Paper 271-27

Making your SAS/IntrNet® System Fault Tolerant

Glen Keer, American Express, Phoenix, Arizona
Charles Biggs, American Express, Phoenix, Arizona
Sarah Mitchell, Qualex Consulting Services, West Jordan, Utah

ABSTRACT

An overview of a SAS/IntrNet® System's Architecture with redundancy and flexibility built in for fault tolerance along with ease of growth potential.

Software: Windows 2000, SAS/IntrNet®, SAS/SHARE®, SAS/ACCESS and Base SAS Software Version 8.2, IIS 5.0, Application Center 2000, HTML, JAVA Script, ASP and other various web tools.

INTRODUCTION

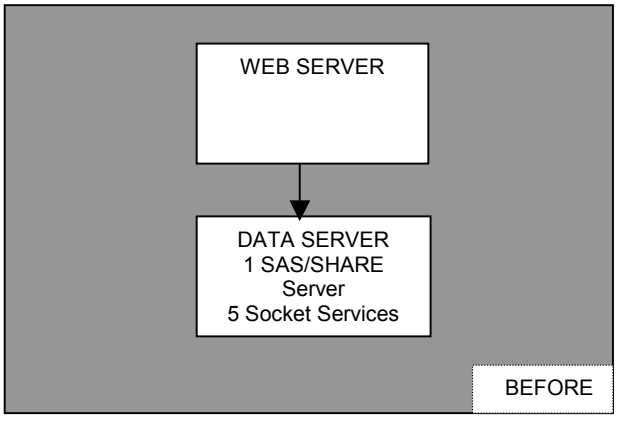
The project team started with an existing SAS/IntrNet system using version 6.12 SAS software. This system required changes because of frequent and significant down time due to many problems, including applications stepping on each other. The plan was to strengthen all tools, provide redundancy and further increase stability by isolating applications. The project went from having a single SAS/SHARE server and 5 socket services for all applications to each application having an individual SAS/SHARE server and pool service.

Duplicating all programs and data across multiple servers created redundancy. By keeping all items identical, including the names of SAS/SHARE servers, all SAS programs ran without regard as to which physical server was executing the intranet request. This way if one machine went down the other machine would pick up the slack.

The plan was kept simple yet flexible. Currently, the project has two Compaq Proliant ML570R01 Application Center 2000 clustered Web Servers, a component server for the Load Manager, and two Compaq Proliant ML570R01 data servers balanced by the Load Manager. This system has the flexibility of growth with minimal changes.

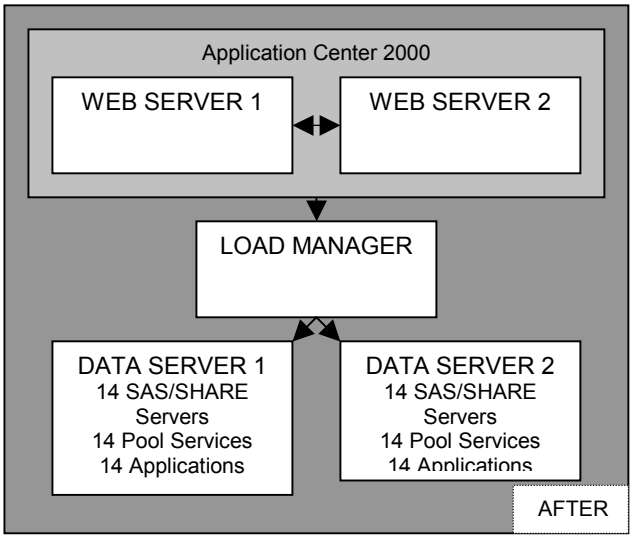
SYSTEMS ARCHITECTURE

The initial infrastructure was comprised of two Compaq 3000 Servers running Windows NT 4.0 with 2GB RAM, 1 IIS4.0 Web Server and 1 Data Server. The Web applications were comprised of ASP pages connected to the Data Server thru the SAS Broker.



