Optimization of non-contiguous MPI-I/O operations

Enno Zickler

Arbeitsbereich Wissenschaftliches Rechnen
Fachbereich Informatik
Fakultät für Mathematik, Informatik und Naturwissenschaften
Universität Hamburg

26.05.2015

Outline

Non-Contiguous Data Access

What is Non-Contiguous Data Access?

- access on data which is scattered over the storage
- common data layouts often lead to non-contiguous access
- e.g. accessing only one element of a triples in an array
- non collective parallel access may also induce such access

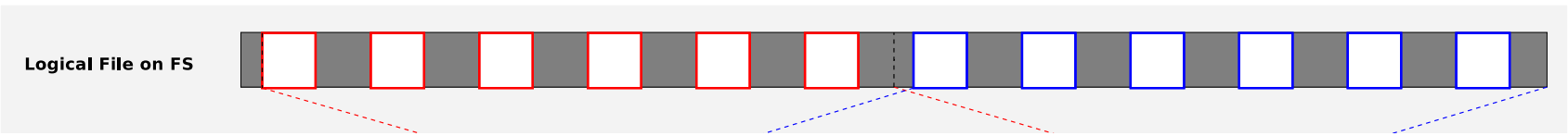

Figure 1: Non-Contiguous Data

Why is Non-Contiguous Data Access a problem?

- generally poor performance on storage systems
  - overhead for many individual I/O calls on the file system
  - long seek time on HDDs
- as I/O is a major bottleneck in HPC optimization of non-contiguous access is important

Data Sieving

What is Data Sieving?

- method to increase performance on non-contiguous data
- areas of unwanted (holes) data are also accessed
- the desired data is sieved out
- additional memory for buffers is needed (typically 2MB-4MB)
- much better performance for small holes and small data sizes

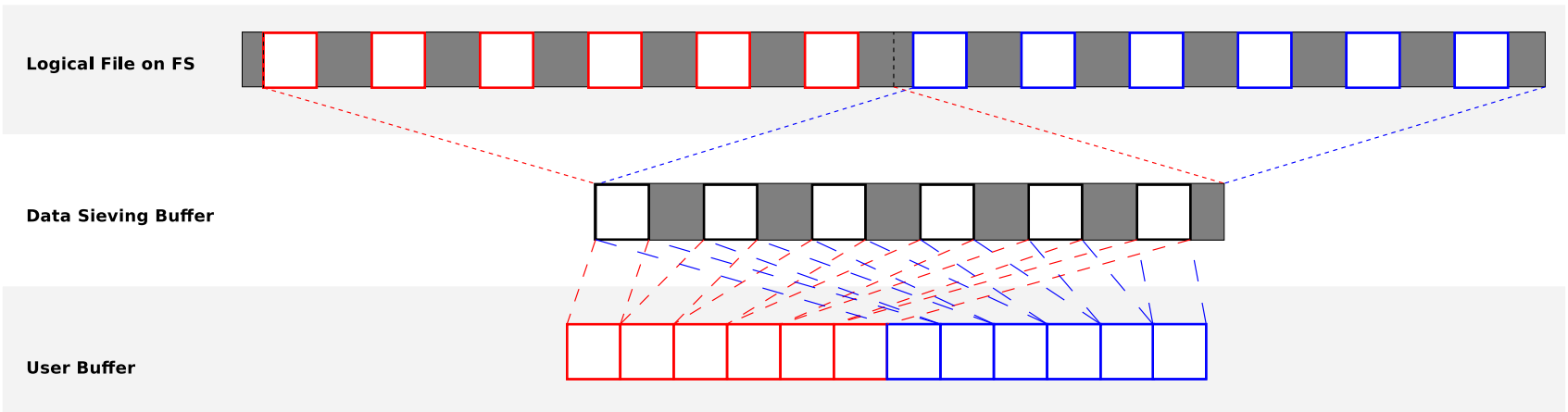

Figure 2: Data Sieving

Available implementations

- ROMIO[1]
  - \+ good performance for small data and hole size
  - \+ sustain detailed information about data layout
  - – but even harmful for big data and hole size
  - – requires detailed knowledge about the system
  - – MPI-I/O only
  - – hard to customize
- Performance model-directed data sieving [2]
  - \+ automatic decision based on system characteristics
  - \+ bandwidth, latency and number of I/O nodes
  - – properly MPI-I/O only (not released yet)

NCT Library

Goals

1. **Flexibility**
   - allow algorithms to adopt to the underlying system
   - easy to modify behavior of data sieving
2. **Automation**
   - possibility to use sophisticated algorithms which do not need further user adjustments
3. **Universality**
   - MPI Independence for wider use of data sieving
   - POSIX level API

Design & Implementation

- C library
- Features
  - view based file access functions
  - POSIX compatible
  - modularity through optimization plugins
  - provides helper functions for plugins
- Processing of data sieving
  - decision making: when and how to use data sieving
  - helper functions access data and copy data between buffers
- 3 differn algorithms were implemented
  - naive
  - ROMIO
  - simple performance model

API

```
1  typedef struct nct_tuple_t
2  {
3      uint32_t size;
4      uint32_t deltaOffset;
5  } nct_tuple;
6
7  typedef struct nct_info_t
8  {
9      double latencyFS;
10     int bandwidth;
11     int numNodes;
12     double latencyNetwork;
13     uint64_t stripeSize;
14     uint64_t blocksize;
15     int dsMethod;
16 }nct_info;
17
18 nct_view nct_create_view( uint32_t initalOffset, size_t ds_bufsize, void *
       ds_buf, int tupleCount, nct_tuple* list, nct_info* info);
19
20 void nct_destroy_view(nct_view view);
21
22 size_t nct_read(int fd, void* buff, uint64_t offset, size_t byte_count, nct_view
       view);
23
24 size_t nct_write(int fd, void* buff, uint64_t offset, size_t byte_count,
       nct_view view);
```

Listing 1: nct.h

Evaluation

Methodology

- custom benchmark tool to use different data pattern
  - vary hole and data size
  - read and write of 10GB or at least 100sec
- WR- Cluster
- 1-10 lustre nodes
- 128KiB and 2MiB Stripe size
- different pattern
- min 3 iterations per setting
- all 3 runs are in a 20% range of the mean value
- model created for expected results

