SESUG Paper 230-2019

Exploring Efficiency in Data Manipulation with SAS®: How to Get the Most Out of My Software and Hardware

Jonathan W. Duggins, NC State University, Raleigh, North Carolina;

Jim Blum, UNCW, Wilmington, North Carolina

ABSTRACT

An exploration of efficiency in combining data sets (merging/joining) is considered under a variety of conditions. Methods are employed in each of the DATA step without hash objects, DATA step with hash objects, and PROC SQL for several data sets. Data sets considered include those with large numbers of observations, large numbers of variables, or both. Non-indexed and indexed versions of data sets are considered, with time required to construct the index included in the efficiency calculation. Under selected settings, multiple operating systems and hardware platforms are considered along with options available to set the resources available in the SAS® session(s).

INTRODUCTION

SAS provides a number of options for controlling the performance of a SAS session, with some options only available at invocation and others available during an active session. Additionally, the programmer must make choices regarding programming elements that affect the efficiency of a particular program. This paper focuses on the common task of combining data sets, where it is reasonable for the programmer to expect that the best choice among the DATA step, SQL, or even whether or not to use the DS2 procedure would depend on the aforementioned settings as well as the attributes of the data sets used in the combination process.

While previous explorations have been done in this area, hardware and software updates necessitate the establishment of new benchmarks. Further, due to the number of settings available and their interactions, this paper explores a number of options that may offer efficiency gains for experienced users while also providing newer SAS users an introduction to a number of relevant settings. In particular, this paper measures efficiency using several metrics provided in the SAS Log: Real Time, CPU Time, Memory, and Operating System Memory. For all metrics, lower values are indicative of a more efficient result.

METHODS

DEFINING THE SCENARIOS

The first step in investigating relative efficiencies is determining what characteristics are most important to users. The programmer's choice of join technique was the first factor under consideration. The second factor under consideration is actually a collection of SAS system options since these are also under the control of the programmer, but within the confines of the available hardware. The final factor are the attributes of the data sets themselves since this is the factor least likely to be controllable by the programmer. In selected scenarios, the operating system is also considered.

Join Techniques

As mentioned in the Introduction, the choice of join technique is of obvious interest, so the DATA step and PROC SQL are compared here. PROC DS2 is not included in favor of comparing multiple versions of the more traditional approaches. In particular, the following join techniques are considered here under one-to-one, one-to-many, and one-to-one-to-many scenarios.

- MERGE statement in the DATA step using data sets pre-sorted via PROC SORT
- MERGE statement in the DATA step using data sets pre-sorted via PROC SORT and an index created with PROC DATASETS
- MERGE statement in the DATA step using an index created with PROC DATASETS

- Hash join in the DATA step
- PROC SQL
- PROC SQL with using an index created with PROC DATASETS

These six distinct approaches represent the most common join techniques currently in use and identify small modifications, such as the use of indexes or hash objects, that may be considered for improving efficiency. In the hash join scenario, only one data set is loaded into a hash object for the one-to-one and one-to-many joins and two data sets are loaded into separate hash objects in the one-to-one-to-many join. In all six of the scenarios provided in the list above, data sets are unsorted unless otherwise specified and are returned in sorted order in all scenarios. When using the hash objects, sorted data is produced by using the ORDERED argument tag and when using PROC SQL ordered data sets are produced via the ORDER BY clause in the CREATE TABLE statement.

System Options

SAS provides a plethora of settings that are useful for controlling various aspects of the efficiency of a SAS session. Table 1 below lists a subset of these options that may be most useful for a programmer looking to tweak their settings. For each setting a brief description is provided as well as a column that indicates whether the setting is only available at invocation (when SAS is launched) or both from within SAS (e.g. via the OPTIONS statement) – these are represented as I and B, respectively.

Option Name	Description	Availability
BUFNO	Specifies the number of buffers for processing SAS data sets.	B
BUFSIZE	Specifies the size of a buffer page for output SAS data sets.	B
COMPRESS	Specifies the type of compression to use for observations in output SAS data sets.	B
MEMSIZE	Specifies the limit on the amount of virtual memory that can be used during a SAS session.	I
REALMEMSIZE	Specifies the amount of real memory SAS can expect to allocate.	I
SORTSIZE	Specifies the amount of memory that is available to the SORT procedure.	B

Table 1: Selected options relevant for affecting efficiency.

