External Sorting Algorithm: State-of-the-Art and Future Directions

Wenhan Chen¹, Yang Liu¹, Zhiguang Chen²*, Fang Liu², Nong Xiao¹,²

¹College of Computer, National University of Defense Technology, Changsha, Hunan, 410073, China
²School of Data and Computer Science, Sun Yat-sen University, Guangzhou, Guangdong, 510000, China

*Zhiguang Chen’s e-mail: zhiguang.chen@nscc-gz.cn

Abstract. The advent of the era of big data provides new opportunities and more challenges to sorting algorithms. The traditional internal sorting algorithm cannot adapt to the explosive growth of data, and the memory cannot accommodate all the data for sorting, so the external sorting algorithm arises at the historic moment. Because of the different application scenarios, storage devices and improvement strategies, there are many kinds of external sorting algorithms. Traditional main memory architecture based on DRAM faces the problems of capacity, energy consumption and reliability. Emerging nonvolatile memory technologies are non-volatile, high-density, byte-addressable, low-power, so they can replace persistent storage, main memory or storage class memory. Though NVM devices provide new choices to the revolution of traditional memory and storage system, traditional external sorting algorithms cannot achieve its performance. This paper first sorts out the development of external sorting algorithm, and summarizes it into four kinds of external sorting algorithm based on HDD, embedded device, SSD and NVM. In addition, the classical external sorting algorithms based on different storage devices are listed, and our opinions are put forward. Finally, this paper proposes three problems that need to be solved urgently in the future development of external sorting algorithm.

1. Introduction

With the large-scale application of sensor devices and the blowout growth of mobile Internet devices, more and more data are generated nowadays. Field data plays an important role in guiding industrial applications, and many related technologies, such as edge computing, cloud computing and so on, emerge as the times require. The rapid processing of huge amount of data has gradually become the bottleneck of data management. Sorting algorithm is one of the most basic algorithms in data management system. The speed of sorting algorithm directly affects the efficiency of data management system. After more than ten years of development of sorting algorithm, the development of internal sorting is more mature. However, with the increase of the amount of data, internal sorting has been unable to meet the current sorting needs of the amount of data. Thus, an external sorting algorithm is proposed. External sorting is mainly used in the scenarios where the amount of data is too large to be loaded into limited memory at one time. When the amount of data is particularly large, due to the limited memory capacity, it is impossible to sort all the data in memory at once, so it is necessary to use external sorting. External sorting is an I/O-intensive algorithm that produces a large number of I/O operations on external memory devices, thus affecting the performance of data processing. Nowadays, the amount of data is exploding, and processing data at the TB and PB levels has become a reality [1].
By 2025, the total amount of data is estimated to reach 185ZB [2]. The capacity of memory has been unable to keep pace with the growth of the data explosion. For data-intensive applications, it is an urgent problem to process a large amount of data quickly. However, at present, no scholar has summarized the research on the external sorting algorithm. In order to facilitate the relevant researchers to do further research quickly, and to point out the direction for the researchers in this field for the first time, this paper collates and summarizes the research on external sorting algorithms in recent years and summarize our direction and content. This is also the first summary of the external sorting algorithm in the past 30 years, which is of great significance to the data management system.

2. Definition of external sort
When the amount of data is particularly large, due to the limited memory capacity, it is impossible to sort all the data in memory at one time, so it is necessary to use external sorting. In 1973, Knuth et al. [3] provided a lot of basic knowledge of sorting algorithm, including the replacement selection and merge sort. Merge sort is the algorithm most commonly used for external sorting. External merge sorting is generally divided into two phases: run formation phase (see figure 1) and run merge phase (see figure 2). Run formation phase means the formation of initial runs. Replacement selection is a popular run formation method. Run merge phase means the merge of the runs into a sorted file [4].

Run formation phase generally has two modes: load-sort-store and replacement selection. In load-sort-store mode, the input data is loaded into memory and sorted, and the sorted data is written to the external memory device to form a run. The length of the run generated in this way is generally the same, except for the last run.

