SCSUG 2017

From FREQing Slow to FREQing Fast: Facilitating a Four-Times-Faster FREQ with Divide-and-Conquer Parallel Processing

Troy Martin Hughes

ABSTRACT

With great fanfare, the release of SAS® 9 delivered multithreaded processing to a single-threaded SAS world. Procedures such as SORT, SQL, and MEANS could now run faster by taking advantage more fully of system resources through parallel processing paradigms. Multithreading commonly implements divide-and-conquer methodologies in which data sets or data streams are decomposed into subsets and processed in parallel rather than in series. Multithreaded solutions are faster (but typically not more efficient) than their single-threaded counterparts because execution time (but not system resource utilization) is decreased. As the costs of memory and processing power have continued to decrease, however, there remains no excuse for not implementing multithreaded processing wherever possible. To this end, and because SAS unfortunately abandoned some hapless procedures in single-threaded Sheol, this text aims to reunite the single-threaded FREQ procedure with its multithreaded bedfellows. The FREQFAST macro is introduced and espouses divide-and-conquer parallel processing that performs a frequency analysis more than four times faster than the out-of-the-box FREQ procedure. Non-environmental factors affecting FREQ performance (e.g., number of observations, number of unique observations, file size) are elucidated and modeled to demonstrate and predict performance improvement delivered through FREQFAST.

INTRODUCTION

All parallel processing is not created equal. Multithreading parallelizes processes at the thread level and is a staple of object-oriented programming (OOP) languages such as Python and Java. Several Base SAS procedures do rely on unseen multithreading to boost performance, including: SORT, SQL, SUMMARY, MEANS, REPORT, and TABULATE[i]. The Base SAS Procedures Guide 9.4[ii] and the SAS 9.4 Language Reference[iii] provide further information about SAS multithreaded procedures. However, SAS practitioners remain unable to create their own multithreaded processes in the Base SAS language. This deficiency was partially overcome with the introduction of the SAS DS2 language which enables developers to build their own multithreaded SAS solutions. Although DS2 is not further discussed in this text, Peter Eberhardt provides a fantastic overview in his book, *The DS2 Procedure: SAS Programming Methods at Work*[iv]. A separate text by the author also elucidates many scenarios in which DS2 multithreading is not faster than single-threaded processing, and underscores the importance of performance testing in one's own environment.[v]

Although user-developed *multithreaded* solutions cannot be created in Base SAS, parallel processing at a higher level—*multiprocessing*, at the process level—can be creatively and effectively implemented to increase performance. Multiprocessing is thoroughly demonstrated in the "Execution Efficiency" and "Automation" chapters of the author's text: *SAS Data Analytic Development: Dimensions of Software Quality*[vi]. By spawning multiple, concurrent SAS sessions through the SYSTASK statement, X statement, or batch jobs initiated from the operating system (OS), separate SAS programs can run in parallel to do more in less time. Increased system resources are typically consumed during parallel processing, not only because more is being done at once, but also due to system overheard required for process coordination and communication. Given the tremendous increase in speed that can be achieved, however, this seems a fair price to foster performance.

Divide-and-conquer techniques are a common parallel processing paradigm that can be implemented through both multithreading and multiprocessing, although this text will demonstrate only the latter due to the Base SAS multithreading restrictions mentioned previously. Divide-and-conquer solutions can be operationalized by reading distinct subsets of a data set in parallel, by subsequently performing some analysis or transformation in parallel, and by typically aggregating the results to produce a final, combined data product or output. *Multithreading* divide-and-conquer solutions parallelize processing through separate threads (run in the same SAS session) whereas *multiprocessing* solutions parallelize processing through separate processes (run across two or more SAS sessions).

Although SAS user-defined multithreading is unavailable in Base SAS (outside of DS2), many of the programmatic concepts central to multithreading can be extrapolated successfully to design SAS multiprocessing solutions.

FREQFAST dynamically spawns multiple SAS sessions in which parallel FREQ procedures run concurrently on separate subsets of the input data set. For example, the first SAS session might run the FREQ procedure on the first million observations, the second session on the second million observations, and so on. When all FREQ procedures have completed and produced their respective frequency data sets (via the OUT option), the primary SAS session joins the frequency data sets into an aggregate frequency data set that can be printed to produce frequency output or used by subsequent processes. Because both the input/output (I/O) read process and analysis process are parallelized, FREQFAST can perform more than four times faster than the out-of-the-box FREQ procedure.

PROC FREQ BACKGROUND

Before diving into parallel frequency analysis, a brief foray into plain-old FREQ is warranted, including an understanding of what attributes predict shorter vs longer execution time and what attributes predict successful vs failed jobs. Data set attributes that can affect FREQ performance include:

- number of observations
- number of unique observations (i.e., distinct values)
- number of fields in the TABLES statement
- variable type (character vs numeric)
- variable length
- order of values in variable
- file size
- file compression

Not all of these attributes are examined in this text. For example, the sample data sets are uncompressed and contain only one character variable of varying length, a varying number of observations in randomized order, and a varying number of unique values. The MAKEDATA macro is used in all examples to create data sets randomized across these attributes. All file attributes are known prior to running the FREQ procedure—with the exception of the number of unique values, which FREQ calculates—so these metadata could be used to model formulas that predict the speed of FREQ as well as the conditions under which FREQ could fail with an out-of-memory runtime error.

The MAKEDATA macro is included in Appendix A and its definition follows:

```
%macro makedata(dsn= /* data set name in LIB.DSN format */,
    obs= /* number of observations */,
    obsuni= /* number of unique observations */,
    charvar= /* number of character variables (creates CHAR1, CHAR2, etc.) */,
    charlen= /* length of character variables */,
    numvar=0 /* number of numeric variables (creates NUM1, NUM2, etc.) */,
    numlen=0 /* length of numeric variables (3 to 8) */);
```

To run examples in this text, first create a folder and initialize it into the global macro variable &FREQPATH, after which the PINCHLOG and MAKEDATA macros must be saved in this folder as pinchlog.sas and makedata.sas, respectively. The PINCHLOG macro is described in (and must be downloaded from) a separate text by the author, *Pinching Off Your SAS® Log: Adapting from Loquacious to Laconic Logs To Facilitate Near-Real Time Log Parsing, Performance Analysis, and Dynamic, Data-Driven Design and Optimization*[vii]. It parses, saves, and aggregates the performance metrics created by a SAS DATA step or procedure (in this case, FREQ):

```
%let freqpath=d:\sas\freqfast; *** Modify to actual folder location;
```

```
%include "&freqpath\pinchlog.sas"; *** Download from separate PINCHLOG text;
%include "&freqpath\makedata.sas";
libname freqfast "&freqpath";
options fullstimer;
```

The following FREQYEAH macro iteratively tests the effects of file attributes on FREQ performance and functionality:

```
%macro freqyeah(dsn= /* data set name in LIB.DSN or DSN format */,
    dsnout= /* output data set name in LIB.DSN or DSN format */,
    dsnmetrics= /* metrics data set in LIB.DSN or DSN format */,
    logfile= /* path, file name, and extension of log created */,
    tempfile= /* path, file name, and extension of log created for PINCHLOG */,
    iterations= /* number of iterations to run */,
    maxobs= /* number of observations */,
    maxobsuni= /* maximum number of approximate unique observations */,
    maxcharlen= /* max length of character variable being FREQqed */);
%local i cpucount memsize filesize obs obsuni charlen obsunival;
%let memsize=%sysevalf(%sysfunc(getoption(xmrlmem))/(1024*1024*1024)); * GB;
%let cpucount=%sysfunc(getoption(cpucount));
* LIB required so PROC SQL can obtain filesize;
%if %scan(&dsn,1,.)=0 %then %let dsn=WORK.&dsn;
%if %scan(&dsnout,1,.)=0 %then %let dsnout=WORK.&dsnout;
%do i=1 %to &iterations;
    %let obs=%sysevalf(%sysfunc(int(%sysevalf(%sysfunc(rand(uniform))*
       &maxobs)))+1);
    %let maxobsuni=%sysfunc(min(&obs,&maxobsuni));
    %let obsuni=%sysevalf(%sysfunc(int(%sysevalf(%sysfunc(rand(uniform))*
       &maxobsuni)))+1);
    %let charlen=%sysevalf(%sysfunc(int(%sysevalf(%sysfunc(rand(uniform))*
       &maxcharlen)))+1);
    %makedata(dsn=&dsn, obs=&obs, obsuni=&obsuni, charvar=1, charlen=&charlen);
    proc sql noprint;
       select filesize format=15.
       into :filesize
       from dictionary.tables
       where libname="%scan(%upcase(&dsn),1,.)"
          and memname="%scan(%upcase(&dsn),2,.)";
    quit;
    %let syscc=0;
    proc printto log="&tempfile" new;
    run;
    proc freq data=&dsn noprint;
       tables char1/ out=&dsnout;
    run;
    proc printto log="&logfile";
    run;
    proc sql noprint;
       select count(*) format=15.
       into :obsunival
       from &dsnout;
    quit;
    %pinchlog(logfile=&tempfile, dsnmetrics=&dsnmetrics,
       othervars=(var=procedure, val=PROC FREQ, len=$20, form=$20.,
              label=Test Procedure /
          var=err, val=&syscc, len=8, form=8., lab=SYSCC error code /
```

```
            var=obs, val=&obs, len=8, form=comma15., lab=Obs /
            var=obsuni, val=&obsuni, len=8, form=comma15., lab=Unique Obs /
            var=obsunival, val=&obsunival, len=8, form=comma15.,
               lab=Unique Obs (Validated) /
            var=charlen, val=&charlen, len=8, form=8., lab=Character Length /
            var=filesize, val=&filesize, len=8, form=comma15., lab=File Size /
            var=cpucount, val=&cpucount, len=8, form=8., lab=CPUCOUNT /
            var=memsize, val=&memsize, len=8, form=8.2, lab=MEMSIZE in GB));
    %end;
run;
%put ALL DONE!;
%mend;
```