To obtain the descriptions and the current values for these options (and others) during a SAS session, run the following code:

```
proc options group = (performance memory);
run;
```

Of the options listed in Table 1, BUFSIZE, COMPRESS, and SORTSIZE were omitted from consideration. BUFSIZE must be set during data set creation, which is limiting and may not be under the control of the programmer when data sets are received from an external source. The COMPRESS option was not considered because the decision to compress a data set should be based on the specific properties of the current data set, affect the CPU times for reading the file, and is mainly focused on reducing I/O time which, as discussed below, is not one of the metrics under consideration. Finally, SORTSIZE was not considered because it only affects the amount of memory available to PROC SORT, while REALMEMSIZE affects the amount of memory available to all procedures.

The remaining options, BUFNO, MEMSIZE, and REALMEMSIZE, represent ways for the programmer to control the buffers used to read data sets and the amount of memory, both virtual and real, available during the session. For each of these options, the scenarios cover a set of reasonable values. Table 2 shows the values for each option and indicates the default value on a 64-bit version of Windows 10. Not all combinations of option values are considered due to constraints on the computing time and space available.

Option Name	Values	Default
BUFNO	1, 100, 250, 500	1
MEMSIZE	2GB, 4GB, 8GB, 16GB	2GB
REALMEMSIZE	1GB, 2GB, 4GB, 8GB, 16GB	1GB

Table 2: Value sets for options under consideration.

Data Set Properties

The final factor under consideration are the properties of the data sets, namely the number of records, the number of variables, the variable attributes, and the number and type of key variables needed during the joins. The expectation of an impact due to the number of records (height) and number of variables (width) is certainly reasonable. However, the types of variables present (character versus numeric) and the lengths of the character variables, may also play a role. Furthermore, the number, type, and length of key variables is included as a factor to determine what, if any, impacts it may have on indexing, sorting, or other aspects of the join process.

Regardless of data set height and key variable properties, the remaining variable properties determine the variable set that makes up the remaining columns in a given data set. Table 3 shows the four variable sets under consideration and gives the number of numeric variables and the number and length of the character variables. In all cases, numeric variables have length eight. These variable sets provide reasonable representations of data sets used in a variety of industries such as clinical trials and financial risk management.

Variable Set	Number of Numeric Variables	Number of Character Variables (Length)
1	8	8 (8)
2	16	16 (8)
3	16	2 (16), 2 (32), 2 (64), 2 (128)
4	16	4 (16), 4 (32), 4 (64), 4 (128)

Table 3: Variable set descriptions.

Each data set also includes four variables, two numeric and two character, available for use as key variables. As with the variable sets, both numeric variables have length eight and the character variables have different lengths – one has length 25 and the other has length 50.

The last data set property, replication, is necessary to support the one-to-many and one-to-one-to-many joins. In those cases, one of the generated data sets includes replication within the combination of the key variables. Records either had four or eight replicates. Combining the height settings with these variable sets, key variable settings, and replication finalizes the data set properties for the various scenarios. Table 4 provides a consolidated look at the data set properties under consideration.

Property	Settings	Notes
Size	2 million, 4 million	Controls the height of the starting data set
Variable Set	1, 2, 3, 4	Controls the width of the data set
Key Variables	N1, N2, C1, C2	Combinations considered: C1; C2 N1; N1 C2; C2 N1; N1 N2 C1 C2; C2 C1 N2 N1
Replication	4, 8	4 (16), 4 (32), 4 (64), 4 (128)

Table 4: Data set and variable properties under consideration.

As with the options given in Table 2, not all combinations of settings in Table 4 are considered. Selected combinations of the join techniques, system options, and data set properties are presented in this paper in order to prioritize the salient results. Further, minimum resource requirements prevent certain combinations, such as loading the 4 million record data sets into memory – as is needed for the hash method – when using only the default 2GB of memory. Care is taken to point out these types of

restrictions when discussing the results. Any additional results are available online by visiting the website provided in the contact information below.

GENERATING THE TEST DATA SETS

Data sets used as inputs in any scenario are created by first generating the set of two numeric and two character key variables. The numeric key variables (N1 and N2) are generated by setting the desired number of digits – this paper uses 10 for N1 and 15 for N2 – and generating random integers from a uniform distribution. The character key variables (C1 and C2) are also randomly generated – a random uniform value is used to select a capital letter. The key variable values are created by concatenating the number of letters necessary to reach the desired length – 25 for C1 and 50 for C2, in this paper.

Once the key variables are generated, the remaining columns of data are generated based on the requested variable set (1, 2, 3, or 4) and the amount of duplication required (either 4 or 8). The numeric and character variables here are generated using a similar approach to the generation of the key variables – numeric variables are generated from a scaled uniform distribution and the character variables are randomly generated sequences of capital letters using the settings provided in Table 3.

