Extending Beowulf Clusters to the Desktop

By George Hamer [1] and Daniel R. Steinwand[2]

Open-File Report 03-432

U.S. Department of the Interior
U.S. Geological Survey

[1] Department of Computer Science, South Dakota State University, Brookings, SD 57007.
[2] USGS, EROS Data Center, SAIC, Sioux Falls, SD 57198-0001. Work performed under U.S. Geological Survey contract 03CRCN0001

Contents

Illustrations

Abstract

Existing Beowulf clusters are normally limited to the actual number of compute nodes physically connected to the cluster's network switch. At some point, it may become necessary to extend the size of the cluster beyond the capacity of the switch. By using existing computers on a campus network, one can extend the computing power of the cluster by including these machines during times they would normally be idle. We have done a survey of methods for extending clusters and devised a parallel solution to a computationally complex all-possible-regressions algorithm and tested it by using idle computing power on the campus network.

Key Words

Parallel Processing, Beowulf Clusters, High-Performance Computing

Introduction

Most companies have many computers that for the most part are underutilized a large percentage of the time. After normal working hours (or during work breaks), this represents a tremendous computing resource. Exploiting this idle resource makes sense as an attempt to increase the available computing power in a Beowulf cluster (Sterling and others, 1999). Idle computers can be exploited during normal working hours by polling the CPU and checking for load. As long as this load is below a predefined threshold, the CPU can be used.

To test this idea, we modified an all-possible-regressions algorithm to run as a parallel algorithm. The original algorithm exhibits $O(2^n)$ growth characteristics, which limits its use for large numbers of variables. This algorithm was tested using 32 variables on a Dell 530 Rambus 1.6 GHz dual processor Xeon machine with 1 Gigabytes of main memory. The test ran for approximately 8 weeks on a single machine.

The goal of this work was to show that it is possible to create a parallel implementation of a computationally complex algorithm with minimal coding using idle CPUs. Additionally, this task should be performed with minimal software installation on users' workstations. The tests use the Parallel Virtual Machine (PVM) software that is installed on each workstation.

The rest of the paper is organized as follows. In section 2 we examine techniques for extending the cluster to the desktop. In section 3 we describe the parallel algorithm used in the testing of the concept. Section 4 gives the performance results, and section 5 suggests future work.

Clustering Methodologies

Many methods can be used to extend a cluster. These range from the use of PVM (www.csm.ornl.gov/pvm), one of the simplest methods, to the use of the Globus Toolkit (www.globus.org), a more complex method. The Condor system (www.cs.wisc.edu/condor/description.html) from the University of Wisconsin is a middle-ground solution that uses an existing underlying network and offers scheduling and authentication services. Using PVM places most of the work on the programmer to allocate and deallocate compute nodes, while the Globus toolkit allows the grid designer to hide this complexity.

The Message Passing Interface (MPI) (www.lam-mpi.org) was not utilized in this study, even though it is a widely used parallel programming methodology, since at the time of this writing a version for Linux that supported the dynamic allocation of nodes was not available. This is to be supported in MPI version 2.

Parallel Virtual Machine – PVM

PVM was one of the first technologies for performing parallel computations on a cluster of workstations. PVM was developed at Oak Ridge National Laboratory in 1989 by Vaidy Sunderam and Al Geist. The first public release occurred in 1991, with the help of the University of Tennessee. It has been continuously updated since that time. Version 3.4.4, which was released in late 2001, was used for this study.

PVM is designed to create a parallel computer that runs on heterogeneous hardware and supports multiple operating systems. This can be used to create a loosely coupled supercomputer. Each node runs a daemon process that allows the remote execution of tasks on that node. PVM uses the message-passing model to allow distributed parallel computing.

The installation process is well documented, and the installer needs to make only two decisions. One is whether to use the remote commands rsh, rlogin, and rcp or their secure replacements ssh, slogin, and scp. This decision can be based on the security level needed. The second is how to support a single user on multiple machines. User accounts can be handled in one of two ways. The Network File System (NFS) can be used to mount all users' directories on all machines or, as an alternative, a single "pvm_user" account can be created on each machine. NFS, however, has the disadvantage of increasing network traffic when performing file reads and writes. If one is careful to write all temporary output to the local /tmp directory, this traffic can be minimized.

