Block Locality Caching for Data Deduplication

Dirk Meister (*)
Jürgen Kaiser
Andre Brinkmann

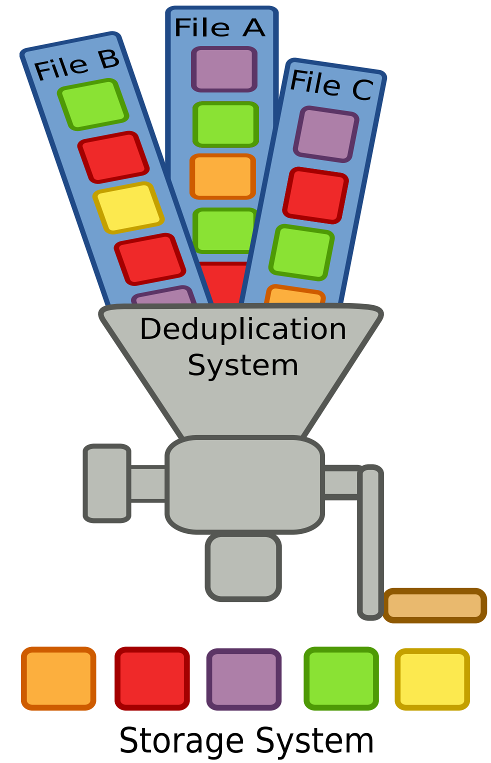
(*) work done while working at University of Mainz

JOHANNES GUTENBERG UNIVERSITÄT MAINZ

Data Deduplication

- Data deduplication
  - Class of storage reduction approaches
  - Remove coarse-grained redundancy
  - Major usage: Backup systems

- Fingerprint-based Deduplication
  - Chunking
  - Fingerprinting
  - Duplicate Chunk Detection



Chunk Lookup Disk Bottleneck

- Baseline approach
  - Disk-based Chunk Index
  - ≈ 12,000 IO/s for each 100 MB/s throughput
  - With ≈ 200 random IO/s per disk, throughput is limited

- Way out: Using special properties of backup workloads
  - High deduplication ratios from week to week
  - High likelihood of similar chunk order between weeks (chunk order locality property)

State of the Art approaches

- Prediction and caching schemes
  - **Predict** likely next chunks
  - **Pre-fetching** on-disk data structure containing likely next chunks
  - **Caching** of recently pre-fetched chunks

  - Avoid chunk index lookups for already existing chunks
  - Approaches differ on how backup locality is “captured”
  - Examples: [Zhu 2008], [Lillibridge 2009], …

Container Caching (Zhu et al. 2008)

- Pre-fetching and caching scheme
- New chunks are stored in container (of 4 MB)
  - In order, stream-aware
- Pre-fetch container (meta data) into container cache
- Use container cache to avoid chunk index lookups

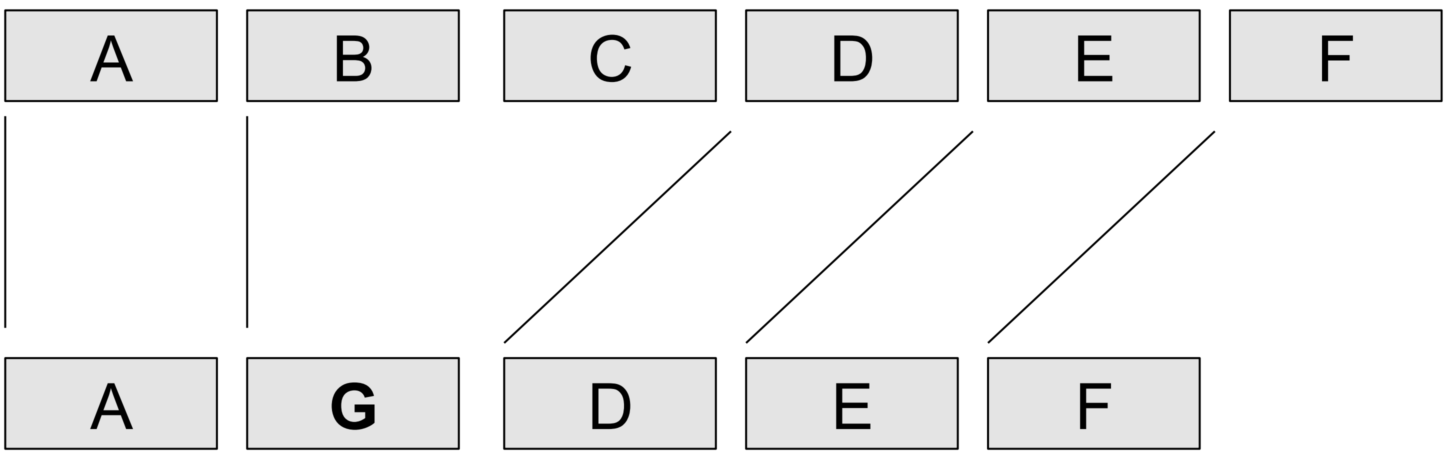
- ➔ Approach is based on order in which chunks have been written initially.