DEFINING THE METRICS

Each scenario is compared based on four metrics: CPU time, real time, operating system memory, and memory. The system option FULLSTIMER places these values in the SAS log; by default, this option separates CPU time into system CPU time and user CPU time. System CPU time represents the time the system used on operating system tasks that support the execution of submitted SAS code while user CPU time is the time spent executing the submitted SAS code. For simplification, the Results section presents the total CPU time which is the sum of system and user CPU times for a given task. CPU times are measured and displayed in seconds and total real time is the primary metric used for comparing the results.

Operating system (OS) memory measures the maximum amount of memory that a given step requests from the operating system. The remaining metric, memory, simply records the amount of memory required to run a given step. Both memory-related metrics are displayed in gigabytes but are typically displayed in the SAS Log in kilobytes.

SIMULATION PROTOCOL

Because the CPU time and real time metrics are affected by resource use by programs and tasks other than SAS, each scenario is benchmarked using the median from three replicates. Time permitting, results presented and hosted online may include results from additional replicates. To ensure that all resources are released at the conclusion of a scenario, the simulation program exits the current SAS session and launches a new session before running another simulation – including replications within a scenario. To facilitate this, separate SAS programs are maintained for each join technique as well as a SAS program that writes a batch file. This batch file controls the execution of all requested scenarios and stores the resulting log files in a user-specified location. Finally, the SAS log files are scraped for the relevant metrics provided the by FULLSTIMER system option.

RESULTS AND DISCUSSION

Despite investigating a large combination of join techniques, SAS system options, and data set properties, the results rather easily condensed into several categories. Certain factors had no substantive impact on performance, some factors were found to consistently degrade performance, and a few choices can lead to significant changes in performance but only in specific scenarios.

FACTORS WITH LOW IMPACTS

The MEMSIZE option had no appreciable impact, with a notable caveat. The value of MEMSIZE must be at least large enough to load the requisite data sets into memory when using hash objects. As such, hash objects tended to require higher memory settings since tables must be loaded into memory, but once the required amount of memory was provided additional memory was not beneficial in reducing total real time

in scenarios using hash objects. Similarly, the non-hash approaches were not impacted by the choice of MEMSIZE unless values were chosen that prevented the program from executing.

The other factor that failed to have substantive impacts on total real time was the number and type of key variables. Using only numeric keys, only character keys, or a mixture of the two had no practical implications. Furthermore, the length of the character keys also played no noticeable role. These results held whether or not indexing was used.

FACTORS WITH DISPARATE IMPACTS

Indexing produced inconsistent results as measured by total real time. When combined with SQL, indexing reduced the real time associated with the SQL procedure itself, but not by enough to offset the time required to build the index. As a result, indexing is only recommended in those cases if the data set is stable – so the index does not need to be rebuilt – but the data set is used in several steps so that the program can recoup the time taken to build the index. Conversely, when using the DATA step, indexing is potentially advantageous when used in conjunction with PROC SORT. However, as Figure 1 shows, the improved efficiency is not substantial. As with indexing in PROC SQL, it may only be beneficial in cases where the index can be reused across multiple steps. Graphs in the left panel use default values for MEMSIZE and REALMEMSIZE, the middle panel uses the default value for MEMSIZE but sets REALMEMSIZE to 8 GB, and the last panel uses 8 GB for MEMSIZE and the default value for REALMEMSIZE. Note that the different memory settings have no noticeable impact on the results.

