.G.: A Survey of Scheduling in Multiprogrammed Parallel Systems. Research Report RC 19790 (87657), IBM T. J. Watson Research Center. 1994.

[14] Feitelson, D.G.: "Memory Usage in the LANL CM-5 Workload". Job Scheduling Strategies for Parallel Processing. LNCS, vol. 1291, pp. 78–94. 1997.

[15] Polze, A.: How to Partition a Workstation. In Proceedings of the IASTED International Conference Parallel and Distributed Computing and Systems. 1996.

[16] Arpaci, R.H., Dusseau, A.C., Vahdat, A.M., Liu, L.T., Anderson, T.E. and Patterson, D.A.: The Interaction of Parallel and Sequential Work-

loads on a Network of Workstations. In Proceedings of the ACM SIGMETRICS'95/PERFORMANCE'95, pp. 267–278. 1995.

[17] Arpaci-Dusseau, A.C., Culler, D.E. and Mainwaring, A.M.: Scheduling with Implicit Information in Distributed Systems. In Proceedings of the ACM SIGMETRICS'98/PERFORMANCE'98. 1998.

[18] Dusseau, A.C., Arpaci, R. H. and Culler, D. E.: Effective Distributed Scheduling of Parallel Workloads. In Proceedings of the ACM SIGMETRICS'96. 1996.

[19] Wong, Frederick C., Arpaci-Dusseau, Andrea C., Culler and David E.: Building MPI for Multi-programming Systems Using Implicit Information. 6th European PVM/MPI User's Group Meeting. LNCS, vol. 1697, pp. 215–222. 1999.

[20] Poovendran, R., Keleher, P. and Baras, J.S.: A Decision Process Analysis of Implicit Co-scheduling. In Proceedings of the IEEE International Parallel and Distributed Computing Symposium. Cancun (Mexico). 2000.

[21] Anglano, C.: A Comparative Evaluation of Implicit Coscheduling Strategies for Networks of Workstation. In Proceedings of the 9th International Symposium on High Performance Distributed Computing (HPDC). Pittsburgh, PA. 2000.

[22] Anglano, C.: A Fair and Effective Scheduling Strategy for Workstation Clusters. In Proceedings of the 2*nd* IEEE International Conference on Cluster Computing. Chemnitz (Germany). 2000.

[23] Petrini, F. and Feng, W.: Buffered Coscheduling: A New Methodology for Multitasking Parallel Jobs on Distributed Systems. In Proceedings of the International Parallel and Distributed Processing Symposium 2000, IPDPS'2000. Cancun (Mexico). 2000.

[24] von Eicken, T., Culler, D. E., Goldstein, S. C. and Schauser, K. E.: Active Messages: a Mechanism for Integrated Communication and Computation. In Proceedings of the 19th ISCA, pp. 256–266. 1992.

[25] Culler, D. E., Karp, R. M., Patterson, D. A., Sahay, A., Schauser, K. E., Santos, E., Subramonian, R. and von Eicken, T.: LogP: Towards a Realistic Model of Parallel Computation. In Proceedings of the 4th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, pp. 262–273. 1993.

[26] Martin, R. P., Vahdat, A. M., Culler, D. E. and Anderson, T. E.: Effects of Communication Latency, Overhead, and Bandwidth in a Cluster Architecture. In Proceedings of the 24th Annual International Symposium on Computer Architecture, pp. 85-97. 1997.

[27] Gropp, W., Lusk, E., Doss, N. and Skjellum, A.: A High-Performance, Portable Implementation of the (MPI) Message Passing Interface Standard. Parallel Computing, vol. 22 (6) pp. 789–828. 1996.

[28] Sobalvarro, P.G. and Weihl, W.E..: Demand-based Coscheduling of Parallel Jobs on Multiprogrammed Multiprocessors. In Proceedings of the IPPS'95 Workshop on Job Scheduling Strategies for Parallel Processing, pp. 63–75. 1995.

[29] Sobalvarro, P.G., Pakin, S., Weihl, W. E. and Chien, A. A.: Dynamic Coscheduling on Workstation Clusters. In Proceedings of the IPPS'98 Workshop on Job Scheduling Strategies for Parallel Processing. 1998.

[30] Geist, A., Beguelin, A., Dongarra, J., Jiang, W., Manchek, R. and Sunderam, V.: PVM: Parallel Virtual Machine - A User's Guide and Tutorial for Networked Parallel Computing. MIT Press. 1994.

[31] Geist, G.A, Kohl, J.A., Papadopoulos, P.M.: PVM and MPI: A Comparison of Features. Calculateurs Paralleles, vol. 8(2), pp. 137–150. 1996.

[32] Message Passing Interface Forum: MPI: A Message-Passing Interface Standard. www-unix.mcs.anl.gov/mpi. 1995.

[33] Message Passing Interface Forum: MPI-2: Extensions to the Message-Passing Interface. www-unix.mcs.anl.gov/mpi. 1997.