The following FREQYEAH invocation runs the FREQ procedure 250 times, creating files that include up to 1 billion observations, up to 100,000 unique values, and one field up to 32 characters in length:

```
%freqyeah(dsn=freqfast.somedata, dsnout=freqfast.somedatafreq,
    dsnmetrics=freqfast.freqmetrics, logfile=&freqpath\freq.txt,
    tempfile=&freqpath\temp.txt, iterations=250, maxobs=1000000000,
    maxobsuni=100000, maxcharlen=32);
```

The Freqmetrics data set is created automatically by the PINCHLOG macro and is updated after each iteration with the FREQ performance metrics. The FULLSTIMER metrics Realtime and User CPU Time are saved into the Realtime and UserCPUTime fields, respectively, and demonstrate the time to execute FREQ. If FREQ fails, which most commonly results from an out-of-memory (aka 1016) runtime error, this failure code is captured in the Err field. Thus, FREQYEAH could be used not only to derive a formula to predict execution time but also to predict the conditions under which FREQ will fail.

Table 1 demonstrates the first few lines of lines of the metrics data set created automatically with PINCHLOG (when run in the SAS 9.4 Display Manager with 8GB ram and four processors). These data are more fully depicted in Figures 1 and 2.

Obs	Real Time	User CPU Time	System CPU Time	Memory in MB	OS Memory in MB	SYSCC error code	Obs	Unique Obs	Unique Obs Validated	Character Length	File Size
1	184.43	176.81	1.39	3.646	25.484	0	622,224,337	22,315	22,315	19	11,906,121,728
2	279.08	277.25	1.32	9.460	31.754	0	749,854,547	98,585	98,542	8	6,094,913,536
3	50.67	50.23	0.42	1.768	23.043	0	194,468,761	9,861	9,861	15	2,942,763,008

Table 1. Sample Metrics Created Automatically with PINCHLOG Macro

Although MAKEDATA aims to create the exact number of unique observations specified in the OBSUNI parameter, in some cases, fewer unique values will be created than specified. This occurs when the character length specified (or randomized) permits fewer unique values. For example, because only the ten letters A through J are randomized into the character fields, a one-character field will have a maximum of ten unique values (A through J), a two-character field a maximum of 100 unique values (AA, AB through IJ, JJ), and so on. To record this disparity, &OBSUNIVAL records the actual number of unique values created by MAKEDATA by analyzing the &DSNOUT data set.

Figure 1 demonstrates a correlation matrix among several attributes, including file size, the number of observations, the number of unique observations, character length, and FULLSTIMER Memory. Given the standardized data sets created by MAKEDATA, file size is roughly predicted by multiplying 1) the number of observations and 2) variable length, so these variables are highly correlated. Thus, a stepwise regression on Memory demonstrated that only 1) number of unique observations and 2) character length were significant ($p < 0.0001$), with number of unique observations accounting for a tremendous 0.944 R^2 of the total model 0.974 R^2.

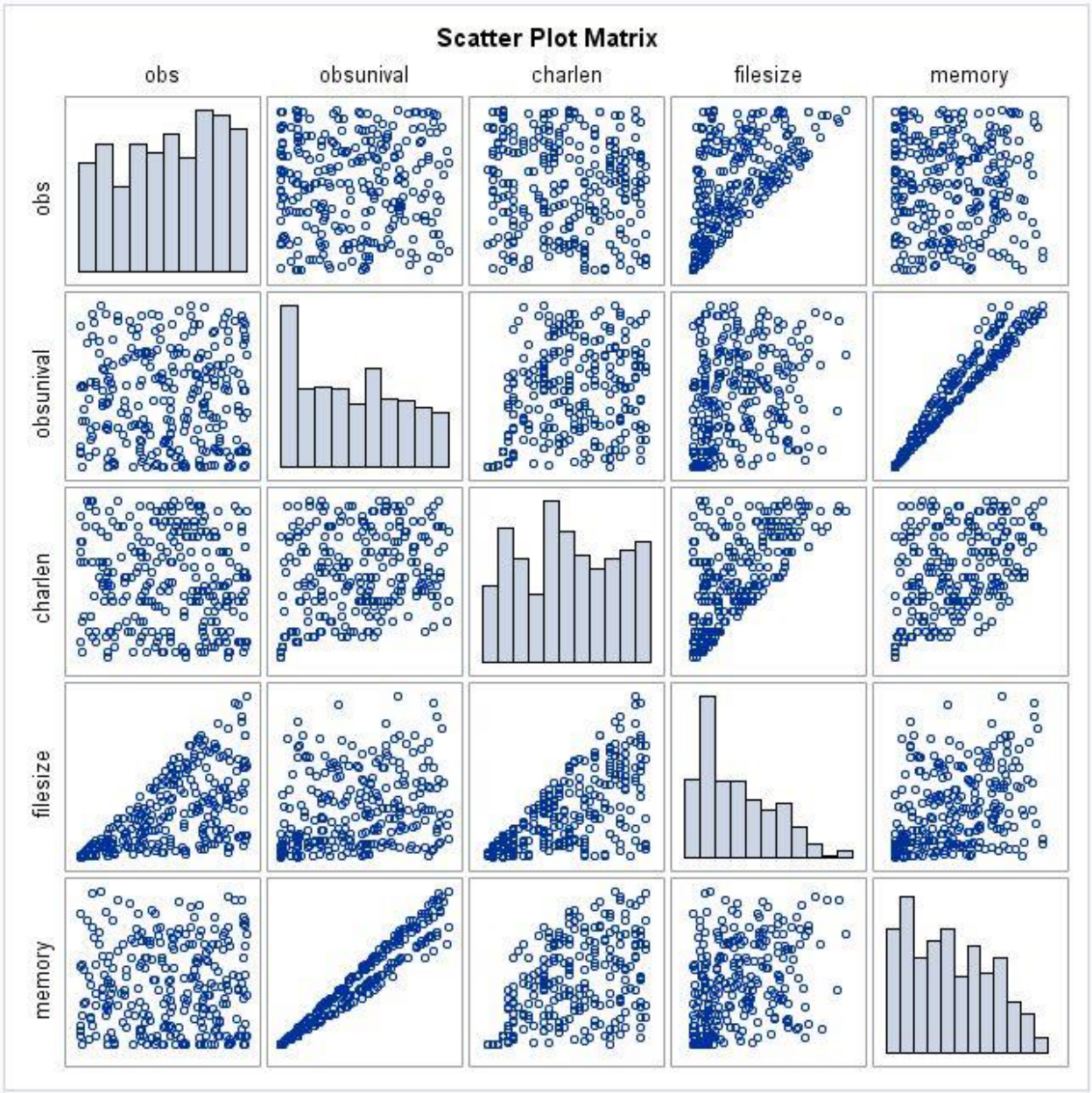

FIGURE 1. Correlation Matrix of Data Set Attributes and PROC FREQ Memory

Given the significant linear relationship between unique observations and memory, as the number of unique observations in a FREQ procedure increases ad infinitum, the SAS system will eventually run out of memory and fail. Although these cases are uncommon (stemming from either too many unique observations or too little memory), FREQFAST can be run successfully in these big-data/small-memory (BDSM) scenarios by specifying the PARALLEL=NO parameter. This option is not demonstrated in this text but causes FREQFAST to spawn batch processes synchronously (i.e., in series) rather than asynchronously (i.e., in parallel) so that the system uses significantly less memory than either the FREQ procedure or a parallel FREQFAST invocation. The remaining FREQ procedures specify a maximum of 100,000 unique observations in sample data sets.

This text focuses, however, on improving execution time rather than memory consumption, so the remaining analyses will examine only the FULLSTIMER Realtime metric. Figure 2 demonstrates the correlation matrix among several attributes, including file size, number of observations, number of unique observations, character length, and Realtime.