Reading model

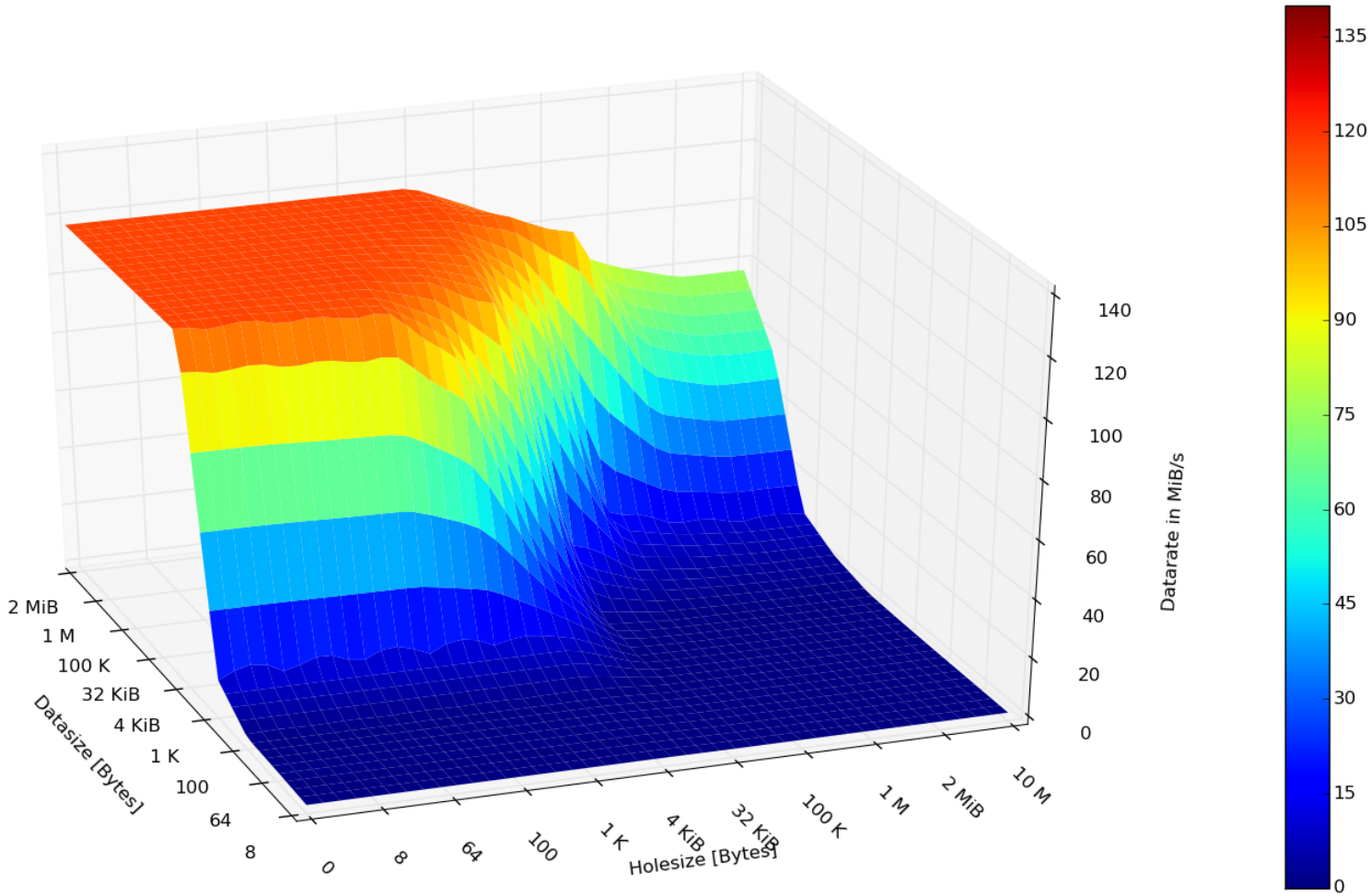

Figure 3: Model WR reading default pattern

Reading naive

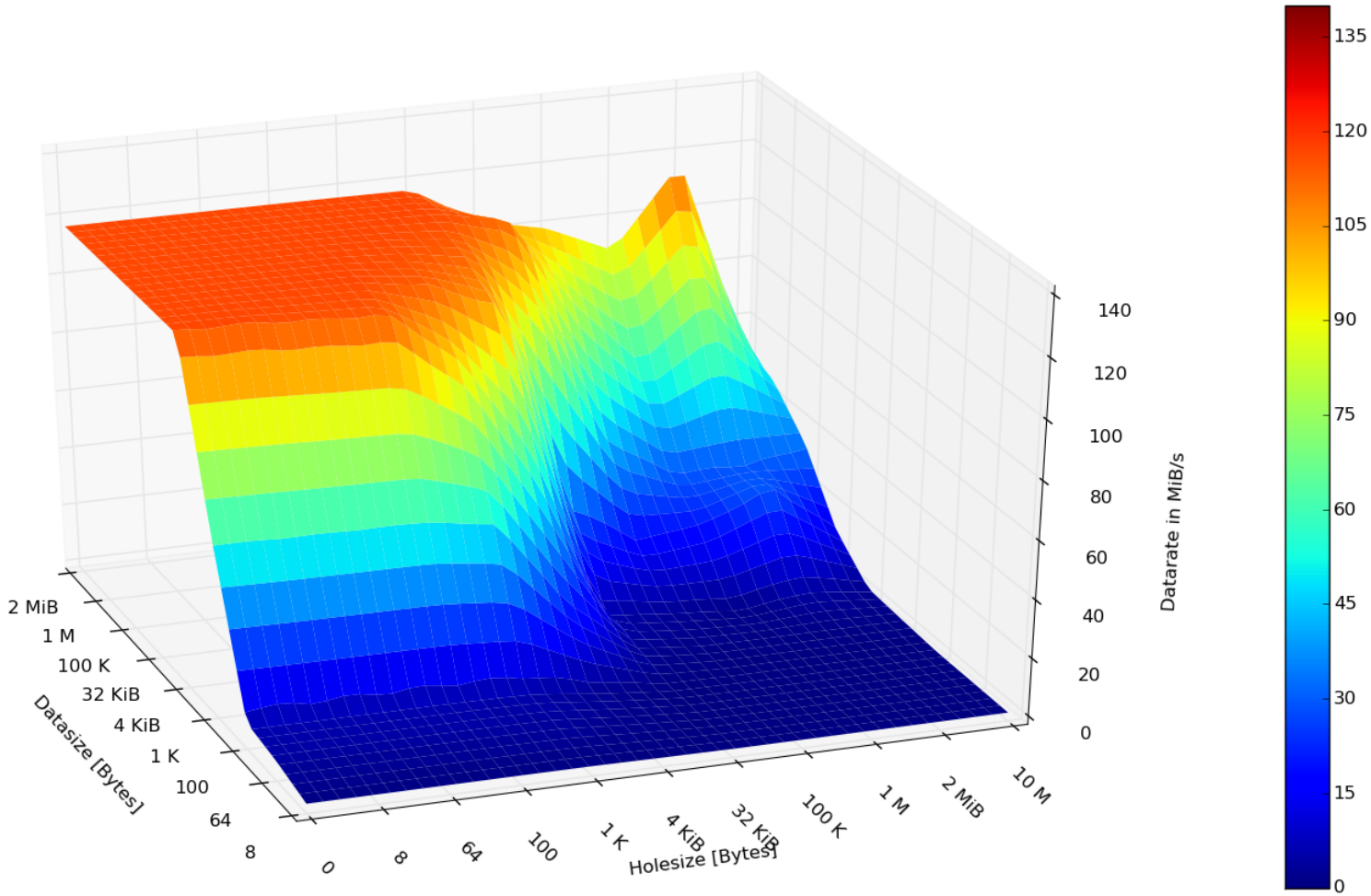

Figure 4: Naive 1 node WR 128 KiByte stripe reading default pattern