[34] Buyya, R.: High Performance Cluster Computing: Architecture and Systems, vol. 1. Prentice Hall. 1999.

[35] Buyya, R.: High Performance Cluster Computing: Programming and Applications, vol. 2. Prentice Hall. 1999.

[36] Tanenbaum, A.S.: Structured Computer Organization (4th ed.). Prentice Hall.1999.

[37] Solsona, F., Giné, F., Hernández, P. and Luque, E.: Synchronization methods in distributed processing. In Proceedings of the 7th IASTED International Conference. Applied Informatics (AI'99), pp. 471–473. 1999.

[38] Solsona, F., Giné, F., Molina, F., Hernández, P. and Luque, E.: Implementing and Analysing an Effective Explicit Coscheduling Algorithm on a NOW. 4th VECPAR Conference (VECPAR'2000). LNCS, vol. 1981, pp. 75–88. Porto (Portugal). 2000.

[39] Solsona, F., Giné, F., Hernández, P. and Luque, E.: Implementing Explicit and Implicit Coscheduling in a PVM Environment. 6th International Euro-Par Conference (Europar'2000). LNCS, vol. 1900, pp. 1165–1170. Munich (Germany). 2000.

[40] Solsona, F., Giné, F., Lérida, J. LL., F., Hernández, P. and Luque, E.: Monito: a Communication Monitoring Tool for a PVM-Linux Environment. 7th European PVM/MPI User's Group Meeting. LNCS, vol. 1908, pp. 233–241. Balatonfüred (Hungary). 2000.

[41] Solsona, F., Giné, F., Hernández, P. y Luque, E.: CMC: A Coscheduling Model for non-Dedicated Cluster Computing. In Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'2001). San Francisco. 2001.

[42] Solsona, F., Giné, F., Hernández, P. y Luque, E.: Coscheduling Policies: Simulation Vs Implementation. In Proceedings of 5th World Multi-Conference on Systemics, Cybernetics and Informatics (SCI'2001), vol. XV, pp. 579–584. Orlando (Florida). 2001.

[43] Solsona, F., Giné, F., Hernández, P. y Luque, E.: Predictive Coscheduling Implementation in a non-dedicated Linux Cluster. 7th International Euro-Par Conference (Europar'2001). LNCS, vol. 2150, pp. 732–741. Manchester (UK). 2001.

[44] Kohl, J.A. and Geist, A.: XPVM 1.0 User's Guide. Technical Report ORNL/TM-12981, Computer Science and Mathematics Division, Oak Ridge National Laboratory. 1995.

[45] Yan, J.C., Schmidt, M. and Schulbach, C.: The Automated Instrumentation and Monitoring Systems (AIMS) - Version 3.2 User's Guide. NAS Technical Report NAS-97-001. 1997.

[46] Heath, M.T. and Etheridge, J.A.: Visualizing performance of parallel programs. IEEE Software, vol. 8 (5), pp. 29–39. 1991.

[47] Information Networks Division. HP Co.: Netperf: A Network Performance Benchmark. http://www.netperf.org/netperf/NetperfPage.html. 1996.

[48] Miller, B.P., Hollingsworth, J.K. and Callaghan, M.D.: Environments and Tools for Parallel Scientific Computing. J.J. Dongarra and B. Tourencheau (eds.), SIAM Press. 1994.

[49] Du, X. and Zhang, X.: Coordinating Parallel Processes on Networks of Workstations. Journal of Parallel and Distributed Computing, vol. 46 (2), pp. 125–135. 1997.

[50] Zhang, X. and Yan, Y.: Modeling and Characterizing Parallel Computing Performance on Heterogeneous Networks of Workstations. In Proceedings of the 7th IEEE Symposium on Parallel and Distributed Processing, pp. 25–34. 1995.

[51] Atallah, M.J.; Lock, C.; Marinescu, D.C.; Siegel, H.J.; Casavant, T.L.: Co-scheduling compute-intensive tasks on a network of workstations: models and algorithms. In Proceedings of the 11th International Conference on Distributed Computing Systems, pp. 344 –352. 1991.

[52] Atallah, M.J., Black, C., Marinescu, D.C., Siegel, H.J. and Casavant, T.L.: Models and Algorithms for Co-Scheduling Compute-Intensive Tasks on a Network of Workstations. Journal of Parallel and Distributed Computing, vol. 16, pp. 319–327 (Special Issue on Scheduling and Load-Balancing). 1992.

[53] Devarakonda, M.V. and Iyer, R.K.: Predictability of Process Resource Usage: A Measurement-Based Study on UNIX. IEEE Trans. on Software Engineering, vol. SE-15(12), pp. 1579-1586. 1989.

[54] Ferrari, D. and Zhou, S.: An Empirical Investigation of Load Indices for Load Balancing Applications. In Proceedings of Performance '87, 12th International Symposium on Computer Performance Modeling, Measurement, and Evaluation. Amsterdam. pp. 515–528. 1987.

