ACM SIGMOD'19 Programming Contest

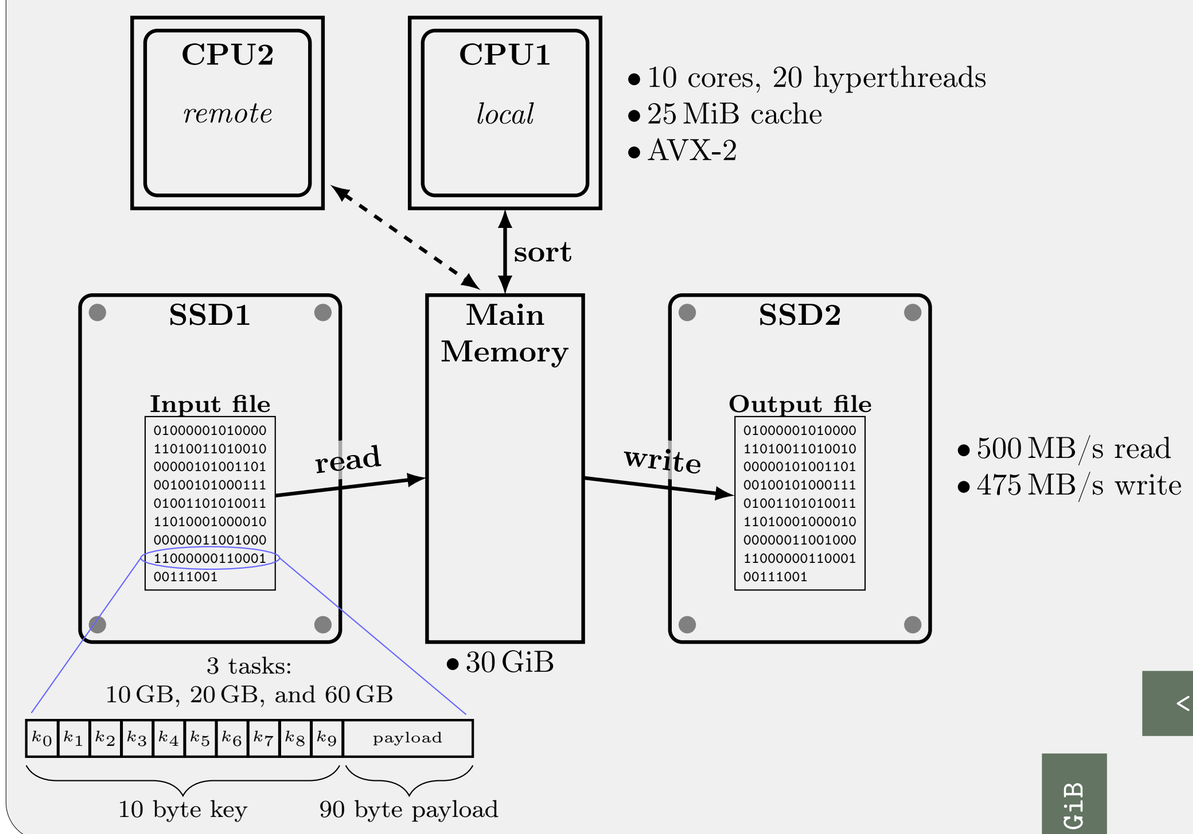
teamsic: Immanuel L. Haffner Advisor: Jens Dittrich