- Successfully used in Data Domain backup systems
  - Together with Bloom Filter to lookups for new chunk

Block Locality Caching

- A novel approach to overcome disk bottleneck
- Called “Block Locality Caching” (BLC)
- Uses a different “clue” to capture backup locality

- Basic Idea:
  Find an alignment between block recipes of two backups



Estimate Alignment(s)

- “Block hint” per chunk
  - Last block number/offset that accessed a chunk
  - Extension of Chunk Index

- Difference cache
  - Difference: current block number – block hint of chunk
  - A difference denotes a possible alignment
  - LRU cache of “successful” differences

- Block cache
  - LRU cache of pre-fetched block recipes

Request for block *b* with chunk *f*

If block cache contains *f*:

    Mark *f* as redundant

Else:

    […]

Request for block *b* with chunk *f*

If block cache contains *f*:

    Mark *f* as redundant

Else:

    For all differences *d* in difference cache (in order):

        $p = b - d$

        Fetch block recipe *p* from disk into block cache

        If block cache contains f:

            Mark *f* as redundant

    […]

Request for block *b* with chunk *f*

If block cache contains *f*:

    Mark *f* as redundant

Else:

    For all differences *d* in difference cache (in order):

        $p = b - d$

        Fetch block recipe *p* from disk into block cache

        If block cache contains f:

            Mark *f* as redundant

    If chunk index contains *f:*

        Load block recipe based on *f*'s block hint h into block cache

        $d = b – h$

        Add *d* to difference cache

        Mark *f* as redundant

    Else:

        Mark *f* as new

BLC Illustration

Block Index:
Block 104 – [0xA, 0xC, 0xE]
Block 105 – [0xB, 0xD, 0xF]

Chunk Index:
0xA – block hint 104
0xB – block hint 105
0xC – block hint 104
0xD – block hint 105
0xE – block hint 104
0xF – block hint 105

Difference Cache:
<empty>

Block Cache:
<empty>

New requests:
Block 200, chunks [0xA, 0xC, 0xG]
Block 201, chunks [0xE, 0xB, 0xD]

BLC Illustration

<u>Block Index:</u>
Block 104 – [0xA, 0xC, 0xE]
Block 105 – [0xB, 0xD, 0xF]

<u>Chunk Index:</u>
0xA – block hint 104
0xB – block hint 105
0xC – block hint 104
0xD – block hint 105
0xE – block hint 104
0xF – block hint 105

<u>Difference Cache:</u>
<empty>

<u>Block Cache:</u>
<empty>

New requests:
Block 200, chunks [<u>0xA</u>, 0xC, 0xG]
Block 201, chunks [0xE, 0xB, 0xD]

BLC Illustration

Block Index:
Block 104 – [0xA, 0xC, 0xE]
Block 105 – [0xB, 0xD, 0xF]

