An evaluation of distributed scheduling algorithms within the DESPOT architecture

Daniel Andresen, Jeffrey Lebak, Ethan Bowker
Department of Computing and Information Sciences
234 Nichols Hall, Kansas State University
Manhattan, KS 66506

Abstract

Current systems for managing workload on clusters of workstations, particularly those available for Linux-based (Beowulf) clusters, are typically based on traditional process-based, coarse-grained parallel and distributed programming. The DESPOT project is building a sophisticated thread-level resource-monitoring system for computational, storage and network resources [2, 3]. In this paper we present an evaluation of several scheduling algorithms within DESPOT, our architecture for low-overhead, fine-grained resource-monitoring tools for per-process network and other resource usage. We also present experimental results show our performance using a genetic algorithm, the MOSIX default scheduler, and a range of parameters for the multi-facetted DESPOT algorithm. We also give several examples where our scheduling algorithm outperforms the MOSIX scheduler.

1 Introduction

Current systems for managing workload on clusters of workstations, particularly those available for Linux-based (Beowulf) clusters, are typically based on traditional process-based, coarse-grained parallel and distributed programming [13, 4]. In addition, most systems do not address the need for dynamic process migration based on differing phases of computation. The DESPOT project is building a sophisticated, thread-level resource-monitoring system for computational, storage, and network resources. We plan to use this information in an intelligent scheduling system to perform adaptive process/thread migration within the cluster. The DESPOT architecture was presented in a previous paper [2]. In this paper we discuss in detail our scheduling algorithms, including an iterative and genetic algorithm-based system, and compare their performance with the default MOSIX load-balancing scheduler.

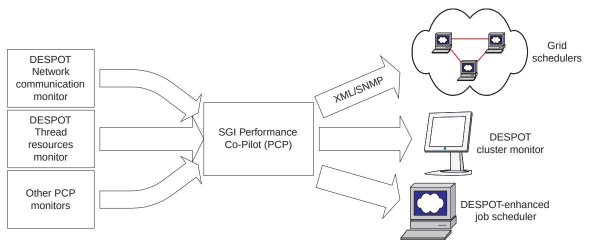
DESPOT Network communication monitor
DESPOT Thread resources monitor
Other PCP monitors
SGI Performance Co-Pilot (PCP)
XML/SNMP
Grid schedulers
DESPOT cluster monitor
DESPOT-enhanced job scheduler

Figure 1. Original DESPOT architecture.

Of central importance to process scheduling systems such as DESPOT is the collection of system performance monitoring data which is used by the scheduling algorithm. Since DESPOT incorporates per-process and per-thread data into the scheduling algorithm, it has demanding performance monitoring requirements. The original implementation of DESPOT was based on SGI's Performance Co-Pilot (PCP) to facilitate the collection of performance monitoring data and to provide an API for the scheduling algorithm to retrieve the data. Unfortunately, the PCP proved to have too much overhead which slowed the performance of the scheduling algorithms.

Our paper is organized as follows: Section 2 discusses related work. Section 3 covers the design of our system. Section 4 offers our experimental results, and finally, Section 5 offers our conclusions and final thoughts.

2 Background

We first briefly discuss Beowulf clusters, and how they differ from a network of workstations (NOW) [13, 1]. We then go on to discuss various monitoring and scheduling tools such as PCP.

Beowulf clusters The first Beowulf cluster computer was built by Thomas Sterling and Don Becker at CESDIS from 16 Intel 486DX4 processors connected by channel-bonded Ethernet. "The machine was an instant success and their idea of providing COTS (commodity off the shelf) base systems to satisfy specific computational requirements quickly spread through NASA and into the academic and research communities" [4, 13, 1]. Beowulf systems are marked by a reliance on COTS machines, publicly available software (in particular, the Linux operating system, the GNU compilers and programming tools, and the MPI and PVM message-passing libraries), and fast node interconnects (typically Ethernet or Myrinet). This software and hardware environment provides a robust system for today, with the ability to migrate easily to faster and better systems in the future due to standards like the Linux API, and message passing based on PVM[1] and MPI[2] [7, 5].

Performance monitoring tools Several commercial and open-source tools have been developed to monitor the performance of a large number of computers such as a typical computing cluster. In contrast with existing systems, which usually display information only graphically, the DESPOT project integrates performance monitoring with scheduling systems. In the following sections, we discuss open-source cluster-monitoring tools.

