Solid State Drives And The Sort-Merge

Dwight A. Haworth, University of Nebraska at Omaha, USA

ABSTRACT

This paper discusses the history of the sort-merge routine and the impacts of hardware limitations on the performance of sort-merge processing. The results of comparing a single-step sort-merge with a two-step sort-merge in a hard-disk drive (HDD) environment are presented to show that a two-step sort-merge can reduce total processing time. An evaluation is made of the total transfer time of three sort-merge variations without reference to seek time or rotational delay. This evaluation prepares the statistics for application to the solid-state drive (SSD) environment, and the conclusion is that sort-merge routines that are optimized for the HDD environment are sub-optimal if applied to the SSD environment. In addition, the sizes of the work files used by the three sort-merge routines are analyzed, and it is demonstrated that sort-merge routines that are optimized for the HDD environment will generate unnecessary wear if applied to the SDD environment. Further, it is demonstrated that the key sorting routine should be preferred over the other sort-merge routines in a SSD environment.

Keywords: Sort-Merge; Key Sorting; Solid-State Drives

INTRODUCTION

Beginning with tape drives and early disk drives up to the appearance of economically practical solid state drives, the performance of sort-merge routines has been an item of concern. To justify this concern, Knuth (1998, p. 3) gives examples that range from 25% to over 50% of computer time being taken up with sorting. Sort-merge is employed whenever the file being sorted is too large to fit entirely into memory at one time. Sort-merge performance has been bound up in the amount of RAM available and the performance of the storage devices holding the original file and the intermediate runs produced in the Sort-Merge process. As a result, a number of processing schemes have been devised to wring the most performance out of hard disk drive (HDD) technology.

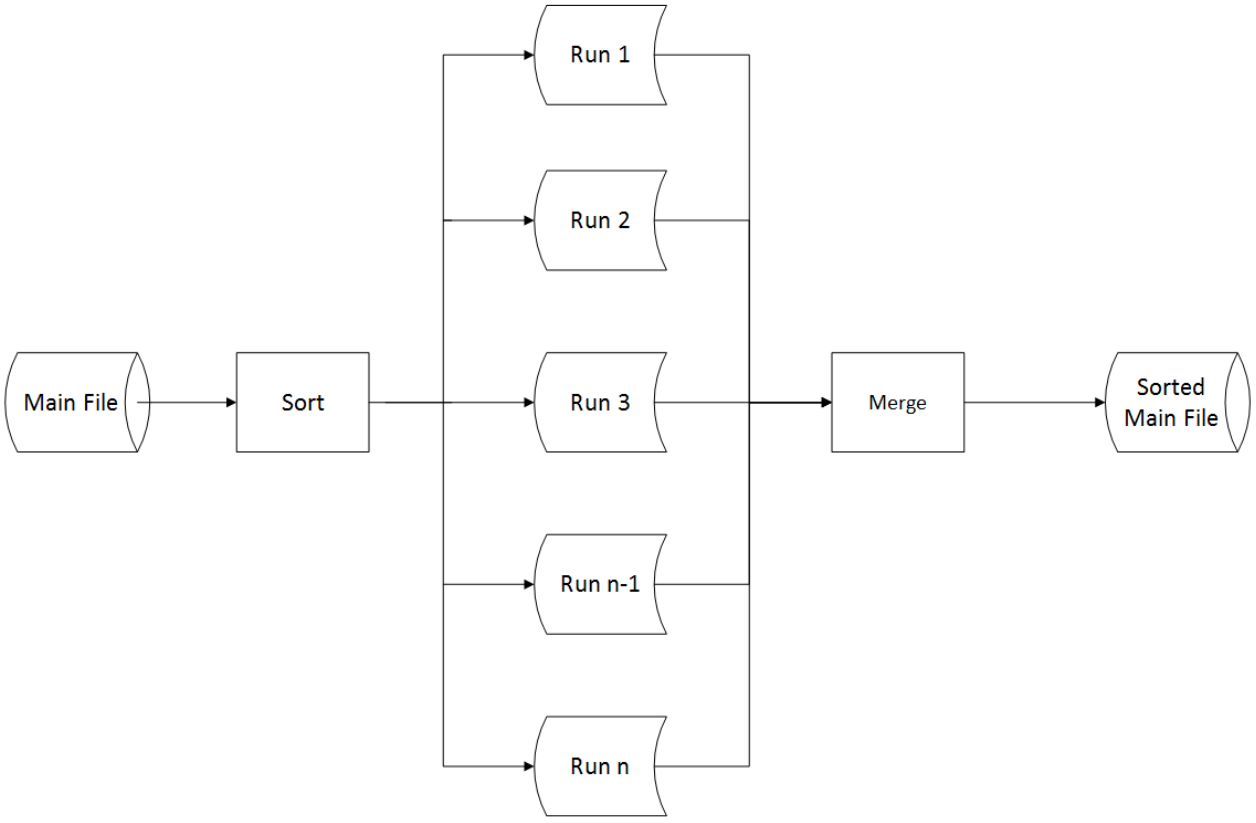
The solid state drive (SSD) has emerged to supplant the HDD. The major problem of the SSD is wear and has been well documented in *Solid-State Disks: Coming to a System Near You* (Ruth, 2008), among others. The problem achieved a headline in September, 2013 when Computerworld announced "SSDs do die, as Linus Torvalds just discovered" (Mearian, 2013, p.1). Mearian (2013, p. 1) implies that this wear problem manifests itself with such dramatic effect when the habits and processes of the HDD environment are applied to a SSD environment.