Condor

The University of Wisconsin's Computer Science Department has been involved in distributed resource sharing for more than 15 years. Dr. Myron Livny has lead his research team in investigating High Throughput Computing (HTC), which is defined to be the number of floating point calculations executed per year. High-performance computing is concerned with the number of floating point operations per second, but HTC concerns itself with problems that may have compute times measured in months.

Computers on the University of Wisconsin's main campus run the Condor system and allow idle computers to be used in distributed computing problems. Currently more than 1000 computers are linked together through software that provides 650 CPU days of computing power to campus researchers on an average day (www.globus.org/about/faq/general.html). Condor can be used to manage an existing Beowulf cluster or extended to search out idle computers that can be added to the computing pool.

Condor has an extensive set of features and provides flexibility for both the users and the owners of machines. The two most important features are remote procedure calls and check-pointing. The remote procedure calls feature allows a master node to execute tasks on a remote machine using a set of primitive calls. Check pointing allows parts of a parallel job to be executed on one machine and then moved to another if the CPU becomes busy. The job then resumes at that stopped point on the new processor.

Globus

Condor was designed to share resources within a single network and works best with homogeneous hardware. Conversely, the I-way project was started by Argonne National Laboratories (ANL), the University of Illinois at Chicago, and the National Center for Super-Computing Applications (NCSA) as a way to connect 10 networks into 1 computing environment. Ian Foster and Steven Tuecke of ANL later received funding to begin the Globus Toolkit project.

The Globus Toolkit is characterized by the following set of protocols and tools:

1. Globus Resource Allocation Manager (GRAM) – allows users to submit grid-enabled jobs to the cluster(s).
2. Gridftp – provides support for high-performance, secure, and reliable data transfers.
3. GSI-enabled SSH – allows authorized users and administrators to login to the front-end node of a cluster.
4. Grid Resource Information Services (GRIS) – collects and publishes information about grid resources.
5. Grid Security Infrastructure (GSI) – provides a single sign-on system for authentication.
6. Monitoring and Discovery System (MDS) – an extension to the Lightweight Directory Access Protocol (LDAP) that provides a uniform method for accessing grid nodes.
7. Heartbeat monitor – allows system administrators or normal users to evaluate the overall state of the grid at a high level.

Each component of the toolkit provides the user with an API written in C. In most cases, command line tools and Java classes are provided for the programmer.

The Globus developers hope to create for grid computing what Apache has created for Web servers. Their plan for the future is to have a production-ready system derived from their current research project; that system is currently in the beta testing stage. Globus is currently being used in test environments at TerraGrid, the Department of Energy's Science Grid, the European DataGrid, and GriPhyN (the grid physics network).

Other Solutions

OSCAR (www.openclustergroup.org/), Grid-in-a-Box (www.ncsa.uiuc.edi/ TechFocus/Deployment/GiB/overview.html), and Condor-G (Frey and others, 2001) are other open source distributed computing solutions that are currently available. Parabon, Entropia, and United Devices have commercial solutions available for users who wish to leverage multiple computers and do not have access to them. These solutions were not investigated but were left as future work.

All Possible Regressions Algorithm

The all-possible-regressions algorithm (Neter and others, 1996), typically used in the experimental design process, is used to find potentially good subsets of variables from a larger set of experimental variables. It lets the designer ask the question: Are all predictor variables needed in this experiment or is a subset adequate?

Consider, for example, that N variables were collected during an experiment. The all-possible-regressions algorithm could be used to fit all 2^N combinations of variables and identify the best (based on some criteria) combinations of variables for further analysis.

Our algorithm will consider datasets with up to 32 variables—the largest dataset the algorithm can handle as implemented. Each combination of variables is fit with a least squares routine implemented using QR factorization and Householder transformations (Golub and Van Loan, 1989). The goodness-of-fit criterion implemented are the SSE_p and MSE_p Criteria (Neter and others, 1996).

Parallel Regression Algorithm

The conversion to a parallel algorithm went quickly since the original serial algorithm exhibits a high degree of parallelism. This means that the work can be subdivided into many subtasks without the need for much communication between nodes. In essence, each node performs 1/Nth of the work. The basic parallelization is as follows:

1. Check to see if I am the master node; if true spawn N-1 new jobs; otherwise proceed to step 2.
2. Determine the group id number of the node.
3. Open an output file in /tmp, avoiding the need to write to an NFS mounted directory.
4. For each iteration of the main loop, check to see if the remainder of the loop control variable divided by the number of hosts is my group id.