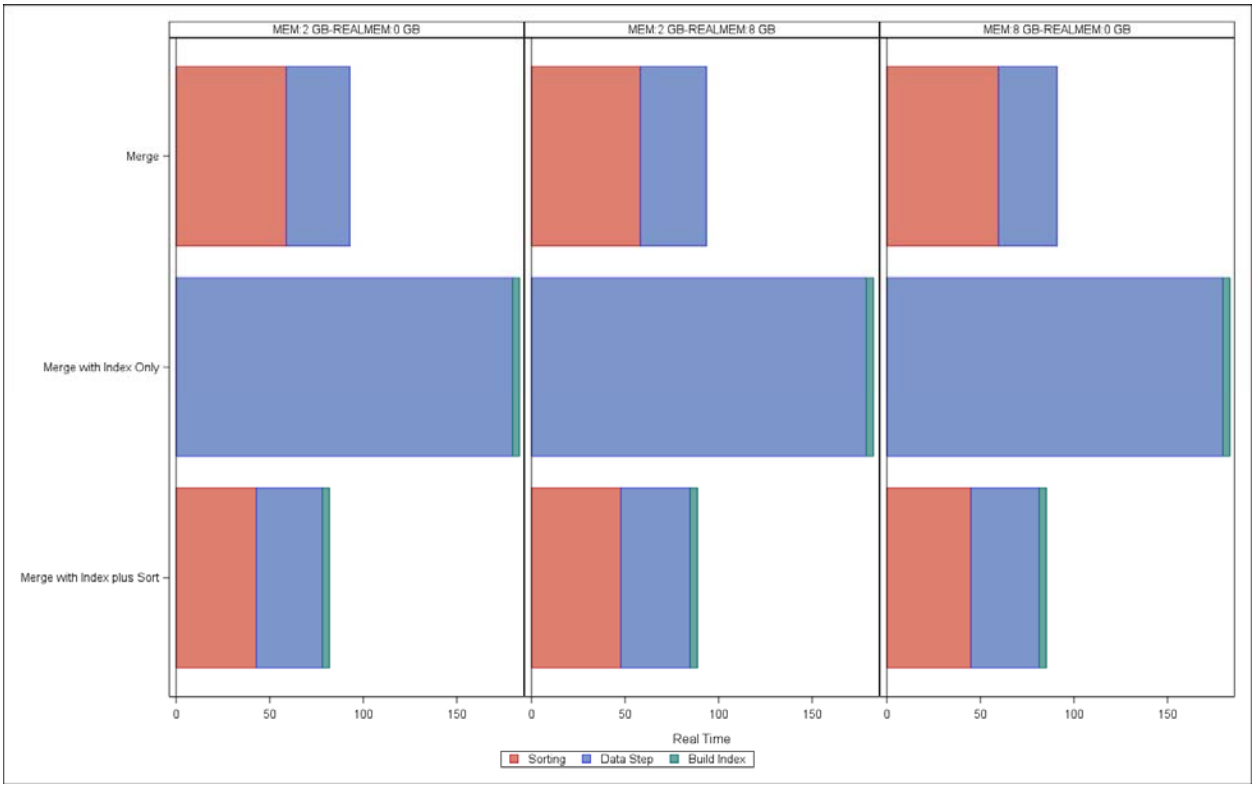

Figure 1: Effects of indexing on DATA step-based joins using the MERGE statement.

Additionally, the manual selection of the number of buffers via the BUFNO option had substantial negative impacts on total CPU time when using hash objects. SAS uses the default value, BUFNO=1, in conjunction with an optimization routine specifically for sequential access to data sets. Setting any other value resulted in severe degradation of performance and as such changing this value is strongly discouraged without careful control of the buffer size (via BUFSIZE) and other options. However, even using the DATA step-based methods without hash objects produces different results as shown in Figure 2. It shows that there is no practical difference between total real time for these scenarios.