The purpose of this study is to establish qualitative values for the practice of using HDD sort-merge procedures on SSDs and to demonstrate the gains possible by using other procedures. This study will review the sort-merge, key-sorting, and the multi-step merge. Next, the effect of solid state drives on the various sorting routines is examined to establish a qualitative view of the differences between the HDD and the SSD, and that is followed by a brief qualitative summary.

SORT-MERGE

The sort-merge routine has a number of variations. Figure 1 shows a general sort-merge process. It may be implemented with tape storage or disks. Further, it requires only sequential access to the storage devices. An optimized sorting process in the first phase minimizes the run time, and with no seeking being required, the only other component of the total time is the file transfer time (Tf). To elucidate this transfer time, follow the arrows in Figure 1. The full file is read into the sort process once (Tf); the full file is written out to working storage as runs (Tf); the full file is read into the merge process (Tf); and finally the full file is written in sorted sequence to storage (Tf). The total transfer time for this whole process is 4 Tf .

Figure 1. The Basic Sort-Merge.



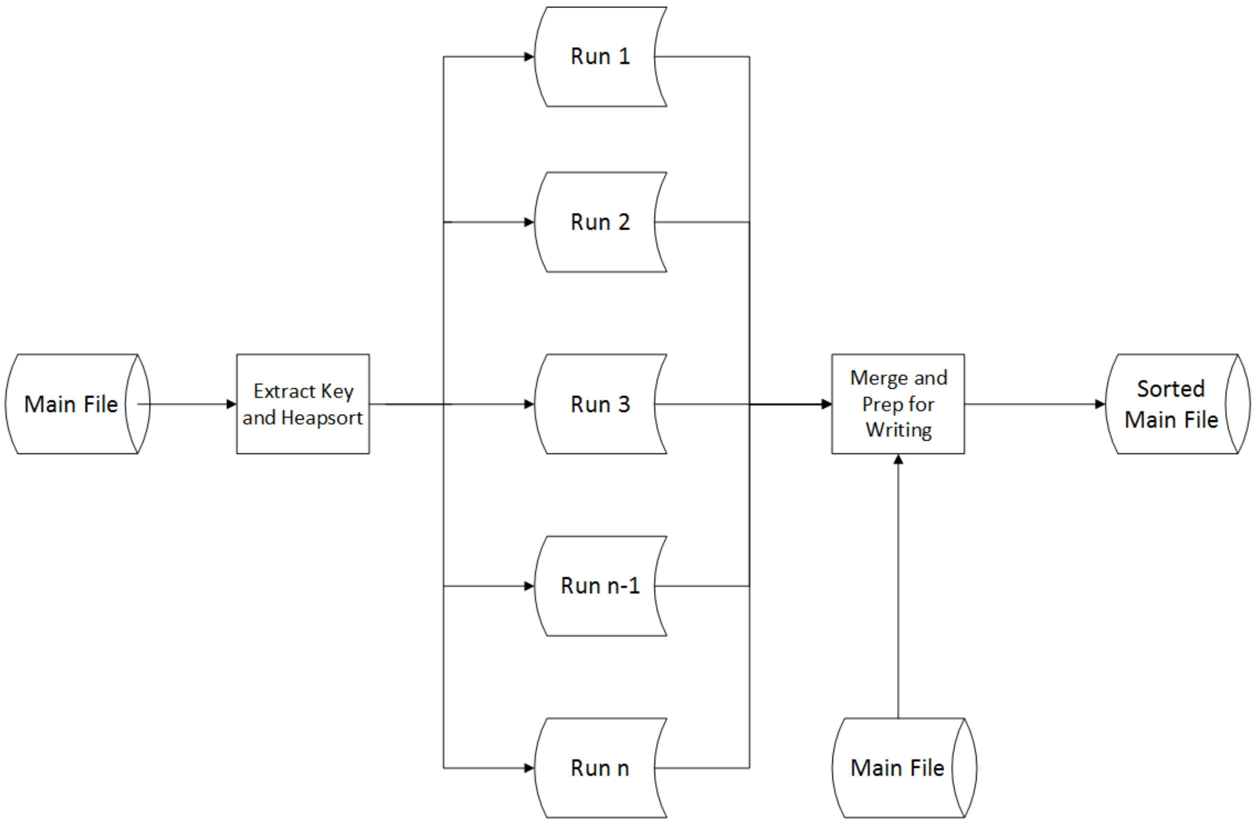
KEY-SORTING

Many of the routines used in sorting may be traced back to the early days of computing when various clever schemes were employed to overcome the limitations of the available hardware. Multi-step merges and key-sorting are two of those schemes. Multi-step merges are useful in overcoming limitations of main storage, and Schick (1963) resorted to a key-sorting scheme to overcome the memory limits of the IBM 1401. Today's students of file structures often forget that the main memory of the IBM 1401 was either 4000, 8000, 12000, or 16000 characters ("The IBM 1401", 2015). Key-sorting uses only keys and the disk address of the corresponding record, thereby allowing more records of the original file to be represented in the limited main memory and in fewer work files.

