Advancing Independent Non-Contiguous I/O

Julian Kunkel, Daniel Schmidtke, Enno Zickler, Eugen Betke,
Michaela Zimmer

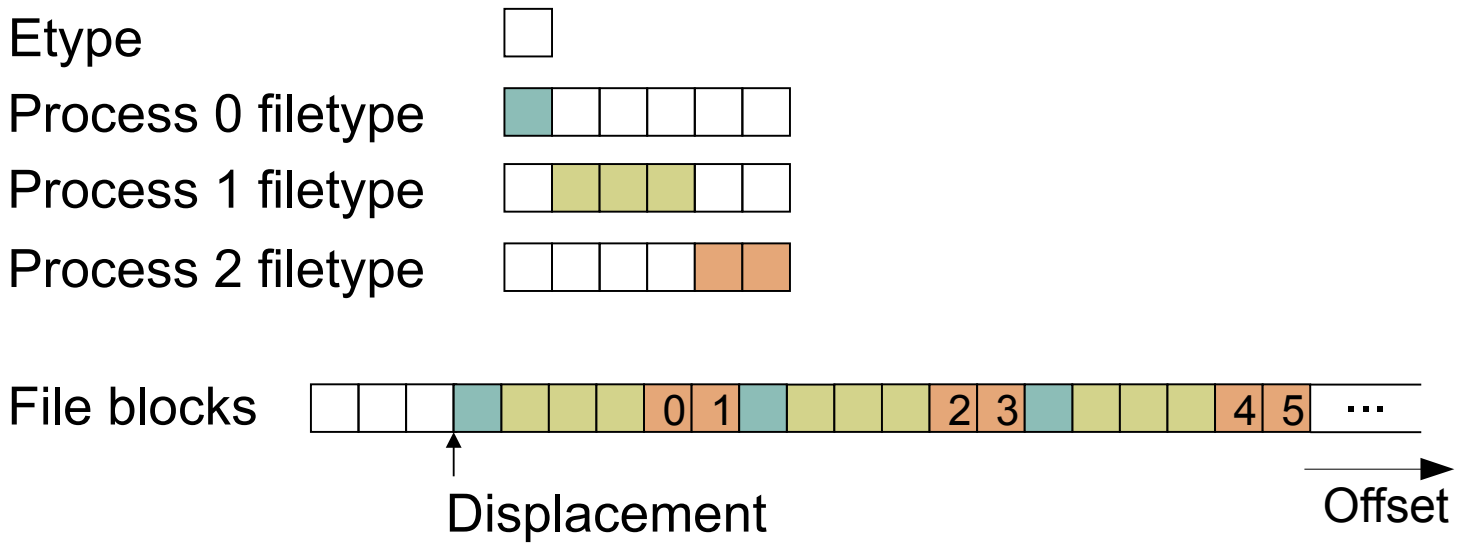
19. June 2014

Outline

Non-Contiguous I/O

- MPI-I/O allows non-contiguous I/O access
- One I/O call accesses multiple file regions
- Depends on MPI file views

Example:



Optimizing Non-Contiguous I/O

Existing Optimizations

- Collective I/O: Two-Phase, Multiphase, ...
- Independent I/O: Datasieving
  - Reads a larger block of data including holes
  - Read-modify-write cycle for modifications

Aim

- Analysis and optimization of data sieving
- (Approach could be used by each collective I/O aggregator too)

Research Perspective

- Completed:
  - Performance analysis of data sieving (BSc thesis Daniel Schmidtke)
- Ongoing:
  - Adaptive data sieving (BSc thesis Enno Zickler)
  - Machine learning of parameter space (MSc thesis Eugen Betke)
- Michaela will advance the ML further in her PhD

Performance Analysis

Benchmark

- `mpi-pattern-bench`
- Creates arbitrary datatype in file view
  - Define by tuples of (incremental offset, size)
- Sets hints: ind_*_buffer_size, romio_ds_*

Example

```
./mpibench-pattern 1 8048 "0-1024,1024-1024,1024" test.dat enable read 104857600
```