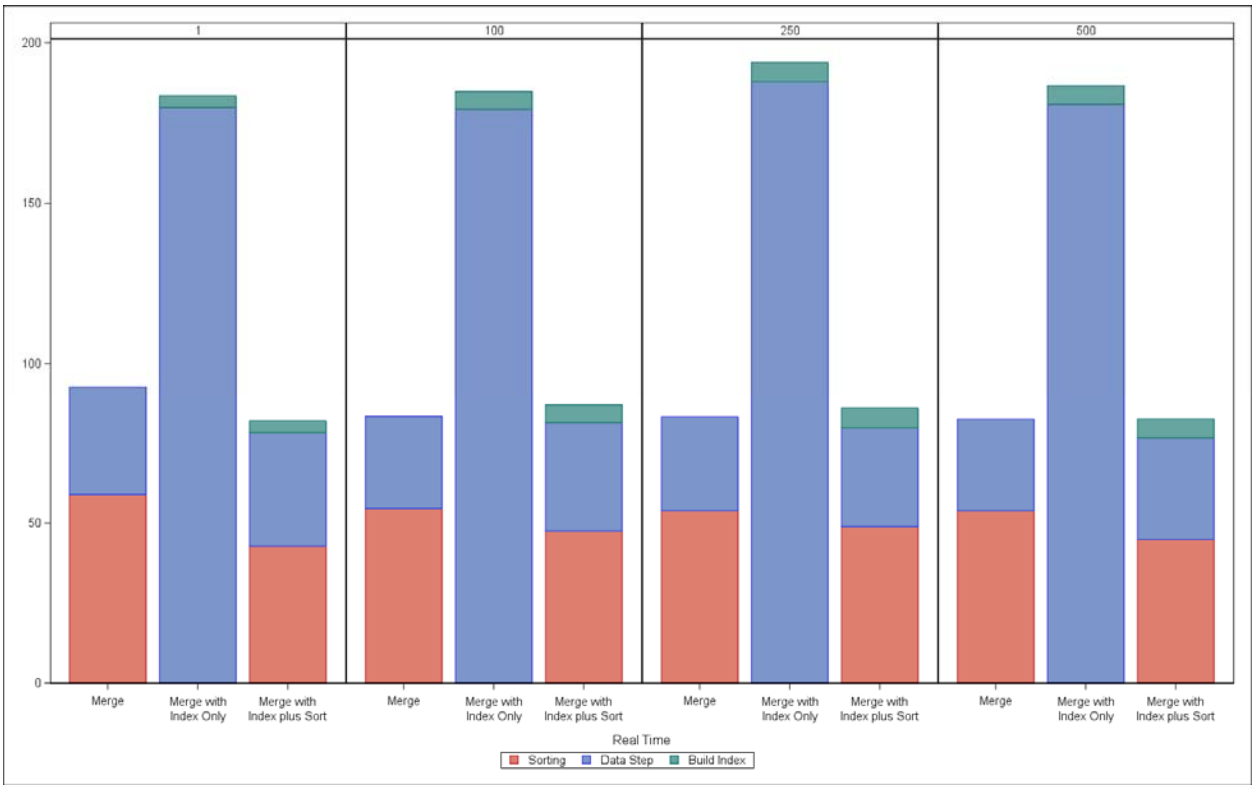

Figure 2: Effects of BUFNO in DATA step-based joins without the use of hash objects

Finally, several factors either had consistent types of impact but with varying degrees or had disparate impacts. As expected, data set size – both in terms of width and height – affects total CPU time. Figure 33 shows a subset of the results based on selected join techniques, memory settings, and size of the file. In this graph, the times plotted correspond to the median total time.

The columns represent both the size of the file and the amount of memory allocated via the MEMSIZE option. For example, the first header in each cell contains values like “One-to-Many (8) 3,4” which represents a one-to-many join with eight replicates per BY-group where the left data set in the join used variable set 3 and the right data set in the join used variable set 4. In the hash join, the left data set is loaded into the hash object. The second header in each cell represents the amount of memory, in gigabytes.

Figure 3 highlights multiple themes found in the results. First, note that within a row – i.e. for fixed file size – there is virtually no difference in results as the amount of allocated memory increases. The notable exceptions are in rows two and four where the hash join is missing in the first column. This is because with files of this size, the hash object is beyond what it can store with the default 2 GB of memory.

Note that by comparing rows the impact of increasing the file size is apparent in that both the DATA step merge and the SQL procedure increase substantially – almost doubling when the right data set uses variable set four instead of variable set three and more than doubling when the right data set has double the number of replicates.