The other characteristic of the IBM 1401 that is often forgotten is that the computer was single-tasking; there was no need to worry about another task moving the heads in-between the IO operations of the sort-merge job. The consequence of that fact was that, with proper placement of all files in the same or adjacent cylinders, seek time was minimized. In the early 1960's, the worrisome component of disk access was rotational delay, as evidenced by Schick's (1963) attention to it and his development of an interleaving scheme to minimize rotational delay.

Figure 2 shows Schick's (1963) key-sorting process. Again, the full file is read into the sort process (Tf). Now the outputs are runs of keys with associated main storage addresses, much smaller than full data records and with proportionately much smaller transfer time (Tk). The runs of keys are read into the merge process (Tk) and the results are used to fetch the associated records (Tf) and write them in sorted sequence to storage (Tf). The total transfer time for this process is 3 Tf + 2 Tk. The assumption is that 2 Tk is much smaller than Tf, and the additional cost of retrieving the records does not exceed the savings gained in handling only the keys. Of course, this assumption depends on the performance of the secondary storage, and in the context of the hard-disk drive (HDD), the assumption fails (Folk & Zoellick, 1992, p. 212). The extra seeking needed to retrieve the full-data record exceeds the savings generated by handling only keys and addresses, making it impractical in a HHD environment. Key sorting will be revisited in the context of a SSD environment.

Figure 2. Key-sorting Process (Schick, 1963).

