A Parametric I/O Model
for Modern Storage Devices

Tarikul Islam Papon
papon@bu.edu

Manos Athanassoulis
mathan@bu.edu

presentation at DaMoN 2021
Modeling Performance

“Algorithm/Data Structure $X$ has $O(f(N))$ performance, where $N$ is the number of data pages on disk”

... is probably one of the most commonly read phrases in SIGMOD papers.
The diagram illustrates a comparison between different storage and processing units, highlighting their speeds and capacities. From top to bottom:

- **CPU**: ~5ns
- **RAM**: ~100ns
- **USB Flash Drive**, **Hard Drive**, **CD/DVD**: ~1 ms

The diagram indicates that as the speed increases, the capacity decreases, with CPUs being the fastest but also the smallest in terms of storage. Conversely, external storage devices like USB Flash Drives and CDs have larger capacities but slower access times. The upward trend from CPU to external storage signifies the trade-off between speed and capacity.
Memory Hierarchy
Traditional I/O Model

Small, fast main memory (size M)
Traditional I/O Model

Small, fast main memory (size M)

Large, slow external memory
Traditional I/O Model

One I/O at a time

Small, fast main memory
(size M)

Large, slow external memory
Traditional I/O Model

Small, fast main memory (size M) 0 access cost

Large, slow external memory
Traditional I/O Model

Small, fast main memory (size M)

Transfer cost 1 unit

Large, slow external memory

0 access cost
Traditional I/O Model

total cost $\approx$ total # reads/writes to disk

- 0 access cost
- Small, fast main memory (size M)
- Large, slow external memory

Transfer cost 1 unit
Traditional I/O Model

Two (outdated) assumptions

- **Symmetric** cost for **Read & Write** to disk
- **One I/O** at a time

Small, fast main memory (size M)  
Large, slow external memory
Hard Disk Drives
Hard Disk Drives

Two assumptions of Traditional I/O Model

**Symmetric** cost for **Read** & **Write** to disk

**One I/O** at a time
HDD Stopped Evolving

- Generally, the slowest component
- Slowest increase in performance

<table>
<thead>
<tr>
<th>Device</th>
<th>Size</th>
<th>Seq B/W</th>
<th>Time to read</th>
</tr>
</thead>
<tbody>
<tr>
<td>HDD 1980</td>
<td>100 MB</td>
<td>1.2 MB/s</td>
<td>~ 1 min</td>
</tr>
<tr>
<td>HDD 2020</td>
<td>4 TB</td>
<td>125 MB/s</td>
<td>~ 9 hours</td>
</tr>
</tbody>
</table>

HDDs are moving deeper in the memory hierarchy, and new algorithms are designed for new faster storage devices.

*How do these modern storage devices perform?*
Solid State Drives & NVMs

SSDs
- SATA SSDs
- PCIe SSDs (NVMe SSDs)
- Zoned SSDs
- Open SSDs

NVMs
- PCM
- MRAM
- STT-RAM
- 3D Xpoint (Intel’s Optane)
Modern Storage Devices

- Symmetric cost for Read & Write
- One I/O at a time
- Read/Write Asymmetry
- Concurrency
Read/Write Asymmetry
Out-of-place updates cause invalidation

Invalidation causes garbage collection
Writing in a free page isn’t costly!
Writes in SSD

Update A, B, C, D

Block 0

A B C
D E F
G H Free
Free Free Free

Block 1

Free Free Free
Free Free Free
Free Free Free
Free Free Free
Not all updates are costly!
What if there is no space?

Writes in SSD

Block 0

...
What if there is no space?

**Garbage Collection!**

<table>
<thead>
<tr>
<th>Block 0</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>D</td>
<td>E</td>
<td>F</td>
</tr>
<tr>
<td>G</td>
<td>H</td>
<td>A'</td>
</tr>
<tr>
<td>B'</td>
<td>C'</td>
<td>D'</td>
</tr>
</tbody>
</table>

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>M</td>
<td>N</td>
<td>O</td>
</tr>
<tr>
<td>P</td>
<td>Q</td>
<td>R</td>
</tr>
<tr>
<td>M'</td>
<td>N'</td>
<td>O'</td>
</tr>
<tr>
<td>P'</td>
<td>Q'</td>
<td>R'</td>
</tr>
</tbody>
</table>

Block N
Writes in SSD

What if there is no space?