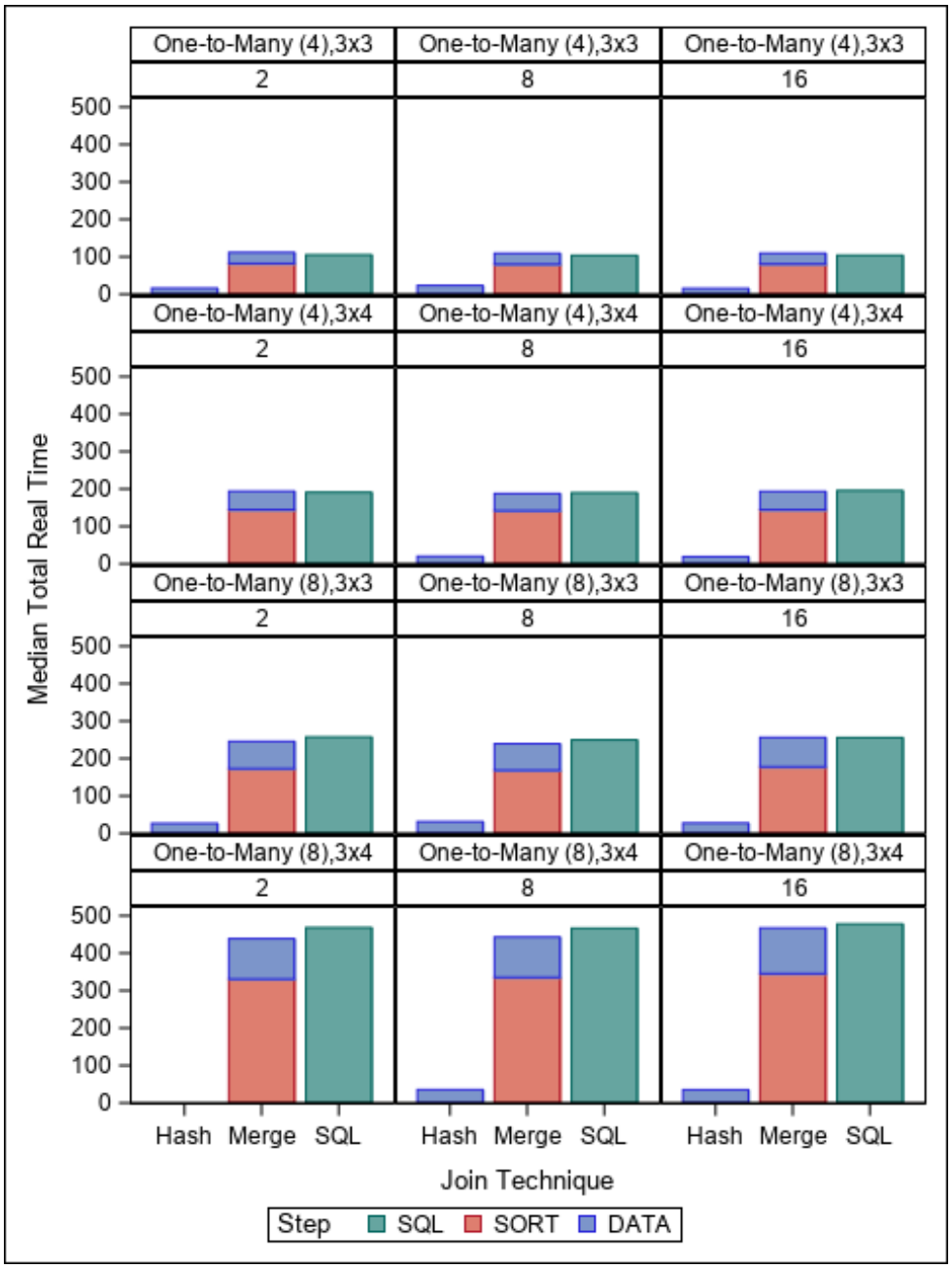

Figure 3: Comparison of median total real time for three join techniques under multiple settings.

However, in all cases note the superiority of the hash object. There is an increase in times as the physical size of the file increases, but the times are drastically reduced compared to the other methods. Compare these results from Figure 3 with those shown in Figure 44 below.

Unlike the total real time, total combined CPU time – which is the sum of system CPU time and user CPU time – follows a different trend. Note that system CPU time follows the same trend shown here while user CPU time is roughly constant within each panel. In Figure 44, it is clear that while the hash object is still superior in terms of total time, its dominance is lessened overall. For the smaller files shown here, there is no practical difference in the median total combined CPU times. However, as file sizes grow, the disparity between the more traditional DATA step approach and the PROC SQL approach become apparent and their difference from the hash object-based technique are noticeable.

