

# Datamation: a quarter of a century and four orders of magnitude later

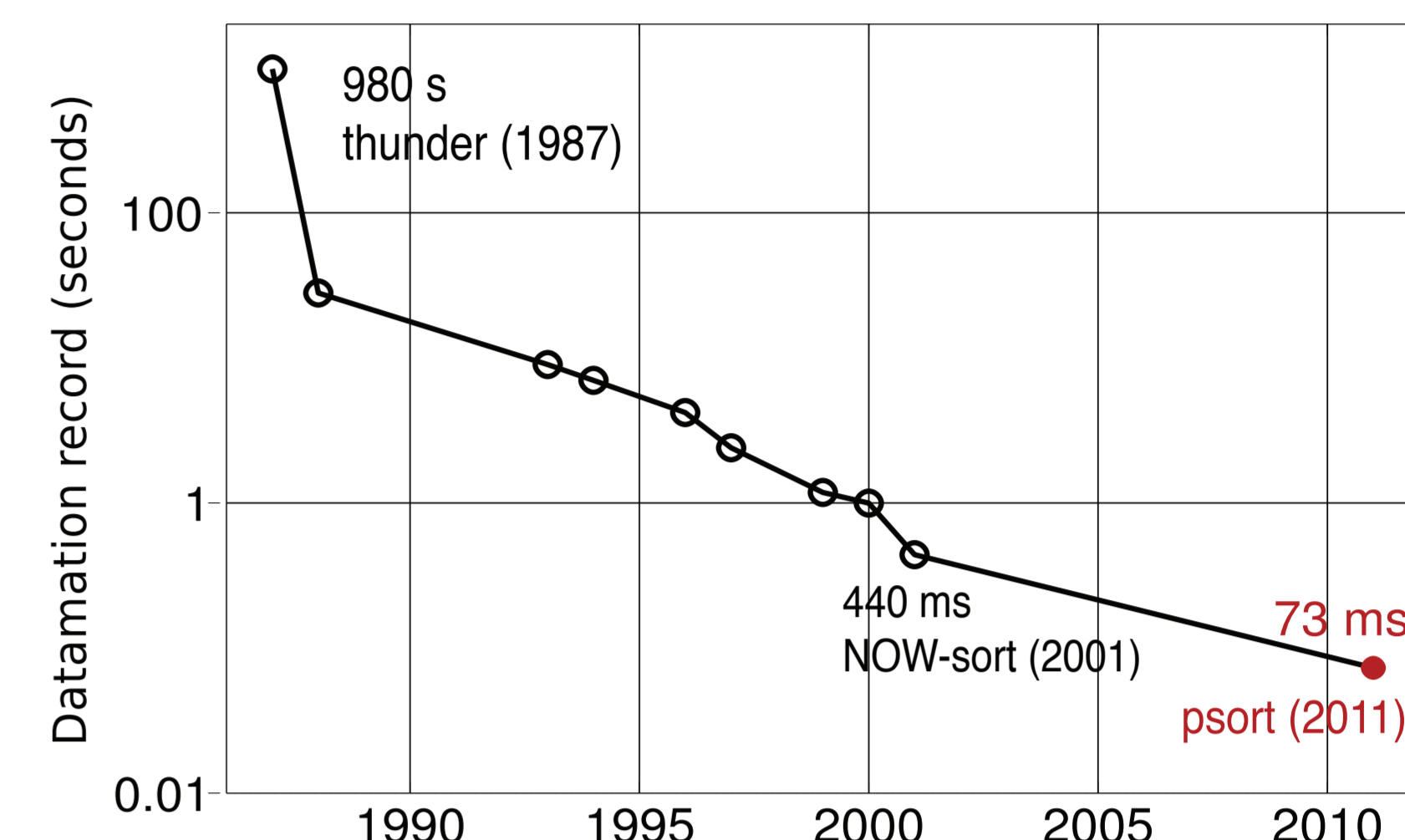
P. Bertasi, M. Bonazza, M. Bressan, E. Peserico

## Breaking the Datamation benchmark

The Datamation sort benchmark requires sorting one million 100-byte records (100MB) according to the first 10 bytes from disk to disk as rapidly as possible. This used to provide a simple, well-rounded evaluation of most parts of a computing system – processor(s), memory, disks and, for computing clusters, interconnection network.