Read 1 datatype(s) at a time
8048 MiB file, (offset-size),(offset,size),(hole)

System Configuration

Test system

- 1 compute node with one process (!)
- 10 I/O nodes with Lustre

Compute Nodes

- Dual-socket Intel Xeon X5650@2.67 GHz
- Ubuntu 12.04
- Applications are compiled with: GCC 4.7.2, OpenMPI 1.6.5

I/O Nodes

- Intel Xeon E3-1275@3.4 GHz, 16 GByte RAM
- Seagate Barracuda 7200.12 (ca. 100 MiB/s)
- CentOS 6.5, Lustre 2.5

Experiments

Pattern

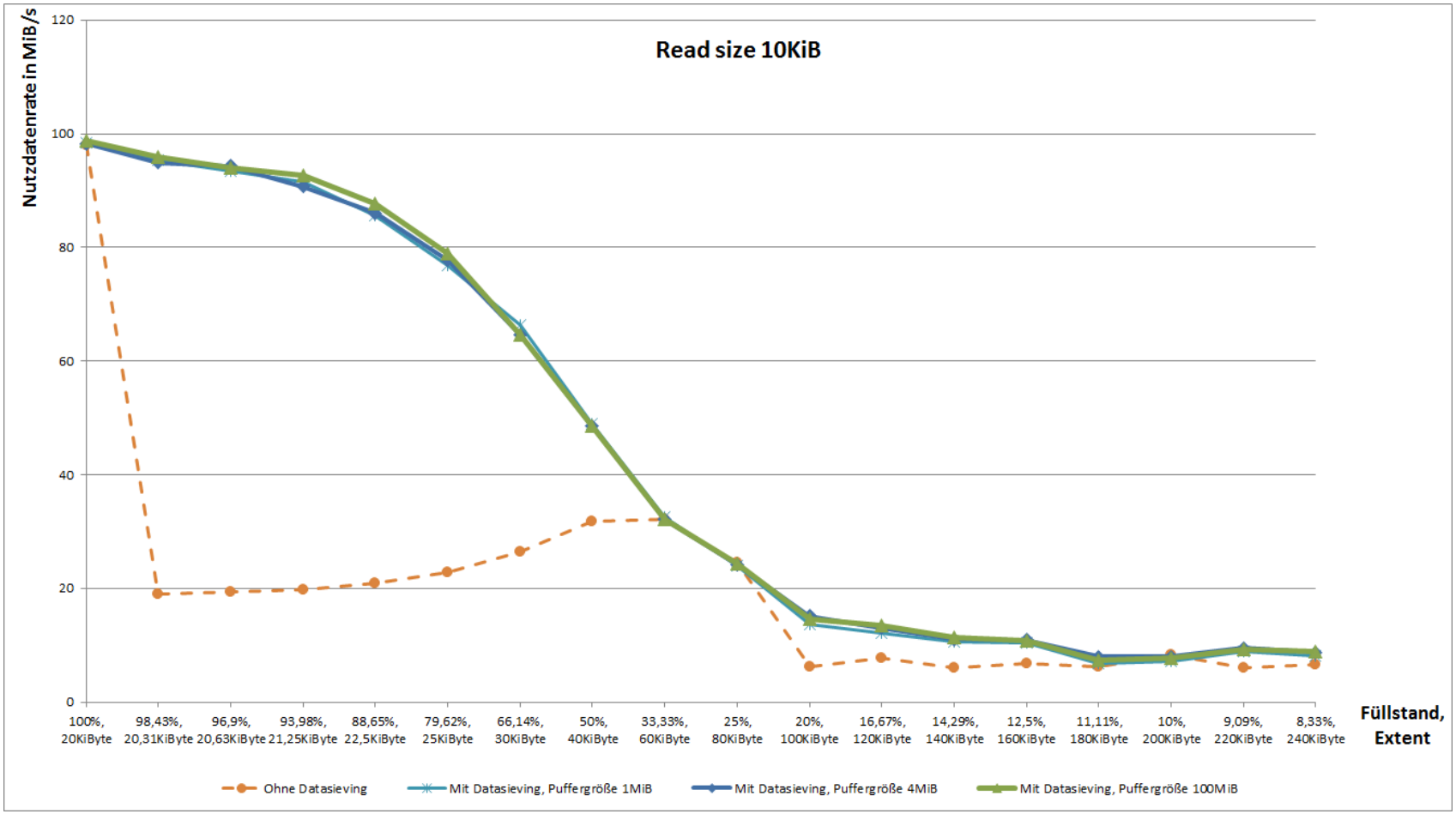
- Basic pattern -X-X
- Vary amount of data and hole size
- Characterized by fill-level = size / extent

