Paper 230-29

Financial COWs: Using SAS to Manage Parallel Clusters for Simulating Financial Markets

Haftan Eckholdt, DayTrends, Brooklyn, New York

ABSTRACT

Using SAS to manage and run a LINUX / WINDOWS-NT based cluster of workstations (COW) to simulate a parallel processor puts the power of supercomputing well within the reach many analysts. The criteria for using COW's will be discussed in terms of their job symmetry and intermediate solution timing. Special problems, and their solutions are discussed including the installation and maintenance of COWs especially in terms of work environments and hardware constraints. Using layers of file monitoring, job distribution, and data analysis, and data visualization, SAS can provide clever and cost effective paths to COW management and usage [X command, %MACRO, DATA, STAT]. Parsing the problem, and distributing it symmetrically across the COW and executing the randomized solutions are discussed in the context of an example from computation finance: simulating financial markets to backtest investment strategies. Backtests are the first line of investigation on quantitative investment models. Developing adequate out of sample databases is an enormous computational task that few laboratories can approach. Using SAS to manage a cluster, and simulate financial markets provides analysts with insights to quantitative strategies that could never be accessed otherwise.

INTRODUCTION

Research in disciplines with high density data have been limited by the state of hardware and software in supercomputing. Examples include simulations, model building, and graphics in weather, finance, and genetics. The barriers to supercomputing have traditionally been costs associated with hardware and programming. Using a cluster of workstations (COW) to simulate a parallel processor puts the price of a teraflop of hardware under $1,000,000. More importantly, perhaps, is that the scalability of a COW -- 50 gigaflops for $50,000 -- well within most research budgets.

The growing history on parallel processing can best be accessed through the Computer Society of Institute of Electrical and Electronics Engineers (IEEE) web site [http://www.computer.org/parascope/] which also sponsors the International Symposium on Parallel Architectures, Algorithms and Networks. The state of the art can be summarized by the simple return of 66,500 URL's from the www.google.com search for " cluster of workstations" as of September 2000. Readers should refer to the brief bibliography for recent text books on the topic.

The real criteria for using COW's are the nature and eligibility of the problem under study. Approaches to parallel processing, in bioinformatics, are classified by the instruction set (single versus multiple) and data set (single versus multiple) to yield four types of parallel processors SISD/MISD/SIMD/MIMD (Buyya, 1999b; Foster, 1994; Wilkinson and Allen, 1999). The distinction between these four types of processing are not always obvious, so it seems that using SAS to manage and run the COW gives the user access to all four approaches, depending upon which one is logically possible, and then, which one is most efficient. In general, questions that lend themselves to parallel processing on a COW are those that can be broken down into parts or jobs. " Granularity" is what COW programmers use to describe just how small, how similar, and how many jobs can be parallel-ized from the overall problem. It does make coding much easier if jobs do not depend on eachother.

Jobs that are the same in terms of analysis would require the same instruction set (single instruction) copied to each workstation, while jobs that are analytically different require different instruction sets (multiple instruction). Likewise, the data demands of the jobs will determine whether the same data set is used (single data) by all workstations or different data sets are used (multiple data).

Most importantly, there must be at least one job for each processor / workstation, no matter how large or small the individual job may be. Those problems that benefit most from parallel processing COWS are those that have either: (1) a large number of symmetric jobs, or (2) jobs that benefit from eachothers' intermediate solutions. Problems that fit criteria (1) are problems that can be distributed rapidly and evenly across the COW, whereby no processor is idle, and each job solution is posted back to the server. The type (1) problems are those that have what might be considered a large set of non-dimensional jobs. This long list of jobs must be completed for the problem to be solved, but there is no logical or necessary order in which the jobs must be completed. In this example, a COW outruns a SoW (single workstation) in that the throughputs (area/access to critical devices: Hard Drive, RAM, CPU) have been radically expanded, and repeated in parallel. In some cases, the resulting throughput of a COW exceeds

that of a similar capacity supercomputer. Type (2) problems are problems that benefit from the " folding of time" that can occur in a parallel process. For instance, some solutions might occur faster with access to solutions from jobs later in the process, something that would never occur under traditional computing environments – SoW/supercomputer.

This talk, like most books on the topic, will fall back on demonstration to explain the challenge of problem parsing. The following sections will provide a more detailed description of the process of setting up and using a COW including hardware designs, and cluster management, and job execution. Readers should note that the overall approach to cluster management and job execution is a patent pending technology, although this paper provides specific approaches to cluster management and job execution using SAS.

HARDWARE

There are several essential layers constructing a COW. A fileserver is needed to hold code, hold/distribute data, collect answers. A primary workstation is needed to write code, submit code (direct/ftp/etc), collect answers, proxy server in peer to peer network. And finally, many workstations are needed to share executions in parallel cluster of workstations. Other issues to be discussed include sharing peripherals (monitors, keyboards, and mice), power, cooling, and communications. Readers should note that the approach to parallel COW processing discussed here is not very efficient in terms of network communications. Making clusters more efficient is discussed in great detail in Buyya (1999).