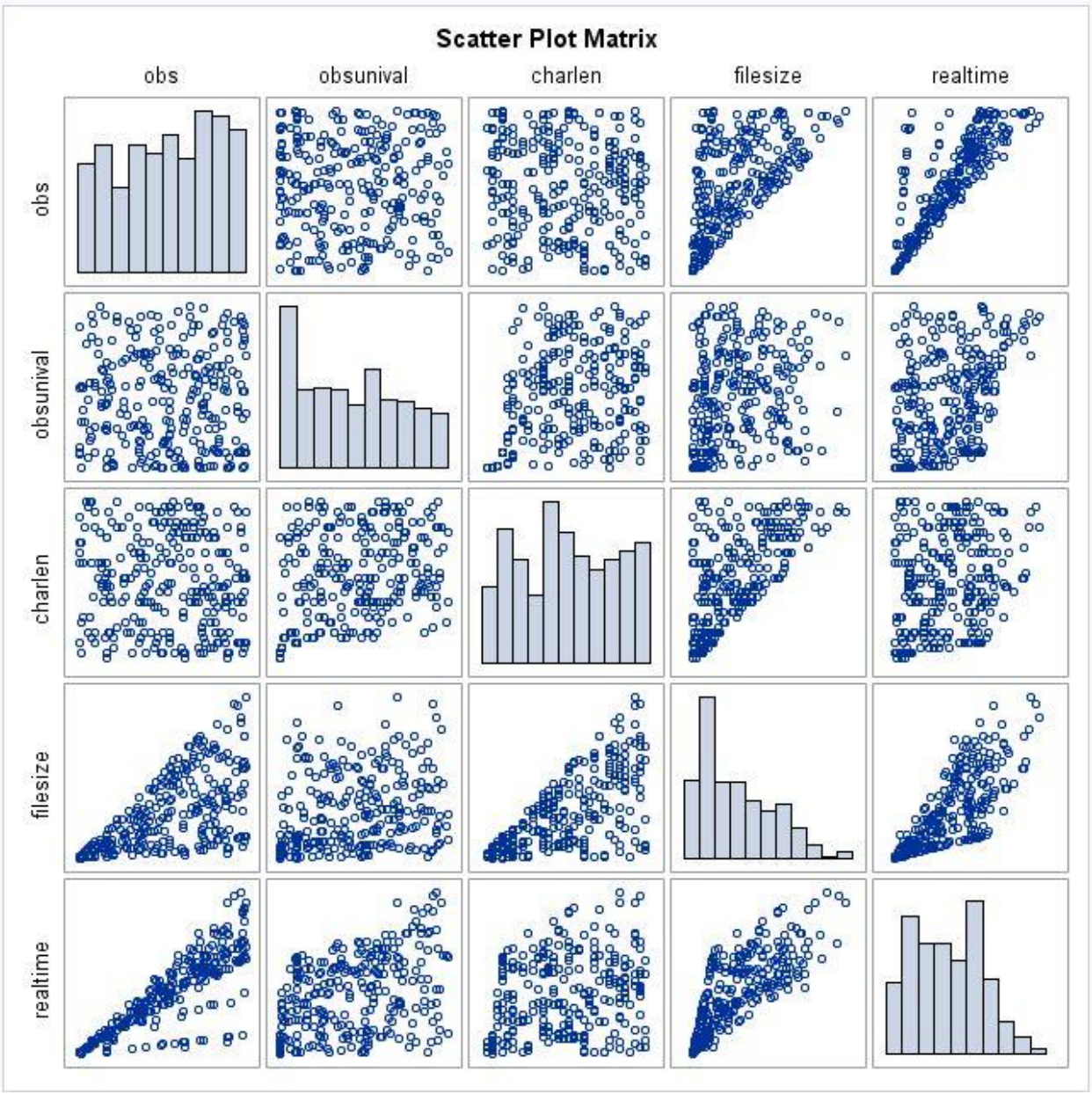

FIGURE 2. Correlation Matrix of Data Set Attributes and PROC FREQ Realtime

Figure 2 demonstrates that Realtime is highly correlated with both total number of observations and file size—both of which are highly correlated with each other (and character length) as previously described. A stepwise regression on Realtime demonstrates a 0.714 R^2 for total observations and a 0.874 model R^2 when the number of unique observations is added. However, although it's important to demonstrate the effect of the number of unique observations on FREQ speed, the number of unique observations won't be known until *after* FREQ has completed so it's ultimately not included in the model. Thus, a subsequent stepwise regression *without* unique observations demonstrates that only total number of observations (0.714 R^2) and file size (0.827 model R^2) are significant ($p < 0.0001$).

Other factors such as whether a data set is sorted, compressed, or has complete fields (vs empty space) also drive FREQ performance. For example, the FREQ procedure runs faster on a sorted data set, but this is not further discussed in this text as all sample data sets are intentionally randomized. The next sections introduce parallel design and parallel frequency analysis and compare FREQFAST performance to the traditional FREQ procedure.

DIVIDE-AND-CONQUER DESIGN

Divide-and-conquer techniques have been prominent for decades, used successfully in both multithreading and multiprocessing solutions that solve an array of programmatic challenges. Data input processes are one of the most commonly parallelized functions, as well as data analytic functions in which single observations can be analyzed independently, thus allowing data sets to be subset without effort. Figure 3 compares a serialized FREQ procedure with the parallelized FREQFAST input and analysis functions. Although not causing significant delay, the rectangular FREQFAST join process represents the system overhead required to combine output from the parallel processes.

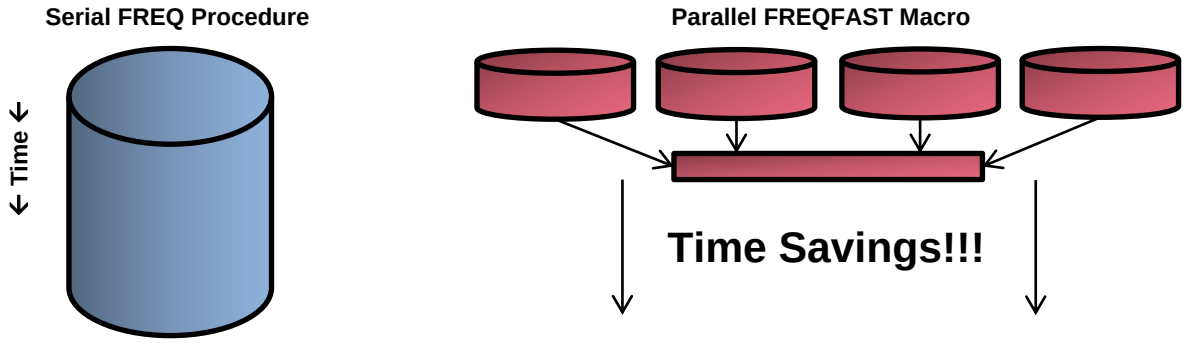

Figure 3. Relative Performance of FREQ Procedure and FREQFAST Parallel Solution

Input-centric data processes in which input data outnumber or outsize output data most benefit from divide-and-conquer solutions because data can be read in parallel but must typically be written in series. For example, the FREQ procedure synthesizes data by producing only one observation per unique value. Thus, unless the analyzed variable comprises unique values, the resultant output data set will have fewer observations than the input data set and thus expend fewer I/O resources on output than input. Conversely, the SORT procedure (which must output the same quantity of data that it inputs) benefits relatively less (than FREQ) from parallel processing, albeit still benefitting significantly. The author demonstrates automating a divide-and-conquer sort in a separate text: *Sorting a Bajillion Records: Conquering Scalability in a Big Data World*[viii].

Figure 3 demonstrates not only the time savings of parallel processing but also the additional system resource utilization required for communication and coordination of the processes. For example, the parent process must spawn the child processes in separate SAS sessions, each of which requires overhead to initialize, run, and close. The respective frequency data sets subsequently must be joined (represented as the rectangle), requiring additional system resources. It is this overhead that, despite the increased performance of divide-and-conquer solutions, dictates that they are typically less efficient than their serial counterparts. Moreover, when system resource utilization overhead is significant enough, or when its ratio to total system resource utilization is large enough, divide-and-conquer solutions can actually perform *slower* than serial solutions in addition to being less efficient.

Thus, factors such as file size, number of observations, and the number of parallel processes running concurrently will affect divide-and-conquer performance. For example, given a relatively small data set, FREQFAST offers no performance improvement over the FREQ procedure (and can even perform slower) because the increased system overhead outweighs the benefit. The optimal number of parallel processes should be selected which requires testing FREQFAST in its intended environment. However, after this nominal load testing, the benefits of FREQFAST are immediately apparent and lasting.