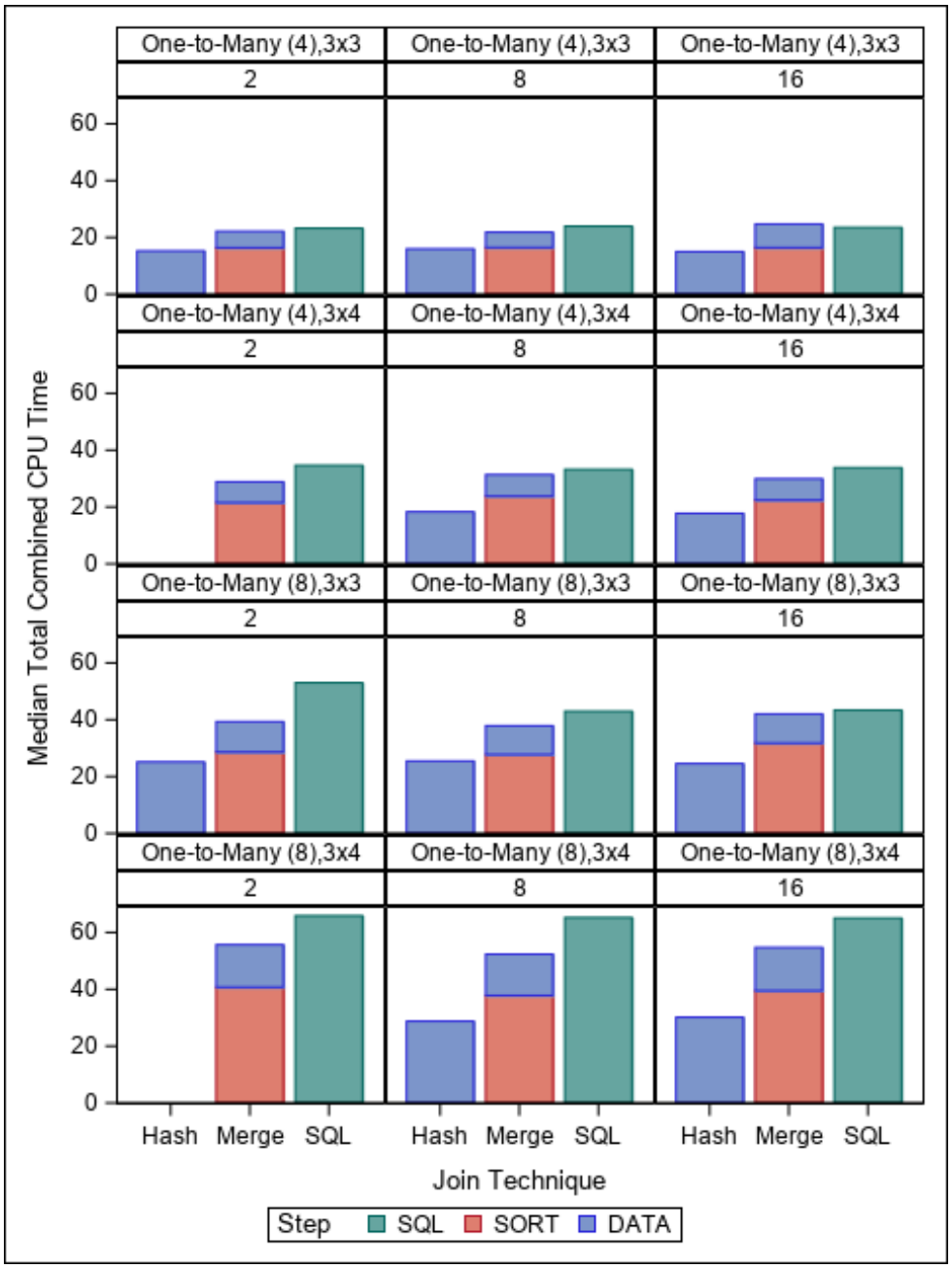

Figure 4: Comparison of median total combined CPU time (system + user) for three join techniques under multiple settings.

The final comparison shown in Figure 5 is in regards to the memory-related metrics: required memory and requested memory. Requested memory is denoted by OS Memory in the FULLSTIMER results and the actual memory required is represented by Memory in the FULLSTIMER results. For these scenarios, the maximum difference in these two memory measurements was 0.03 GB – as such, only the memory actually required by the step is discussed further.

