ACM SIGMOD Programming Contest 2019

vsb_ber0134 (Jakub Beránek)

Advisor: Radim Bača

VSB TECHNICAL UNIVERSITY OF OSTRAVA

Task

- Sort files containing 100 B records (10 B key, 90 B payload)
- Datasets:
  Small - 10 GiB, uniform distribution
  Medium - 20 GiB, uniform distribution, only ASCII bytes
  Large - 60 GiB, skewed distribution
- Memory limited to 30 GiB (external sort necessary for the Large dataset)
- Two SSD disks (one for input, one for output)

In-memory sort

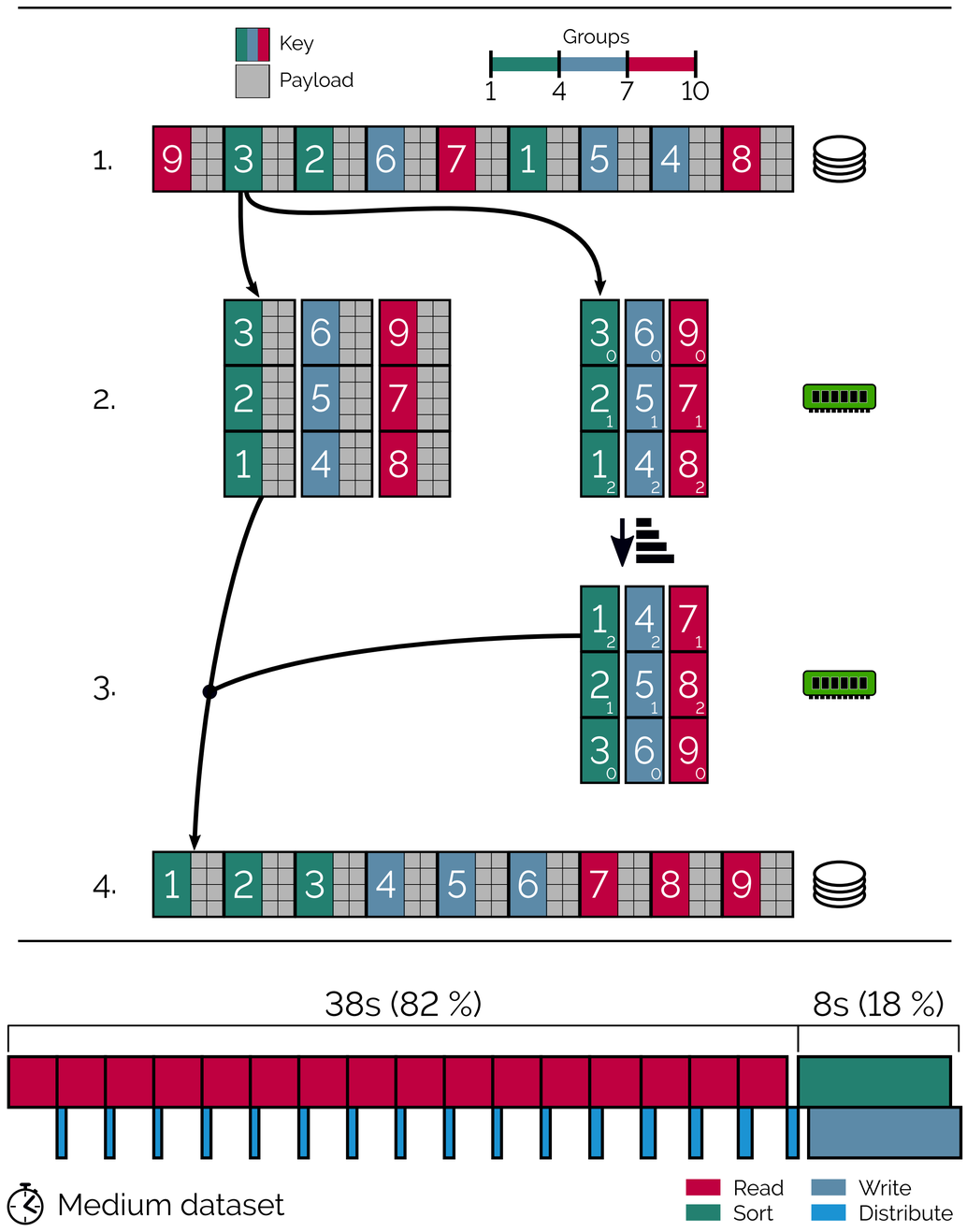
If the dataset fits into memory, no actual write I/O is necessary.
The key idea is to aggressively deallocate memory as soon as it becomes redundant in every step of the sort to maximize space for the Linux page cache.