The TESTFREQ macro (included in Appendix D) is used for stress testing and load testing of the FREQ procedure and FREQFAST macro. For example, the following invocation runs the FREQ procedure once on a 500 million observation data set (9.8 GB) having one 20-character variable with 5,000 unique values. After the FREQ procedure completes, FREQFAST runs nine times on the same data set, iteratively varying the number of concurrent processes from between two and ten. TESTFREQ saves the performance metrics to the FREQFAST.Metrics data set:

```
%testfreq(dsnmetrics=freqfast.metrics, dsn=freqfast.disdata,
    dsnoutserial=freqfast.serial_out, dsnoutpara=freqfast.para_out,
    obs=500000000, obsuni=5000, charlen=20, path=&freqpath,
    iterations=1, minprocesses=2, maxprocesses=10, childmem=4);
```

Figure 4 demonstrates the clear performance advantage of FREQFAST over the FREQ procedure, with FREQFAST performing 4.16 times faster (28.69 seconds vs 119.39 seconds) when FREQFAST is run on eight processes.

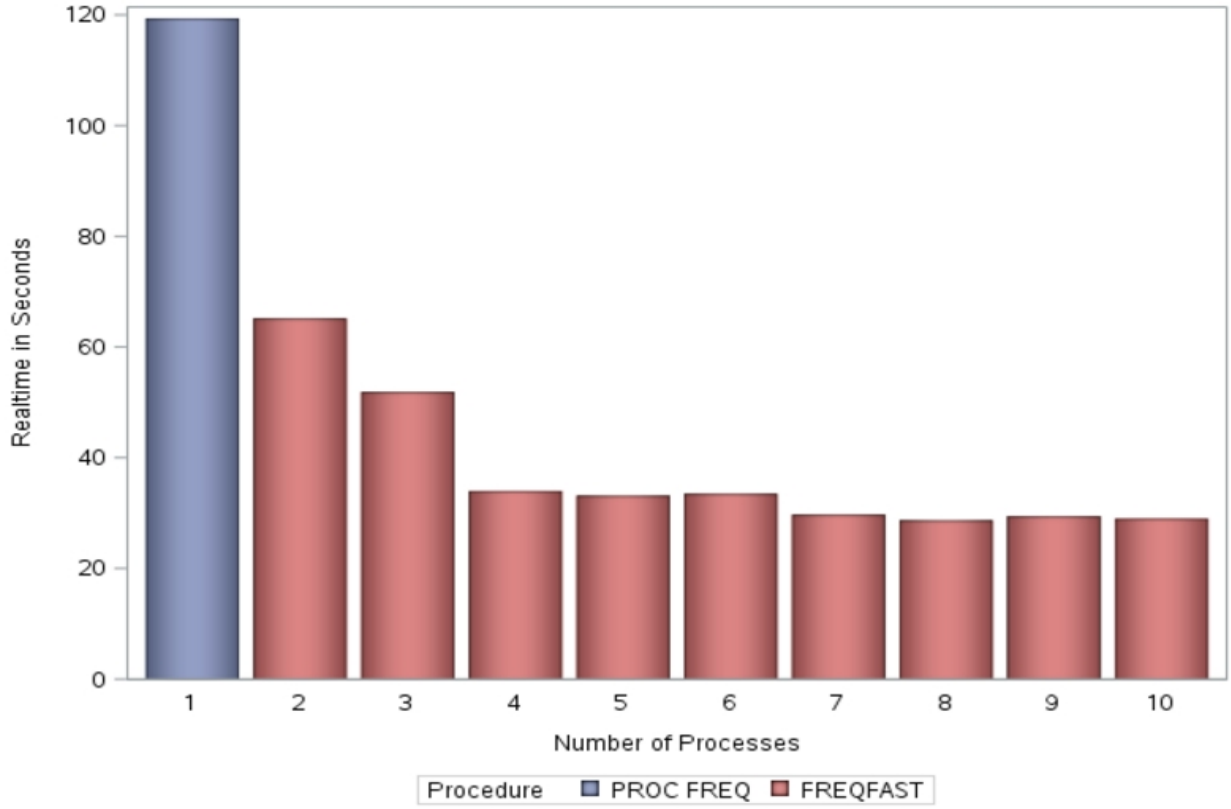

Figure 4. Realtime Comparison of FREQ Procedure and FREQFAST with Incrementing Parallel Processes

In this test environment, FREQFAST demonstrated optimal performance (in 95 percent of all tests) when running eight parallel processes, similar to the results demonstrated in Figure 4. Thus, in the remaining examples, the number of processes is held constant at eight. However, the optimal number of processes will vary by available system resources and, to a lesser extent, the data being analyzed.

FREQFAST STEP 1. INITIALIZATION AND INVOCATION

Note that because FREQFAST dynamically evaluates the path in which it resides, FREQFAST always must be run from a named SAS program file; it will fail if run from an unnamed editor window. FREQFAST requires a path into which the batch processes (i.e., child programs) and their logs will be written. For example, the following code initializes the global macro variable &FREQPATH and references (via %INCLUDE) the necessary SAS files, including pinchlog.sas, makedata.sas, freqfast.sas, and freqfast_child.sas:

```
%let freqpath=d:\sas\freqfast; *** Modify to actual folder location;
%include "&freqpath\pinchlog.sas"; *** Download from separate PINCHLOG text;
%include "&freqpath\makedata.sas";
libname freqfast "&freqpath";
```

The FREQFAST macro definition follows:

```
%macro freqfast(dsn= /* input data set name in LIB.DSN or DSN format */,
    dsnout= /* output data set name in LIB.DSN or DSN format */,
    logpath= /* path for FREQFAST child logs */,
    freqvar= /* variable on which FREQ is performed */,
    processes= /* number of concurrent processes to run */,
    mem=2 /* memory size to allocate in gigabytes */,
    parallel=yes /* runs in parallel, NO runs in series */);
```

Thus, to invoke FREQFAST to perform frequency analysis on the LIB.test data set with eight parallel processes each having 4 GB of memory, the following parameters can be specified:

```
%freqfast(dsn=freqfast.somedata, dsnout=freqfast.freqout,
    logpath=&freqpath, freqvar=char1, processes=8, mem=4);
```

The equivalent FREQ procedure (albeit single-threaded and run with default memory allocation) follows:

```
proc freq data=freqfast.somedata noprint;
    tables char1 / out=freqfast.freqout;
run;
```

The Char1 variable is consistently used throughout this text because the MAKEDATA macro names its first character variable Char1. Because FREQFAST executes multiple, concurrent FREQ procedures, it saves the respective frequency tables to temporary data sets that are later joined. The resultant combined frequency data set is identical to the data set that the FREQ procedure produces with the OUT option and can be printed or used in subsequent processes. Thus, to ensure comparability with FREQFAST, all instances of the FREQ procedure compared throughout this test use the NOPRINT and OUT options.

FREQFAST STEP 2. MAKEGROUPS MACRO

The MAKEGROUPS macro is included in the freqfast.sas file (Appendix B) and determines the observation break points at which to break a data set into two or more subsets for parallel processing. For example, given a data set of 1,000 observations, four parallel processes could each process a separate 250-observation subset of the original data set utilizing the FIRSTOBS and OBS options. Thus, the first process would ingest observations 1 through 250, the second process 251 through 500, the third process 501 through 750, and the final process 751 through 1,000. In this example, MAKEGROUPS would first determine the total number of observations and subsequently create a space-delimited list of value pairs to be used with FIRSTOBS and OBS statements.

For example, the following MAKEGROUPS invocation on a data set having 1,000 observations produces the following output:

```
%makegroups(dsn=freqfast.somedata, groups=4);

%put &grouplist;
1 250 251 500 501 750 751 1000
```

Thus, the &GROUPLIST global macro variable is used to assign values dynamically to FIRSTOBS and OBS options to subset data sets as demonstrated in the FREQFAST macro.

FREQFAST STEP 3. RUN CHILD PROCESSES

FREQFAST calls the freqfast_child.sas program (Appendix C) as a batch process via the SYSTASK statement:

```
systask command """%sysget(SASROOT)\sas.exe"" -noterminal -nosplash
    -memsize &mem.G -sysin ""&freqfastpath\freqfast_child.sas""
    -log ""&logpath\freqfast_childlog&i..txt""
    -sysparm ""logpath=&logpath, dsn=&dsn, dsnout=&dsnout&i, dsnlib=&dsnlib,
    dsnoutlib=&dsnoutlib, freqvar=&freqvar, low=&low, high=&high""""
```

```
taskname=task_freq&i status=rc_freq&i;
```

SYSTASK is not described in detail but executes the freqfast_child.sas program asynchronously once for each requested process. All processes run concurrently but wait at the WAITFOR _ALL_ boundary until all concurrent processes have completed. Because SYSTASK spawns a new SAS session, each batch process must define libraries, initialize macro variables, and specify system options that are utilized. The SYSPARM parameter can pass multiple tokenized values from the parent process to its child processes. As operationalized within FREQFAST, SYSPARM is tokenized into several sub-parameters that pass information about &DSN, &DSNOUT, and the specific range of observations that each process should ingest and analyze (i.e., FIRSTOBS and OBS values).