Reading ROMIO

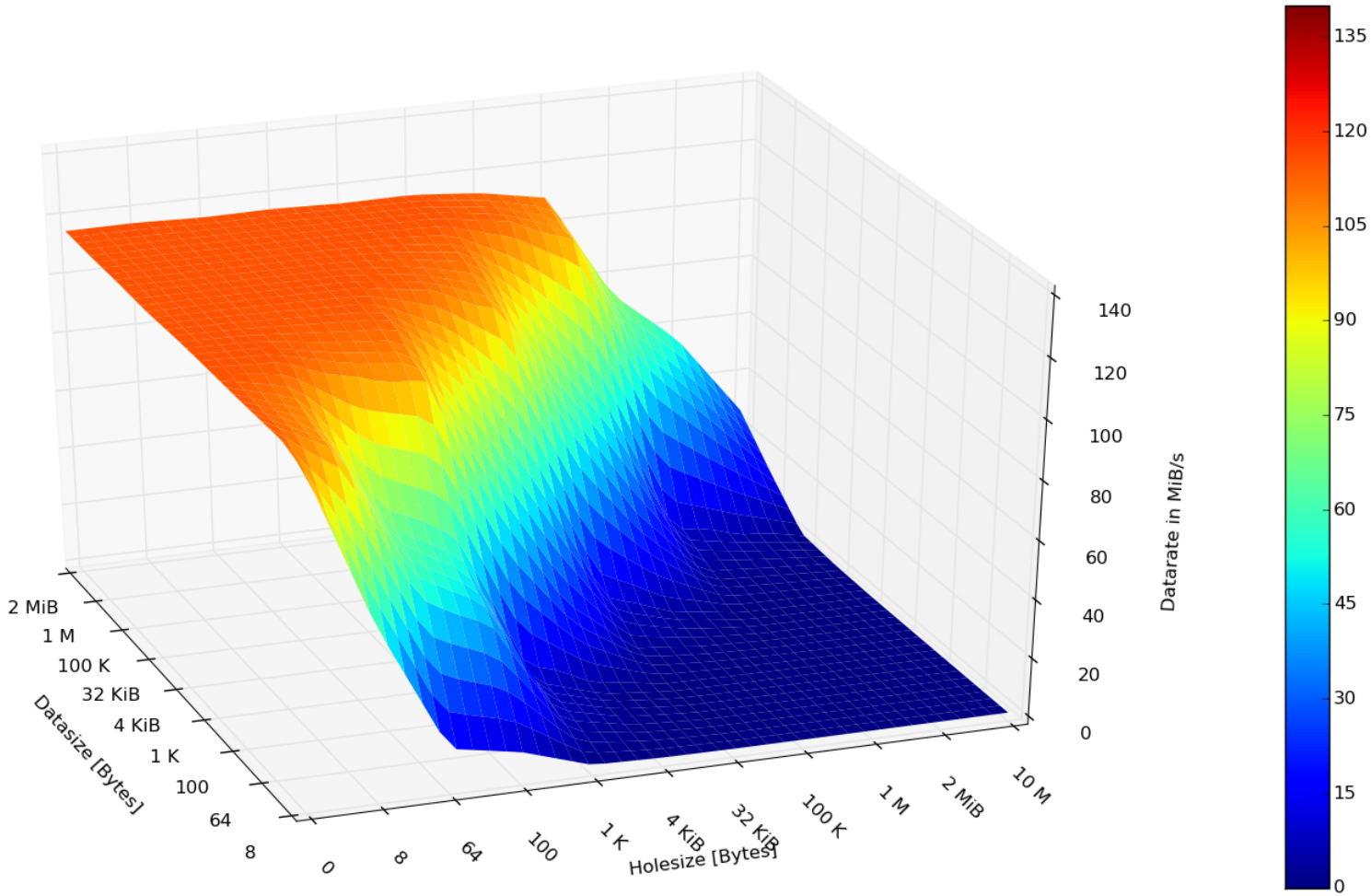

Figure 5: ROMIO WR cluster read 2 nodes 2 MiByte stripes

Reading ROMIO delta to naive

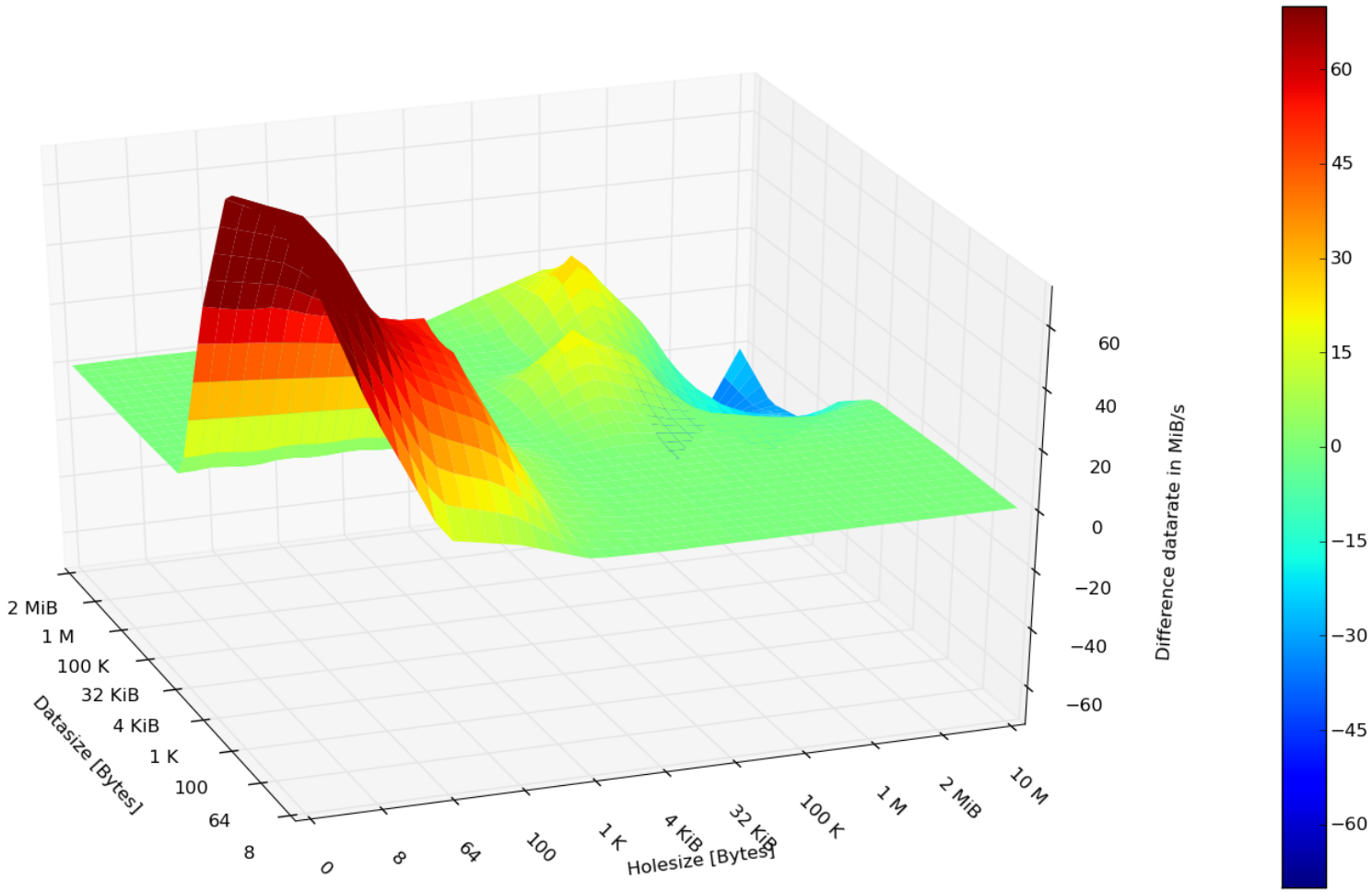

