ACM SIGMOD Programming Contest 2019

Team Leyenda: Zhaoxing Li, Yuanjing Shi, Tianyin Xu (Advisor)

{zl50, ys26, tyxu}@illinois.edu, Department of Computer Science, University of Illinois at Urbana-Champaign

1. INTRODUCTION

Task:
Sort as fast as possible three different datasets (10GB, 20GB and 60GB) with constrained memory (30GB).

Format: Each dataset consists of 100-byte records with 10-byte key and 90-byte value. Record is binary for 10 & 60 GB datasets and ASCII for 20 GB dataset.
Distribution: Key distribution is set to uniform for 10 & 20 GB and skewed for 60 GB.

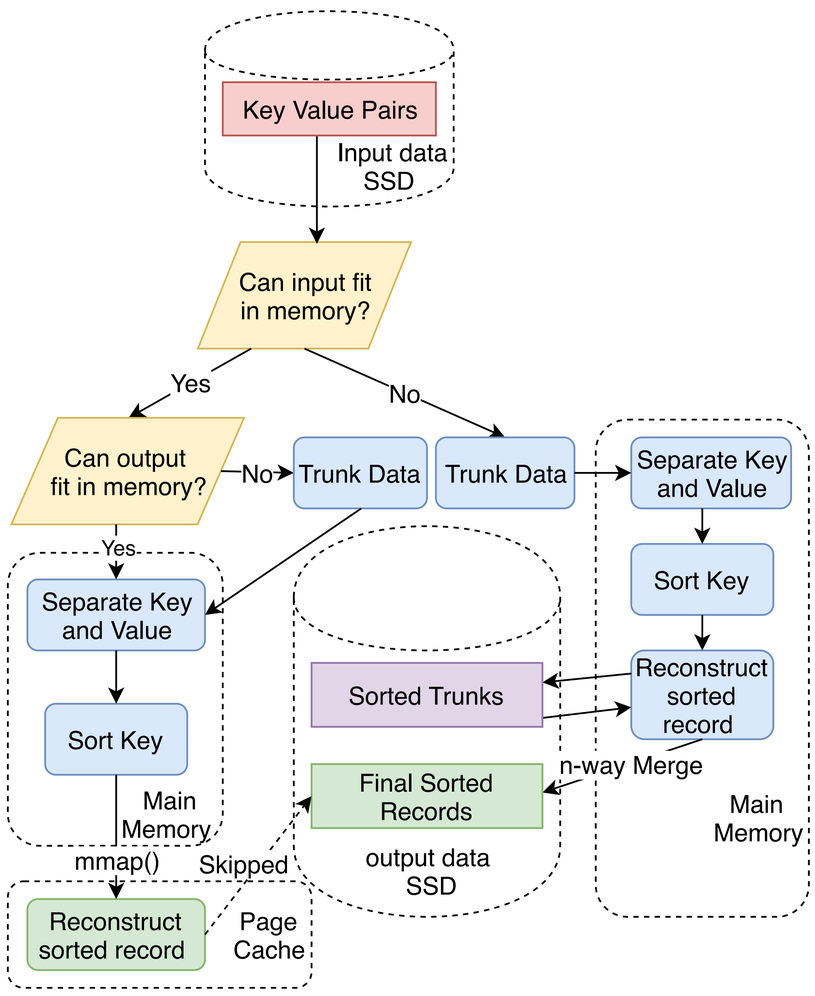
Testing machine configuration:
2x Intel Xeon E5-2640 v4 CPU (2.40 GHz), 20 cores / 40 hyperthreads, 30 GB RAM, SATA3 SSD (up to 600MB/s)

2. OUR APPROACH

We tackle the sorting task with five key ideas:

- **No Silver Bullet.** Use different algorithms depending on if data can fit in memory.
- **Adaptive sorting.** Combining *Radix Sort* and *Comparison Sort* to create a sort kernel that adapts well with skewed data.
- **I/O Overlapping.** Design an external sort schedule which overlaps disk I/O with computation as much as possible.
- **Keep data in memory** whenever possible during and after sorting with fine-grained memory management and Linux page cache.
- **Hardware Friendliness.** Utilize characteristics of CPU and memory architecture in a NUMA system to speed up sorting.