Chunk Index:
0xA – block hint 200
0xB – block hint 105
0xC – block hint 104
0xD – block hint 105
0xE – block hint 104
0xF – block hint 105

Difference Cache:
Diff 96

Block Cache:
0xA, 0xC, 0xE

New requests:
Block 200, chunks [0xA, 0xC, 0xG]
Block 201, chunks [0xE, 0xB, 0xD]

BLC Illustration

Block Index:
Block 104 – [0xA, 0xC, 0xE]
Block 105 – [0xB, 0xD, 0xF]

Chunk Index:
0xA – block hint 200
0xB – block hint 105
0xC – block hint 104
0xD – block hint 105
0xE – block hint 104
0xF – block hint 105

Difference Cache:
Diff 96

Block Cache:
0xA, 0xC, 0xE

New requests:
Block 200, chunks [0xA, 0xC, 0xG]
Block 201, chunks [0xE, 0xB, 0xD]

BLC Illustration

<u>Block Index:</u>
Block 104 – [0xA, 0xC, 0xE]
Block 105 – [0xB, 0xD, 0xF]

<u>Chunk Index:</u>
0xA – block hint 200
0xB – block hint 105
0xC – block hint 200
0xD – block hint 105
0xE – block hint 104
0xF – block hint 105

<u>Difference Cache:</u>
Diff 96

<u>Block Cache:</u>
0xA, 0xC, 0xE

New requests:
Block 200, chunks [0xA, 0xC, <u>0xG</u>]
Block 201, chunks [0xE, 0xB, 0xD]

BLC Illustration

<u>Block Index:</u>
Block 104 – [0xA, 0xC, 0xE]
Block 105 – [0xB, 0xD, 0xF]

<u>Chunk Index:</u>
0xA – block hint 200
0xB – block hint 105
0xC – block hint 200
0xD – block hint 105
0xE – block hint 104
0xF – block hint 105
0xG – block hint 200

<u>Difference Cache:</u>
Diff 96

<u>Block Cache:</u>
0xA, 0xC, 0xE

New requests:
Block 200, chunks [0xA, 0xC, <u>0xG</u>]
Block 201, chunks [0xE, 0xB, 0xD]

BLC Illustration

<u>Block Index:</u>
Block 104 – [0xA, 0xC, 0xE]
Block 105 – [0xB, 0xD, 0xF]
Block 200 – [0xA, 0xC, 0xG]

<u>Chunk Index:</u>
0xA – block hint 200
0xB – block hint 105
0xC – block hint 200
0xD – block hint 105
0xE – block hint 104
0xF – block hint 105
0xG – block hint 200

<u>Difference Cache:</u>
Diff 96

<u>Block Cache:</u>
0xA, 0xC, 0xE

New requests:
Block 200, chunks [0xA, 0xC, 0xG]
Block 201, chunks [<u>0xE</u>, 0xB, 0xD]

BLC Illustration

<u>Block Index:</u>
Block 104 – [0xA, 0xC, 0xE]
Block 105 – [0xB, 0xD, 0xF]
Block 200 – [0xA, 0xC, 0xG]

<u>Chunk Index:</u>
0xA – block hint 200
0xB – block hint 105
0xC – block hint 200
0xD – block hint 105
0xE – block hint 201
0xF – block hint 105
0xG – block hint 200

<u>Difference Cache:</u>
Diff 96

<u>Block Cache:</u>
0xA, 0xC, 0xE

New requests:
Block 200, chunks [0xA, 0xC, 0xG]
Block 201, chunks [0xE, <u>0xB</u>, 0xD]

BLC Illustration

<u>Block Index:</u>
Block 104 – [0xA, 0xC, 0xE]
Block 105 – [0xB, 0xD, 0xF]
Block 200 – [0xA, 0xC, 0xG]

<u>Chunk Index:</u>
0xA – block hint 200
0xB – block hint 105
0xC – block hint 200
0xD – block hint 105
0xE – block hint 201
0xF – block hint 105
0xG – block hint 200

