Reducing Disk I/O Performance Sensitivity for Large Numbers of Sequential Streams

George Panagiotakis, Michail D. Flouris and Angelos Bilas

Foundation for Research & Technology - Hellas (FORTH-ICS)

ICDCS 2009
Motivation

• Media servers, scientific data applications
  – Write-once, read-many workloads
  – Large sequential files: Media (HD video), Scientific data
  – Parallel retrieval of sequential I/O streams from disks
• Sequential access: simple & efficient for disks
• Challenge
  – Maintain max read throughput while scaling to large number of I/O streams per disk
• Disk capacity increase ➔ less spindles per stream
  – 2TByte disk holds 440 full-size DVD movies
Linux I/O Schedulers

1 stream: 60MB/sec

Number of Concurrent Seq. Streams
Iozone on Ext3: Reading Sequential Files, 4KB Block

Parallel reading of sequential streams on 1 SATA disk
Traditional Solutions

• Caching & aggressive/static prefetching
• Efficient I/O schedulers
  – Anticipatory, Fair-queuing
• Work well
  – Small number of streams
  – Prefetching buffers fit in memory
• However
  – Various workloads need large number of streams
  – Storage controllers: many disks and limited memory
Other Solutions

• SSDs: expensive & low capacity
  – Behavior with high performance workloads not well understood
  – Used as a prefetching buffer?

• Data placement not practical solution
  – Predict which streams read together?
  – Stream playout short-lived vs. time to reorganize data
Overview

• Motivation
• Related work & contributions
• Disk & controller-level prefetching
• Our approach
• Evaluation
• Conclusions
Related Work

• Modeling & optimizing disks
  – [Ganger95], [Jacobson & Wilkes 91], [Ruemmler & Wilkes 94],
    [Shriver 97], [Varki et al. 04], [Zhu & Hu 02]
• I/O performance & scheduling optimizations
  – [Bachmat02], [Iyer & Druschel 01], [Kim et al. 06], [Mokbel et al. 04],
    [Shenoy & Vin 98], [Wijayaratne & Reddy 01], [Hsu & Smith 04],
    [Carrera & Bianchini 02], [Coloma et al. 05], [Yu et al. 06]
• Prefetching
  – [Shriver et al. 99], [Cao et al. 95], [Kimbrel & Karlin 00],
    [Li et al. 07], [Patterson et al. 95], [Ding et al. 07]
• Storage caching (non-sequential workloads)
  – [Chen et al. 03], [Dahlin et al. 94], [Johnson & Shasha 94], [Zhou et al. 02]
• I/O for multimedia applications
  – [Chen et al. 94], [Dey-Sircar et al. 94], [Rangan & Vin 91],
    [Reddy & Wyllie 94], [Dan et al. 95]
Contributions

• Analysis of the problem
• Solution at the host level
  – Up to 4x higher throughput with 100 streams / disk
  – Improved disk utilization with limited memory
• Our approach relies on
  – Identifying & separating sequential streams
  – Buffering & coalescing small requests in host memory
  – Notion of working set for servicing multiple I/O streams
• Validation through
  – Disksim simulation and real system experiments
  – Multiple disk & controller configurations
I/O Path

- I/O path components that perform caching & queuing
- Caches become smaller towards bottom

- Disk cache: limited size, divided into fixed segments
Disk-level Prefetching

• Achieved by
  – Increasing application request size
  – Increasing disk segment size to prefetch full segments
• Measurements with Disksim and microbenchmarks
• Larger request sizes improve throughput, if there is enough disk cache for all I/O streams
• When number of streams x req. size > cache size throughput degrades dramatically
• Increasing disk cache size and prefetching improves throughput for large number of streams
• However, disk cache size fixed by manufacturer
Controller-level Prefetching

- Prefetching at controller-level is effective when there is enough memory for all streams

- Not a solution, because one controller may have 4-16 disks and should handle thousands of streams (need GBytes of memory)
Host-level Approach

- Block-level operation, file system agnostic
- System receives block I/O requests
- Classifier detects sequential requests using bitmap
- Non-sequential requests sent directly to disks
- Requests in sequential streams sent to scheduler
Scheduling

- **Dispatch Set** (D): stream set currently in scheduler issues I/O
- **Read-ahead size** (R): size of requests actually issued to disks
- Streams remain in D until having issued **N disk requests**
- Replacement policy for streams in D: Round-Robin
- Disk req completion ➔ scheduler completes block I/O request
Staging prefetched data