Replacement selection has the following advantages over load-sort-store algorithm. First, it can generate longer runs, averagely generating runs twice the size of memory, thereby reducing the number of runs generated and the number of merge passes [5]. Moreover, replacement selection can take advantage of the partial order of the data to generate a longer run than when the input data is randomly distributed. Besides, replacement selection can overlap CPU processing and I/O, which reduce elapse time, but there exists frequent cache miss problem.

I/O time normally determines the elapsed time for external merge sort. So many external sorting research focus on optimizing algorithms to reduce I/O operations.

3. HDD-based external sorting
The traditional research of external merge sorting is mainly based on hard disk drive (HDD). The performance of read and write operations are symmetric, but sequential I/O operations are more efficient than random ones. The main research direction of disk-based external sorting is to reduce I/O
operations to reduce the random read access to external memory devices, reduce seek time and rotational latency.

In 1996, Luoquan Zheng et al. [6] proposed two strategies: interleaved layout and a new reading strategy, to speed up the execution of external merge sorting and improve the performance of external merging sorting based on disk. Interleaved layout reduces the seek time of the disk during the merge phase by placing the blocks of different run in a continuous position on disk. A new reading strategy calculates block consumption sequence, and then allocates extra buffer in memory to prefetch data in that order, thereby reducing seek overhead and read time. Block consumption sequence is the order in which blocks are processed during the merge phase, which can be formed by sorting the maximum key values for each block during the run formation phase. A new reading strategy reduces seek overhead and read time by calculating block consumption sequence and prefetching data to memory buffer in that order.

Traditional merge sorting in the merge phase can only load one block at a time, and cannot prefetch the data, so it cannot overlap CPU process and I/O operations. Forecasting can prefetch the data during the merge phase. In run formation phase, keep the last key of each block, to determine which block will be emptied first. During the merge phase, an extra buffer is set aside in memory to prefetch next block from the run which block was emptied first. Only one input buffer is available for reading, so the disks cannot work in parallel.

Double buffering [7] can better overlap CPU process and I/O operation, which is a space-for-time strategy. During the merge phase, half of the buffer in memory is used as input buffer, and the other half of the buffer is used for prefetching the next block from each run on disk. However, double buffering does not know in advance which block in the input buffer will be emptied next. As a result, the eager block needs to wait for the previous block to finish processing before it can be loaded into memory for processing, so the algorithm is not very efficient.

AlphaSort [8] is a fast, cache-conscious disk-to-disk sort. It uses QuickSort to generate runs and uses replacement-selection to merge the runs, which has good cache performance and overlaps CPU process and I/O operations. Per-Ake Larson has proposed a cache-conscious version of replacement selection [9], called batch replacement selection, which is similar to AlphaSort. Traditional replacement selection can only handle fixed-length records, but batch replacement selection can handle variable-length records through better memory management. Batch replacement selection creates small run in memory and merges them to generate output runs, and both input buffer and output buffer are processed in batches. Overall, batch replacement selection can solve the problem of cache miss very well, reduce the number of comparisons, and run faster than the traditional replacement selection algorithm.

Md Rafiqul Islam et al. [10] proposed a new external sorting algorithm, which does not need to make use of additional disk space and generate no intermediate data. The algorithm makes use of fast sorting and special merging technology, which can reduce the number of comparisons in the sorting process and has a good performance.

4. External sorting for flash memory

Compared to traditional storage media, flash is non-volatile memory introducing many advantages: shock-resistance, fast read access, low production cost and power efficiency. There are two types of flash memory, NOR and NAND [11].

Four critical issues have a significant impact on the efficiency of external sorting in flash memory: (1) write once with bulk erase, (2) the endurance issue, (3) asymmetrical read and write operations, and (4) no mechanical latency. Flash memory has no disk head and disk platter, no seek or rotational latency will be needed. Therefore, random access and sequential access to flash memory will not cause mechanical latency problems [12].

4.1. External sorting on flash-based embedded devices

In recent years, with the rapid development of flash technology, the capacity of flash memory has
increased and the price has decreased a lot. Flash memory is widely used in mobile devices, embedded devices and sensing devices. Many embedded devices need to store and calculate huge amount of data. Embedded devices process data internally more efficiently and faster than sending data over the network for analysis. However, the biggest challenge of external sorting in embedded devices is its limited memory space and very few CPU resources.