<u>Difference Cache:</u>
Diff 96

<u>Block Cache:</u>
0xA, 0xC, 0xE
0xB, 0xD, 0xF

New requests:
Block 200, chunks [0xA, 0xC, 0xG]
Block 201, chunks [0xE, <u>0xB</u>, 0xD]

BLC Illustration

Block Index:
Block 104 – [0xA, 0xC, 0xE]
Block 105 – [0xB, 0xD, 0xF]
Block 200 – [0xA, 0xC, 0xG]

Chunk Index:
0xA – block hint 200
0xB – block hint 201
0xC – block hint 200
0xD – block hint 105
0xE – block hint 201
0xF – block hint 105
0xG – block hint 200

Difference Cache:
Diff 96

Block Cache:
0xA, 0xC, 0xE
0xB, 0xD, 0xF

New requests:
Block 200, chunks [0xA, 0xC, 0xG]
Block 201, chunks [0xE, 0xB, 0xD]

BLC Illustration

Block Index:
Block 104 – [0xA, 0xC, 0xE]
Block 105 – [0xB, 0xD, 0xF]
Block 200 – [0xA, 0xC, 0xG]
Block 201 – [0xE, 0xB, 0xD]

Chunk Index:
0xA – block hint 200
0xB – block hint 201
0xC – block hint 200
0xD – block hint 201
0xE – block hint 201
0xF – block hint 105
0xG – block hint 200

Difference Cache:
Diff 96

Block Cache:
0xA, 0xC, 0xE
0xB, 0xD, 0xF

New requests:
Block 200, chunks [0xA, 0xC, 0xG]
Block 201, chunks [0xE, 0xB, 0xD]

Trace-based Simulation

- Replay of two weekly full backup traces

	UPB	JGU
Weekly Backups	15	13
Total Capacity	6.6 TB	16.1 TB
Deduplication Factor	1:20	1:11.5

- Comparing: BLC, Container Caching

- Key Metric:
  - Number of IO operations used for Pre-fetch operations + Chunk Index Lookups
  - Proxy for system throughput