Several tools have been developed to monitor a large number of machines as stand-alone hosts as well as hosts in a cluster. These tools can be useful because they monitor the availability of services on a host and detect if a host is overloaded, but they do not generally provide performance-monitoring information at the level of detail needed to tune the performance of a Beowulf cluster. Examples of these systems are PaRe Procps [14], BWatch [12], Mon [15], Nocol [10], and Netsaint [6].

The *SMILE Cluster Management System* (SCMS) is an extensible management tool for Beowulf clusters [8]. SCMS provides a set of tools that help users monitor, submit commands, query system status, maintain system configuration, and more. System monitoring is limited to heartbeat-type measurements.

MOSIX is a popular platform for supporting distributed computing. It enhances the Linux kernel with cluster computing capabilities. In a MOSIX cluster, there is no need to modify applications to run in the cluster, or to link applications with any library, or even to assign processes to different nodes. MOSIX does it automatically and transparently. The resource sharing algorithms of MOSIX attempt to equalize the processing load of the machines in the cluster. However, the scheduling algorithms primarily consider the total CPU load and memory usage of each machine. Per process load and network load measurements are relatively minor factors [9].

MOSIX was useful for our experiments for two reasons. First of all, it provides a framework and an API for migrating processes between machines. Thus it is a convenient platform for the development of prototype scheduling algorithms. Secondly, the built-in MOSIX scheduling algorithm offers a baseline measuring stick for comparing our own scheduling algorithms.

3 System design

DESPOT allows for easy testing of different scheduling algorithms and heuristics. By using dynamically linked libraries (dlls) scheduling algorithms may be swapped in and out at the command line without recompilation of DESPOT. Adding another scheduler is simple – the program must be compiled as a dll and must follow DESPOT's dll interface. Currently, two scheduling libraries are in use: the DESPOT sched-

[1]Parallel Virtual Machine

[2]Message Passing Interface

uler and the genetic algorithm scheduler. Each uses the same heuristic back end, which is currently a set of functions and macros that determines the effectiveness of a scheduling. Eventually, DESPOT will allow for dynamic linking of heuristics using the same method as schedulers. With DESPOT's flexible architecture, new algorithms can be tested with no code rewriting and no recompilation.

DESPOT begins by initializing scheduler dll functions based on a command line parameter that indicates which scheduler to use. Next, it sets up the process monitoring tool, *distop*, on each node in the cluster. After initialization, DESPOT repeats the following: First, it accesses the shared memory where each distop client places its process information and organizes it into an array of hash tables. The process information structure and a collection of cluster parameters (number of nodes, number of processors on each node, etc) are passed in a single structure to the chosen scheduler. The scheduler produces mapping of processes to nodes in the form of a list to send to each node. DESPOT uses RPC to forward each node's migration list to its distop server. Once it receives the RPC call, distop server uses MOSIX to migrate the specified processes to their new location. Returned from the RPC call is an indication of success or failure for each attempted migration. Using the returned list, DESPOT updates its process information structure to reflect successful migrations. Finally the loop ends by sleeping a set amount of time.

The *distop* package is intended to be used in a distributed computing environment to provide a lightweight means of collecting the overall system usage. The package is divided into three applications: a daemon, a server, and a client. The daemon program runs in the background collecting current system information, comparing it to previously collected system information, and storing the information as a ratio of difference over time. The server, upon request from the client, collects non-ratio system information, as well as obtaining the stored information from the daemon, and returns it to the client. Further details can be found in [2].

The DESPOT scheduling algorithm Using a list of active processes provided by DESPOT, the DESPOT scheduler iterates through all possible combinations of placing p processes on n nodes. The combinations are formed through simple binary (for 2 machines) logic on the number of processes and nodes. Consider a 2 node cluster with one process on each of them. A binary combination of values from 0 to $2\hat{2}$ will be 00, 01, 10, 11. (In general, total combinations equals $n\hat{p}$.) The first combination suggests assigning both processes to node 0, while the third combination considers the first process to be on node 1 and the second on node 0.

Once it obtains a particular combination, the scheduler may then determine how fit this combination is using a heuristic developed specifically for DESPOT. A so called 'happiness' value is computed for every process based on the current suggested configuration and a sum of all process happiness values is the happiness of the current configuration. A process is considered to be 'happy' if it receives the total amount of CPU, memory, and network bandwidth it requires. Desired resource usage is estimated from usage during the previous time interval. The combination with maximum total happiness is chosen as the desired scheduling for the next time period. In calculating happiness, if a process is using less than its 'fair share' of CPU or network bandwidth then its corresponding happiness value is equal to 1.

