PharmaSUG 2015 – Paper QT13

Is Your SAS® System Reliable?

Wayne Zhong, Accretion Softworks

ABSTRACT

If you ask your SAS Server to add 1 + 1, how often will you receive the answer 2? As professionals working in the pharmaceutical industry, we take for granted that our computer systems will follow the Code of Federal Regulations (CFR) Title 21 Part 11, namely that our SAS systems possess 'accuracy, reliability, consistent intended performance'. Computer systems however are complex and difficult to validate completely by IT, and quality lapses result in SAS Systems that get it right “most of the time”. This paper discusses ways a SAS System can perform inconsistently, its outward symptoms, and how to prevent it all by running a QA test program. The program is in SAS and full code is provided.

INTRODUCTION

One of the first lessons a prospective programmer learns is that computers do not make mistakes. If a program does not perform as expected, the fault lies with the instructions provided by the operator or the written code itself, not the computer which faithfully executes the instructions it is given. This is a good lesson in that these two types of error are almost always the cause of trouble.

There is however a third possibility: the computer system is defective, and will not reliably deliver correct results even with sound code and user operation. Before anyone blames their computer for their log error messages however, let’s first discuss the symptoms of an unstable system in everyday use.

This paper was developed with SAS environments in mind, with SAS code examples and a test tool written in SAS provided in Appendix A.

SYMPTOMS

While computers that do not start up or crash frequently can be considered defective, it is unlikely that one with severe symptoms will remain undetected as a SAS server in a business environment. Instead, the challenge is to find defective systems that appear functional, making calculation mistakes with rare enough frequency to escape attention in casual testing and regular use.

The key difference between unreliable systems and human errors is: running a program with code errors will always result in incorrect outputs, whereas running any program using a defective system can result in incorrect outputs on rare occasions. The latter is not reproducible. In pharmaceutical environments, SAS programs produce outputs such as SAS Datasets, RTF files, and PDF files. These outputs are expected to remain unchanged when the input remains static. If it is noticed that outputs produced by the same programs, run under the same conditions with the same inputs, can sometimes change, this is a symptom of an unstable system.

Visual inspection is unlikely to stumble across this type of issue due to its low occurrence. It takes automated processes that monitor output versions for changes (usually designed with other purposes in mind) to reliably detect these issues. This is due to the odds of the same random error repeating being close to non-existent. Even when detected, simply rerunning the program will fix the offending output, which may lead to a failure to link events to a systematic risk and creates the possibility for unnoticed errors.

DETECTION

The previous section discusses encountering symptoms in a live environment, a worst case scenario. Before this happens, is it possible to pre-emptively test reliability while evaluating a new SAS server? The answer of course is yes. Put simply, if an unreliable system will make mistakes, a test can be constructed by looping a sample SAS program a billion times and comparing the output – all done programmatically with the push of a button. If no errors are detected after such a test, and in live usage only a thousand similar programs are expected to be run per day, a high degree of confidence in the system can be established.

The test program should perform tasks that a live SAS program would, such as creating SAS Datasets and computing statistics. It should also perform quality assurance tasks, checking whether each step is executed successfully. Consider the macro code below.

```
%macro test1();

data total;
    x=.;
    stop;
run;

%do i=1 %to 100;
    data add;
        x=&i;
    run;

    data total;
        set total add;
    run;
%end;

data result;
    do x=1 to 100;
        output;
    end;
run;

proc compare base=result compare=total;
run;

%if &sysinfo ne 0 %then %put calculation error occurred;

%mend test1;
```

This code creates two SAS Datasets, TOTAL and RESULT, with the same content: 100 records with 1 variable X and ascending values from 1 to 100. The approaches are very different. For each record in TOTAL, 2 datasets (ADD and TOTAL) are read and written. On the other hand, the dataset RESULT is achieved with one dataset written and no reads. While the latter code is more efficient, the former is more susceptible to storage read/write errors and contrary to common programming sense serves as better test code. By giving more chances to the computer to error, a more challenging test is created.