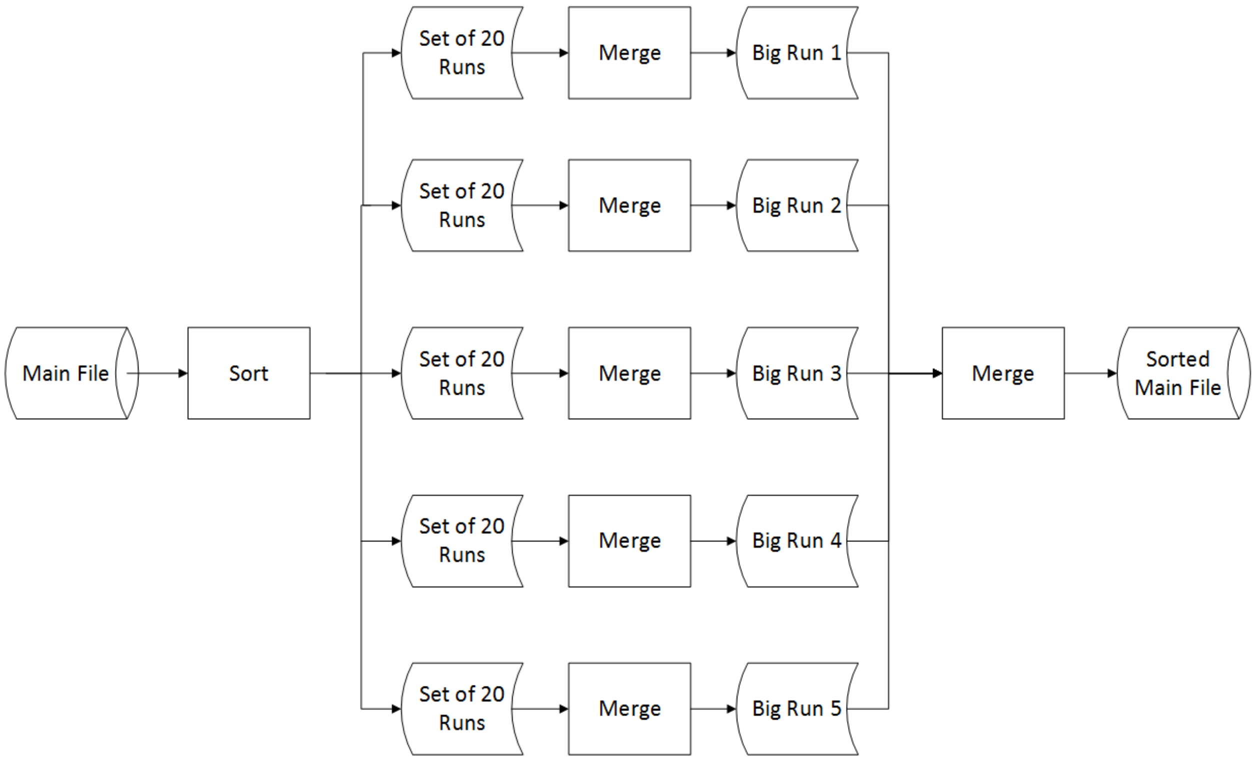


MULTI-STEP MERGE

Once a multiprogramming environment is entered, seek time becomes the troublesome component of access time. Folk and Zoellick (1992, pp. 290-298) work out an example that shows, in the context of a large file, first a one-step merge, as shown in Figure 1, and then a two-step merge as in Figure 3. Following the flows in Figure 3, the transmission time of the two-step merge includes the following components. First, the full file is read into the sort process (Tf); next the full file is written to the runs (Tf); then the full file, as sets of small runs, is input to the first merge step (Tf); the full file is then written to the big runs (Tf); the full file is read from the big runs into the second merge step (Tf); and finally, the full file is written to the sorted main file (Tf). The total transmission time for this 2-step merge is 6 Tf.

Under the scenario posited by Folk and Zoellick (1992, p. 296-298), the difference between the one-step merge and their two-step example is an 81.5 percent reduction in seek time and a 100 percent increase in the transmission time used in merge operations. Because transmission time is so small in relation to seek time, the net effect is to reduce the overall processing time by 69.2 percent. This is an effective argument for a two-step merge. Folk and Zoellick (1992, p. 297) also conclude that with the two-step merge of their example, seek time and transmission time are close enough that any further reduction in seek time will be offset by the increase in transmission time, as would be the case if a three-step merge were attempted. IBM (2012 a) reports a similar conclusion, "In general, using more than three work data sets does not reduce elapsed time any further...."

Figure 3. 2-Step Sort-Merge (Folk & Zoellick, 1992)



Considering all of the foregoing, the point to be recognized is that a sort-merge optimized to run in a spinning disk environment will probably have used the trade-off between seek time and transmission time and employed at least a two-step merge, like that demonstrated by Folk and Zoellick. IBM (2012b) implies a similar approach in their DFSORT utility when they acknowledge "The Blockset technique might require more intermediate work space than Peerage/Vale [another of IBM's sorting techniques]." In the analysis and comparisons below, the 2-Step Sort-merge is the HDD standard for sorting large files.