Happiness (H_{ij}) for a particular process (i) on machine (j) is

$$H_{ij} = \alpha H_{CPU_{i,j}} + \beta H_{NET_{i,j}}$$

where α and β are typically both 0.5, meaning that the weighting between network and CPU is equal. $\alpha = 1.0$ and $\beta = 0$ is equivalent to a load-balancing algorithm such as that used by MOSIX. We define

$$H_{CPU_{i,j}} = \begin{cases} 1 & \text{if } CPU_{usage} < CPU_{avail} \\ \frac{CPU_{avail}}{CPU_{usage}} & \text{otherwise} \end{cases}$$

$$H_{NET_{i,j}} = \begin{cases} 1 & \text{if } NET_{usage} < NET_{avail} \\ \frac{NET_{avail}}{NET_{usage}} & \text{otherwise} \end{cases}$$

where

CPU_{usage} = % system and user time used by a process

$$CPU_{avail} = \frac{\text{processors on a node}}{\text{CPU procs assigned}}$$

$NET_{usage} =$ bandwidth consumed by a process

$$NET_{avail} = \frac{\text{network capacity}}{\text{NET procs assigned}}$$

Total happiness simply becomes $\sum_{i=1,j=1}^{p,n} H_{ij}$

In addition to CPU and network, memory is considered using a simple heuristic. If the current configuration would cause thrashing on any given node (i.e. the memory of all processes scheduled on a node is greater than the memory of the node) that combination is skipped and scheduling continues. Also, a slight penalty to happiness is incurred for every process that is moved. This helps to prevent scheduling in which a pair of identical processes on identically scheduled nodes would be swapped back and forth with no performance gain. After determining an optimal scheduling, a list of processes to migrate is created and returned from the scheduler.

Techniques for improving scheduler performance The DESPOT scheduler looks at all possible combinations of processes to nodes. This works for very small numbers of processes and nodes only. In order to prevent an explosion in the number of combinations to check, the DESPOT scheduler orders processes from least to most 'happy' where happiness is a measure of well a particular process fits in the proposed scheduling. Moving the most unhappy processes first reduces scheduling overhead significantly. This allows for an incremental scheduling approach. Since the cluster computing environment is dynamic, scheduling small chunks of processes yields better performance if monitoring and scheduling overhead can be kept at a minimum.

Genetic algorithms Efficiently scheduling p processes over n nodes requires special consideration. One cannot simply evaluate every possible combination of processes to nodes in a reasonable amount of time for large p and n. Using an exploratory search method such as a genetic algorithm, a satisfactory scheduling of the processes may be achieved in a realistic time period. A genetic algorithm has a population of solutions to the problem in question. The population in this case is a set of possible process-to-node mappings. Each of these population members or chromosomes is given a fitness, or value measuring the effectiveness of a chromosome. New populations are formed by selectively manipulating the current population. Better solutions are preserved while poor solutions are replaced. This process is repeated until certain stopping conditions are met, such as a set number of iterations reached or a certain solution fitness value exceeded.

PGAPack, a C language genetic algorithm library, is used in the DESPOT architecture to implement one of the two main scheduling algorithms [11]. Fitness values are calculated based on happiness of a current configuration in exactly the same way as described earlier.

Apart from the way each produces a set of combinations, the GA and DESPOT schedulers are nearly identical. Both have the same dll function and parameter interface, return the same list of processes to migrate and evaluate happiness the same way. Another difference in the DESPOT scheduler is that it sorts processes by happiness and then only considers the first few for migration. While this is not currently done in the GA scheduler, such functionality could easily be added in a later version.

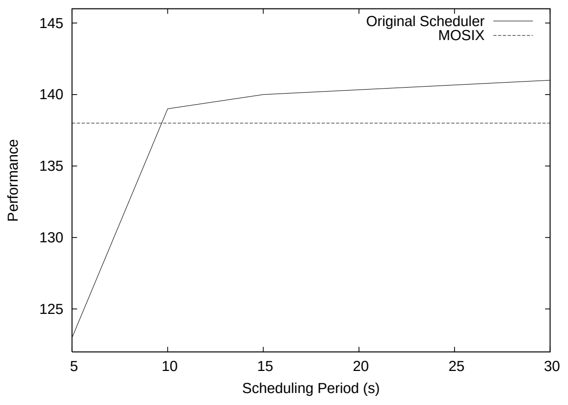

Figure 2. MOSIX vs. DESPOT (network-biased mixed processes, higher is better).