Does this mean we set 100 to 1 billion in the macro above and call our test program complete? While any resulting mismatch would be a red flag, there are still improvements that may be made. The first is test diversity. The test program included in Appendix A includes similar loops using sort, merge, and modify steps as well as mathematical operations to more closely emulate code found in live SAS programs. This allows the test to be more representative of a live usage scenario. The second is test duration control. A shorter test step can be looped continuously until a set duration limit is reached.

```
%macro runtest1(seconds=);

%let dt1=%sysfunc(datetime());
%let duration=0;

%do %while (%sysevalf(&duration < &seconds));
    %test1;
    %let dt2=%sysfunc(datetime());
    %let duration=%sysevalf(&dt2-&dt);
%end;

%mend runtest1;
```

The third is reporting. Writing error messages in the log works, however a more professional look can be achieved by collecting information on total checks executed and number of errors encountered. Counts can be collected using macro variables and written to a neat .html file using a put statement with the code below.

```
data _null_;
    file "&path/report.html";
    set write;
    put line;
run;
```

The dataset WRITE contains one variable LINE, with values making up the necessary html code to create the image below. Full code is provided in Appendix A.

SERA Report

Measurement Description	Attempts	Errors
Read Operation	4,512	0
Write Operation	4,512	0
Arithmetic Calculation Set	48,320,000	0
Sort Operation	320	0
Merge Operation	320	0
Overwrite Operation	800	0
Delete Operation	2,912	0
Duration: 0 Hours 1 Minutes	.	.
Performance Score: 8.6	.	.

CONCLUSION

The possibility of an unreliable SAS computing environment exists only if reliability is taken for granted. This paper discussed how to recognize symptoms of instability in a live environment, and how to design a test program to prevent or confirm the presence of defective systems.

This paper did not discuss possible causes of defects, because there are many due to the complexity of modern systems – storage/memory/CPU/network/software any subsystem of which can cause issues. If errors are detected, and a determination is made that user error is not the cause, IT should be contacted.

In Appendix A of this paper, a sample SAS program is provided to perform reliability testing without the installation of special software, with a workload that is typical in pharmaceutical environments, and produces an html report. A performance score is also calculated, which is helpful in ensuring that shiny new server is not slower than what it is intended to replace.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact Wayne at:

Author Name: Wayne Zhong
Company: Accretion Softworks
Email: wayne@asoftworks.com