Other factors

- Datasieving On/off
- Independent buffer size (1 MiB, 4 MiB or 100 MiB)
- Pre-filled file, $\geq$ 3x measured

Only reads for the moment (writes behave worse)

Results

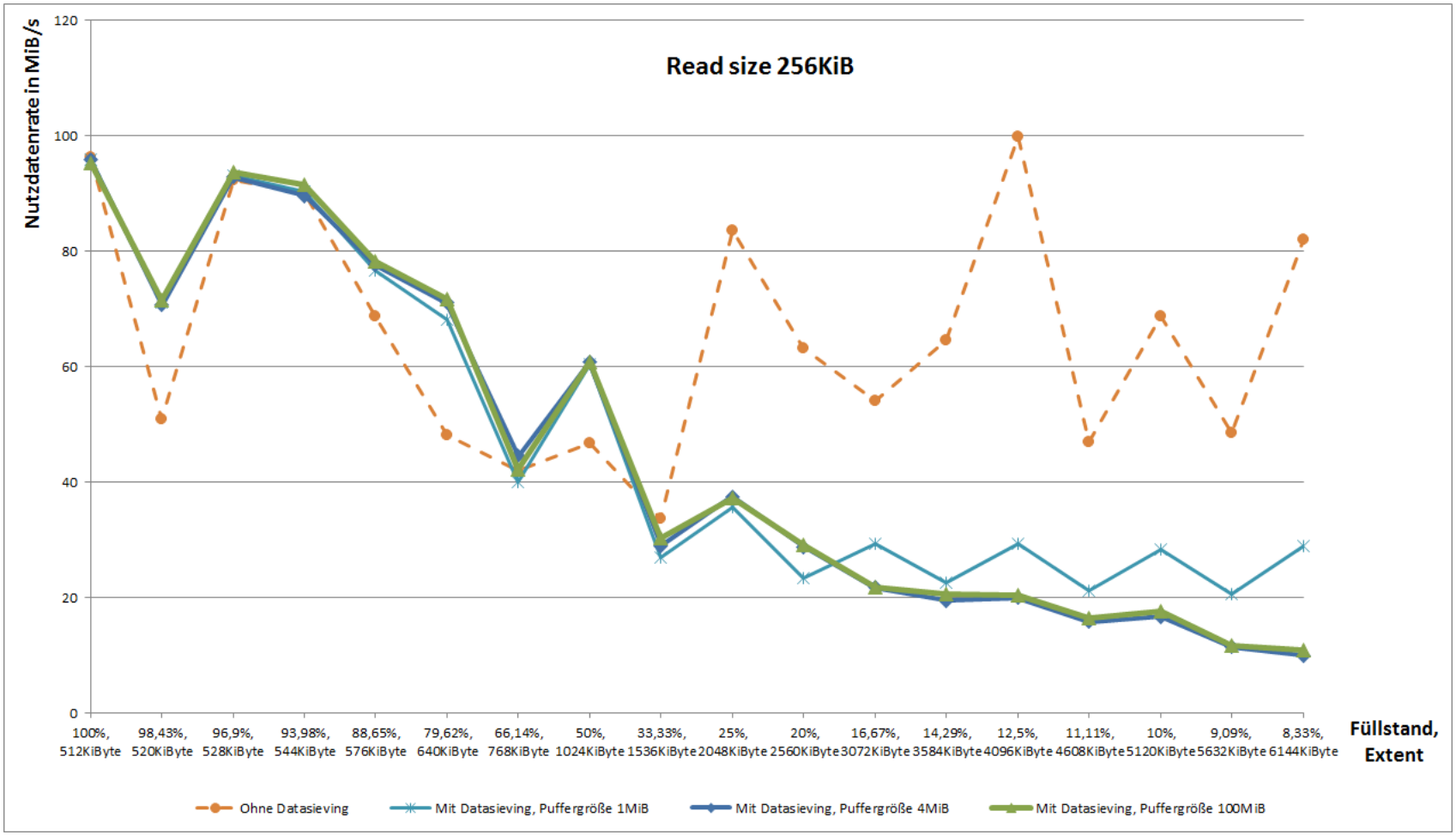