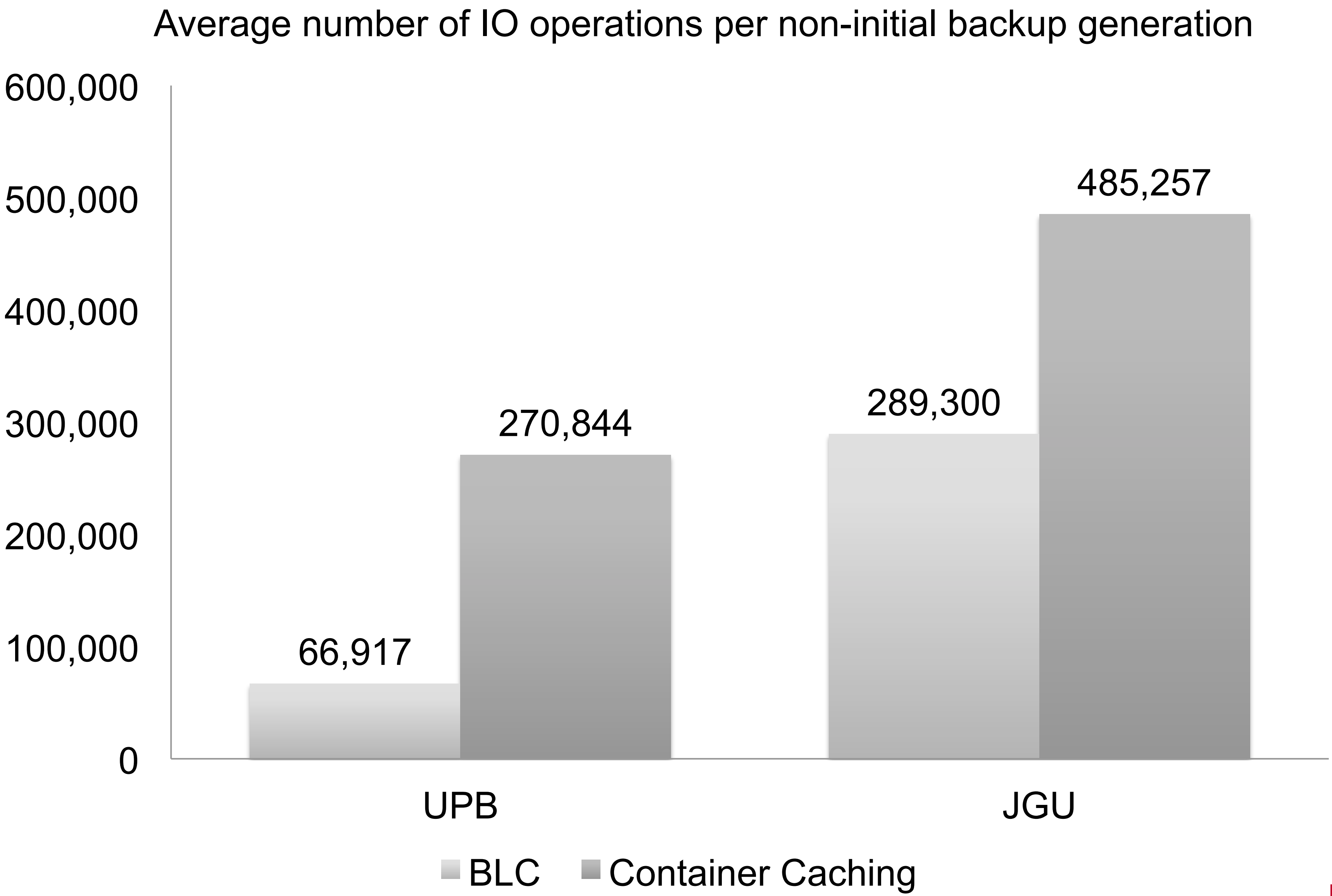
Configurations

- Large configurations space evaluated
- Best configuration for each approach is reported

- BLC approach
  - Block chunk cache with 2,048 blocks (@ 32 chunks)
  - Difference cache with 4 values
  - LRU chunk cache with 1,024 chunks