Fsort [13] is an external sorting algorithm on embedded flash-based sensor devices with small memory capacity. Flash memory has the characteristics of low access delay, power efficiency and shock-resistance [14]. However, the cost of writing access (about 800us) on flash memory is higher than that of reading (about 60us). And the number of times a block can be written is limited (between 10000 to 100000times). This algorithm can minimize the write-delete operation on flash memory. Fsort uses a top-down replacement selection algorithm in run formation phase to generate a longer run, with an average size of twice the memory size. Generating a longer run makes it more likely to decrease the number of runs generated and the number of merge passes, thus reducing the number of I/O operations. The paper compares the performance of online sorting and offline sorting in time and energy consumption, and the effect of online sorting is 2-3 orders of magnitude worse than that of offline sorting. The algorithm is improved based on offline sorting.

FAST [15] is an external sorting algorithm applied to NAND-flash based mobile database system. Because the write cost of flash memory is much higher than the read cost, the algorithm reduces write operations to flash memory by increasing additional read operations, thus improving the execution time and response time, as well as prolonging the lifetime of flash memory. In the merging phase, FAST scans the input data several times and constructs the maximum tree to get the first m minimum values, and the values are sorted and written directly to the output file. Until all the input data is processed, the algorithm terminates. The algorithm also needs to keep the max key value written out and its record number, to skip the data that has been processed in the previous iteration. To our opinion, this algorithm may work better in NVM. By taking profit of byte addressability, we only need to index the last processed data for each run to avoid reloading the processed data into memory again.

Similarly, Flash Minsort is also external sorting algorithms specifically for embedded devices based on flash storage system [16]. Flash Minsort combine run generation and merge phase into a single phase. The main idea of the algorithm is to retrieve only the tuples required by random reads instead of sequential scans. The input data is statically divided into different regions, and the smallest value in each region is stored to form a minimum index. The algorithm reads only the region containing the smallest value, and output the record, and then update minimum index. When all values are output to the external memory device in order, the algorithm is terminated. Performing the algorithm depends on the distribution of the input data. If the number of different values in the input data is particularly large, the effect is not ideal. When the key values of the input data are clustered, the performance is better. However, the base version of flash Minsort does not adapt to the input relation size. In 2013, Ramon Lawrance et al proposed a modified version of Flash Minsort which are adaptive to memory limits [17]. The number of regions can be adjusted dynamically when the actual size of the input data is unknown. Once the memory limit is reached, the algorithm dynamically constructs the minimum index by merging adjacent regions.

Standard external merge sorting, merging in limited memory requires at least three memory buffers, but the algorithm proposed by Riley Jackson [18] can use only two memory buffers in memory to complete the merge operation, at the expense of more data comparison operations and data movement operations. This is very effective in embedded devices with minimal memory capacity.

4.2. external sorting on flash-based SSD

With the development of flash memory technology, flash-based solid-state drives (SSDs) has been widely used in various fields, as an alternative to HDDs. Compared with HDD, SSD has high I/O bandwidth and short access latency. Moreover, flash-based SSD has the characteristics of good internal parallelism. The performance of sequential read and random read of are symmetrical, but not the performance of read and write operations are [19].
YANG LIU et al. [20] put forward an external sorting algorithm on flash memory via natural page run generation. The algorithm speeds up the running generation phase by taking advantage of the partial sorting nature of the data. The algorithm finds pages without overlapping value intervals, which form naturally occurring page runs. Figure 3 shows an example of a naturally occurring page run of 3 pages in partially sorted data \((P_1, P_2, P_3)\). The algorithm writes the index entries for the pages of naturally occurring page run, instead of sorting it and writing it back to memory, which reduces writes to flash memory. In this paper, a fast heuristic method is proposed to find a high percentage of naturally occurring page runs using a minimum of computational overhead. Firstly, separation distance loading of pages increases the chance of finding naturally occurring page run. Secondly, find non-overlapping intervals using a minimum interval tree (Min tree) implemented by red and black tree. Thirdly, if there are no enough non-overlapping pages can be found to form a naturally occurring page run, remove long intervals using an interval length tree (Length Tree) to form a normal run. In the merge phase, pages of naturally occurring runs are loaded into RAM buffer and sorted before participating in the merging process. However, the performance of the algorithm depends on the distribution of the input data. If the input data is partially ordered, the performance of the algorithm is better. Moreover, the algorithm assumes that it merges in a single pass, and the case of multiple merge passes is not discussed.