The data server contained one SAS/SHARE server, which managed data for 5 separate applications. Five socket sessions were available as services for all applications. The data server housed all SAS data and SAS programs in version 6.12.

The new infrastructure includes 5 Compaq servers, all running Windows 2000 SP2, two Proliant ML370 Application Center 2000 Web Servers with 2GB RAM, one Compaq DL380 component server that houses the load manager with 2GB RAM, and two Compaq Proliant ML570R01 Data Servers with 4GB RAM. The Web Servers act as one server due to clustering with Virtual IP. The data servers manage 14 applications, each with its own SAS/SHARE server and Pool Services.



DATA/APPLICATIONS

There were many changes to the methods of managing applications. One of the biggest improvements to version 8.2 over 6.12 was work space isolation. Each intranet request has its own work space. This makes isolating applications much easier.

Changing from 1 set of socket services for all applications to an individual pool service, this allowed full utilization of resources to further isolate the data from one application to another. Changing from one SAS/SHARE session for all applications on one data server, to one SAS/SHARE session per application duplicated across two data servers, maintaining the same SAS/SHARE server names across both machines.

Batch update jobs run on the first data server. All programs were written independent of the machine they were running on. This way if the primary batch server gets overloaded the schedule can be switched to run parts on a different data server with zero code changes. With any expansion of data servers, updating programs only need an additional PROC COPY, in order to update the additional machine.

Intranet requests start with the second data server and roll over to the first data server when capacity per application is hit. All intranet request programs contain zero knowledge of what server is executing the job. From time to time changes can be made with regard to certain applications as to which of the two servers will control primary execution. This is useful for additional resource balancing.

Windows back up software runs on the second data server. The project team chose to separate the update and back up processes due to resource conflicts. Since both machines have redundant data, only one needs to be backed up.

LESSONS LEARNED

The first plan was to use a Virtual IP so that the either of the two Web Servers could talk to either of the two Data Servers as if we just had one pair of servers. Problem: SAS/IntrNet does not support Virtual IP. Also, since SAS/SHARE was being used there would have had to be different names for each SAS/SHARE server, which would have crippled the flexibility efforts. There were many potential solutions for this problem, but it was finalized by having the Load Manager run on a server alone and clustering only the Web Servers.

The knowledge base of SAS/SHARE was increased. The project team wrote a member-locking macro, %SYSLOCK, that will loop until locks are achieved. Note the following SAS Macro:

```
%macro syslock(_dsn_=,_sec_=5,_loop_=5) /
store
 des='Locks shared data';

  options mprint mlogic symbolgen;

  %global _pass_;
  %let _pass_ = 0;

  %do _i_ = 1 %to &_loop_;

    lock &_dsn_;

    %if &syslckrc > 0  %then %do;
      data _null_;
        x = sleep(&_sec_);
      run;
    %end;
    %else %do;
      %let _i_ = %eval(&_loop_ + 1);
      %let _pass_ = 1;
    %end;

  %end;

  options nomprint nomlogic nosymbolgen;

%mend syslock;
```

Just before performing an update to a database the macro is invoked with the following code.

```
%syslock(_dsn_=somedb.somefile, _sec_=2,
_loop_=20);

data _null_;
   if &_pass_ then call symput('endit',' ');
   else call symput('endit','%put JOB WAS
     KILLED; endsas;');
run;

&endit;
```

This macro will try to obtain a lock up to 20 times, sleeping 2 seconds between each attempt. If the lock cannot be achieved the program will end. Sometimes it is not desirable for a job to end or there may be a need to perform other activities, in which case the code will be slightly different. After the update is complete the database lock is released with a lock statement:

```
lock somedb.somefile clear;
```

The project team wrote another macro, %SYSOPEN, that will determine if a dataset is locked or can be read. Unlike %SYSLOCK, %SYSOPEN does not continue to loop a specified number of times. We did this because reports can be re-ran and some update processes may be long. If a guarantee is required then you can alter the following code:

```
%macro sysopen(_dsn_=,_textin_=) / store
des='checks for lock';

  %global _pass_ _blocker_ _blocker_beg_
          _textout_;
  %let _pass_ = 1;
  %let _textout_ = &_textin_;

  data _null_;
    rc=open("&_dsn_");
    if rc=0 then do;
      call symput('_pass_','0');
      call symput('_blocker_','*');
      call symput('_blocker_beg_','/*');
      call symput('_textout_',' ');
    end;
  run;

%mend sysopen;
```