Big Data Analytics Group, Saarland Informatics Campus

bigdata.uni-saarland.de

git.io/fjazH

Task Description



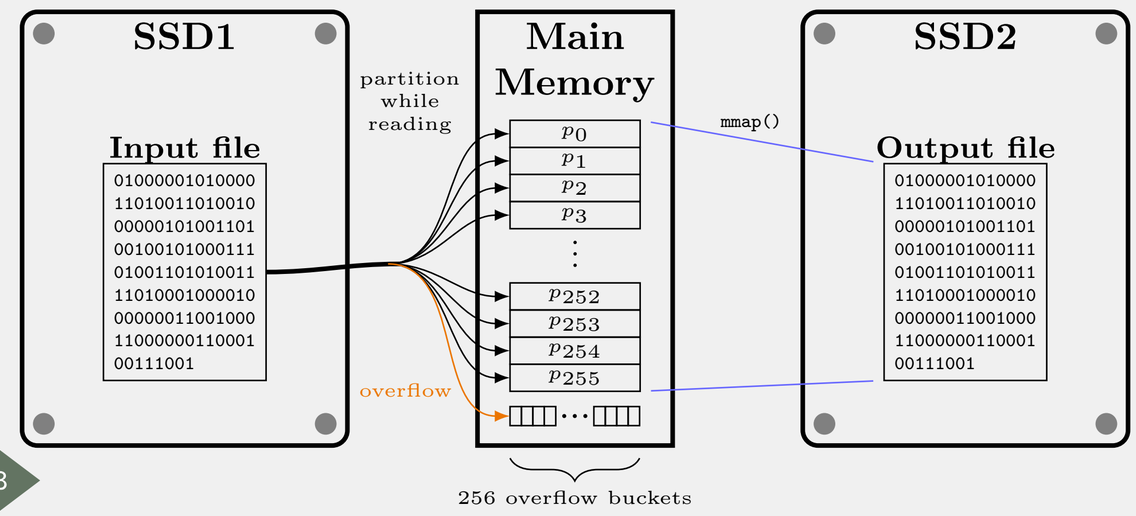
< 30 GiB

≥ 30 GiB

In-Memory Sorting

Algorithm:

1: predict record type (ASCII [32-126] or binary [0-255])
2: estimate bucket sizes
3: partition into 256 buckets
4: sort each bucket with *sicsort*



Tricks:

- avoid write to disk with `mmap()` *#thankskernel*
- keys are uniformly distributed; no skew
- partition into `mmap`'d output file while reading from disk:

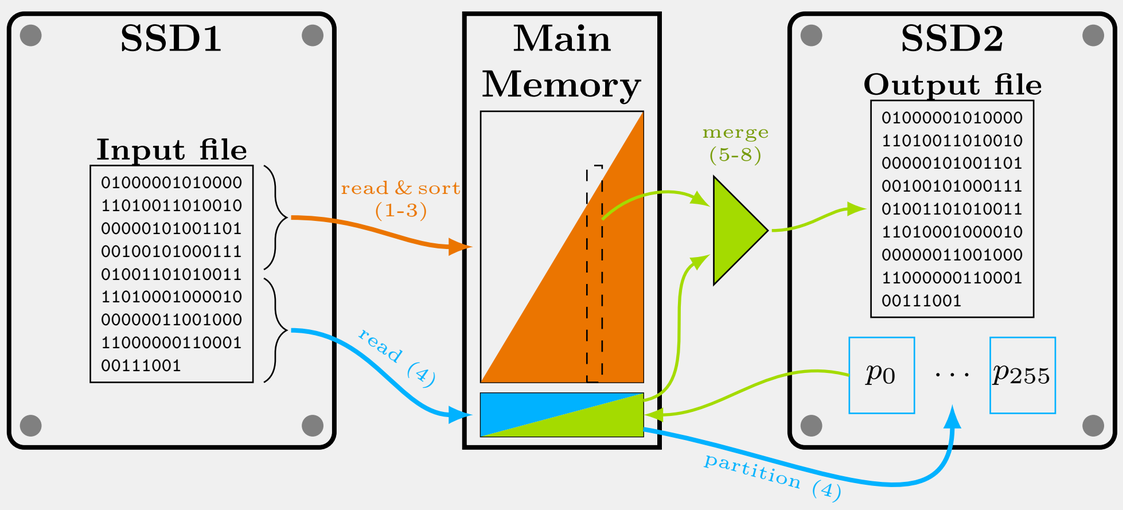
```
buckets ← mmap(output file, ...)        ▷ consecutive memory
for record r in input file do                          ▷ read from disk
    bucket b ← buckets[r.k_0]                  ▷ initial partitioning
    if b is full then                  ▷ underestimated bucket size
        append r to overflow bucket of b
    else
        append r to b
    end if
end for
patch_overflow()
```