3. SYSTEM ARCHITECTURE



4. SORT KERNEL

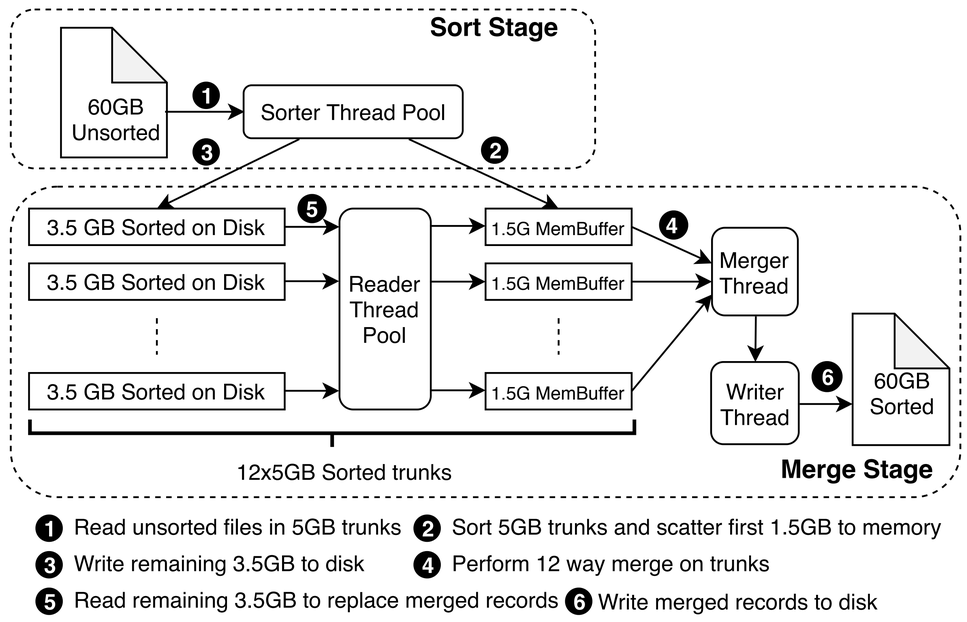
- **Separate keys with values,** which transforms the problem into sorting fixed sized keys.
  - This has *O(wn)* complexity using *Radix Sort*, where *w* is key size and *n* is data size.
  - 10 times faster than sorting record as a whole due to less memory copy operations.
- **Use Most Significant Bit Radix Sort** which does not require the scattering process to continue until the last digit. When bucket size is small apply *Comparison Sort* directly to shortcut Radix Sort.
- **Handling Skewed data** by load-balancing small buckets among threads, and distributing big buckets onto multiple threads.
- **Maximize memory bandwidth.** Scatter records to CPU cores' L2 cache lines before flushing to main memory. Only schedule threads on CPU local to the main memory in a NUMA system.

5. MEMORY MANAGEMENT

- **Write sorted data to page cache whenever possible** so it can be consumed by the judging program directly.
- **Use Memory map** to reconstruct sorted data records directly in the page cache in parallel, given the sorted *(key, pointer)* tuples. This reduces disk write time by at most 50%.
- **Garbage Collection.**
  - Implement a counter based garbage collector in C++ to aggressively free user space memory when it is no longer needed.
  - Combined with a fine-grained memory allocation of 4KB buffers, we allow OS to reclaim user space memory in time to continuously write sorted data to page cache.

6. EXTERNAL MERGE SORT SCHEDULE

- **Random read with 256KB request size on 20 threads using O_DIRECT**, which is slightly faster than sequential read.
- Steps in sort stage and merge stage run in parallel.



7. WORKLOAD AND BEST CONTEST RESULT

Workload	Read	Sort/Scatter	Write/Merge	Total
10GB	18.716	0.696	5.192	25.91
20GB	38.092	1.638	12.29	52.02
60GB	114.896	N/A	174.419	290.91

ILLINOIS