This macro sends back a couple of different styles of commenting, which can be used to conditionally comment SAS code. The macro variables &_TEXTIN_ and &_TEXTOUT_ are used to hold the SET Statement or blank in the event that a lock was in place. Note the following SAS code:

```
%sysopen(_dsn_=somedb.somefile,
         _textin_= set somefile);

&_blocker_beg_;

data somefile;
  set somedb.somefile;
        .
        .
        .
  Many more statements;
run;

***********/;

data _null_;
  file _webout;

  &_textout_;

  if _n_ = 1 then do;
        .
        .
        .
    Many more statements;

    if &_pass_=0 then do;
      put '<tr><th align=center>Sorry the
           data is getting updated - Try
           again in a couple of
```

```
            minutes</th></tr>';
        put '</body>';
        put '</html>';
        stop;
      end;
    end;
        .
        .
        .
    Many more statements;
  Run;
```

The development of the log analyzer system has proved to be crucial to determining problems or inefficiencies throughout all applications. This system interrogates SAS Log Files, looking for problems and keeping statistics.

The interfaces and SAS programs were separated. Since interfaces were easier than programs it was found that a single web programmer could support several applications. A series of reviews and system standards were also incorporated. The data should be pre-manipulated as much as possible in order to keep the SAS/IntrNet programs simple and quick.

Email notification to text pagers is included for batch update abends. This way any problems can be fixed before the user community was affected.

Initially strange abend problems were encountered. The Hot Fix 82ba33 was applied and took care of the abends. This fix can be downloaded:

```
http://ftp.sas.com/techsup/download/hotfix/v82baa33.html.
```

Sometimes batch programs would stop in the middle of the job. This happened because Windows Terminal Server puts processes to sleep if they have not received either a mouse click or keyboard input in a given time frame. Microsoft has a knowledge Base article, Q186628 that states how to control the sleep timings in the registry. The project team turned the sleep timing off entirely:

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows2000\CurrentVersion\TerminalServer\Compatibility\Applications\SAS]
"Flags"=dword:00000008
"NthCountMsgQPeeksSleepBadApp"   =dword:0000FFFF
"FirstCountMsgQPeeksSleepBadApp"=dword:0000FFFF
"MsgQBadAappSleepTimeInMillisec"=dword:00000000
```

Some batch jobs do not always clear out of Windows when finished. When enough jobs remain on the machine problems with throughput and SAS/SHARE were encountered. By scheduling weekly re-boots of the data servers all of these "dead jobs" were removed, thus freeing up capacity.

CONCLUSION

The purpose of the changes outlined in this paper was to provide a scalable infrastructure with no single point of failure. The goal was completed with the exception of a single point of failure with the Load Manager. However, since the Load Manager was isolated to a component server, it has never been down. The data, SAS and Web programs are redundant on different machines, which also helps throughput. Complete separation of the applications provided additional stability, since one application is no longer able to take down another.

There are many ways to build SAS/IntrNet infrastructures; this is not necessarily recommended as the solution for all needs. However, for this environment, it was a huge win.

ACKNOWLEDGMENTS

Personal and professional thanks should go to Bubba Talley, of SAS Institute, Technical Support. Bubba has been crucial to the success of this project.

A special thanks is also extended to the management within American Express, the PQM Department, Corey Boschee and Steve Hudson for their vision and support.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. You may contact any of the three authors listed:

Glen Keer, American Express, Phoenix, Arizona
Glen.A.Keer@aexp.com,

Charles Biggs, American Express, Phoenix, Arizona
Charles.A.Biggs@aexp.com,

Sarah Mitchell, Qualex Consulting Services, West Jordan, Utah
Sarah.Mitchell@qlx.com,