APPENDIX A

```
/*

SERA: SAS® Environment Reliability Assessment
Version: 1.00
SERA is a test program designed to detect instability in SAS computer systems.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED,
INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
PARTICULAR PURPOSE AND NONINFRINGEMENT.  IN NO EVENT SHALL THE CONTRIBUTORS OR
COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN
AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH
THE SOFTWARE OR THE USE OR OTHER DEALINGS WITH THE SOFTWARE.

SAS and all other SAS Institute Inc. product or service names are registered
trademarks or trademarks of SAS Institute Inc. in the USA and other countries.
® indicates USA registration.


** DIRECTIONS: **
1. store this SAS program on the logical drive the user wishes to test
2. specify test run time in hours and minutes below, in integers >=0, and press run
3. open report.html in newly created folder at the same location as this program
4. (optional for servers) for each physical CPU core, have a separate user run this
    test concurrently for a more accurate test


** SUGGESTIONS: **
This program executes common code found in SAS programs via loops. Additional code
counts errors occurred and creates a summary. It is recommended to review the
program code to understand what it does.

General test durations:
1  minute:  performance testing only
10 minutes: moderate chance to catch issues
1  hour:    high chance to catch issues
24 hours:   best chance to catch issues

Even after a 24 hour reliability assessment, the only acceptable number of errors
is 0. Contact IT to remedy errors identified.

Frequency of testing: Reliable systems age and wear out over time. Once every 1/2
or 1 year is sufficient, or upon suspection of instability. Consult with IT.


** PERFORMANCE TESTING: **
A one minute test produces a performance score that is normalized to workloads in
the Pharma SAS industry. A score of 5 indicates the system can serve 5 active
programmers concurrently without appearing "slow" or "sluggish". If multiple tests
are run concurrently, scores should be summed together.

The test does not evaluate connection speed in the case of virtual desktop.
Connectivity issues can cause a server to feel sluggish but programs still run
quickly.

*/


** set test run duration, use integers >=0 **;
%let hours=0;
%let minutes=1;



** turn off most harmless log messages to avoid a huge log file **;
options nonotes nosource compress=no noquotelenmax;
** important: don't stop processing after encountering errors **;
options nosyntaxcheck nodmssynchk;


** create folder for temp testing data and writing results **;
%global currentpath dt runid;
%let dt=%sysfunc(datetime());
%let runid=%sysfunc(translate(%sysfunc(putn(&dt,is8601dt.)),_,:));
```

```
data _null_;
    if envlen('sas_execfilepath')>0 then i=sysget('sas_execfilepath');
    else i=getoption('sysin');
    path=substrn(i,1,find(i,'\',-1E3));
    x=dcreate("SERA_D&runid",path);
    call symput('currentpath',strip(path));
    if x ne '' then put 'Created Dir: ' x;
    else do;
        put 'ERROR: unable to create test folder, check permissions ';
        abort cancel;
    end;
run;

%put runid=&runid;
%put currentpath=&currentpath;


libname testlib "&currentpath.SERA_D&runid";
%put &currentpath.SERA_D&runid;

** use macro variables to store total check runs and errors **;
** doubled up to check for errors in processing macro variables **;
%global
readtot  readerr
writetot writeerr
arithtot aritherr
sorttot  sorterr
mergetot mergeerr
ovrwrtot ovrwrerr
delettot deleterr
x loop duration dt1
e1 e2 e3 e4
;

%let readtot=0;  %let readerr=0;
%let writetot=0; %let writeerr=0;
%let arithtot=0; %let aritherr=0;
%let sorttot=0;  %let sorterr=0;
%let mergetot=0; %let mergeerr=0;
%let ovrwrtot=0; %let ovrwrerr=0;
%let delettot=0; %let deleterr=0;


** helper macros used in subsequent checks **;

** increment error tracking macro variable **;
%macro counterr(chk);
    %let &chk.err=%eval(&&&chk.err+1);
%mend counterr;

** increment arithmetic error count from within data step **;
%macro counterr1();
    call execute('%let aritherr=%eval(&aritherr+1);');