In 1987, the first Datamation record was of 980 seconds, a value that dropped to less than 1 second in 2000, turning Datamation into a measure of the responsiveness of a system and of the delay in accessing the mass storage layers of its memory hierarchy, a key feature for a large and growing number of application domains: web services, cloud computing, Weather Research and Forecast, Massive Multi-player Online Games, distributed large dataset processing.

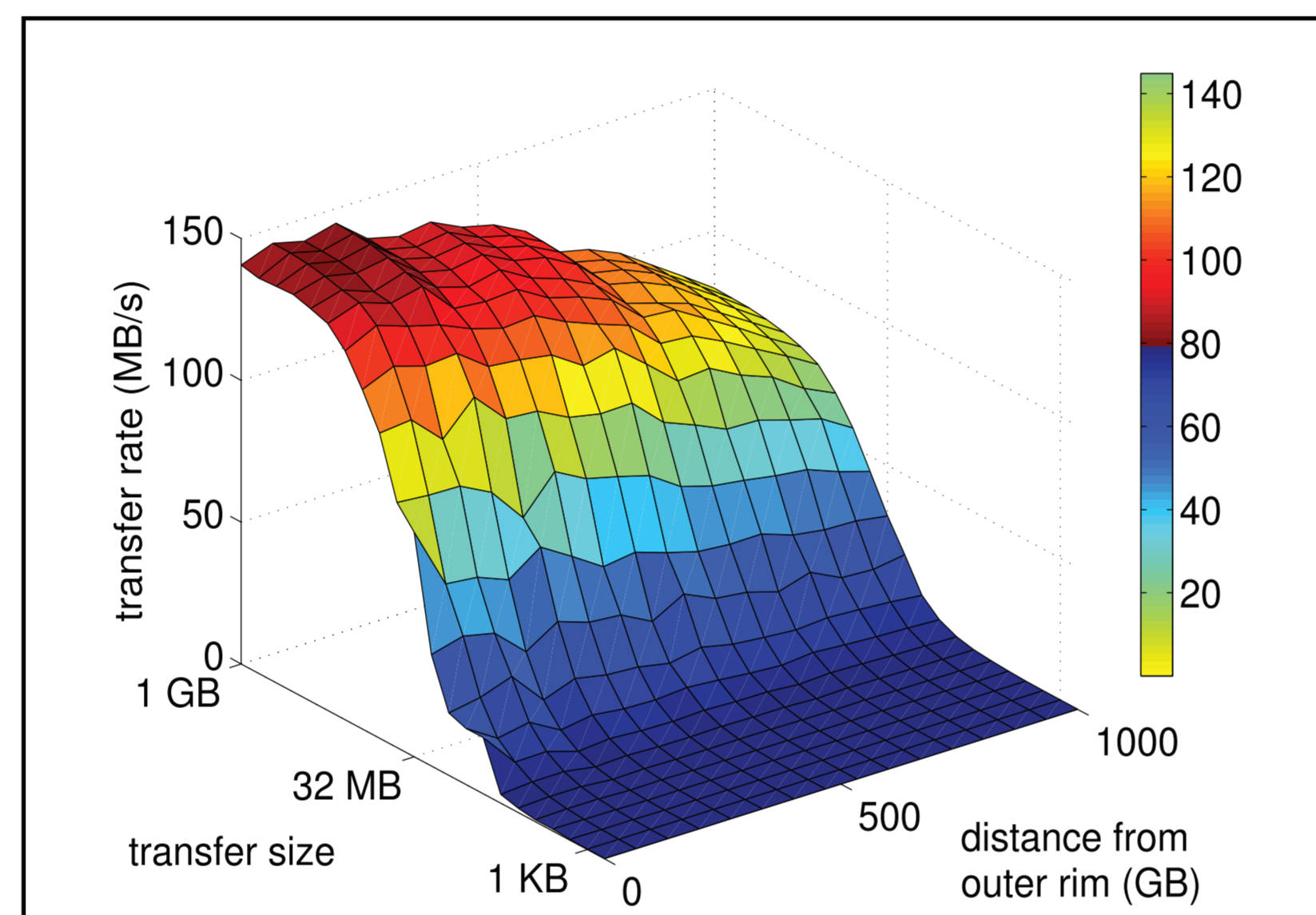
Our 16-node desktop-class cluster Eridanus, using a tuned version of our fast sorting library `psort`, runs the Datamation benchmark in less than 75 milliseconds (average execution time).