Figure 6: ROMIO WR cluster write 2 nodes 2 MiByte stripes difference to naive

Reading simple performance model

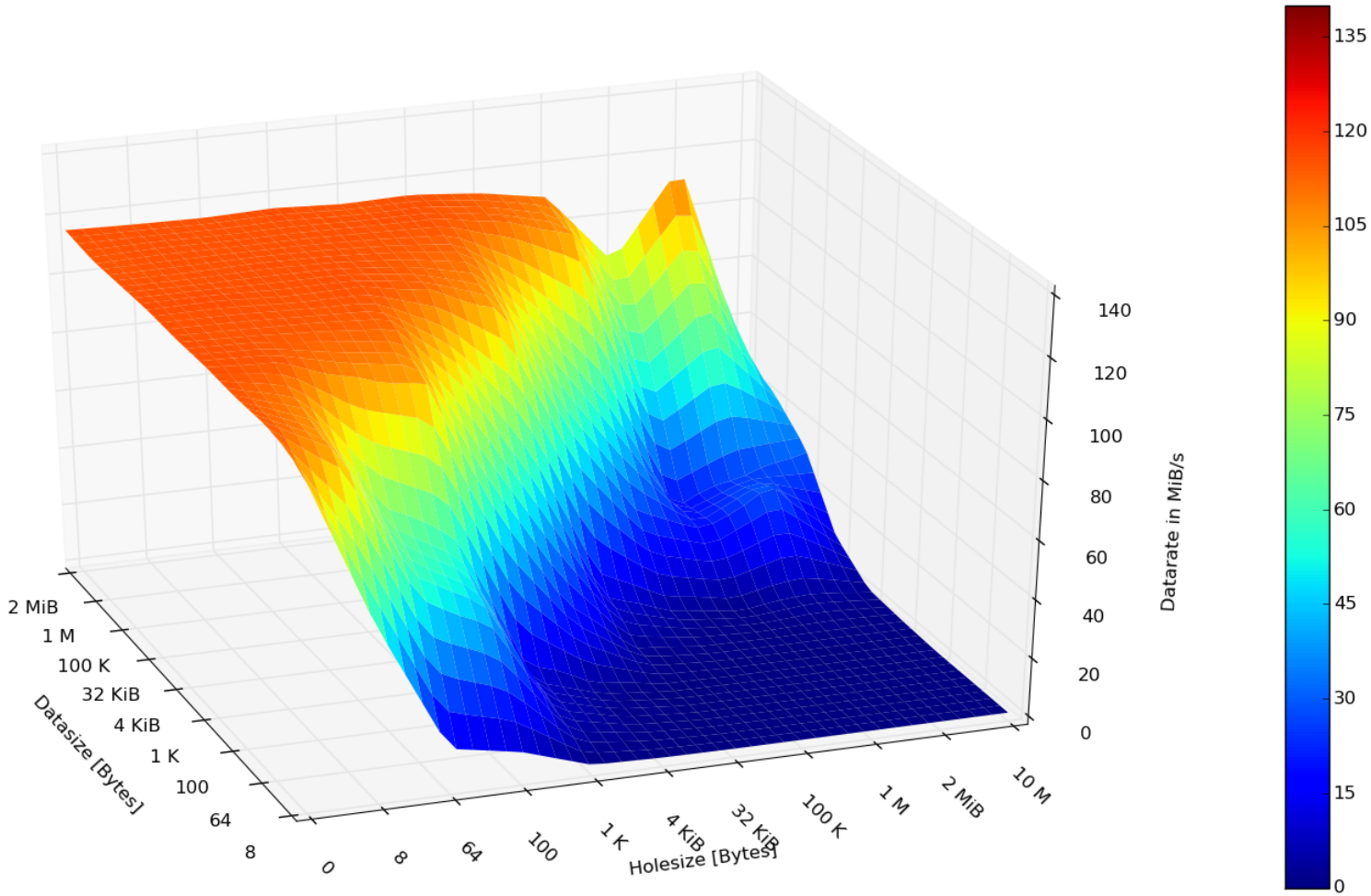

Figure 7: Simple performance Model WR cluster reading 2 nodes 2 MiByte stripes