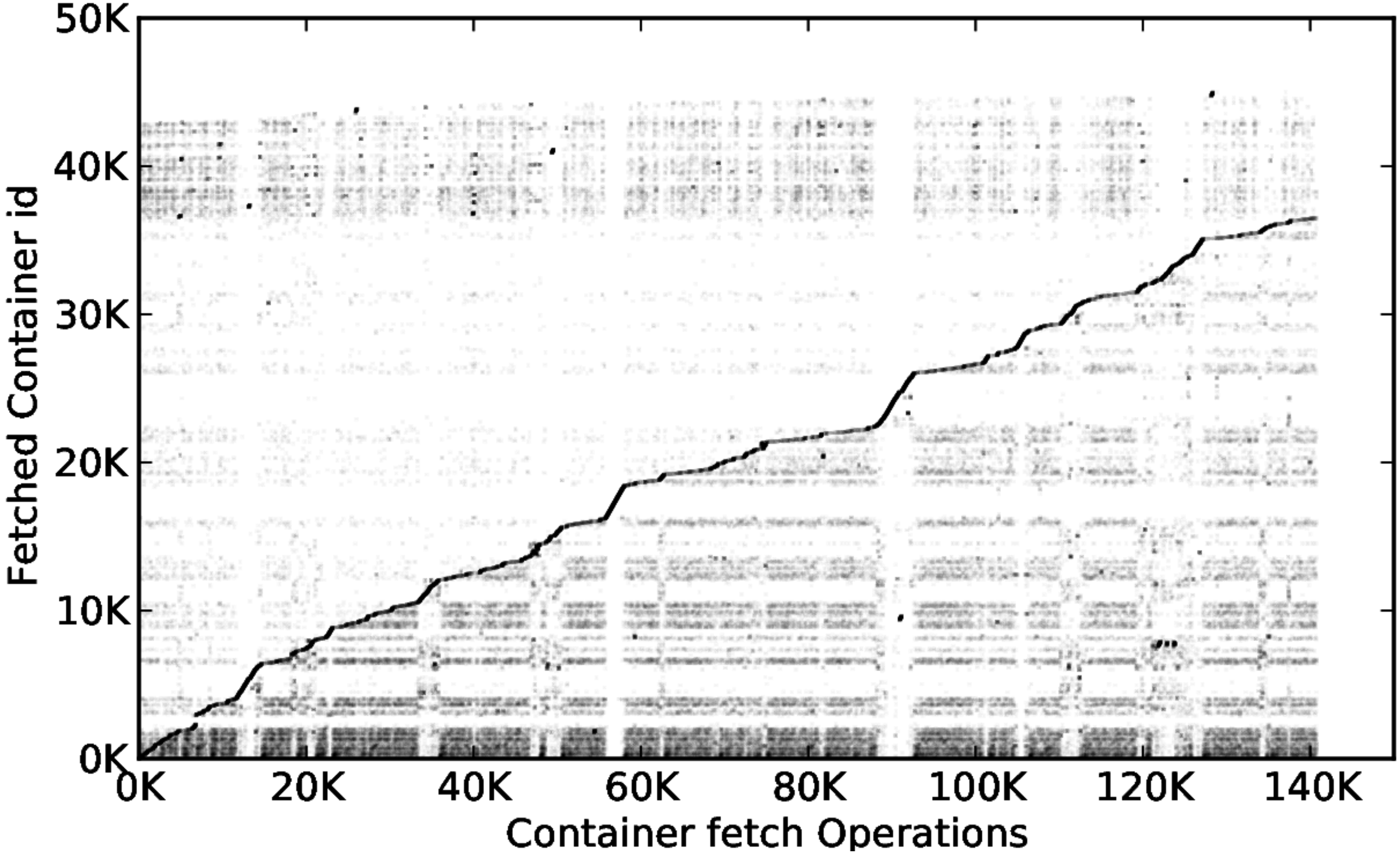
- Container Caching Approach
  - 4 MB container (on average 1,024 chunks)
  - LRU container cache with 1,024 containers
  - LRU chunk cache with 1,024 chunks