For example, to ensure that the SAS library associated with the &DSN data set is defined (i.e., known by the child process), the path of the &DSN library is dynamically assessed (with the PATHNAME function) and transmitted to the child via the DSNLIB sub-parameter. Within the child process, the GETPARM macro subsequently parses the &SYSPARM global macro variable (passed from the parent as the SYSPARM parameter) to define the necessary libraries (for &DSN and &DSNOUT) and to initialize the necessary macro variables (e.g., &FIRSTOBS and &OBS).

If a warning or runtime error occurs during any child process, this failure is communicated to the parent via the SYSTASK return code (e.g., &RC_FREQ1 for the first process) which will indicate 1 for a warning or 2 for a runtime error. Note that this return code paradigm differs from other automatic macro variables such as &SYSERR or &SYSCC. If any of the child processes encounters a warning or runtime error, the global macro variable &FREQFASTCHILDRC is initialized to the value of the SYSTASK return code. If no warning or runtime error is encountered, &FREQFASTCHILDRC is set to 0, which can be used in exception handling to validate success. Apt exception handling is critical whenever SYSTASK is used to spawn batch processes because failure in a child process will not affect the value of &SYSCC or &SYSERR in the parent process.

FREQFAST STEP 4. VALIDATION AND CLEANUP

After each of the child processes has completed and their individual successes are validated, the respective frequency data sets (produced by each child process) are joined into an aggregate frequency data set &DSNOUT that mirrors frequency data sets created by the OUT option of the FREQ procedure. Thereafter, each of the subset frequency data sets is deleted although the log files remain in case further analysis is warranted. With a few lines of code (not demonstrated), the log files could also be deleted to further clean the environment.

Because even FREQFAST can fail when memory or other system resources are exhausted or other conditions are encountered, exception handling must be implemented to detect not only failure in the FREQFAST macro but also in its respective child processes. Thus, if a warning or runtime error is detected in FREQFAST, this exception is passed to &FREQFASTRC whereas freqfast_child.sas failures are passed &FREQFASTCHILDRC as previously described.

To facilitate FREQFAST performance testing, four date-time global macro variables are created which decompose specific FREQFAST functionality:

- &FREQFAST_DTG1 — date-time when FREQFAST is initiated
- &FREQFAST_DTG2 — date-time when the FREQFAST internal FREQ procedure completes
- &FREQFAST_DTG3 — date-time when the FREQFAST frequency data set join completes
- &FREQFAST_DTG4 — date-time when the FREQFAST file cleanup completes

For example, the time required for FREQFAST to perform its internal FREQ procedure (i.e., &FREQFAST_DTG2 minus &FREQFAST_DTG1) can be compared to the total time (i.e., &FREQFAST_DTG4 minus &FREQFAST_DTG1) to complete FREQFAST frequency, join, and cleanup functions. In general, as the ratio of time spent performing the frequency analyses to time spent performing the join increases, FREQFAST will perform relatively faster. Stated another way, data sets that have very few observations or in which the number of unique observations is a relatively high percent of total observations will benefit less from FREQFAST because a greater proportion of time (and resources) are invested in joining—as compared with freqqing—the data.

STRESS TESTING FREQFAST: REPEATED MEASURES

With subtle modifications, the FREQYEAH macro (previously demonstrated) is transformed not only to test the performance of FREQ but also to compare its performance to FREQFAST. The TESTFREQ macro (included in Appendix D) measures the FULLSTIMER Realtime metric and its definition follows:

```
%macro testfreq(dsnmetrics= /* metrics data set in LIB.DSN format */,
    dsn= /* raw data set that is created in LIB.DSN format */,
    dsnoutserial = /* the frequency table created by the serial process */,
    dsnoutpara = /* the frequency table created by the parallel process */,
    iterations = /* number of repeated measures to run */,
    obs = /* number of observations in input data set */,
    obsuni = /* approximate number of unique values in input data set */,
    charlen = /* character length of variable to be FREQqed */,
    path = /* file path of logs that are created by FREQFAST */,
    maxprocesses = /* maximum number of processes to run concurrently */,
    minprocesses = /* minimum number of processes to run concurrently */,
    childmem= /* number of GB of memory specified by SYSTASK */);
```

TESTFREQ is more dynamic than FREQYEAH in part because it allows the user to vary not only the number of concurrent processes being tested but also the number of repeated measures to run for each scenario. For example, although Figure 4 demonstrates the variability both between the FREQ procedure and FREQFAST and among various process levels of FREQ FAST, each scenario is represented only once. Because Realtime will vary even when the program and data do not (as exemplified by fluctuations in system resource availability), repeated measures analysis can provide a more accurate portrayal of FREQ and FREQFAST performance and variability.

Figure 5 represents the same TESTFREQ invocation that produced Figure 4 with the solitary exception that the ITERATIONS parameter is changed from 1 to 10, thus causing the FREQ procedure and each of the FREQFAST concurrent process levels to run ten times. This demonstrates only subtle variability because no other SAS or non-SAS processes were running concurrently; however, repeated measures stress testing during peak business hours can show wildly varying performance as processing speeds ebb and flow with user demand and resource availability.

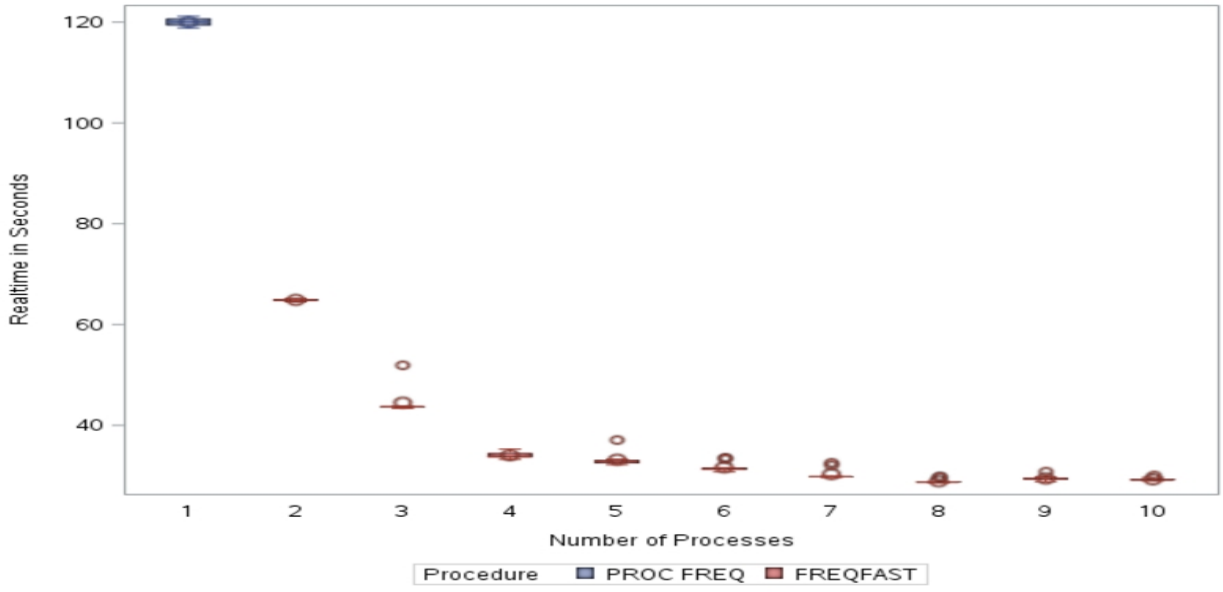

Figure 5. Repeated Measures Analysis Comparing PROC FREQ and FREQFAST Macro Across Ten Samples

STRESS TESTING FREQFAST: VARYING OBS, UNIQUE OBS, AND CHARACTER LENGTH

Figures 4 and 5 demonstrate the clear performance advantage of FREQFAST but do so in a very limited scope—i.e., a 500 million observation data set with 5,000 unique values 20 characters in length. To more appropriately demonstrate the breadth of data set scenarios for which FREQFAST outperforms FREQ, these attributes must be varied. The COMPAREFREQ macro randomizes and parameterizes the number of observations, number of unique observations, and character length before invoking TESTFREQ:

```
%macro comparefreq(samples= /* number of sample data sets to build and FREQ */,
    iterations= /* number of repeated measures to run */,
    maxobs= /* maximum number of observations */,
    maxobsuni= /* maximum number of unique observations */,
    mincharlen= /* minimum character length */,
    maxcharlen= /* maximum character length */);
%local obs obsuni charlen i;
%do i=1 %to &samples;
    %let obs=%sysevalf(%sysfunc(int(%sysevalf(
       %sysfunc(rand(uniform))*&maxobs)))+1);
    %let obsuni=%sysevalf(%sysfunc(int(%sysevalf(
       %sysfunc(rand(uniform))*&obs)))+1);
    %let charlen=%sysevalf(%sysfunc(int(%sysevalf(
       %sysfunc(rand(uniform))*%eval(&maxcharlen-&mincharlen+1))))+&mincharlen);
    %testfreq(dsnmetrics=freqfast.comparemetrics, dsn=freqfast.disdata,
       dsnoutserial=freqfast.serial_out, dsnoutpara=freqfast.para_out,
       obs=&obs, obsuni=&obsuni, charlen=&charlen, path=&freqpath,
       iterations=&iterations, minprocesses=8, maxprocesses=8, childmem=4);
    %end;
%mend;
```

The following invocation of COMPAREFREQ builds 300 sample data sets, each of which has a random number of observations, unique observations, and character length. The number of observations can vary between one and 4 billion, unique observations between ten and 3 million, and character length between one and 45:

```
%comparefreq(samples=300, iterations=1, maxobs=4000000000, maxobsuni=3000000,
    mincharlen=1, maxcharlen=45);
```

TESTFREQ builds the Comparemetrics data set that captures execution time for both the FREQ procedure and FREQFAST although these two different methods are distinguished by the class variable Procedure. This configuration is useful for regression and other analyses that can utilize a CLASS=procedure statement to compare FREQ and FREQFAST. However, because FREQ and FREQFAST results reside in alternating observations, this data set is less desirable to calculate the difference between FREQ and FREQFAST Realtime metrics.