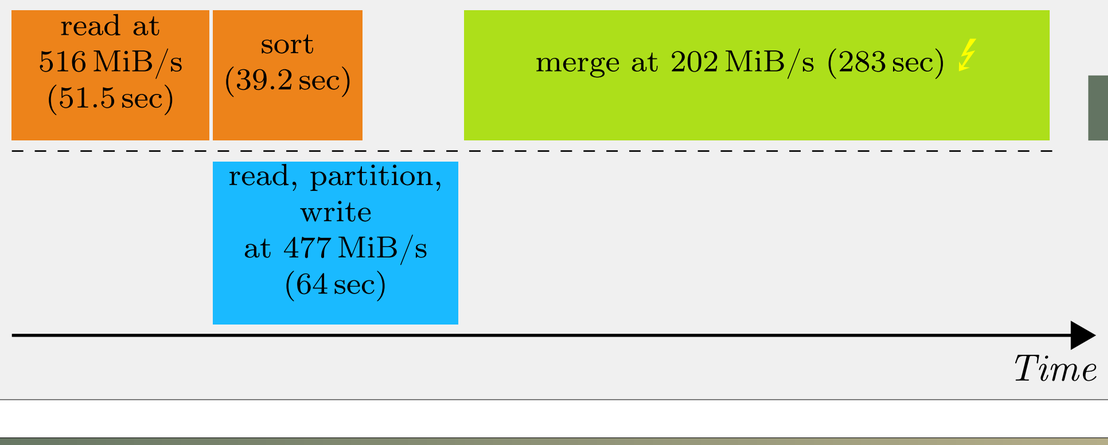
uses

External Memory Sorting

Algorithm:

1: $n \leftarrow$ number of records that fit into main memory
2: $s \leftarrow$ read first n records into main memory
3: sort s with *sicsort* ▷ in separate thread
4: partition remaining records into 256 files
5: **for** every partition file p, sorted by size **do**
6: sort p with *sicsort*
7: merge p with s to output file on disk
8: **end for**



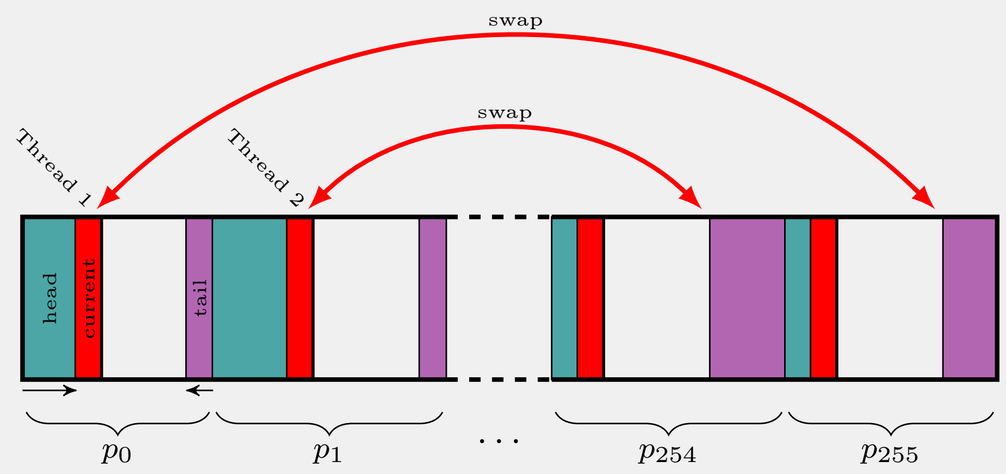


uses

Sorting Algorithm *sicsort*

Algorithm:

1: compute histogram
2: compute partition boundaries
3: **for** every record r **do**
4: **if** r in destination partition **then**
5: proceed to next record
6: **else**
7: $p \leftarrow$ destination partition of r
8: swap r with unpartitioned record of p
9: **end if**
10: **end for**



Details:

- based on *American Flag Sort*, an in-place radix sort
- extended with parallel partitioning (see figure above)
- parallelize recursion into partition
- fall back to `std::sort()` on recursion if number of records is small (i.e. *introsort* with fall back to *insertion sort*)
- atomic counters for head and tail of each partition
- avoid cache contention
  - pad counters to 64 Bytes ⇒ one cache line per counter
  - with 256 partitions requires exactly 32 KiB cache (L1D)

Third-Party Libraries

- Agner Fog's `asmlib` (for `memcmp()`)
- CTPL: Modern and Efficient C++ Thread Pool Library (not in final submission)
- Intel Processor Counter Monitor (PCM) (not in final submission)

Ranking

- preliminary third place
- first in *Small* (10 GB) with 17 % margin to second
- first in *Medium* (20 GB) with 12 % margin to second
- fifth in *Large* (60 GB) with 15 % margin to best