- Data-sieving is beneficial
- Read-ahead of HDD and Linux helps

Results

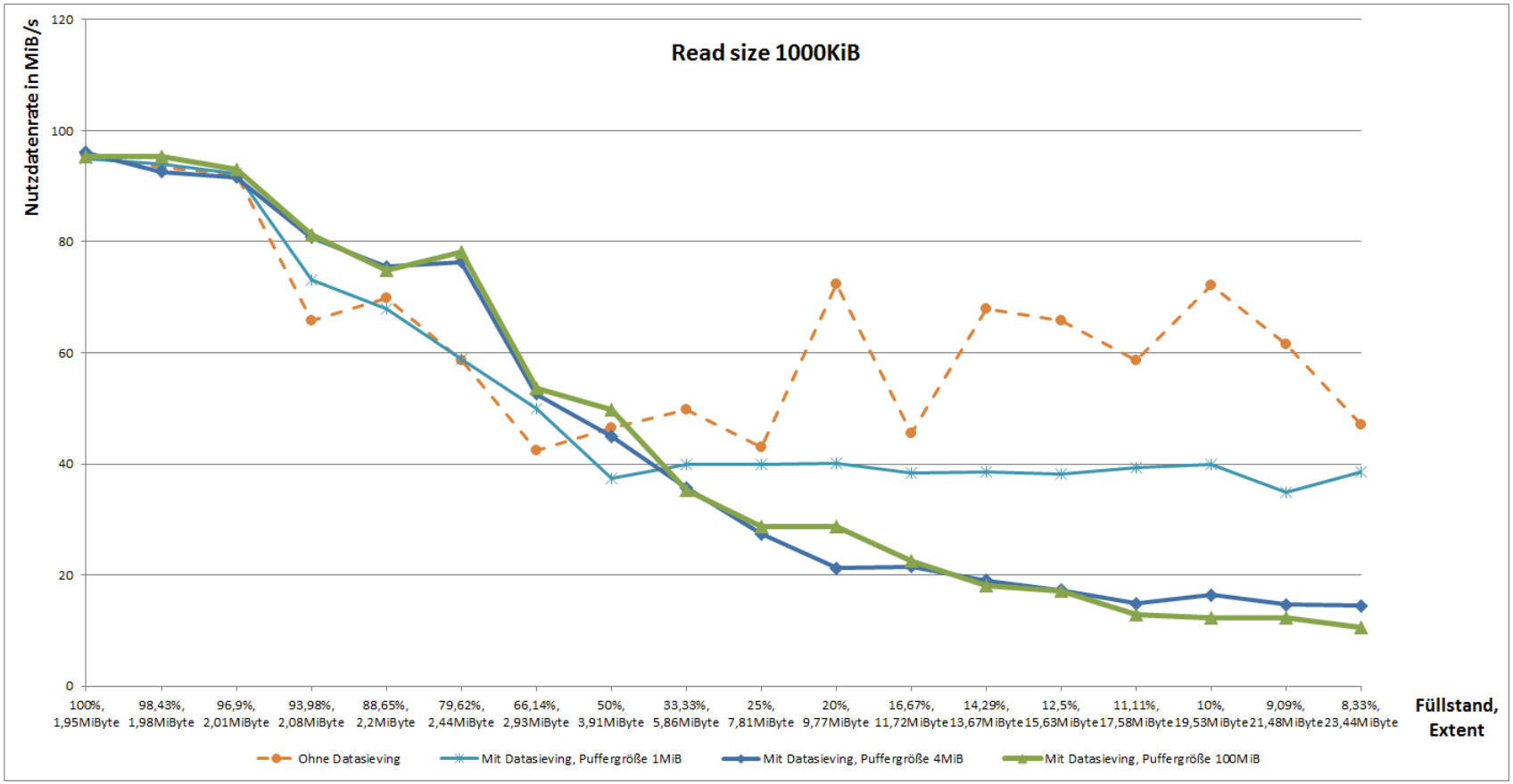
- Data sieving disadvantagous for larger holes
- Cannot be turned off selectively for these cases