FILESERVER

First of all, the fastest and cheapest file server is LINUX! Not only is Linux inexpensive, but it is easy to set up. And, most importantly, there is plenty of research to show that Linux servers are very fast. Once the number of clients exceeds about 8 workstations, there is no competition.

In this case, Red hat 6.2 [www.redhat.com] was downloaded and installed on a modified station (faster throughput with two hard drives [primary master + secondary master] set in RAID0 configuration with two Ethernet cards). You can purchase RedHat 7.0 with a beautiful new user interface, or you can download the very same thing for nothing. RedHat Linux has no limits on the number of attached clients. Buying support is (a) not very necessary, and (b) not very useful. This Samba Server, necessary to connect Microsoft NT workstations, was up and running in 12 minutes. Readers should note that I have faked this part of the COW in the past with a plug and play SNAP! Server which has no limits on the number of clients as well. For long standing COWS, the higher speed and life of a traditional fileserver is worth the additional cost.

WORKSTATION

Any Number of NT workstations were linked in a traditional client/server architecture. The configuration of each workstation was identical. Actually one drive was set up and optimized, and the rest were cloned across the LAN with Symantec GHOST in 45 minutes! GHOST is part of NORTON SYSTEM WORKS, and a shareware version can get you most of the way there for nothing, although the full function version is worth the price. Note that each workstation had a market value of about $450 in September 2000 (500 Mhz, 128 Mb RAM, 10 Gb HDD, 100 mps Nic, SVGA).

COMMUNICATIONS

Workstations shared several keyboards/mice/monitors with Belkin KVM switches. Remember that none of workstations or file server will be touched again after the first login in most cases. The network communication was handled as cheaply as possible using small hubs ($100 each). Long standing COWs should use more expensive buy faster network switches ($1000 each).

POWER & ENVIRONMENT

A few cautions should be noted: (1) all COWs need lots of smoothed electricity (many smaller UPS's are significantly cheaper and a single larger UPS); (2) all COWs produce lots of heat, so there needs to be lots of cool breezes; and (3) in the event of a power outage, the UPS should be set to shut down all systems before the ambient temperature reaches critical limits… a few minutes in most rooms is a good rule of thumb, really; (4) all COWs eat hard drives for lunch, so use swap drawers, have plenty of GHOSTed clones on hand, and carefully monitor the warranty on any drive [hint: an IDE drive on a COW will definitely burn up before its warranty expires].

SOFTWARE

Overcoming the software barrier is the last hurdle, and SAS can provide and clever and cost effective paths to COW management and usage. Readers should note that lots of other programs could be used to manage and execute jobs on a COW (Perl, C++, etc.) and that there are companies selling cluster management software. Managing a COW, with SAS, is the point of this story.
In this example, each workstation needs a copy of SAS. SAS will be used to manage the cluster, submit jobs, distribute jobs, share data, and collect solutions. The general approach in the COW workstation flows like this:

1. Learn identity, location in network, and size of network.
2. Look for a new job to execute from server.
3. Get necessary data from server.
4. Execute relevant section of job.
5. Post solution to server.
6. GOTO step 2.

STARTUP

SAS was set in the NT startup folder so that SAS initiates after login. A startup file placed in autoexec.sas should minimally contain the unique identity of the workstation, its place in the network, and %MACRO loop to check a reference file, and barring changes, go to sleep for a while.he introduction explains the purpose and scope of your paper and provides readers with any general information they need to understand your paper.

```
* future X commands should not wait;
OPTIONS NOXWAIT;

* log files will stop a  cow in it tracks;
FILENAME NOLOG DUMMY;
PROC PRINTTO LOG=NOLOG;

* establish unique machine identity from local directory name;

%MACRO MID;
* send directory name to text file;
X "DIR D:\SASSPACE > :\SASSPACE\MACHINE.TXT";

* acquire text file :: work in  MSNT4;
DATA WORK;
INFILE "D:\SASSPACE\MACHINE.TXT" FIRSTOBS = 8;
 INPUT CREATED MMDDYY8. CREATET TIME7. AMPM $ FORM $ MACHINE $;

IF FORM = "<DIR>";
IF UPCASE(SUBSTR(MACHINE,1,2)) = "DT";

DATA WORK;
 SET WORK;

* parse the identity;

NUM = 1*SUBSTR(MACHIN E,3,3);
TEN = 1*SUBSTR(MACHINE,4,1);
ONE = 1*SUBSTR(MACHINE,5,1);

IF NUM = . THEN DELETE;

DATA WORK;
 SET WORK;

CALL SYMPUT("HOME",TRIM(MACHINE));

PROC PRINT;
PROC PRINT;
```

```
%MEND MID;
 %MID;
```