Reading simple performance model delta to naive

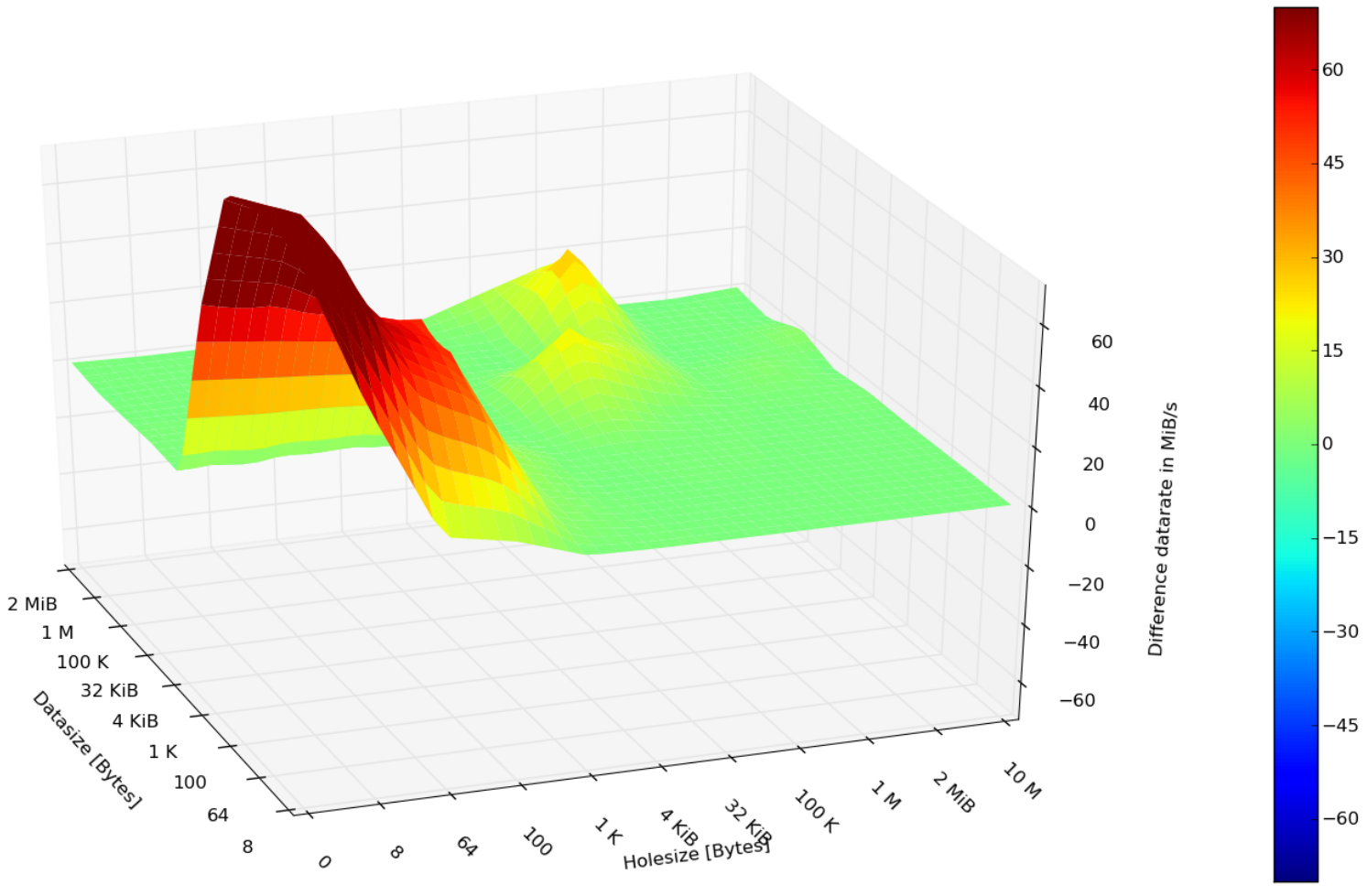

Figure 8: Simple performance Model WR cluster reading 2 nodes 2 MiByte stripes difference to naive