If this is true, perform the regression for this value of the loop control variable; otherwise, another node will perform this calculation. Write the answer in the temp file.

5. Close the temporary file.
6. If I am the master node, gather all the temp files and merge into a final result file.

Test Results

To test for correctness, we ran the serial version of the algorithm using combinations of up to 4 variables (from the 32 variables available). Then the same run was done with the parallel version using four machines. These tests were run on Pentium 4 processors running at 1.8 GHz with 256 megabytes of RAM memory using the RedHat Linux version 7.3 operating system running kernel 2.14.18-10. The serial algorithm required 31 minutes to complete, and the parallel version required approximately14 minutes of computation.

After verifying the results from the output of the previous test, we ran the next test using combinations of 4 variables on 8 and 15 nodes, respectively. The results are shown in table 1.

	Elapsed Time	Speed-up
1 node	30:55	0
4 nodes	13.37	2.3
8 nodes	13.22	2.3
15 nodes	13:19	2.3

Table 1. Run-times for combinations of up to 4 variables and 15 nodes.

The results seemed disappointing until it was discovered that the original algorithm requires 13 minutes to iterate over all 2^{32} values in the loop. Therefore, 13 minutes is the minimum time required for the algorithm to execute in all cases.

Extending the amount of work (computations) led to increased speed-ups. Running a series of tests using combinations of up to eight variables in the regression gave the following results shown in table 2.

	Elapsed Time	Speed-up
1 node	746m16.314s	0
8 nodes	303m5s	2.46
15 nodes	58m44.3s	12.71
30 nodes	48m42s	15.33
40 nodes	27m14.2s	27.40
76 nodes	21m17.4s	35.05

Table 2. Run-times for combinations of up to 8 variables and 76 nodes.

As a test of heterogeneous hardware platforms, the program was run using an 18-node Beowulf cluster running PIII 500 MHz machines and 15 Pentium 4 1.8 GHz machines for a total of 33 CPUs. The results showed the difference in processing speeds between the two different types of processing nodes. The P4 nodes finished their task in 30 minutes, while the PIII nodes required 80 minutes to complete the task. Two important results should be noted here: first, the ease with which new nodes can be added to the cluster through the use of PVM system calls, and second, the need to load-balance the algorithm.

The final test was to use 76 P4 machines on the South Dakota State University campus network and consider all 32 variables. This test required 111 hours to run to completion. It was hoped that the run time would be reduced to less than 63 hours so that the program could be run over the weekend (5 p.m. Friday thru 8 a.m. Monday) when all computers would be idle in a workplace. With the addition of more machines, this time constraint can probably be met.

As a test of creating quick parallel programs for computationally intense problems, this experiment is considered a success. It showed that with minimal coding and hardware configuration complexity it is possible to leverage the use of idle CPUs with very little effort. The only software needed on user workstations was the installation of the PVM daemon.

Future Work

The next step is to use the SETI@home model (setiathome.ssl.berkeley.edu) to extend the calculation to occur whenever a CPU is idle. When a node goes idle and a screen saver activates, the node will make a request of the server to be given a work packet to calculate. This work is currently in progress and is seen as a natural way to load balance the algorithm. Each idle computer will be sent a work packet and then report its results back to the server when completed.

References

Frey, James; Tannenbaum, Todd; Foster, Ian; Livny, Miron & Tuecke, Steve. Condor-G: A computation management agent for multi-institutional grids. In Proceedings *of the Tenth IEEE Symposium on High Performance Distributed Computing (HPDC),* pages 7-9, San Francisco, California, August 2001

Golub, Gene H. and Van Loan, Charles F., Matrix Computations, 2nd Ed., John Hopkins University Press, Baltimore, 1989.

Neter, J., Mutner, M., Nachtsheim, C., Wasserman, W., Applied Linear Statistical Models, 4th Ed., WCB/McGraw-Hill, 1996.

Sterling, T., Salmon, J, Becker, D., & Savarse, D., "How to Build a Beowulf: A Guide to the Implementation and Application of PC Clusters (Scientific and Engineering Computation)", MIT Press, 1999.