%mend counterr1;

** increment total tracking macro variable **;
%macro counttot(chk,num);
    %let &chk.tot=%eval(&&&chk.tot+&num);
%mend counttot;


** does dataset exist? if so set macvar to 1, else 0. **;
** if 0 wait 0.1 seconds and try again **
** set as write error if 0 on second time, read error if 1 **;
** count read/write attempts and errors **;
** reversed for delet chk **;
%macro dexist(ds,macvar);
    %counttot(read,1);
    %counttot(write,1);
    %let &macvar=%sysfunc(exist(&ds));
    %if &&&macvar=0 %then %do;
        data _null_;
            call wait(0.1);
        run;
        %let &macvar=%sysfunc(exist(&ds));
        %if &&&macvar %then %counterr(read);
        %else %counterr(write);
    %end;
%mend dexist;
```

```
** compare two datasets that should be identical, count attempts of operation used to create datasets and
errors **;
%macro comprdata(ds,chk);
    %counttot(&chk,1);
    proc compare base=&ds.1 compare=&ds.2 noprint;
    run;

    %if &sysinfo>0 %then %counterr(&chk);
%mend comprdata;


** delete a dataset **;
%macro killdata(lib,ds);
    %counttot(delet,1);
    proc datasets lib=&lib nolist nodetails;
        delete &ds;
    quit;
    %if %sysfunc(exist(&lib..&ds)) %then %counterr(delet);
%mend killdata;


** check macros perform calculations and helper macros determine if errors occurred **;
%macro check1(lib);

%local i;
%do i=1 %to 30;

%counttot(arith,100000);

** do simple arithmetic calculations twice and error if results are not equal **;
data &lib..check1;
    do i=1 to 100000;
        x=(ranuni(0)-.5)*100;
        y=(ranuni(0)-.5)*100;
        if x=. or y=. then %counterr1
        else do;
            sum1=x+y;
            dif1=x-y;
            prod1=x*y;
            if y ne 0 then div1=x/y;
            exp1=x**2*y**2;
            sum2=x+y;
            dif2=x-y;
            prod2=x*y;
            if y ne 0 then div2=x/y;
            exp2=x**2*y**2;
            if sum1 ne sum2 or dif1 ne dif2 or prod1 ne prod2 or div1 ne div2 or exp1 ne exp2 then %counterr1;
        end;
        output;
    end;
run;

%dexist(&lib..check1,e1);
%if &e1 %then %killdata(&lib,check1);

%end;

%put check1 duration = %sysevalf(%sysfunc(datetime())-&dt1);
%let dt1=%sysfunc(datetime());

%mend check1;


** sort one dataset twice and merge them twice, error if results do not compare equal **;
%macro check2(lib);

%local i;
%do i=1 %to 20;

%counttot(arith,1000);

data &lib..check2a;
    length str1 str2 $10 num1 8;
    do i=1 to 1000;
        str1=byte(floor(65+26*ranuni(0)))||byte(floor(65+26*ranuni(0)));
        str2=byte(floor(65+26*ranuni(0)))||byte(floor(65+26*ranuni(0)));
        num1=round(100*ranuni(0));
        if length(compress(str1)) ne 2 or length(compress(str2)) ne 2 or num1=. then %counterr1;
        output;
    end;
run;
```

```
%dexist(&lib..check2a,e1);

%if &e1 %then %do;
    proc sort data=&lib..check2a out=&lib..check2b1 nodupkey;
        by str1 num1;
    run;

    proc sort data=&lib..check2a out=&lib..check2b2 nodupkey;
        by str1 num1;
    run;

    %killdata(&lib,check2a);

    %dexist(&lib..check2b1,e1);
    %dexist(&lib..check2b2,e2);

    %if &e1 and &e2 %then %do;
        %comprdata(&lib..check2b,sort);

        data &lib..check2c1;
            merge &lib..check2b1 &lib..check2b2(rename=(str2=str3));
            by str1 num1;
        run;

        data &lib..check2c2;
            merge &lib..check2b1 &lib..check2b2(rename=(str2=str3));
            by str1 num1;
        run;

        %dexist(&lib..check2c1,e3);
        %dexist(&lib..check2c2,e4);

        %if &e3 and &e4 %then %comprdata(&lib..check2c,merge);
        %if &e3 %then %killdata(&lib,check2c1);
        %if &e4 %then %killdata(&lib,check2c2);
    %end;

    %if &e1 %then %killdata(&lib,check2b1);
    %if &e2 %then %killdata(&lib,check2b2);
%end;

%end;

%put check2 duration = %sysevalf(%sysfunc(datetime())-&dt1);
%let dt1=%sysfunc(datetime());

%mend check2;


%macro check3(lib);

data &lib..check3a1;
    stop;
run;

data &lib..check3a2;
    stop;
run;

%local i;
%let i=1;
%dexist(&lib..check3a1,e1);
%dexist(&lib..check3a2,e2);

%do %while ((&e1 and &e2 and &i<=50));
    data &lib..check3b;