## Lowering latency in clusters

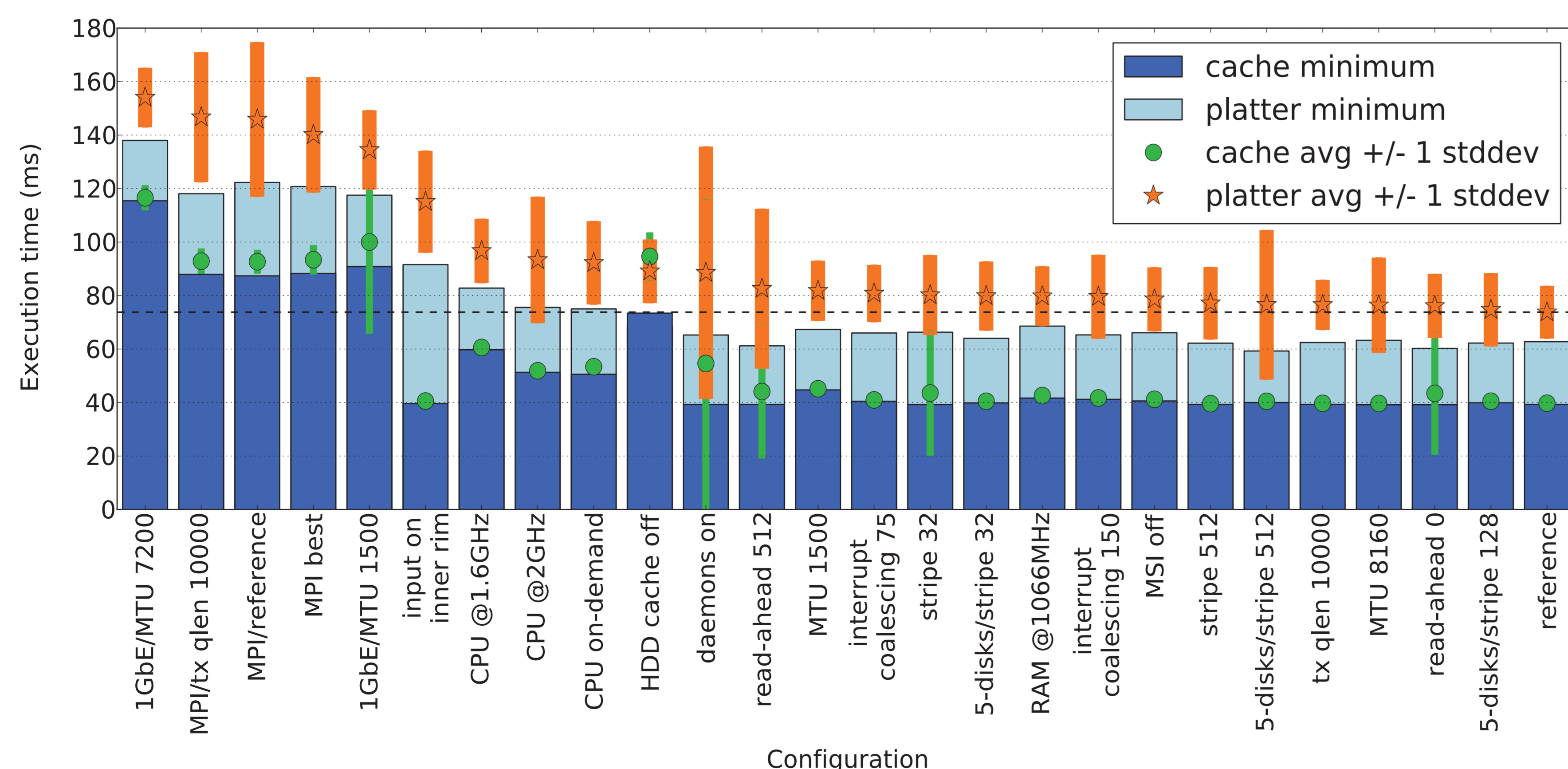
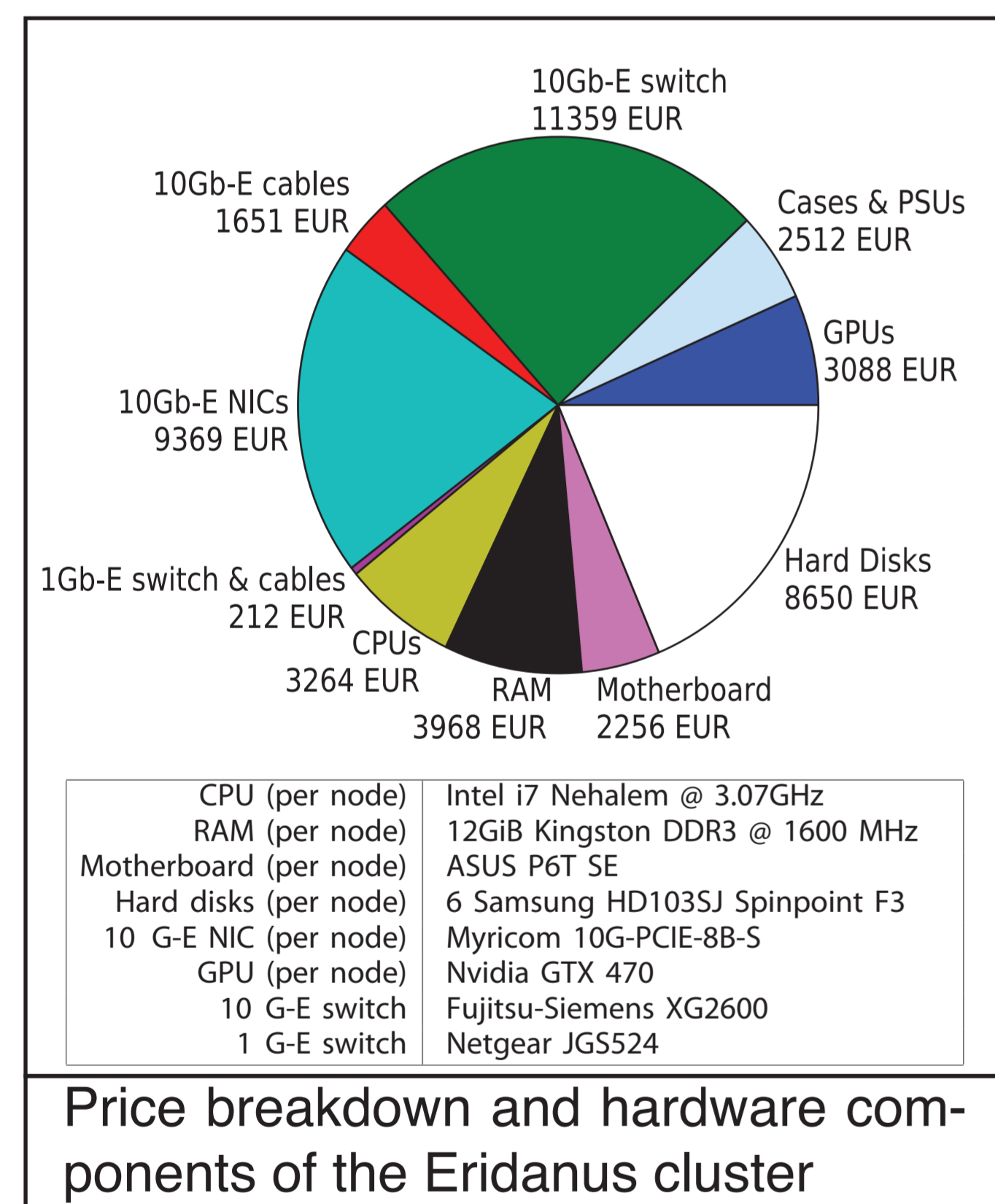
We investigated on some of the many sources of latency that affect high-performance clusters.