Results



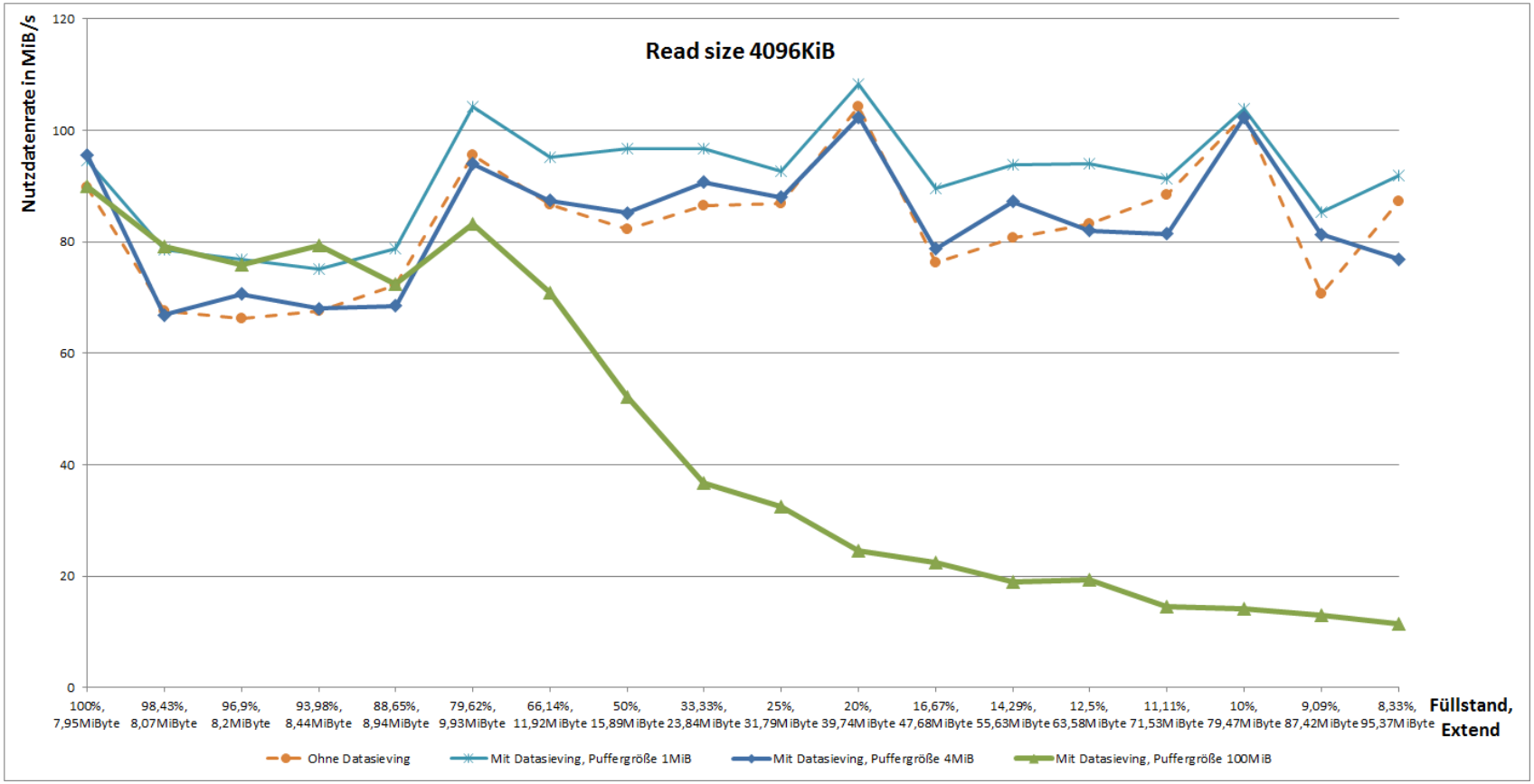
- Wait a second, larger buffers are even slower?
- Some weird patterns...