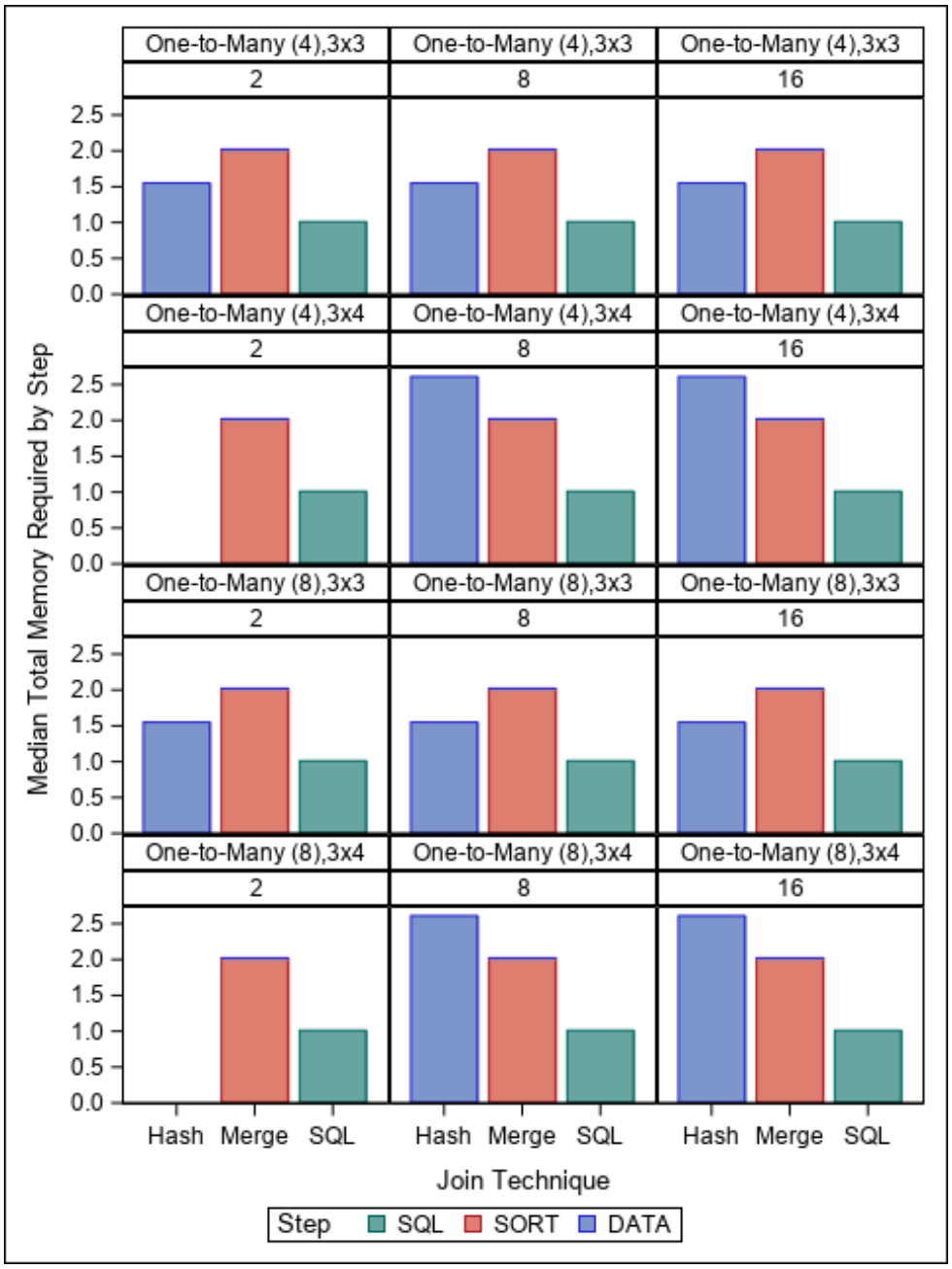

Figure 5: Comparison of median total memory required by the steps in three join techniques under multiple settings.

Figure 5 shows that as the physical size of the file increases, the hash tables require additional memory. Note that the required memory in the One-to-Many (4), 3x3 case for the hash object is constant at roughly 1.5 GB. In fact, for all techniques this value is constant within a row since additional memory-related resources are not utilized here. However, moving to the One-to-Many (4), 3x4 case shows that the hash object required nearly 2.5 GB of memory which is why no results were possible in the 2 GB column – the program produces an error in the log indicating there are insufficient resources to continue processing the DATA step.

CONCLUSION

The results discussed here are similar across the included operating systems, whether data sets began with two million or four million records, and join type. Two potential factors, use of multithreading and impacts of I/O speed, were not considered here because neither were under control of the authors. Further work is necessary to determine if the use of multithreading or techniques to reduce I/O time would significantly impact these results.

In cases where memory is at a premium, SQL is viable alternative but with a cost of longer CPU and real times. As data set size grows the SQL procedure becomes less attractive in favor of the traditional DATA step approach when hash objects are not an option. However, given the substantial differences between the hash-based methods and all other techniques, it is clear that whenever possible that hash objects should be incorporated when carrying out data set joins. There are substantial memory limitations for larger data sets, but given the relatively low cost of memory and the rate at which data set sizes are growing in modern computing applications, there is a clear preference for the hash-based methods.

REFERENCES

Watson, Richann and Mullins, Lynn. 2016. “Exploring HASH Tables vs. SORT/DATA Step vs. PROC SQL.” Proceedings of the PharmaSUG 2016 Conference, Denver, CO: PharmaSUG.

ACKNOWLEDGMENTS

Thanks to Terry Byron at North Carolina State University for providing substantial technical support to allow the storage of the large data sets required for this investigation.

RECOMMENDED READING

- *SAS Hash Object Programming Made Easy by Michele Burlew*
- *Getting Started with PROC DS2 by James Blum and Jonathan Duggins*

CONTACT INFORMATION

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

Jonathan Duggins
North Carolina State University
jwduggin@ncsu.edu
http://duggins.wordpress.ncsu.edu