SOLID STATE DRIVES

When seek time and rotational delay are removed from the processing parameters by using Solid State Drives (SSDs), then the additional transmission time of the multi-step merge is for nothing. Moreover, the additional intermediate runs used in a multi-step merge will increase the wear on the solid state drive. Consider a simple one-step merge as in Figure 1; it will use three times the space of the original file: the original file (Sf), the space of sorted runs that will be input to the merge (Sf), and the final sorted output (Sf). This makes the space required by a one-step merge 3 Sf. A two-step merge will create an additional copy of the data as intermediate, partially-merged runs, thereby adding a fourth copy of the original data to the usage of the drive space, making the total 4 Sf. Each additional step in the merge will produce one additional copy of the original data. When switching from HDDs to SSDs, sort-merge routines must be modified to single-step merges to reduce the transmission time and the unnecessary wear on the SSD.

But something better than the one-step merge is possible. At this point, return to the key sorting process. Recall from the earlier analysis, keysorting time is 3 Tf + 2 Tk, and this time is less than the 4 Tf of the basic (one-step) sort-merge because there is no seeking to offset the savings in transfer time. Referring to Figure 2, observe that the key-sorting process uses the space of the original file (Sf), the space to store the runs of keys and addresses (Sk) (less than the full file), and the space for the final sorted file (Sf) for a total of 2 Sf + Sk. And the best part of all, 2 Sf +

Sk is less than 3 Sf. Thus, the key-sorting process produces less wear on the SSD than either of the two other routines considered here.

SUMMARY

The above discussion presents several sets of performance parameters for various types of sort-merge routines. Seek time and rotational delay have been ignored because they do not exist in SSDs. These parameters are collected into Table 1 for easy comparison.

Table 1. Storage Space and Transmission Time of Various Sort-Merge Routines

Routine	Space Used	Transmission Time
Basic (one-step) Sort-Merge	3 Sf	4 Tf
Key-Sorting	2 Sf + Sk	3 Tf + 2 Tk
2-Step Sort-Merge	4 Sf	6 Tf

The bottom line for the practitioner is that in an environment of SSDs, legacy sort-merge routines will incur unnecessary processing time and wear on the SSDs. Sort-merge processes should be revised to key-sorting processes to minimize transmission time (and the resulting run time) and to reduce wear on the drives.

AUTHOR BIOGRAPHY

Dwight A. Haworth received his B.S. degree from the United States Air Force Academy, CO, in 1963. He retired from the United States Air Force in 1981. He received his Ph.D. in Management Information Systems from Texas Tech University, Lubbock, TX, in 1990. His research interests are information assurance and systems development and performance. E-mail: haworth@unomaha.edu

REFERENCES

Folk, M. J. & Zoellick, B. (1992). *File Structures* (2nd ed.). Reading, MA: Addison-Wesley.

IBM (2012 a). z/OS DFSORT Application Programming Guide, Number of Devices. Retrieved May 13, 2015 from http://www-01.ibm.com/support/knowledgecenter/SSLTBW_1.13.0/com.ibm.zos.r13.icea100/ ice1ca61303.htm% 23wq1426.

IBM (2012 b). z/OS DFSORT Application Programming Guide, Specify Efficient Sort/Merge Techniques. Retrieved May 13, 2015 from http://www-01.ibm.com/support/knowledgecenter/SSLTBW_1.13.0/ com.ibm.zos.r13.icea100/stech.htm%23stech.

Knuth, D.E. (1998). *The Art of Computer Programming, v. 3* (2nd ed.). Boston, MA: Addison-Wesley.

Mearian, L. (2013). SSDs do die, as Linus Torvalds just discovered. *Computerworld*, September 12, p. 1. Retrieved May 13, 2015 from http://www.computerworld.com/article/2484998/solid-state-drives/ssds-do-die--as-linus-torvalds-just-discovered.html.

Ruth, G. (2008). *Solid-State Disks: Coming to a System Near You*. Midvale, UT: Burton Group.

Schick, T. (1963). Disk File Sorting. *Communications of the ACM*, 6, 330-331, 339.

NOTES