![Figure 3. Example of a naturally occurring page run of 3 pages.](image)

MONTRES [21] takes advantage of SSD performance to accelerate the sorting process. MONTRES mainly to solve the problem that other external sorting algorithms do not work well when the amount of input data is too large and the memory capacity is too small. During run formation phase, ascending block selection and continuous run expansion to generate larger runs are used to optimize the algorithm. Besides, by introducing a merge on-the-fly mechanism in run formation phase, the algorithm continuously writes some small values directly to the final output file, thus reducing the generation of temporary data and reducing I/O amount. During run merge phase, MONTRES proceeds in a single pass by indexing the minimum value of each block and using a min-max heap to retrieves the small values. The performance of the algorithm is closely related to the distribution of input data. If the input data is partially ordered, the performance will be much better.

FMsort [22] takes advantage of the short access delay and high random I/O bandwidth of flash-based SSD. Flash-based SSD has the characteristics of good internal parallelism [23]. The algorithm can overlap CPU process and I/O operations very well. In run formation phase, FMsort calculates a block read order by sorting the minimum value of each block for each run, which is the order of blocks needed in the merge phase. In the merge phase, the next L blocks are prefetched according to the block read order, which overlaps CPU process and I/O operations, ans speeds up the merge phase.

Yaron Kanza et al. [24] proposed an external sorting algorithm for flash-based SSD. The algorithm can reduce cell wearing, improve the efficiency and prolong lifetime of the flash memory, by reducing the intermediate write operations. The algorithm is mainly used in the case that the input data is only a little larger than the memory, and the input data is partially ordered.

The ActiveSort [25] proposed by Young-Sik Lee makes use of Active SSD [26][27]. Active SSD is a special kind of SSD. Active sort offloads the merge operation to the SSD, which can reduce the number of I/O operations to preserve SSD lifetime. In run formation phase, the input data is loaded into memory and sorted to form a run, and written to SSD. When the host sends a standard read
request to read the final sorted data, Active Sort creates the sorted output on-the-fly within the active SSD and sends the sorted results to the host side. The application of SSD into Hadoop MapReduce, can take advantage of the high network bandwidth of SSD and increase parallel storage access, thus speeding up the execution of Hadoop MapReduce [28]. Integrating ActiveSort into MapReduce framework can improve the performance of MapReduce framework and prolong lifetime of SSD. Active sort can speed up the data processing process of data-intensive applications based on Hadoop framework. However, this paper does not discuss how to optimize merge phase when the number of runs is greater than the number of chunks, nor does it discuss how to better place data on the active SSD, so as to reduce the time extension caused by channel congestion.

5. External sorting on NVM
Emerging memory technologies has developed rapidly in recent years. Emerging nonvolatile memory (NVM) technologies such as Phase-Change Memory (PCM), Spin Torque Transfer Magnetic RAM (STT-RAM), and Memristor-based Resistive RAM (ReRAM) offer the promise of significantly lower energy and higher density (bits per area) than DRAM [29]. Non-volatile memory (NVM) has the following advantages: high-density, high-speed, low-power, random-accessibility, non-volatility [30]. However, unlike DRAM, there exists endurance problems in NVMs (a cell wears out after $10^8$-$10^{12}$ writes) [31]. There is a significant asymmetry between read and write costs [32], which requires NVM-based algorithms to minimize write operations.