- **Disks.** Head movement and rotational delay impose a heavy performance penalty. Sorting from/to disk platter imposes an overhead of 30+ms and introduces a significant noise over sorting from/to disk cache.
- **Network infrastructure.** Small bandwidth affects responsiveness. Switching from the 10G to a 1G-Ethernet imposes an overhead >60ms.
- **Communication protocols.** A tuned MPICH version of `psort` is significantly slower (+70ms) than standard `psort`. Increasing the Linux kernel's `txqueuelen` to 10kB, which could in theory improve TCP's efficiency, did not yield better results.



Read transfer rate of a Samsung Spinpoint F3 HD103SJ as a function of the transfer size and of the distance from the disk's outer rim

- **Data placement.** Careful data positioning on the disk(s) is crucial: placing the input closer to the disk's spindle slows `psort` by at least 40 milliseconds.
- **CPU frequency.** Decreasing the CPU frequency to 2GHz and 1.6GHz increases the average execution time by, respectively, ~10ms and ~20ms, while "on demand" frequency setting incurs a 10ms increase.
- **System daemons.** Debian Linux daemons periodically wake up and consume resources, increasing the average execution time (+15ms) and its standard deviation (+60ms).
- **Filesystem read-ahead.** Decreasing it from 2048 to 512 sectors imposes an overhead of 8ms, but more surprisingly it takes the standard deviation of execution time to 30ms.



## psort, a fast external sorting library

`psort` is our fast C++ library for stable sorting on external memory. Since 2008, `psort` ranks first among the external sorting software for PC-class machines according to the Sort Benchmark (<http://sortbenchmark.org>).

`psort` is highly optimized to match the machine architecture and overlap I/O with computation, outperforming other libraries for large data sets (GB, TB, ...). A version of `psort` has been created for low-latency sorting on clusters and multicore architectures, and will be included in future releases.