Thus, Comparemetrics can be transformed from a stacked to a packed data set by decomposing the composite variable Paratotal (representing Realtime) into two fields Procfreqtime and Freqfasttime, which respectively capture execution time for FREQ and FREQFAST:

```
data packed (drop=procedure processes);
    set freqfast.comparemetrics (rename=(paratotal=realtime));
    length procfreqtime 8 freqfasttime 8 ratio 8 timesavings 8;
    format ratio 8.3 timesavings 8.2;
    if procedure='PROC FREQ' then do;
       procfreqtime=realtime;
       freqfasttime=.;
       end;
    else do;
       freqfasttime=realtime;
```

```
        ratio=procfreqtime/freqfasttime;
        timesavings=procfreqtime-freqfasttime;
        if err_build=0 and err_freq=0 and err_freqfast=0
           and err_freqfastchild=0 then output; * save only if no errors exist;
        end;
     retain procfreqtime;
run;
```

This transformation generates two comparative metrics:

- Timesavings — This represents the number of seconds saved (or, if negative, lost) by implementing FREQFAST instead of FREQ.
- Ratio — This represents the ratio of FREQ to FREQFAST Realtime. For example, if FREQ completes in 60 seconds and FREQFAST in only 15 seconds, the ratio is 4 and indicates FREQFAST completed 4 times faster. A ratio of 1 demonstrates no performance improvement from implementing FREQFAST although a ratio less than 1 demonstrates FREQFAST actually ran slower than the comparable FREQ procedure.

COMPAREFREQ generated 300 sample data sets all of which were 25 GB or less. Figure 6 demonstrates that for all but one of these samples, FREQFAST was at least twice as fast as FREQ; for 92.1 percent of the samples, FREQFAST was at least three times faster than FREQ, and; for 49.6 percent of the samples, FREQFAST was at least four times faster than FREQ!

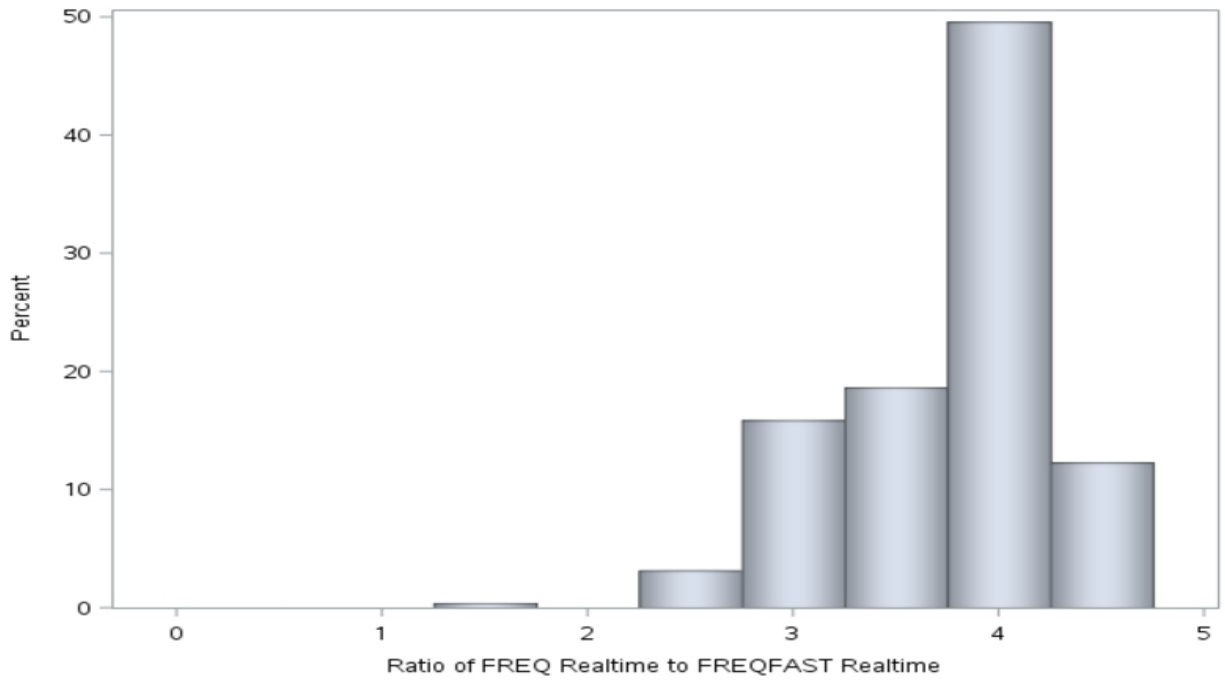

Figure 6. Distribution of Ratio of PROC FREQ to FREQFAST Realtime

Similar to the FREQ procedure, FREQFAST performance is most significantly driven by the number of unique observations (outside of environmental factors such as number of CPUs, not discussed here). This is expected considering that FREQFAST is comprised of internal FREQ procedures. Figure 7 demonstrates the correlation matrix among several attributes, including file size, number of observations, number of unique observations, character length, and Ratio.

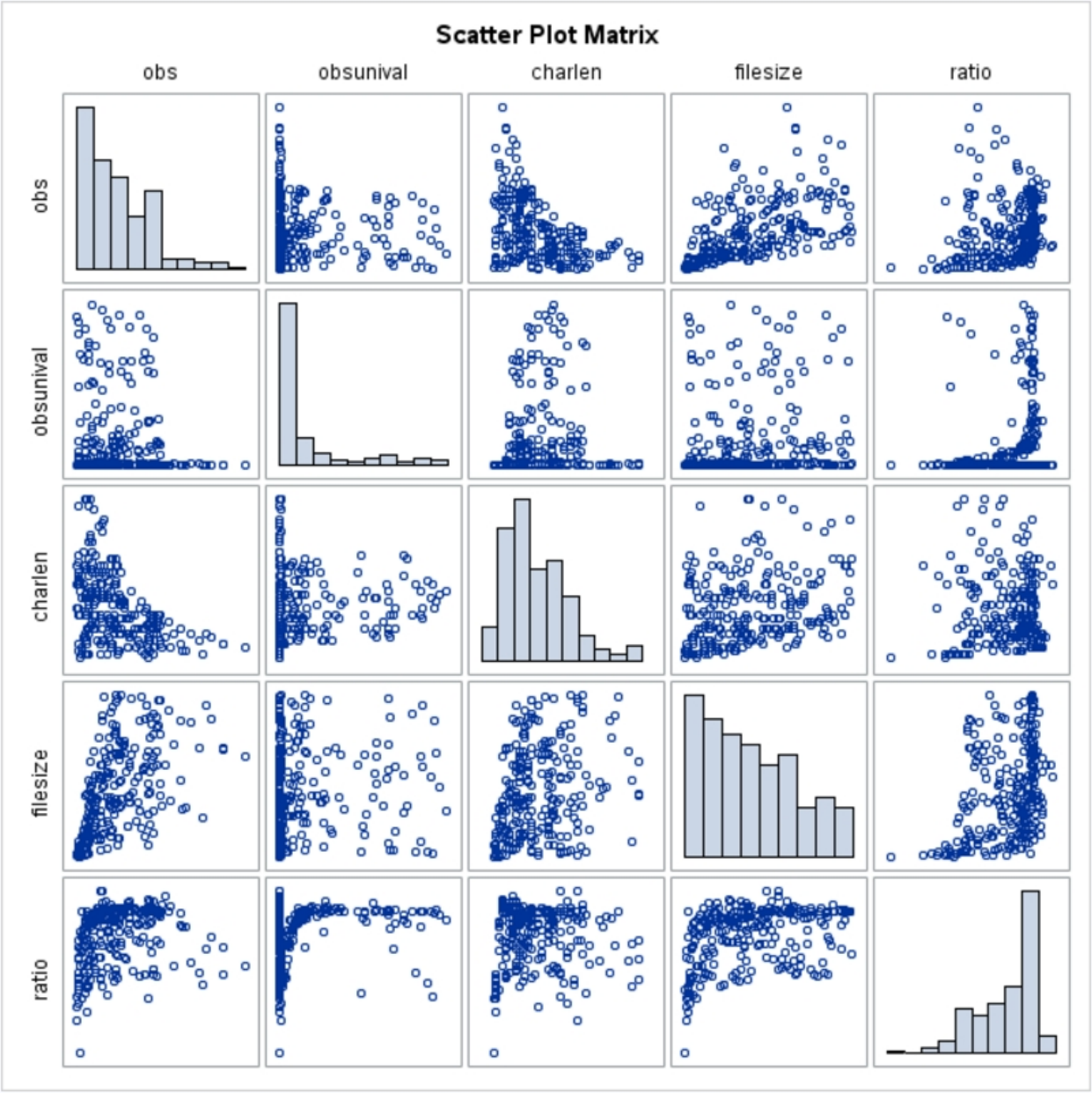

Figure 7. Correlation Matrix of Data Set Attributes and Ratio of FREQ to FREQFAST Execution Time