Phase change memory (PCM) [33] is a byte-addressable memory that has higher density and better endurance than NAND flash memory. However, compared with DRAM, there is a higher access delay and write operation is more expensive. In 2018, Arezki Laga et al proposed MONTRES-NVM [34], an external sorting algorithm based on hybrid memories that consist of PCM and DRAM. MONTRES-NVM has made optimization based on MONTRES. MONTRES-NVM uses a small amount of DRAM to speed up the sorting process and minimize writes to NVM. This paper mainly makes use of byte addressability of PCM to detect already sorted sequences in the input data and treat the sorted sequences as runs, so as to reduce the write operations to PCM.

The algorithm uses the merge on-the-fly mechanism of MONTRES to write the small values directly to the final sorted file, so as to reduce the generation of intermediate data. MONTRES can merge in a single pass. In the merge phase, the minimum value is retrieved by constructing a minimum heap and some small values are written to the final output file. The performance of the algorithm is closely related to the distribution of the input data. When the input data is partially ordered, it can play a very good performance.

6. External sorting with other technologies

6.1. External sorting with compression
Data compression technology is also often used in external sorting. Data compression can increase the information density of transferred data, and can relieve the I/O bottleneck of high performance database management system [35]. John Yiannis et al. [36] apply data compression to external sorting algorithms, through better management of memory space, which can accelerate the execution of external sorting, reduce the cost of data transmission, and generate a smaller number of runs. During run formation phase, first load the key values of the records into memory and sort them, then compress the records, and write the generated run to the external memory device. In the merge phase, the data is loaded into memory from each run for merging, and the data is decompressed and output to the external memory device. The performance of three data compression techniques applied to external sorting is also compared in the experiment: Canonical Huffman coding of bigrams [37], Bytewise bigram coding, Bytewise common-quadgram coding. In this paper, it is mentioned that for small-scale data, the speed of data compression and data decompression is the main factor affecting external sorting; however, the amount of data compression is the key factor to determine the sorting time for large-scale data.
6.2. Distributed external sorting
Multiway merge sort is a good algorithm for parallel external sorting. Two algorithms based on the multiway merging algorithm are proposed for sorting a large amount of data on massively parallel machines by Mirko Rahn et al [38]. Merge sort with global striping minimizes I/O operations and works with just two passes over large-scale input data. Canonical Merge sort algorithm has close to minimal communication overhead and uses more conventional specifications of the result.

DEMSort is a distributed external memory sort proposed in 2009 [39]. The algorithm consists of three phases: run formation, distribution, local merging. It supports at the same time distributed-memory parallelism and shared-memory parallelism, which work well in sorting huge volumes of data.

The distributed external sorting algorithm is also modified based on bucket sorting (an internal sorting) [40]. The input data is divided into different sets with non-overlapping intervals, and the interval of each bucket is defined. The data in each set is then distributed to different buckets for internal sorting. Finally, the ordered data of each bucket form the final ordered file together.

7. Conclusion
To conclusion, traditional external sorting research is mainly based on disk. The main optimization direction is to place the data properly, and to reduce the random read access to external storage, thus reducing seek time and rotational latency. The read and write overhead of flash memory is asymmetric. The cost of writing is much higher than that of reading, and the number of times a block can be written is limited. The research of external sorting based on flash memory is mainly to reduce the write operations to flash memory by increasing additional read operations, and even to avoid the generation of intermediate data. Some methods manage to optimize external sorting based on SSD. The algorithm mainly makes use of the high random I/O bandwidth and internal parallelism of SSD to overlap CPU process and I/O operation, so as to accelerate external sorting. Nowadays, DRAM technology is approaching its scaling limit, and the cost of using traditional technology to increase the capacity of DRAM and improve its energy efficiency and reliability is also greatly increased [41]. NVM has the characteristics of non-volatile, byte addressing, high storage density, low energy consumption, read and write performance close to DRAM, asymmetric read and write overhead, limited lifetime and so on. NVM-based external sorting algorithms mainly utilize the performance asymmetry between reads and writes and the benefits of byte addressing, decreasing write operations and extending lifetime of NVM. Our team mainly studies the external sorting algorithm based on NVM, which makes use of the characteristics of NVM byte addressing to optimize the external sorting algorithm, so as to reduce write amount and random access to NVM. There are three problems that need to be solved urgently in the future. Firstly, the main research direction in the future is to solve the fragmentation problem caused by NVM byte addressing. Secondly, when NVM and DRAM are combined as a hybrid memory, hard-to-find pointers may occur because of non-volatile feature of NVM. Finally, because of endurance of NVM device, so it is necessary to put forward solutions about wear-leveling, reducing write amount and the write operation to enhance lifetime of NVM device.