- Streams removed from D staged in buffered set, until prefetched data are used by new requests or timeout expires
- Classifier looks up req. data in buffered set, completes req. if found
- Overall memory space (M): size of buffered set & dispatch set (D)
- At all times $M \geq D \times R \times N$
- Periodically garbage collect inactive/non-seq streams
Implementation

• Implemented on Linux
• User-space I/O server & stream generators
• Using asynchronous I/O, not threads
• Direct I/O to bypass kernel buffer cache
Evaluation Setup

• One storage node
  – Dual Opteron machine, 1GB memory
  – Broadcom RAID controller for 8 SATA disks
  – WD 7200rpm SATA disks (55-60 Mbytes/sec)

• Multiple client nodes
  – Necessary to saturate 8 disks
  – Issues many seq. stream requests over 1 GigE link
  – Data are not transferred over the network
Read-ahead (R)

- S: number of input streams
- M = S × R × N and S = D (fits in dispatch set)
- Substantial amount of memory required

\[
(M = D \times R \times N) \\
(D = \#S) \\
(N = 1)
\]

![Graph showing throughput vs. number of streams per disk](image)

- **R = 8MBytes** \( (M \approx 800\text{MBytes}) \)
- **R = 2MBytes** \( (M \approx 200\text{MBytes}) \)
- **R = 1MByte** \( (M \approx 100\text{MBytes}) \)
- **R = 512KBytes** \( (M \approx 50\text{MBytes}) \)
- **R = 128KBytes** \( (M \approx 12\text{MBytes}) \)
- **No Readahead**
Memory Size

• Interested in many streams that need much memory
• Fixed R value: increasing $S \rightarrow$ lower throughput
• Increased R important for high throughput

$\left( D = \frac{M}{R*N} \right)$, $N = 1$

Throughput (MB/s) vs. Memory Size (MB)
Multiple disks

- Throughput for 8 disks as S per disk increases
- Throughput drops regardless of read-ahead value R
- Bottleneck: controller due to buffer management
- Need to separate dispatched from staged streams

\[(D = S), (M = D*R*N), (N = 1)\]

- No Readahead
- \(R = 512\text{KBytes}\)
- \(R = 1\text{MByte}\)
- \(R = 2\text{MBytes}\)
Dispatched vs. staged

- 8 disk setup with dispatched < staged streams
- Better behavior with small amount of memory because of lower buffer management overhead
- Potential for high utilization by tuning R, D, N and M

![Graph showing throughput (MBytes/s) vs. number of streams per disk (#S) with equations: R = 512KBytes, D = #disks, N = 128, M = staged*N*R and R = 512KBytes (previous figure).]
Single-disk Throughput

- 1 disk with dispatched < staged streams
- Better behavior with small amount of memory compared to $S = D$ case (fits in dispatch set)

![Graph showing throughput vs. number of streams per disk](slide 17)

- $R = 512$KBytes, $D = 1$, $N = 128$, $M = \text{staged}^*N^*R$
- $R = 2$MBytes, from $S=D$ figure (slide 17)
- $R = 8$MBytes, from $S=D$ figure (slide 17)
Response Time

- Mainly interested in improving disk utilization
- Increasing $S \rightarrow$ high impact on response time
- Increasing $R$ improves response time
- Average request response time not very different among streams because of round-robin policy

![Graph showing response time variations with ReadAhead (KBytes) and different values of $S$ and $M$.]
Conclusions

• Analyze performance of many seq. I/O streams on disk
• Examine the effect of I/O subsystem parameters
• Find certain parameters can improve performance
• Propose solution at host level that
  – Identifies structures needed & parameterizes each
  – Allows setting these parameters (D, R, N, M) independently
• Implement & measure solution on real system
  – Up to 4x higher throughput with 100 streams / disk
  – Makes the I/O subsystem insensitive to number of streams
  – Approach works with limited memory
  – Response time affected by no of streams, not read-ahead
Thank you!

Questions?

“Reducing Disk I/O Performance Sensitivity for Large Numbers of Sequential Streams”
George Panagiotakis, Michail Flouris & Angelos Bilas
Foundation for Research & Technology - Hellas

http://www.ics.forth.gr/carv/scalable