        length str1 $10;
        str1=byte(floor(65+26*ranuni(0)))||byte(floor(65+26*ranuni(0)));
    run;

    %dexist(&lib..check3b,e3);

    %if &e3 %then %do;
        data &lib..check3a1;
            set &lib..check3a1 &lib..check3b;
        run;

        data &lib..check3a2;
            set &lib..check3a2 &lib..check3b;
        run;
```

```
        %dexist(&lib..check3a1,e1);
        %dexist(&lib..check3a2,e2);
        %killdata(&lib,check3b);

        %if &e1 and &e2 %then %comprdata(&lib..check3a,ovrwr);
        %if &sysinfo>0 %then %let i=1000;
    %end;

    %let i=%eval(&i+1);
%end;

%if &e1 %then %killdata(&lib,check3a1);
%if &e2 %then %killdata(&lib,check3a2);


%put check3 duration = %sysevalf(%sysfunc(datetime())-&dt1);
%let dt1=%sysfunc(datetime());

%mend check3;


** call above 3 checks until time limit is reached **;
%macro test();
%let x=testlib;
%let loop=1;
%let dt1=&dt;

%do %while ( %length(&x) );

    %put running loop # &loop;

    %check1(lib=&x);
    %check2(lib=&x);
    %check3(lib=&x);

    proc datasets lib=&x nolist nodetails kill;
    quit;

    %if &x=testlib %then %let x=work;
    %else %if &x=work %then %let x=testlib;

    %let dt2=%sysfunc(datetime());
    %let duration=%sysevalf(&dt2-&dt);
    %if %sysevalf(&duration > (60*&minutes + 3600*&hours)) %then %let x=;
    %let loop=%eval(&loop+1);

%end;

%put Total Duration: &duration seconds;

%mend test;

%test;

%let score=%sysfunc(round(%sysevalf((&loop - 1)/&duration*25),.01));
%put score=&score;


** output report.html file **;
data result;
    length text $100 tot err 8;
    do i=1 to 7;
        text=scan('Read Operation,Write Operation,Arithmetic Calculation Set,Sort Operation,
Merge Operation,Overwrite Operation, Delete Operation',i,',');
        tot=input(scan("&readtot,&writetot,&arithtot,&sorttot,&mergetot,&ovrwrtot,&delettot",i,','),best12.);
        err=input(scan("&readerr,&writeerr,&aritherr,&sorterr,&mergeerr,&ovrwrerr,&deleterr",i,','),best12.);
        output;
    end;
    tot=.;
    err=.;
    i=8;
    text="Duration: &hours Hours &minutes Minutes";
    output;
    i=9;
    text="Performance Score: &score";
    output;
run;

data write;
    length line $1000;
```

```
    set result end=eof;
    if _n_=1 then do;
        line='<!DOCTYPE html> <html> <head> <meta charset="UTF-8" /> <title>SERA Test Results</title>';
        output;
        line='<style type="text/css"> table { width: 80%; margin-left: auto; margin-right: auto;
font-size: 16px; font-size: 1vw;}';
        output;
        line='table, th, td { border: 1px solid black; border-collapse: collapse;} th, td { padding: 8px;
text-align: left; width: 20%;}';
        output;
        line='th.type1 {width: 60%;} tr.odd {background-color: white;}
tr.even {background-color: #F5F5F5;} td.green {background-color: #33D685;}
td.orange {background-color: #FFA347;}';
        output;
        line='div {font-size: 32px; font-size: 2vw; text-align: center;}';
        output;
        line='th {background-color: aqua; color: black; font-size: 20px; font-size: 1.2vw;}
</style> </head> <body>';
        output;
        line='<div> SERA Report </div> <table> <tr> <th class="type1">Measurement Description</th>
<th>Attempts</th> <th>Errors</th>  </tr> ';
        output;
    end;
    if mod(i,2)=0 then line='<tr class="even">';
    else line='<tr class="odd">';
    output;
    line='<td>'||strip(text)||'</td>';
    output;
    if tot<=10**9 then line='<td>'||strip(put(tot,comma12.))||'</td>';
    else line='<td>'||strip(put(tot,e12.))||'</td>';
    output;
    if err=0 then line='<td class="green">';
    else if err>0 then line='<td class="orange">';
    else line='<td>';
    output;
    if err<=10**9 then line=strip(put(err,comma12.))||'</td>';
    else line=strip(put(err,e12.))||'</td>';
    output;
    line='</tr>';
    output;
    if eof then do;
        line='</table></body></html>';
        output;
    end;
run;


data _null_;
    file "&currentpath.SERA_D&runid/report.html";
    set write;
    put line;
run;

%put
&readtot
&writetot
&arithtot
&sorttot
&mergetot
&ovrwrtot
&delettot;


%put
&readerr
&writeerr
&aritherr
&sorterr
&mergeerr
&ovrwrerr
&deleterr;
```