CONCLUSION

The SAS FREQ procedure is not multithreaded which contributes to slower performance than can be achieved through parallel processing. The FREQFAST macro introduced in this text performs a similar frequency analysis through divide-and-conquer techniques that can improve performance dramatically by more fully utilizing system resources. In roughly half of the sample data sets that were analyzed, FREQFAST executed more than four times faster than the FREQ procedure! With very little setup, SAS practitioners can implement this modular solution to accelerate the pace of frequency analysis. Finally, the TESTFREQ and COMPAREFREQ macros are included in this text and can be utilized to perform load testing and stress testing to optimize FREQFAST in any environment.

REFERENCES

[i] Scalable SAS Procedures. SAS Institute. Cary, NC. Retrieved from http://support.sas.com/rnd/scalability/procs/.

[ii] Base SAS® 9.4 Procedures Guide, Seventh Edition. Threaded Processing for Base SAS Procedures. SAS Institute. Cary, NC. Retrieved from http://support.sas.com/documentation/cdl/en/proc/70377/HTML/default/viewer.htm#n0n5mm9l2pmpevn1lsjqccmgp8tx.htm#p0fbq4inzwf57bn1xrcss87xxj8t.

[iii] SAS® 9.4 Language Reference: Concepts, Sixth Edition. Threading in Base SAS. SAS Institute. Cary, NC. Retrieved from http://support.sas.com/documentation/cdl/en/lrcon/69852/HTML/default/viewer.htm#n0czb9vxe72693n1lom0qmns6zlj.htm.

[iv] Peter Eberhardt. 2016. *The DS2 Procedure: Programming Methods at Work*. SAS Press, Inc.

[v] Troy Martin Hughes. 2019. User-Defined Multithreading with the SAS® DS2 Procedure: Performance Testing DS2 Against Functionally Equivalent DATA Steps. *PharmaSUG*. Retrieved from https://www.pharmasug.org/proceedings/2019/AD/PharmaSUG-2019-AD-228.pdf.

[vi] Troy Martin Hughes. 2016. *SAS Data Analytic Development: Dimensions of Software Quality*. John Wiley and Sons, Inc.

[vii] Troy Martin Hughes. 2017. Pinching Off Your SAS® Log: Adapting from Loquacious to Laconic Logs To Facilitate Near-Real Time Log Parsing, Performance Analysis, and Dynamic, Data-Driven Design and Optimization. *Western Users of SAS Software (WUSS)*. Retrieved from http://www.lexjansen.com/wuss/2017/119_Final_Paper_PDF.pdf.

[viii] Troy Martin Hughes. 2016. Sorting a Bajillion Records: Conquering Scalability in a Big Data World. *SAS Global Forum*. Retrieved from http://support.sas.com/resources/papers/proceedings16/11888-2016.pdf.

CONTACT INFORMATION

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

Name: Troy Martin Hughes
E-mail: troymartinhughes@gmail.com

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.

Other brand and product names are trademarks of their respective companies.

APPENDIX A. MAKEDATA MACRO (SAVED AS MAKEDATA.SAS)

```
%macro makedata(dsn= /* data set name in LIB.DSN format */,
    obs= /* number of observations */,
    obsuni= /* number of unique observations */,
    charvar= /* number of character variables */,
    charlen= /* length of character variables */,
    numvar=0 /* number of numeric variables */,
    numlen=0 /* length of numeric variables (3 to 8) */);
%let syscc=0;
%global makedataRC;
%let makedataRC=99;
%local i j maxnum;
%if %length(&numvar)>0 %then %do;
    %let maxnum=%sysevalf(32*(256**(&numlen-2)));
    %end;
%else %let numvar=0;
data &dsn (drop=obs obs2 i);
    length rando 8 i 8 obs 8 obs2 8
    %if %eval(&charvar>0) %then %do i=1 %to &charvar;
       char&i $&charlen
       %end;
    %if %eval(&numvar>0) %then %do i=1 %to &numvar;
       num&i &numlen
       %end;
    %str(;);
    %let j=%eval(&obsuni-%sysfunc(mod(&obs,&obsuni)));
    do obs=1 to &obsuni;
       %if %eval(&charvar>0) %then %do i=1 %to &charvar;
          char&i='';
          do i=1 to &charlen;
             char&i=cats(char&i,byte(int(rand('uniform')*10)+65)); *A to J;
             end;
          %end;
       %if %eval(&numvar>0) %then %do i=1 %to &numvar;
          num&i=int(rand('uniform')*&maxnum);
          %end;
       do obs2=1 to ifn(obs<=&j,%sysfunc(floor(%sysevalf(&obs/&obsuni))),
             %sysevalf(%sysfunc(floor(%sysevalf(&obs/&obsuni)))+1));
          rando=rand('uniform');
          output;
          end;
       end;
run;
* ensures data are not only random but also in random order;
proc sort data=&dsn out=&dsn (drop=rando);
    by rando;
run;
%let makedataRC=&syscc;
%mend;
```

APPENDIX B. FREQFAST AND MAKEGROUPS MACROS (SAVED AS FREQFAST.SAS)

```
%let freqfastpath=%sysget(SAS_EXECFILEPATH);
%let freqfastpath=%substr(&freqfastpath,1,%length(&freqfastpath)-
    %length(%scan(&freqfastpath,-1,\)));

* creates a space-delimited list of pairs of low/high thresholds for each group;
%macro makegroups(dsn= /* data set name in LIB.DSN format */,
    groups= /* number of desired groups into which to break obs */);
%global grouplist makegroups_obs;
%let grouplist=;
%local i obs1 obs2;
proc sql noprint;
    select count(*) format=15.
    into :makegroups_obs
    from &dsn;
quit;
%do i=1 %to &groups;
    %if &i=1 %then %let obs1=1;
    %else %let obs1=%sysevalf(&obs2+1);
    %if &i=&groups %then %let obs2=&makegroups_obs;
    %else %let obs2=%sysevalf(&i*&makegroups_obs/&groups,ceil);
    %let grouplist=&grouplist &obs1 &obs2;
    %end;
%mend;


* creates a data set with frequency distribution through parallel processing;
%macro freqfast(dsn= /* input data set name in LIB.DSN or DSN format */,
    dsnout= /* output data set name in LIB.DSN or DSN format */,
    logpath= /* path for FREQFAST child logs */,
    freqvar= /* variable on which FREQ is performed */,
    processes= /* number of concurrent processes to run */,
    mem=2 /* memory size to allocate in gigabytes */,
    parallel=yes /* runs in parallel, NO runs in series */);
%global freqfastRC freqfastchildRC freqfast_dtg1 freqfast_dtg2
    freqfast_dtg3 freqfast_dtg4;
%let freqfastRC=99;
%let freqfastchildRC=99;
%let freqfast_dtg1=%sysfunc(datetime());
%local dsnlib dsnoutlib i low high;
%if &parallel=yes %then %let parallel=;
%else %let parallel=wait;
* standardize LIB.DSN naming convention in case only DSN was used;
%if %sysfunc(find(&dsn,.))=0 %then %let dsn=WORK.&dsn;
%if %sysfunc(find(&dsnout,.))=0 %then %let dsnout=WORK.&dsnout;
%let dsnlib=%sysfunc(pathname(%substr(&dsn,1,%eval(%sysfunc(find(&dsn,.))-1))));
%let dsnoutlib=%sysfunc(pathname(%substr(&dsnout,1,%eval(
    %sysfunc(find(&dsnout,.))-1))));
* MAKEGROUPS returns the total number of obs as makegroups_obs;
%makegroups(dsn=&dsn, groups=&processes);
%do i=1 %to &processes;
    %let low=%scan(&grouplist,%sysevalf(((&i-1)*2)+1),,S);
```

```
    %let high=%scan(&grouplist,%sysevalf(&i*2));
    systask command """%sysget(SASROOT)\sas.exe"" -noterminal -nosplash
       -memsize &mem.G -sysin ""&freqfastpath\freqfast_child.sas""
       -log ""&logpath\freqfast_childlog&i..txt""
       -sysparm ""logpath=&logpath, dsn=&dsn, dsnout=&dsnout&i, dsnlib=&dsnlib,
       dsnoutlib=&dsnoutlib, freqvar=&freqvar, low=&low, high=&high"""
       taskname=task_freq&i status=rc_freq&i &parallel;
    %end;
waitfor _all_
    %do i=1 %to &processes;
       task_freq&i
       %end;;
%let freqfast_dtg2=%sysfunc(datetime());
* detect failure within child batch processes;
%do i=1 %to &processes;
    %if &&&rc_freq&i=1 or &&&rc_freq&i=2 %then %let freqfastchildRC=&&&rc_freq&i;
    %end;
%if &freqfastchildRC=99 %then %do;
    %let freqfastchildRC=0;
    data &dsnout (keep=&freqvar count percent);
       length count 8 percent 8;
       format count 8. percent 8.6;
       merge
       %do i=1 %to &processes;
          &dsnout&i (drop=percent rename=(count=count&i))
          %end;;
       by &freqvar;
       count=sum(of count1-count&processes);
       percent=(count/&makegroups_obs)*100;
    run;
    %end;
%let freqfast_dtg3=%sysfunc(datetime());
%do i=1 %to &processes;
    %if %sysfunc(exist(&dsnout&i)) %then %do;
       proc delete data=&dsnout&i;
       run;
       %end;
    %end;
%let freqfastRC=&syscc;
%let freqfast_dtg4=%sysfunc(datetime());
%mend;
```