Acknowledgments
Supported by National Key R&D Program of China under Grant NO.2018YFB0203904 and National Natural Science Foundation of China under Grant NO.61832020, NO.61872392, NO.U1611261 and NO.U1811461, the Pearl River S & T Nova Program of Guangzhou under Grant NO. 201906010008, and Natural Science Foundation of Guangdong Province, China under Grant 2018B030312002.

References
[1] Thusoo, A., Shao, Z., Anthony, S., Borthakur, D., Jain, N., Sarma, J. S., ... & Liu, H. (2010). Data warehousing and analytics infrastructure at facebook. international conference on management of data.
[2] J. Boukhobza, and P. Olivier. (2017). Flash Memory Integration: Performance and Energy Issues. 1st. UK: ISTE Press - Elsevier.

[3] Wilkes, M. V. (1974). The Art of Computer Programming, Volume 3, Sorting and Searching. The Computer Journal, 17(4), 324-324.

[4] Dobosiewicz, W. (1984). Replacement selection in 3-level memories. The Computer Journal, 27(4), 334-339.

[5] Estivill-castro, V., & Wood, D. (1992). A survey of adaptive sorting algorithms. ACM Computing Surveys, 24(4), 441-476.

[6] Zheng, L., & Larson, P. (1996). Speeding up external mergesort. IEEE Transactions on Knowledge and Data Engineering, 8(2), 322-332.

[7] Anderson, T. A., Breitbart, Y., Korth, H. F., & Wool, A. (1998). Replication, consistency, and practicality: are these mutually exclusive? international conference on management of data, 27(2), 484-495.

[8] Nyberg, C., Barclay, T., Cvetanovic, Z., Gray, J., & Lomet, D. (1994). AlphaSort: a RISC machine sort. international conference on management of data, 23(2), 233-242.

[9] Larson, P. A. (2003). External sorting: run formation revisited. IEEE Transactions on Knowledge and Data Engineering, 15(4), 961-972.

[10] Islam, R., Adnan, N., Islam, N., & Hossen, S. (2003). A new external sorting algorithm with no additional disk space. Information Processing Letters, 86(5), 229-233.

[11] Okazawa, T. (1996). Electrically erasable programmable non-volatile semiconductor memory device and method for manufacturing the same.

[12] Wu, C. (2012). Data sorting in flash memory. acm symposium on applied computing, 1847-1849.

[13] Andreou, P., Spanos, O., Zeinalipouryazti, D., Samaras, G., & Chrysanthis, P. K. (2009). FSort: external sorting on flash-based sensor devices. data management for sensor networks.

[14] Lee, S., Moon, B., Park, C., Kim, J., & Kim, S. (2008). A case for flash memory ssd in enterprise database applications. international conference on management of data.

[15] Park, H., & Shim, K. (2009). FAST: Flash-aware external sorting for mobile database systems. Journal of Systems and Software, 82(8), 1298-1312.

[16] Cossentine, T., & Lawrence, R. (2010). Fast sorting on flash memory sensor nodes. international database engineering and applications symposium, 105-113.

[17] Cossentine, T., & Lawrence, R. (2013). EFFICIENT EXTERNAL SORTING ON FLASH MEMORY EMBEDDED DEVICES. International Journal of Database Management Systems, 5(1), 1-20.

[18] Jackson, Riley & Lawrence, Ramon. (2019). Faster Sorting for Flash Memory Embedded Devices, 1-5.

[19] Boukhobza, Jalil. (2013). Flashing in the Cloud: Shedding some Light on NAND Flash Memory Storage Systems.

[20] Liu, Y., He, Z., Chen, Y. P., & Nguyen, T. (2011). External Sorting on Flash Memory Via Natural Page Run Generation. The Computer Journal, 54(11), 1882-1900.