Writing ROMIO delta to naive

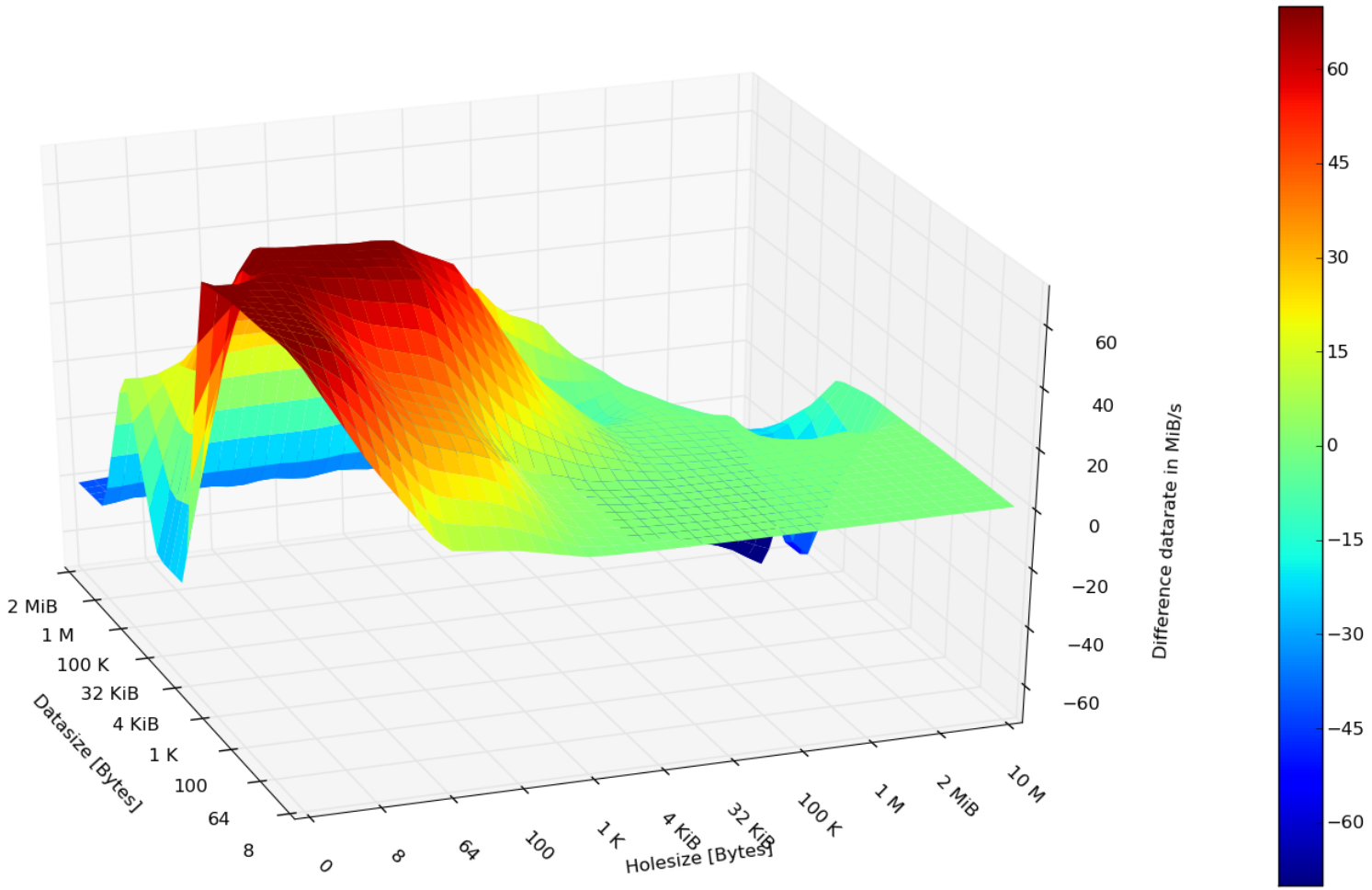

Figure 9: ROMIO WR cluster write 2 nodes 2 MiByte stripes difference to naive

Writing simple performance model delta to naive

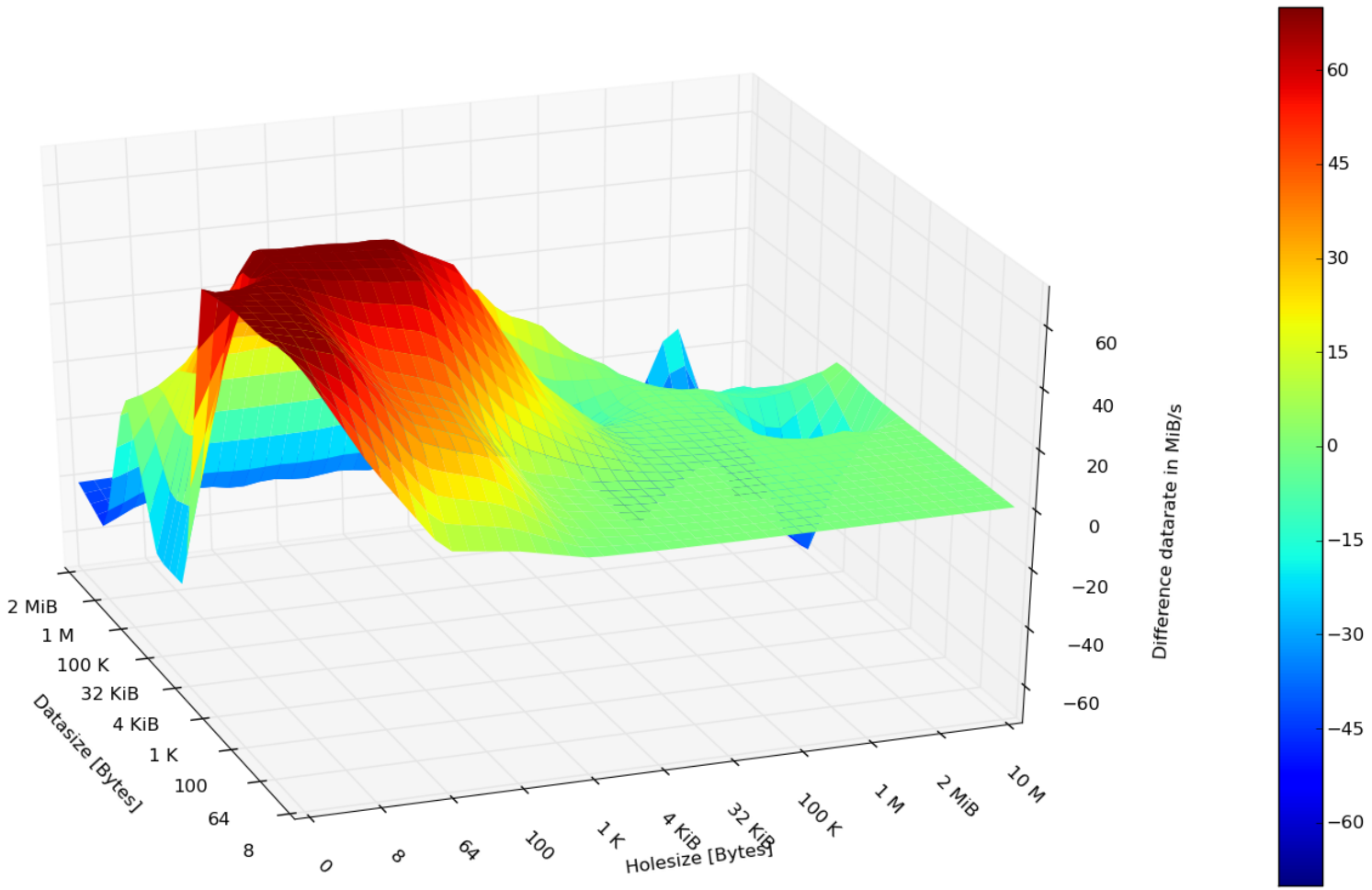

Figure 10: Simple performance Model WR cluster writing 2 nodes 2 MiByte stripes difference to naive