Results



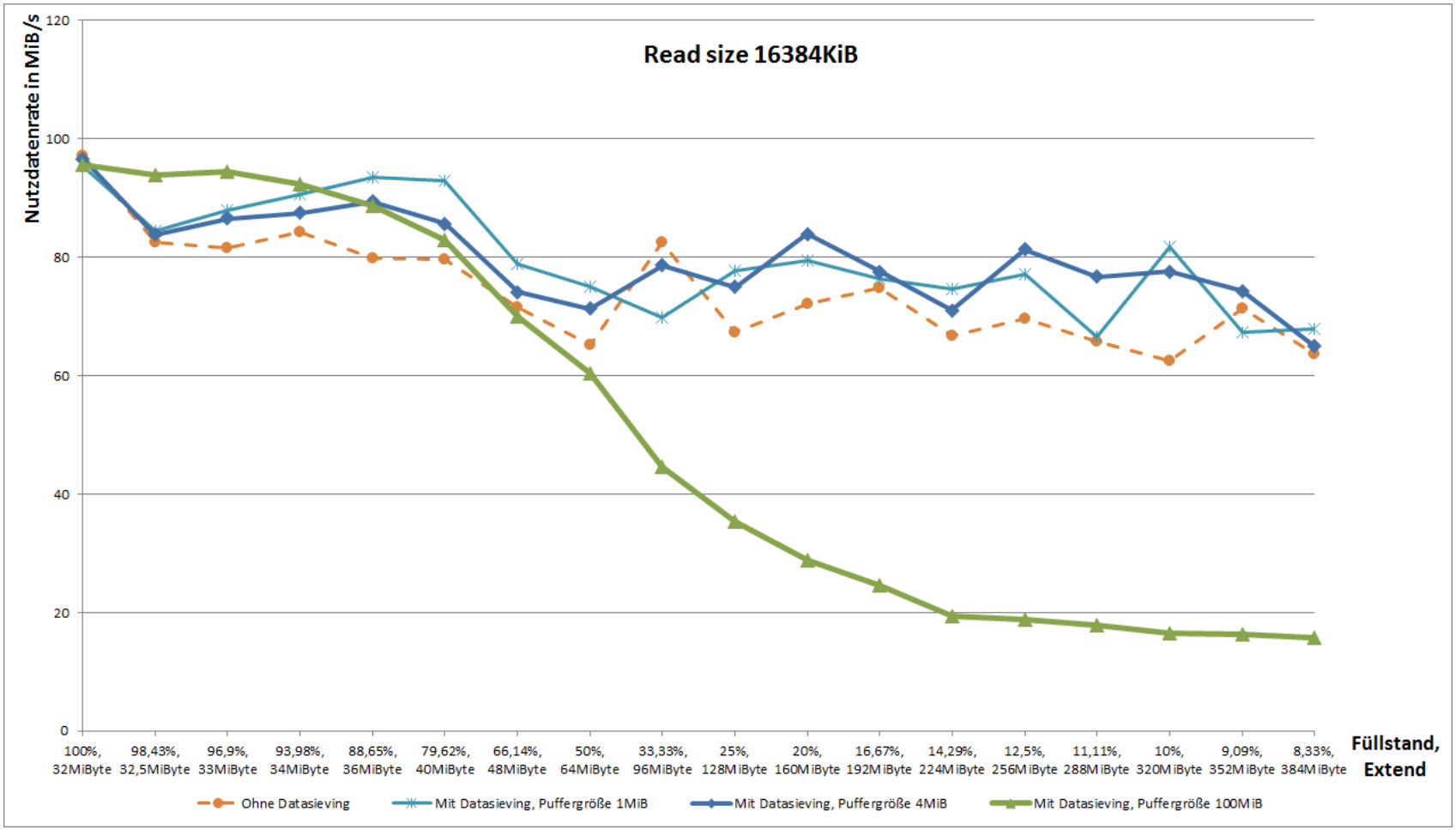
- Wait a second, larger buffers are slower?
- But sometimes slower too ;-)