[21] Boukhobza, J., Rubini, S., Chen, R., & Shao, Z. (2017). Emerging NVM: A Survey on Architectural Integration and Research Challenges. ACM Transactions on Design Automation of Electronic Systems, 23(2).

[22] Lee, J., Roh, H., & Park, S. (2016). External Mergesort for Flash-Based Solid State Drives. IEEE Transactions on Computers, 65(5), 1518-1527.

[23] Chen, F., Lee, R., & Zhang, X. (2011). Essential roles of exploiting internal parallelism of flash memory based solid state drives in high-speed data processing. high-performance computer architecture, pp. 266-277

[24] Kanza, Y., & Yaari, H. (2016). External sorting on flash storage: reducing cell wearing and increasing efficiency by avoiding intermediate writes. very large data bases, 25(4), 495-518.
[25] Lee, Y., Quero, L. C., Kim, S., Kim, J., & Maeng, S. (2016). ActiveSort: Efficient external sorting using active SSDs in the MapReduce framework. *Future Generation Computer Systems*, 76-89.

[26] Cho, S., Park, C., Oh, H., Kim, S., Yi, Y., & Ganger, G. R. (2013). Active disk meets flash: a case for intelligent SSDs. *International Conference on Supercomputing*, 91-102.

[27] Tiwari, D., Boboila, S., Vazhkudai, S. S., Kim, Y., Ma, X., Desnoyers, P., & Solihin, Y. (2013). Active flash: towards energy-efficient, in-situ data analytics on extreme-scale machines. *File and Storage Technologies*, 119-132.

[28] Moon, S., Lee, J., & Kee, Y. S. (2014). Introducing SSDs to the Hadoop MapReduce Framework. *International Conference on Cloud Computing*, 272-279.

[29] Blelloch, G. E., Fineman, J. T., Gibbons, P. B., Gu, Y., & Shun, J. (2015). Sorting with Asymmetric Read and Write Costs. *ACM Symposium on Parallel Algorithms and Architectures*, 1-12.

[30] Li, H., & Chen, Y. (2010). Emerging non-volatile memory technologies: From materials, to device, circuit, and architecture. *International Midwest Symposium on Circuits and Systems*, 1-4.

[31] Meena, J. S., Sze, S. M., Chand, U., & Tseng, T. (2014). Overview of emerging nonvolatile memory technologies. Nanoscale Research Letters, 9(1), 526-526.

[32] Kim, H. J., Seshadri, S., Dickey, C. L., & Chiu, L. (2014). Evaluating Phase Change Memory for Enterprise Storage Systems: A Study of Caching and Tiering Approaches. *File and Storage Technologies*, 10(4), 33-45.

[33] Lee, B. C., Zhou, P., Yang, J., Zhang, Y., Zhao, B., Ipek, E., ..., & Burger, D. (2010). Phase-Change Technology and the Future of Main Memory. *International Symposium on Microarchitecture*, 30(1), 143-143.

[34] Bey Ahmed Khernache, Mohammed & Laga, Arezki & Boukhabza, Jalil. (2018). MONTRES-NVM: an External Sorting Algorithm for Hybrid Memory. *IEEE Non-Volatile Memory Systems and Applications Symposium*.

[35] Graefe, G. (1993). Query evaluation techniques for large databases. *ACM Computing Surveys*, 25(2), 73-169.

[36] Yannis, J., & Zobel, J. (2003). External sorting with on-the-fly compression. *British National Conference on Databases*, 115-130.

[37] Alistair Moffat, & Andrew Turpin. (2002). Compression and Coding Algorithms. Kluwer Academic Pub, Holland.

[38] Rahn, M., Sanders, P., & Singler, J. (2010). Scalable distributed-memory external sorting. *International Conference on Data Engineering*, 685-688.

[39] Rahn, Mirko & Sanders, Peter & Singler, Johannes & Kieritz, Tim. (2010). DEMSort — Distributed External Memory Sort.

[40] Dobosiewicz, W. (1984). Replacement selection in 3-level memories. *The Computer Journal*, 27(4), 334-339.

[41] Mutlu, Onur. (2015). Main Memory Scaling: Challenges and Solution Directions. 127-153.