[55] Kunz, T.: The Influence of Different Workload descriptions on a Heuristic Load Balancing Scheme. IEEE Trans. on Software Engineering, vol. SE-17(7), pp. 725-730. 1991.

[56] Bailey, D., Barszcz, E., Barton, J., Browning, D., Carter, R., Dagum, L., Fatoohi, R., Fineberg, S., Frederickson, P., Lasinski, T., Schreiber, R., Simon, H., Venkatakrishnan, V. and Weeratunga, S.: The NAS Parallel Benchmarks. Tech. Report RNR-94-007. 1994.

[57] Parkbench Committee. http://www.netlib.org/parkbench.

[58] McMahon, F.H.: The Livermore Fortran Kernels: A Computer Test Of The Numerical Performance Range. Lawrence Livermore National Laboratory, Livermore, California, UCRL-53745. 1986.

[59] Dongarra, J.J.: Performance of Various Computers Using Standard Linear Equations Software. University of Tennessee. Computer Science Technical Report CS-89-85. 2002

[60] Mc. Voy, L. and Staelin, C.: lmbench: Portable tools for performance analysis. Silicon Graphics Inc. ftp://ftp.sgi.com/pub/lmbench.tgz. 1997.

[61] The Beowulf Project. http://www.beowulf.org.

[62] Linux Documentation Project. http://www.linuxdoc.org.

[63] Beck, M., Böhme, H., Dziadzka, M., Kunitz, U., Magnus, R. and Verworner, D.: LINUX Kernel Internals. Addison-Wesley. 1996.

[64] Carretero, J., García, F., De Miguel, P. y Pérez, F.: Sistemas Operativos. Una Visión Aplicada. McGraw-Hill. 2001.

[65] Tackett, J. and Gunter, D.: Utilizando Linux (2^{a} ed.). Prentice Hall. 1996.

[66] Card, R., Dumas, E., Mével, F. : The Linux Kernel Book. Wiley. 1998.

[67] Tanenbaum, A.S.: Operating Systems. Design and Implementation. Prentice Hall. 1987.

[68] Stevens, W.R.: Unix Network Programming. Prentice Hall. 1990.

[69] Postel, J.: RFC 768 - User Data Protocol. 1980.

[70] Postel, J.: RFC 793 - Transmission Control Protocol: Protocol Specification. 1981.

[71] Postel, J.: RFC 791 - Internet Protocol: Protocol Specification. 1981.

[72] Pakin, S., Lauria, M., and Chien, A.A.: High Performance Messaging on Workstations: Illinois Fast Messages (FM) for Myrinet. In Proceedings of the Supercomputing'95. 1995.

[73] Burger, D.C., Hyder, R.S., Miller. B.P. and Wood, D.A.: Paging Tradeoffs in Distributed-Shared-Memory Multiprocessors. Journal of Supercomputing, vol. 10(1), pp. 87–104, 1996.

[74] Baker, M.: Cluster Computing White Paper. IEEE Computer Society Task Force on Cluster Computing. http://www.dcs.port.ac.uk/~mab/tfcc/WhitePaper/final-paper.pdf. 2000.

[75] Bizard, S., Dalhem, S., Veigneau, S., Topol, B., Sunderam, V., Alund, A.: PGPVM: Performance Visualization Support for PVM (version 2.0). http://phalanstere.univ-mlv.fr/~sv/PGPVM2.

[76] Kohl, J.A. and Geist, G.A.: XPVM 1.0 User's Guide. Technical Report ORNL/TM-12981, Computer Science and Mathematics Division, Oak Ridge National Laboratory. 1995.

[77] Junius, M., Steppler, M., Büter, M. Pesch, D. et al.: CNCL: Communication Networks Class Library. ftp://ftp.comnets.rwth-aachen.de/pub/ comnets/CNCL.

[78] Setia, S., Squillante, M.S. and Naik., V.: The Impact of Job Memory Requirements on Gang-Scheduling Performance. Performance Evaluation Review. 1999.

[79] Setia, S.: The Interaction between Memory Allocation and Adaptive Partitioning in Message Passing Multicomputers. In Proceedings of the IPPS Workshop on Job Scheduling Strategies for Parallel Processing. 1995.

[80] Batat, A. and Feitelson, D.G.: Gang Scheduling with Memory Considerations. In Proceedings of the 14th International Parallel and Distributed Processing Symposium, pp. 109–114. 2000.

[81] Giné, F., Solsona, F., Hernández, P. and Luque, E.: Coscheduling Under Memory Constraints in a NOW Environment. 7th Workshop on Job Scheduling Strategies for Parallel Processing. LNCS, vol. 2221 pp. 41–65. 2000.

[82] Giné, F., Solsona, F., Hernández, P. and Luque, E.: MemTo: A Memory Monitoring Tool for a Linux Cluster. 8th European PVM/MPI User's Group Meeting. LNCS, vol. 2131, pp. 225–232. Santorini/Thera (Greece). 2001.