APPENDIX C. FREQFAST_CHILD.SAS

```
%let path=%sysget(SAS_EXECFILEPATH);
%let path=%substr(&path,1,%length(&path)-%length(%scan(&path,-1,\)));
options fullstimer;

%macro getparm();
%local i;
%let i=1;
%do %while(%length(%scan(%quote(&sysparm),&i,','))>1);
    %let var=%scan(%scan(%quote(&sysparm),&i,','),1,=);
    %let val=%scan(%scan(%quote(&sysparm),&i,','),2,=);
    %global &var;
    %let &var=&val;
    %let i=%eval(&i+1);
    %end;
%mend;

%getparm;

%macro setuplib();
* DSN and DSNOUT must be in LIB.DSN (not DSN) format;
libname %substr(&dsn,1,%eval(%sysfunc(find(&dsn,.))-1)) "&dsnlib";
%if %substr(&dsn,1,%eval(%sysfunc(find(&dsn,.))-
1))^=%substr(&dsnout,1,%eval(%sysfunc(find(&dsnout,.))-1)) %then %do;
    libname %substr(&dsnout,1,%eval(%sysfunc(find(&dsnout,.))-1)) "&dsnoutlib";
    %end;
%mend;

%setuplib;

proc freq data=&dsn (firstobs=&low obs=&high) noprint;
    tables &freqvar / out=&dsnout;
run;
```

APPENDIX D. TESTFREQ MACRO FOR STRESS TESTING FREQFAST

```
%macro testfreq(dsnmetrics= /* metrics data set in LIB.DSN format */,
    dsn= /* raw data set that is created in LIB.DSN format */,
    dsnoutserial = /* the frequency table created by the serial process */,
    dsnoutpara = /* the frequency table created by the parallel process */,
    iterations = /* number of repeated measures to run */,
    obs = /* number of observations in input data set */,
    obsuni = /* approximate number of unique values in input data set */,
    charlen = /* character length of variable to be FREQqed */,
    path = /* file path of logs that are created by FREQFAST */,
    maxprocesses = /* maximum number of processes to run concurrently */,
    minprocesses = /* minimum number of processes to run concurrently */,
    childmem= /* number of GB of memory specified by SYSTASK */);
%local i j lenmax dtgstart1 dtgend1 dtgstart2 dtgend2
    err_freqserial err_freqfast;
%let memsize=%sysevalf(%sysfunc(getoption(xmrlmem))/(1024*1024*1024)); * GB;
%let cpucount=%sysfunc(getoption(cpucount));
%if %sysfunc(exist(&dsnmetrics))=0 %then %do;
    data &dsnmetrics;
       length procedure $20 processes 8 err_build 8 err_freq 8
          err_freqfast 8 err_freqfastchild 8
          obs 8 obsuni 8 obsunival 8 charlen 8
          filesize 8 buildtime 8
          parafreq 8 parajoin 8
          paraclean 8 paratotal 8
          parentmem 8 childmem 8 cpucount 8;
       format procedure $20. processes 8. err_build 8. err_freq 8.
          err_freqfast 8. err_freqfastchild 8.
          obs comma15. obsuni comma15. obsunival comma15. charlen 8.
          filesize comma15. buildtime 8.2
          parafreq 8.2 parajoin 8.2
          paraclean 8.2 paratotal 8.2
          parentmem comma15.2 childmem comma15.2 cpucount 8.;
       label procedure='Procedure' processes='Number of Processes'
          err_build='MAKEDATA Failure' err_freq='PROC FREQ Failure'
          err_freqfast='FREQFAST Failure'
          err_freqfastchild='FREQFAST Child Failure'
          obs='Obs' obsuni='Unique Obs'
          obsunival='Unique Obs (Validated)' charlen='Character Length'
          filesize='File Size' buildtime='File Build Time'
          parafreq='Paralel FREQ Time' parajoin='Parallel Join Time'
          paraclean='Parallel Clean Time' paratotal='FREQ Time'
          parentmem='Parent Memory' childmem='FREQFAST Child Memory'
          cpucount='CPUCOUNT';
       if ^missing(err_build);
    run;
    %end;
%do i=1 %to &iterations;
    %put ITERATION: &i;
    * create the sample data set to be used;
    %let dtgstart1=%sysfunc(datetime());
    %makedata(dsn=&dsn, obs=&obs, obsuni=&obsuni, charvar=1, charlen=&charlen);
    %let dtgend1=%sysfunc(datetime());
```

```
proc sql noprint;
    select filesize format=15.
    into :filesize
    from dictionary.tables
    where libname="%scan(%upcase(&dsn),1,.)" and
       memname="%scan(%upcase(&dsn),2,.)";
quit;
* run the serial FREQ procedure;
%let syscc=0;
%let dtgstart2=%sysfunc(datetime());
proc freq data=&dsn noprint;
    tables char1 / out=&dsnoutserial;
run;
%let dtgend2=%sysfunc(datetime());
%if &syscc=0 %then %do;
    proc sql noprint;
       select count(*) format=15.
       into :obsunival
       from &dsnoutserial;
    quit;
    %end;
%else %let obsunival=.;

* update metrics data set;
data x;
    if 0 then set &dsnmetrics;
    procedure='PROC FREQ';
    processes=1;
    err_build=&makedataRC;
    err_freq=&syscc;
    err_freqfast=0;
    err_freqfastchild=0;
    obs=&obs;
    obsuni=&obsuni;
    obsunival=& obsunival;
    charlen=&charlen;
    filesize=&filesize;
    buildtime=%sysevalf(&dtgend1-&dtgstart1);
    parafreq=.;
    parajoin=.;
    paraclean=.;
    paratotal=%sysevalf(&dtgend2-&dtgstart2);
    parentmem=&memsize;
    childmem=&childmem;
    cpucount=&cpucount;
run;
proc append base=&dsnmetrics data=x;
run;
* iterate through FREQFAST with incrementally more processes;
%do j=&minprocesses %to &maxprocesses;
    %freqfast(dsn=&dsn, dsnout=&dsnoutpara, logpath=&freqpath,
       freqvar=char1, processes=&j, mem=&childmem);
    %if &syscc=0 %then %do;
```

```
        proc sql noprint;
           select count(*) format=15.
           into : obsunival
           from &dsnoutpara;
        quit;
        %end;
        %else %let obsunival =.;
        data x;
           if 0 then set &dsnmetrics;
           procedure='FREQFAST';
           processes=&j;
           err_build=&makedataRC;
           err_freq=0;
           err_freqfast=&freqfastRC;
           err_freqfastchild=&freqfastchildRC;
           obs=&obs;
           obsuni=&obsuni;
           obsunival =& obsunival;
           charlen=&charlen;
           filesize=&filesize;
           buildtime=%sysevalf(&dtgend1-&dtgstart1);
           parafreq=%sysevalf(&freqfast_dtg2-&freqfast_dtg1);
           parajoin=%sysevalf(&freqfast_dtg3-&freqfast_dtg2);
           paraclean=%sysevalf(&freqfast_dtg4-&freqfast_dtg3);
           paratotal=%sysevalf(&freqfast_dtg4-&freqfast_dtg1);
           parentmem=&memsize;
           childmem=&childmem;
           cpucount=&cpucount;
        run;
        proc append base=&dsnmetrics data=x;
        run;
        %end;
    %end;
%put ALL DONE!!!;
%mend;
```