1. Read
- Input is read in chunks, I/O is overlapped with partitioning

2. Partition
- Records are partitioned according to their first byte
- Parallel partitioning without synchronization - each thread scans all records, but partitions only its assigned byte range
- Each partition has a duplicate that contains only the keys
- Growable mmap buffers are used to avoid copies on reallocations

3. Sort
- All partitions with keys are sorted in parallel using MSD radix sort
- The first byte is already partitioned and is thus skipped during the sort
- The payload is not moved during the sort - better cache utilization

4. Write
- Records are copied in parallel to the output in sorted order
- After a group is written, it is unmapped from memory



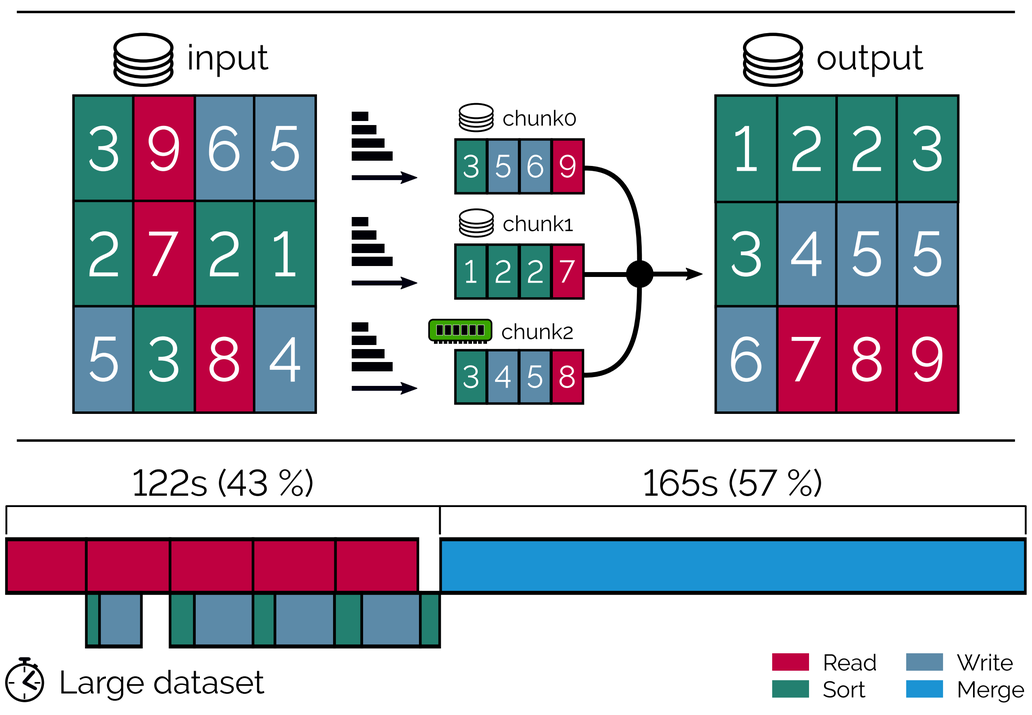
External sort

An external k-way sort is used, the input is divided into sorted ~12 GiB chunks that are subsequently merged at the end.

1. Read
- Input is read in chunks, I/O is overlapped with sorting and writing the previous chunk to disk

2. Sort & write
- Chunks are sorted using the in-memory sort method
- The final chunk is kept in memory to avoid unnecessary I/O
- The last chunk written to the disk is written backwards to keep its beginning in the page cache to accelerate reads during merge

3. Merge
- Sorted chunks are merged to form the final result
- Smallest element amongst the sorted chunks is found using a linear array search - keeping a sorted heap introduces overhead for so few chunks
- Writes to the output are double-buffered
- Reads from the input chunks are not double-buffered to minimize concurrent reads/writes on the same disk
- After every ~500 MiB read from the in-memory chunk, the read data is unmapped to free up more space for the page cache



General optimizations

- **Transparent Huge Pages**
  - Reduced TLB pressure for large buffers accessed in a random order
- **NUMA**
  - Memory for the program was allocated only on NUMA node 0
  - Node 1 cores were disabled to reduce memory bandwidth contention
- **Memory footprint**
  - Minimized memory usage provides more space for the page cache
  - Structures compressed with `__attribute__((packed))`
  - Sequentially accessed buffers gradually unmapped in chunks using `madvise(..., MADV_FREE)`
- **Efficient key comparison**
  - Keys compared as a big endian uint64/uint16 pair (`bswap` + `cmp`)

Code

~2200 lines of C++ 17

Third-party libraries
- OpenMP
- radix sort (`https://github.com/voutcn/kxsort`)

VSB TECHNICAL UNIVERSITY OF OSTRAVA | IT4INNOVATIONS NATIONAL SUPERCOMPUTING CENTER

VSB TECHNICAL UNIVERSITY OF OSTRAVA | FACULTY OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE | DEPARTMENT OF COMPUTER SCIENCE