Simulation Results



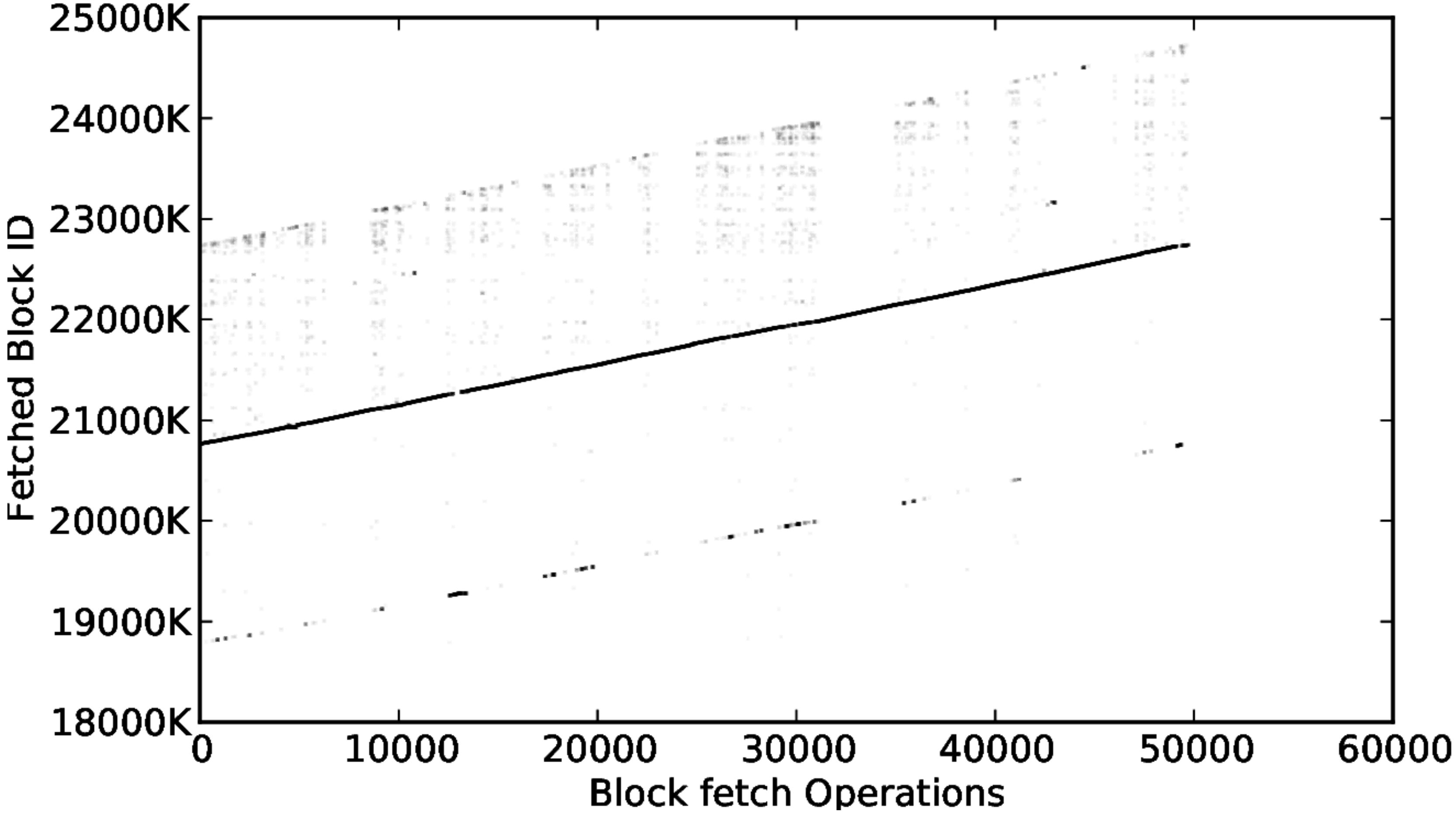
Simulation Result: Container Caching

Average Disk IO per Week: 270,844
Figure: IO Pattern, UPB data set, Week 13



Simulation Result: BLC

Average Disk IO per Week: 66,917
Figure: IO Pattern, UPB data set, Week 13



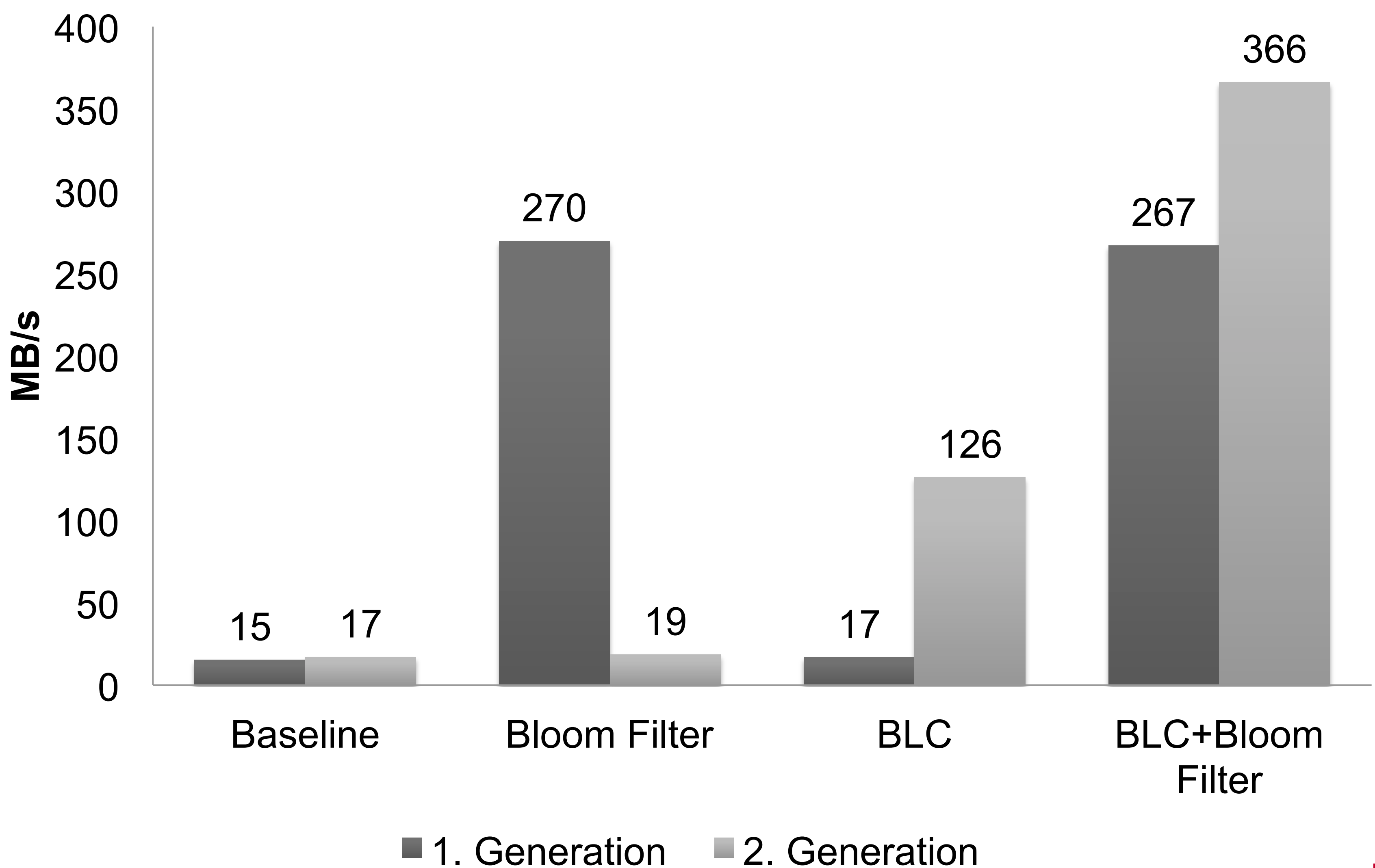
Experimental Evaluation

- Prototype-based on dedupv1 system

- Single-node system
  - 4-core CPU
  - 16 GB RAM
  - 6 500 GB disks as RAID-0
  - 1 Intel 520 SSD for WAL

- Single backup client
  - Generated data stream based on UPB data set
  - Two backup generations
  - 128 GB each

Prototype Evaluation



Conclusion

- Novel exact approach to overcome disk bottleneck
- Uses less IO operations than Zhu et al.'s Container Caching
- Also, likely to be less prone to aging

- Future Work: "Sampling BLC"
  - an approximate version of the BLC
  - Sampled Chunk Index used for 'block hints'
  - Intuition: Should work well with straight pre-fetch line

Thank you

QUESTIONS?

FS-C: https://code.google.com/p/fs-c/
dedupv1: https://github.com/dedupv1/dedupv1

FS-C [SYSTOR'09]

- Tool suite for data deduplication research
  - Tracing: File system walk with content fingerprints
  - Analysis: e.g. of deduplication ratio
    - Exact in-memory, Exact Hadoop-based, Estimated (Harnik's method)
  - Simulation
    - Replay trace files, e.g. to simulate a backup workload.

- Developed since 2008 (Master thesis of the author)
- Foundation for various contributions

FS-C Data Sets

	UPB	JGU	ENG
Weekly Backups	15	13	21
Total Capacity	6.6 TB	16.1 TB	318.7 GB
Deduplication Factor	1:20	1:11.5	1:40