Reading naive aligned

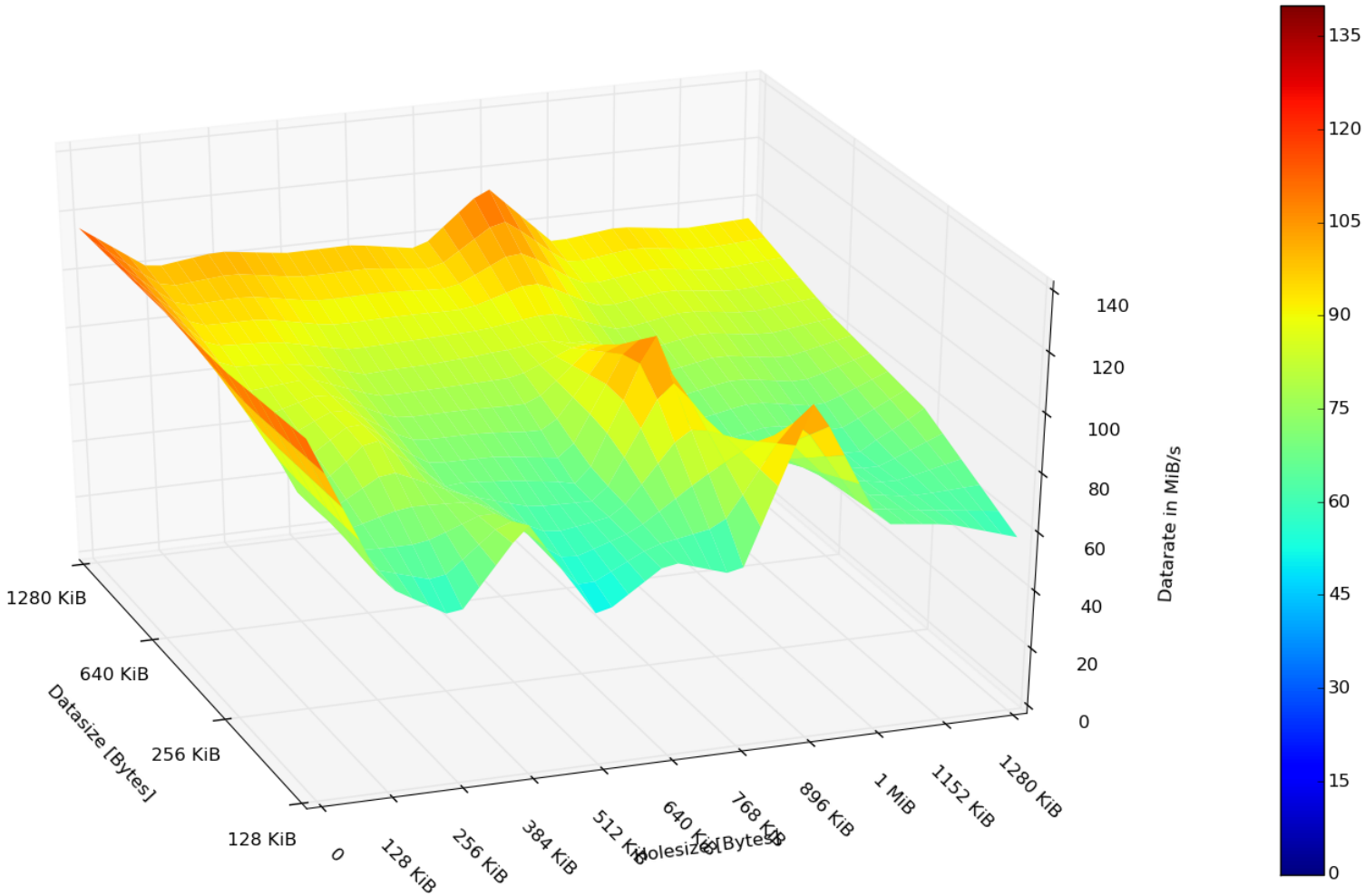

Figure 11: naive reading 2 nodes 128 KiByte stripes aligned

Writing naive aligned

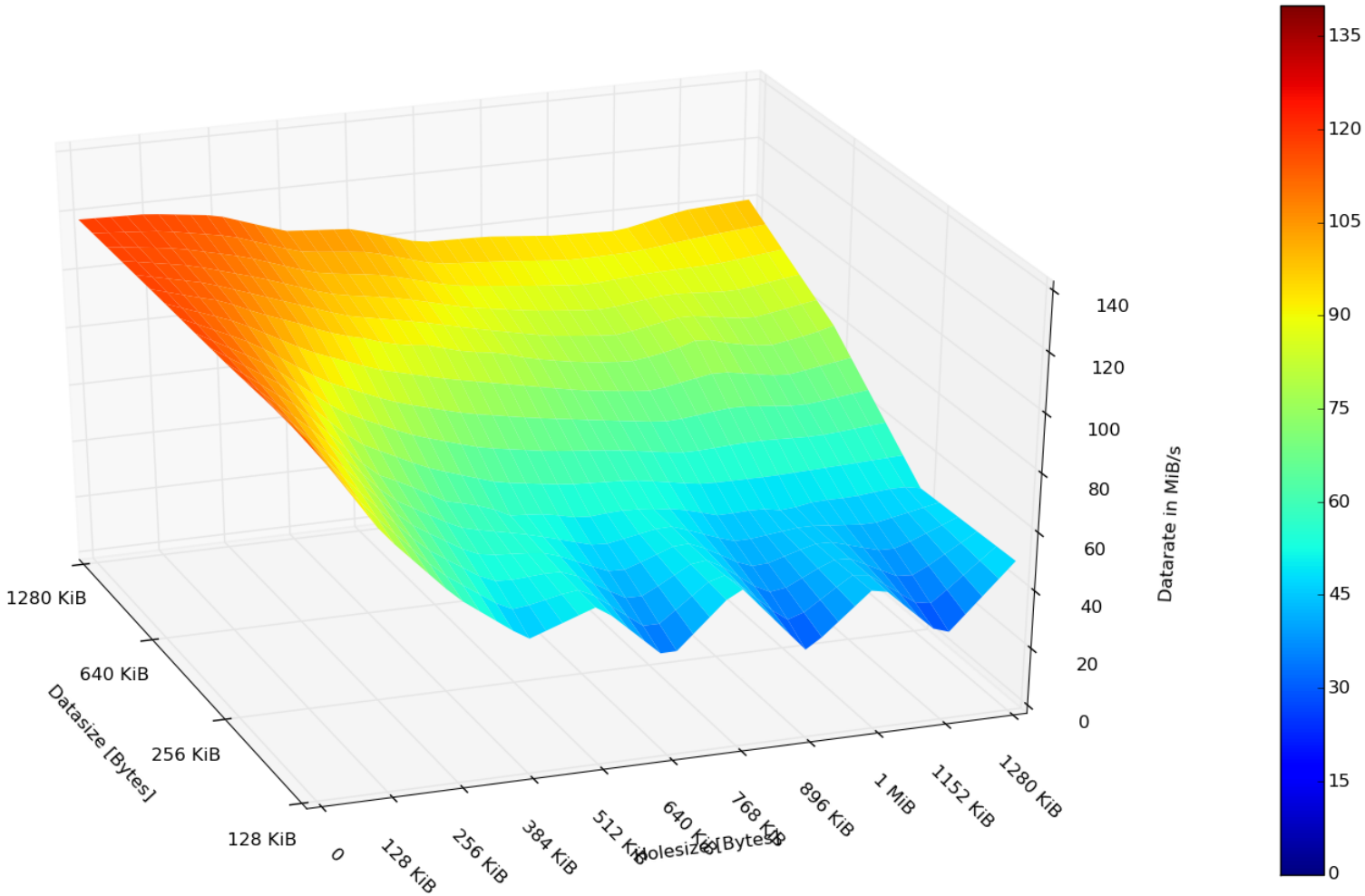

Figure 12: naive writing 2 nodes 128 KiByte stripes aligned

Summary & Conclusions

- POSIX leveled, flexible library makes research much easier
- potential in further optimization based on system characteristics
- implementation of the MPI-I/O interface is planed

Questions?

Literature

Literatur I