After the identity is learned, each workstation can begin monitoring the server for a new SAS command file containing new work. This is accomplished by comparing the file creation information repeatedly: (1) going to sleep if there are no changes, or (2) copying and executing the file if it changes.

```
* get the init ial estimates of file creation date and time;

X "DIR X:\DNA\ > D:\SASSPACE\TIME2.TXT";

DATA TIME2;
 INFILE "D:\SASSPACE\TIME2.TXT" FIRSTOBS = 8 LRECL=65 PAD;
 INPUT CREATED $ 1 -8 CREATET $ 11 -16 FILEN $ 40 -49;

FILEN = UPCASE(FILEN);
IF FILEN = "DOTHIS.S AS";

FTIME = CREATET;
FDATE = CREATED;
TIME = 2;

 DATA _NULL_;
  slept=sleep((60*60*0)+(60*&TWAIT));
 RUN;

* begin looping procedure to assess changes in file creation date and time;

%MACRO WORK;

 %DO I = 1 %TO &TSTOP;

X "DIR X:\DNA\ > D:\SASSPACE\TIME1.TXT";

DATA TIME1;
 INFILE "D:\SASSPACE\TIME1.TXT" FIRSTOBS = 8 LRECL=65 PAD;
 INPUT CREATED $ 1 -8 CREATET $ 11 -16 FILEN $ 40 -49;

FILEN = UPCASE(FILEN);
IF FILEN = "DOTHIS.SAS";

FTIME = CREATET;
FDATE = CREATED;
TIME = 1;

DATA TIME12;
 SET TIME1 TIM E2;

FTIME1 = LAG1(FTIME);
FTIME2 = FTIME;
FDATE1 = LAG1(FDATE);
FDATE2 = FDATE;

DATA TIME12;
 SET TIME12;

IF TIME = 2;
IF FTIME2 NE FTIME1 OR FDATE2 NE FDATE1 THEN DIRX = "DNA";
IF FTIME2 = FTIME1 AND FDATE2 = FDATE1 THEN DIRX = "DNX";
 CALL SYMPUT ("DX ",DIRX);
```

```
  ROUND = &I;

DATA TIME12;
 SET TIME12;

%INCLUDE "X: \&DX\DOTHIS.SAS";

 DATA _NULL_;
  slept=sleep((60*60*0)+(60*&TWAIT));
 RUN;
```

EXECUTION

The existence of the reference file is critical to SAS/COW management. This reference file (here called DOTHIS.SAS), which does not contain the actual job, contains pointers to the real COW jobs. Using the reference file allows the user to overcome network level file access violations, and allows the user to point to any number of new or different COW job on the network without touching any workstation. The reference file in this example utilize operating system shell commands to calculate the number of jobs, and to symmetrically distribute the jobs across the COW using commands like following:

```
***> SET JO B VARIABLES;

%LET CPU = 100;
%LET CODE = MODELA;
%LET SOURCE = WORMS;
LIBNAME DNA "X: \DNA\";
LIBNAME HOME "X: \&HOME\";
LIBNAME TEMP "D: \SASSPACE\";

%MACRO JOBS;

X "DIR X: \DNA\%SCAN(&SOURCE,1) \*.TXT /O:N > D: \SASSPACE\SOURCE.TXT";

DATA TEMP.SOURCE;
 SET DNA.MEMBER;

* INFILE "D: \SASSPACE\SOURCE.TXT" FIRSTOBS = 6  LRECL=65 PAD;
* INPUT CREATED $ 1 -8 CREATET $ 11 -16 MIDDLE $ 17 -39 FILEN $ 40 -49;

IF COMPBL(MIDDLE) = "<DIR>" THEN DELETE;
IF CREATED = " " THEN DELETE;

NAME = COMPRESS(SUBSTR(FILEN,1,INDEX(FI  LEN,".")),".");

DATA _NULL_;
 CALL SYMPUT ("JOB",COUNT);
 STOP;
 SET TEMP.SOURCE NOBS=COUNT;

run;

%MEND JOBS;
 %JOBS;

* CUT UP AND ASSIGN THE WORK LOAD;
%MACRO WORK2;

DATA WORK;
 SET WORK;

PART = INT((&JOB/&CPU)+1);
START = (NUM -11)*PART + 1;
```

```
STOP  = START + PART;

DATA WORK;
 SET WORK;

CALL SYMPUT("MSTART",START);
CALL SYMPUT("MSTOP",STOP);

%INCLUDE "X: \DNA\%SCAN(&CODE,1).SAS";
```

DEMONSTRATION

Of course, the kewlest part of this presentation is the example of using a COW to simulate financial markets.

REFERENCES

Buyya, R. (1999). High Performance Cluster Computing: Architecture and Systems, volume 1. Prentice Hall: New Jersey.

Buyya, R. (1999b). High Performance Cluster Computing: Programming and applications, volume 2. Prentice Hall: New Jersey.

Foster, I. (1994). Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering. Addison_Wesley Publishing Company: New York.

Hill, T. (1998). WINDOWS NT: Shell Scripting. Macmillian Technical Publishing: Indianapolis, IN.

Wilkinson, B. and Allen, M. (1999). Parallel Programming: techniques and applications using networked workstations and parallel computers. Prentice Hall: New Jersey.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Haftan Eckholdt
DayTrends
10 Jay Street
Brooklyn, New York 11201
Work Phone: (718) 522-3170
Fax: (718) 504-
Email: haftan@daytrends.com
Web: www.daytrends.com