SAS® GLOBAL FORUM 2016

IMAGINE. CREATE. INNOVATE.

Sorting a Bajillion Records: Conquering Scalability in a Big Data World

Troy Martin Hughes

#SASGF

Sorting a Bajillion Records: Conquering Scalability in a Big Data World

Troy Martin Hughes

ABSTRACT

"Big data" is often distinguished as encompassing high volume, velocity, or variability of data. While big data can signal big business intelligence and big business value, it also can wreak havoc on systems and software ill-prepared for its profundity. Scalability describes the adequate and efficient response of a system to increased data throughput. But because sorting data is one of the most common as well as resource-intensive operations in any software language, inefficiencies or failures caused by big data often are first observed during sorting routines. This text introduces the SAFESORT macro that facilitates *a priori* exception handling routines (which detect environmental and data set attributes that could cause decreased performance or process failure) and post hoc exception handling routines (which detect actual failed sorting processes.) SAFESORT automatically reroutes program flow from the default sorting routine (i.e., SORT procedure, SQL procedure, or hash object) to a divide-and-conquer method that sequentially sorts the data set in chunks. Because SAFESORT does not exhaust system resources like default SAS sorting routines, it more reliably and effectively sorts big data and, in some cases, performs more than 200 times faster than the SORT procedure!

FAILURE OF BASE SAS DEFAULT SORTING ROUTINES

The performance of Base SAS sorting procedures will vary based on data, system, and environmental factors but, in general, the execution time to sort data represents a linear relationship perfectly correlated with file size, depicted in Figure 1. However, as file size continues to increase, performance and efficiency can plummet as sorting rates reach the "inefficiency elbow" demonstrated in Figure 2 , typically the point at which available RAM is depleted and virtual memory must be utilized. And, with even bigger data, sorting procedures may fail outright with runtime errors.

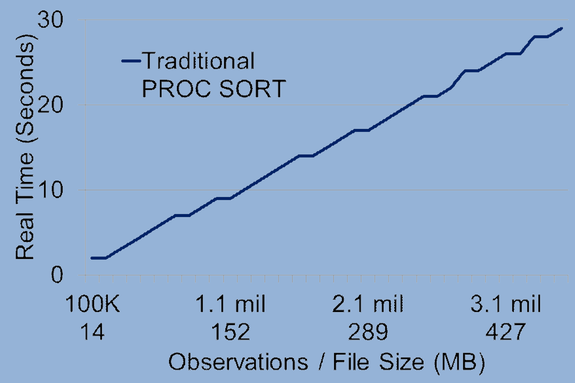

Figure 1. Traditional SORT Procedure

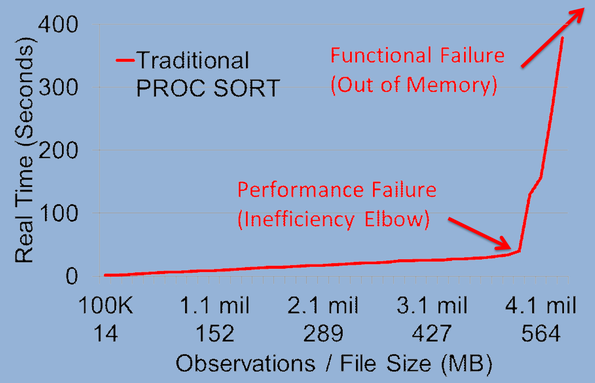

Figure 2. Inefficiency Elbow

SAFESORT SOLUTION

The SAFESORT macro uses a divide-and-conquer method to 1) divide large data sets into smaller chunks (using FIRSTOBS and OBS subset options), 2) individually sort these chunks (in series), and 3) recombine these chunks in order to produce a sorted data set. Because SAFESORT performs smaller sorts in series, it uses fewer system resources and thus is impervious to the inefficiency elbow as well as functional failure caused by memory errors. Given the inefficiency elbow in Figure 2 occurring while sorting approximately 550 MB size (and larger) files, SAFESORT is invoked at a 550 MB threshold. Figure 3 demonstrates that at 4.3 million observations, SAFESORT is over 5x faster than the SORT procedure, taking only 70 seconds compared to 379 seconds. And, Figure 4 demonstrates that at 4.5 million observations, the SORT procedure requires 2 hours 20 minutes while SAFESORT only 74 seconds, over 200 times faster!

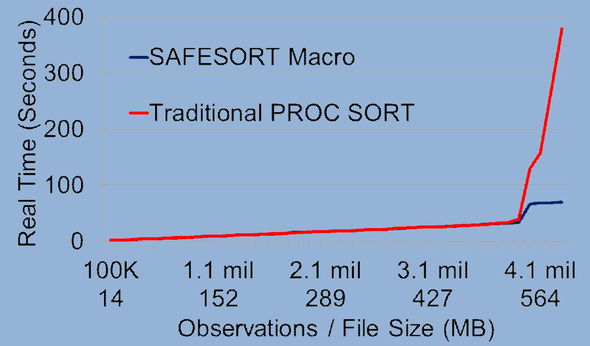

Figure 3. Improvement with SAFESORT

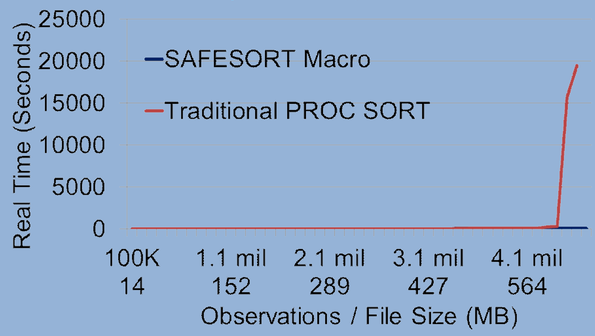

Figure 4. Improvement with SAFESORT

SAFESORT BENEFITS

The SAFESORT macro is described in detail in the accompanying white paper but benefits include:

- Dramatically increased performance and execution time due to more efficient utilization of system resources.
- Dramatically increased functionality and reliability due avoidance of out-of-memory runtime errors.
- *A priori* exception handling to detect the inefficiency elbow BEFORE it occurs and shift to SAFESORT method.
- Post hoc exception handling to detect failed standard sorts and automatically shift to divide-and-conquer method.
- Ability to invoke SAFESORT to utilize the SORT procedure, SQL procedure, or the hash object.
- Straightforward, single-line macro invocation.

```
%macro SAFESORT (meth1=SORT, meth2=SORT, dsn=bigdata1, dsnout=bigdata2, sortvars=char1, thresh=550);
```