4 Experimental Results

Tests were conducted on a four node cluster whose hardware consists of two servers with dual Athlon MP 2000+ (1.7 GHz), 2 GB RAM, and 100 Mb Ethernet, and two servers with dual Athlon MP 1800+ (1.5 GHz), 2 GB RAM, and 1 Gb Ethernet. All systems run Debian Linux kernel version 2.4.22 and Mosix version 1.10.1.

Overhead of the data collection process, distop is minimal during a CPU intensive test. CPU usage sits around 1.5 to 2 percent. A mixed mode test yielded more overhead in the range of 15-20 percent CPU usage. This is due presumably to the overhead associated with packet sniffing within distop, as well as the MOSIX communications overhead.

Test cases Our performance test consists of 12 processes with 2 threads each. One thread is dedicated to computation, the other communication. The computational thread calculates a Fibonacci number of a specified size and then sleeps. The communication thread sends a message of a specified size taken from a 3 MB text file. To make this situation network intensive, computational threads calculate a Fibonacci number of size 40, which takes about 20-30 seconds each on our system. Once the computational threads yield, the communication threads begin sending the entire 3 MB text file to a predetermined node. Each thread sends the same message 10 times in a row with very little pause in between. After sending its 10 messages, the communication thread yields and the procedure repeats for 4 minutes. We discuss two variants below.

- *Mixed CPU and Network test* A similar testbed is used to simulate a mixed computation and communication environment. The network intensive test is modified so that the communication thread yields after each 3 MB file it sends. By keeping the Fibonacci number the same, a good balance was struck between computation and communication. At any given moment half of the processes are doing computation and half are communicating, testing the schedulers on their ability to adapt in a dynamic environment.

- *CPU intensive test* To simulate CPU intensive computing with the same test program, the following modifications were made: Instead of using 3 MB messages, a single 4kB message is sent before yielding. To increase computational time, a Fibonacci number of size 44 was calculated.

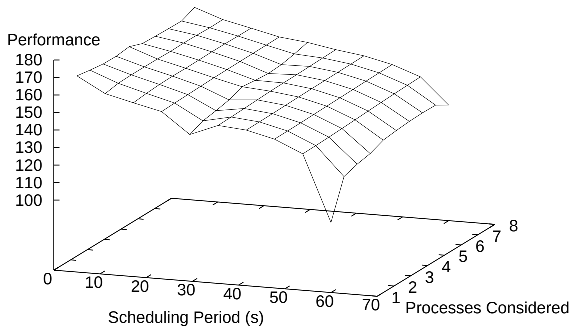

Figure 3. Effects of scheduling frequency on DESPOT scheduling algorithm, CPU-bound tasks, higher is better.

Scheduling times for the two algorithms (general and GA) were nearly constant for the GA at approximately 200 ms., and ranged from 123 μs for moving one process to 16,000 μs. for moving up to six processes. The results for more frequent scheduling and considering the movement of various numbers of processes can be found in Figures 3, 4, and 5. As might be expected, in general scheduling more frequently produced better results. Surprising, changing the number of processes to be moved usually did not substantially effect the results. Table 1 gives the relative performance numbers for the two algorithms. In

As the benchmark processes were more and more biased towards network communication, the DESPOT algorithm began to dominate the CPU-centric MOSIX algorithm, as shown in Figure 2. Changing the weights within the DESPOT algorithm to bias the algorithm itself towards making scheduling decisions based on network activity led to significantly (50%) worse performance with $\alpha = 0$.

5 Conclusions and future work

In this paper we have presented our system for monitoring and scheduling processes within a Beowulf

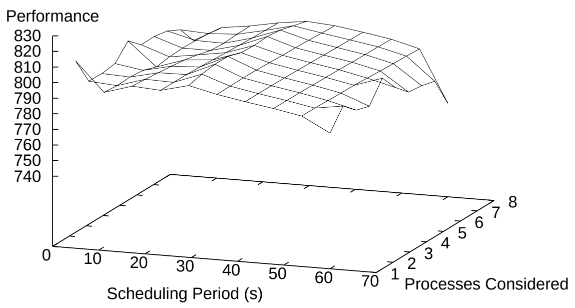

Figure 4. Effects of scheduling frequency on DESPOT scheduling algorithm, network-bound tasks, higher is better.