## Distributed sorting with psort

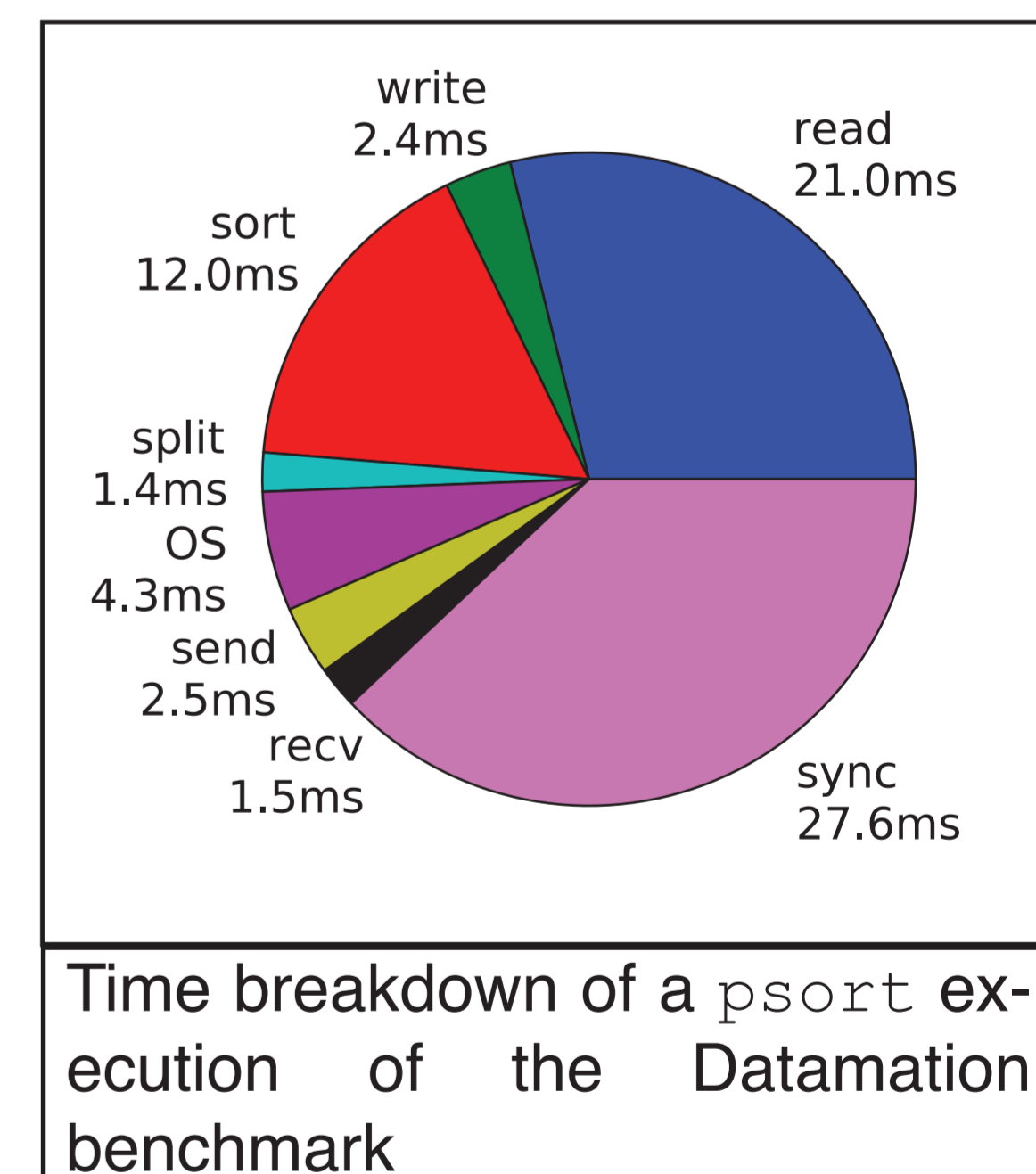
We coupled the Eridanus cluster with a distributed version of `psort` that yielded an extremely responsive sorting system. Sorting is done in the 3 following phases:

### 1. local bucketing

Each node reads 100MB/16 from the fastest portion of its RAID0 storage, using the `O_DIRECT` flag to bypass kernel buffers and reduce latency. It then distributes records among 16 buckets according to the first 4 bits of their 10-byte keys. Each key is conveniently separated, extended with a compressed pointer to the payload, and byteswapped to integer for faster comparisons.

### 2. global distribution

Each node sends the ~6MB of keys and payloads in its *i*-th bucket to the *i*-th node of the cluster via UDP messages. Communications are scheduled in 16 synchronized rounds, each round being completely free of collisions. This yields a large fraction of synchronization operations, but guarantees a low communication time. After the last round, each node has collected ~6MB of keys and payloads from the other nodes.



### 3. local sorting

Each node sorts the collected keys, employing the fast sorting routines provided by the `psort` library. Several optimizations, such as quasi-in-place mergesort and explicit handling of CPU registers, guarantee an extremely high throughput. The sorted keys are then converted back to strings, their payloads concatenated, and the records are then written to disk.

## This work has been supported by:

- Samsung who generously donated the 96 disks of our cluster
- University of Padua under Strategic Project AACSE
- MIUR under Project AlgoDEEP
- PAT, FBK, and INFN under Project Aurora-Science