Rajeev Thakur, William Gropp, and Ewing Lusk.
Data sieving and collective i/o in ROMIO.
In *Frontiers of Massively Parallel Computation, 1999. Frontiers' 99. The Seventh Symposium on the*, pages 182–189. IEEE, 1999.


Yong Chen, Yin Lu, Prathamesh Amritkar, Rajeev Thakur, and Yu Zhuang.
Performance model-directed data sieving for high-performance i/o.
The Journal of Supercomputing, pages 1–25, September 2014.

Writing naive

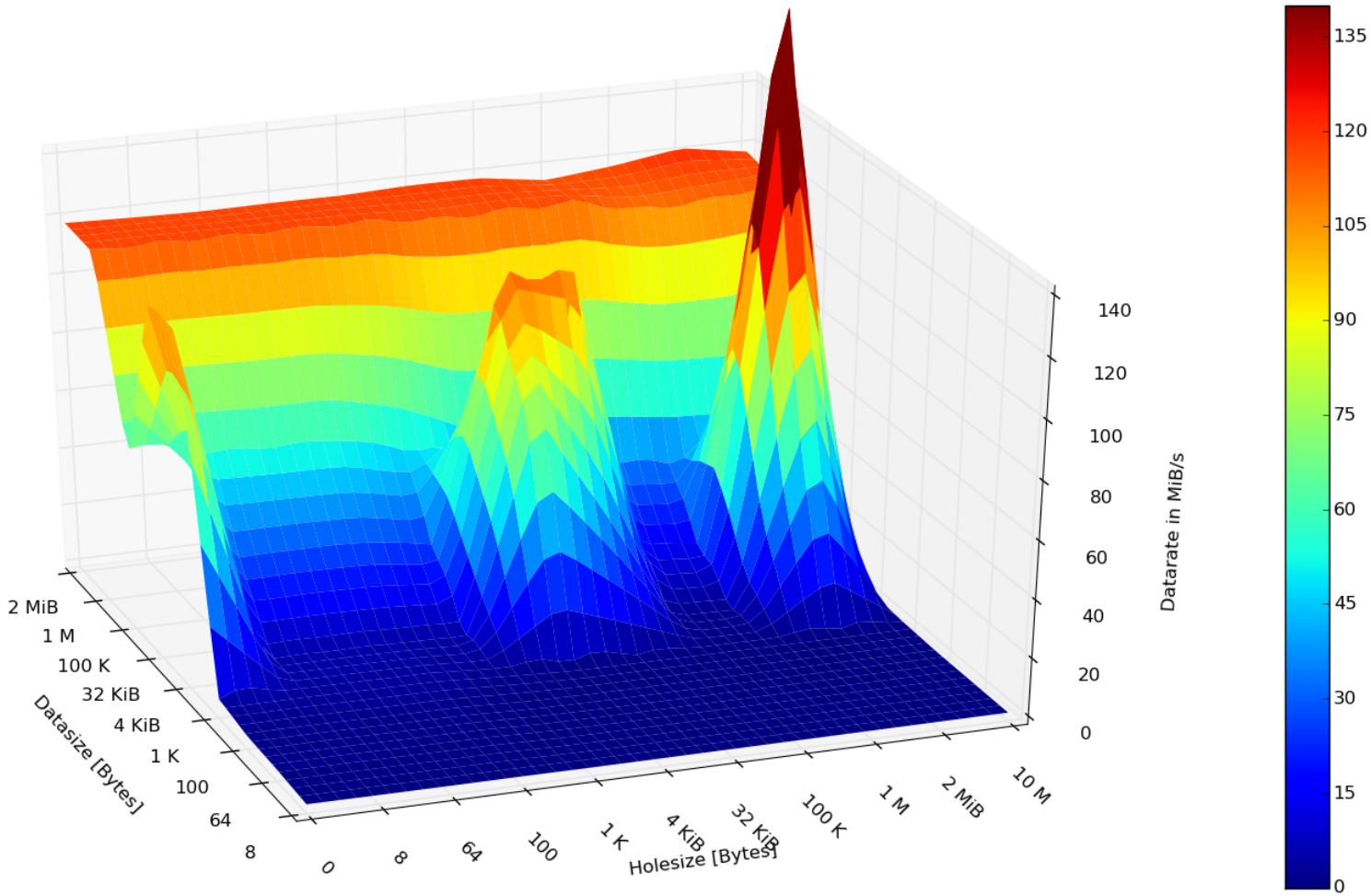

Figure 13: naive writing 10 nodes 2 MiByte stripes