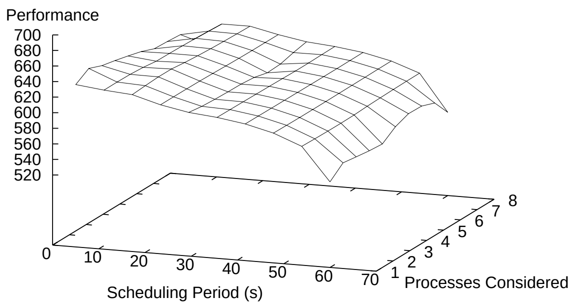

Figure 5. Effects of scheduling frequency on DESPOT scheduling algorithm, CPU- and network-bound tasks, higher is better.

Scheduling	Standard Algorithm			Genetic Algorithm		
Period (s)	CPU	Mixed	Net	CPU	Mixed	Net
5	170	646	792	172	464	n/a
10	168	641	814	172	544	628
15	162	686	803	170	687	681
30	159	605	825	152	584	760
60	126	537	771	140	578	682

Table 1. Benchmark execution times (work units - higher is better).

cluster. We have also discussed our multi-facetted scheduling algorithms, and demonstrated good performance over a variety of test cases, including surpassing that of MOSIX in some cases.

We plan to extend the system and viewing application to a hierarchical organization to increase its scalability over the current, centralized system. We are also working to achieve the ability to monitor individual Java threads through identifying their mapping to kernel-level threads.

Acknowledgments This material is based in part upon work supported by the National Science Foundation under award numbers ITR-0082667 and ACS-0092839. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

[1] T. E. Anderson, D. E. Culler, and D. A. Patterson. A case for NOW (Networks of Workstations). *IEEE Micro*, 15(1):54–64, Feb. 1995.

[2] D. Andresen, S. Kota, M. Tera, and T. Bower. An ip-level network monitor and scheduing system for clusters. In *Proceeding of the 2002 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'02)*, Las Vegas, June 2002.

[3] D. Andresen, N. Schopf, E. Bowker, and T. Bower. Distop: A low-overhead cluster monitoring system. In *Proceedings of the 2003 International Conference on Parallel and Distributed Processing Techniques*

and Applications (PDPTA'03), pages 1832–1836, Las Vegas, NV, June 2003.

[4] D. Becker. *The Beowulf project*, July 2000. http://www.beowulf.org.

[5] M. P. I. Forum. MPI: A message-passing interface standard. Technical Report UT-CS-94-230, Department of Computer Science, University of Tennessee, Apr. 1994. Wed, 7 Jul 99 23:57:06 GMT.

[6] E. Galstad. *Netsaint Network Monitor*. http://www.netsaint.org/.

[7] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. *PVM 3 User's Guide*.

[8] P. R. Group. *SCMS Web Page*. Kasetsart University. http://smile.cpe.ku.ac.th/.

[9] *The MOSIX Project Homepage*. http://www.mosix.cs.huji.ac.il/.

[10] Netplex Technologies Inc. *Nocol System Monitoring Tool*. http://www.netplex-tech.com/software/nocol.

[11] Pgapack, Jan. 2004. www-fp.mcs.anl.gov/CCST/research/reports_pre1998/comp_bio/stalk/pgapack.html.

[12] J. Radajewski. *bWatch - Beowulf Monitoring System*. University of Southern Queensland, Apr. 1999. http://www.sci.usq.edu.au/staff/jacek/bWatch/.

[13] T. Sterling, D. Savarese, D. J. Becker, J. E. Dorband, U. A. Ranawake, and C. V. Packer. BEOWULF: A parallel workstation for scientific computation. In *Proceedings of the 24th International Conference on Parallel Processing*, pages I:11–14, Oconomowoc, WI, Aug. 1995.

[14] F. Strauss and O. Wellnitz. *Procps Monitoring Tools*. Technical University Braunschweig, June 1998. http://www.sc.cs.tu-bs.de/pare/results/procps.html.

[15] J. Trocki. *Mon System Monitoring Tool*. Transmeta Corporation. http://www.kernel.org/software/mon/.