Results



- Bigger holes is faster (but amount of data the same)

Results



- Larger blocks result in slower performance :-)

Outline

Theoretic Considerations

Pattern

- Sequence of holes and data
- Repeats (half a datatype should probably not written)

Performance Factors

- Read-ahead of Linux
- Costs for a (system) call
- Costs for transferring (unneeded) data
- Locking costs
- Data distribution (stripe size)
- Latency, throughput of block storage
- Block alignment

None of these are explicitly included in ROMIO's data sieving

Theoretic Considerations

Pattern

- Sequence of holes and data
- Repeats (half a datatype should probably not written)

Performance Factors

- Read-ahead of Linux
- Costs for a (system) call
- Costs for transferring (unneeded) data
- Locking costs
- Data distribution (stripe size)
- Latency, throughput of block storage
- Block alignment

None of these are explicitly included in ROMIO's data sieving

Adaptive Data Sieving

- Analyze pattern when file view is set
  - Sequence of offset, size pairs
  - Decide for subsequent accesses if they should be blocked
- Several basic parameters:
  - Max mergeable hole size
  - Max buffer size
  - Misalignment penalty
  - File system blocksize
  - (may be extended)
- A machine model could be used to set technical hints
  - According to the system’s performance factors
- Machine learning and SIOX can be used, too

Machine Learning of Parameter Space

Benefit of machine learning

- We can extract rules and learn sth.
- Autonomous operation possible
- Potential feedback with parameter space exploration tool

Issues to address (in the following)

- Workflow (learning and applying) in a real system
- Learning strategy
- Reducing patterns to ML input vector (I/O signature)

Workflow

- A plugin for the SIOX trace-reader feeds in traces
  - Builds the parameters of the machine learning algorithm
- A plugin for SIOX will read these parameters
  - Classify “new” patterns

Status: The generic workflow is almost implemented using OpenCV

Learning Strategy

- Since throughput is measurable, supervised learning is possible
- In a real system performance will vary based on the usage
  - We'll ignore this fact for the moment
  - Later, statistics about runs, e.g. quartiles could be handled
  - Problem: singular observations will skew results
- Testing: Cross-validation with observations
- Offline and online both possible with SIOX

Learning strategy

A1 : Measure performance of various patterns and options

A2 : Use observations as they come in (by apps)

A3 : Extend A2 – explore missing testcases automatically

Status: Generic test with OpenCV are positive; start with A1

Reducing patterns to ML input vector

Problems and open questions

- Any file view is possible; sequence of (offset, size)
- Unbounded dimensionality of parameter space
- “I/O signature” reduces parameter space, e.g.
  - Fill-level (this one is clearly suboptimal)
  - Defined histogram for holes and blocks (2^n stepping)
- Which signature to pick?
- Distance metrics in parameter space?
  - Motivation: Tread hole sizes the same as block sizes?
- Interpolation of “response” parameters (for unknown patterns)
  - Non-linear behavior of several parameters

Summary

- Existing data sieving with its “buffer size” tunable is suboptimal
- Adaptive data sieving supports more technical parameters
- Machine learning approach to parameterize these
- Plenty of open questions, initial approach to pursue is clear
- Maybe machine aware “Intelligent” data sieving is a chance
  - Cost function seems easy to compute
  - But: Hopefully the machine model is correct
- Future: ML to determine parameters or new models for DS