Garbage Collection!

Block 0

...
What if there is no space?

Garbage Collection!

Higher average update cost (due to GC) $\rightarrow$ Read/Write asymmetry
## Read/Write Asymmetry - Example

<table>
<thead>
<tr>
<th>Intel Device</th>
<th>Advertised Random Read IOPS</th>
<th>Advertised Random Write IOPS</th>
<th>Advertised Asymmetry</th>
</tr>
</thead>
<tbody>
<tr>
<td>D5-P4320</td>
<td>427k</td>
<td>36k</td>
<td>11.9</td>
</tr>
<tr>
<td>DC-P4500</td>
<td>626k</td>
<td>51k</td>
<td>12.3</td>
</tr>
<tr>
<td>DC-P4610</td>
<td>643k</td>
<td>199k</td>
<td>3.2</td>
</tr>
<tr>
<td>Optane 900P</td>
<td>550k</td>
<td>500k</td>
<td>1.1</td>
</tr>
<tr>
<td>Optane H10</td>
<td>330k</td>
<td>250k</td>
<td>1.3</td>
</tr>
</tbody>
</table>
Read/Write Asymmetry

Asymmetry-Aware Algorithms
Concurrency
Internals of an SSD

Parallelism at different levels (e.g. channel, chip, die, plane block, page)
Concurrency in SSD (simplified)
Benchmarking
Benchmarking

Tools

• Custom micro-benchmarking infrastructure
• fio
• Intel’s SPDK

Setup

• With File System
• Without File System
Measuring Asymmetry/Concurrency (With FS)

Device: Dell P4510 (1TB)

IOPS vs. # Threads

- 4K Random Read
- 8K Random Read

×10³
Measuring Asymmetry/Concurrency (With FS)

Device: Dell P4510 (1TB)

For 4K random read,

- **Asymmetry**: 2.8
- **Concurrency**: 70

---

For 4K random write,

- **Concurrency**: 1.8x

For 8K random read,

- **Concurrency**: 2.8x

For 8K random write,
Measuring Asymmetry/Concurrency (With FS)

Device: Dell P4510 (1TB)

For 8K random write,

*Asymmetry*: 1.8

*Concurrency*: 10

---

Graph showing IOPS vs. # Threads for 4K and 8K random read and write operations.
Measuring Asymmetry/Concurrency (With FS)

Asymmetry and concurrency depends on request type and access granularity.

Device: Dell P4510 (1TB)

- 4K Random Read × 10^3
- 4K Random Write
- 8K Random Read
- 8K Random Write

2.8x
1.8x
Measuring Asymmetry/Concurrency (Without FS)

Device: Dell P4510 (1TB)

For 4K random reads,

**Asymmetry**: 3

**Concurrency**: 14

- 4K Random Read
- 4K Random Write
- 8K Random Read
- 8K Random Write

IOPS

×10^3

# Threads

1000

400

200

0

0

10

20

30

40

50

3x

3x
Measuring Asymmetry/Concurrency (Without FS)

Device: Dell P4510 (1TB)

Much stable performance without the file system!

Graph showing IOPS vs. # Threads for different operations (4K and 8K Random Read and Write). The graph indicates much stable performance without the file system.
Can the File System be the **Bottleneck**?

**Interrupt-based model**

- Request is submitted to OS
- Driver processes the request
- Data is read from h/w to buffer
  - Interrupt is generated
  - CPU is notified
  - Data is read from the buffer
Can the File System be the Bottleneck?

Device: Dell P4510 (1TB)
### Measuring Asymmetry/Concurrency

#### Table 2: Empirical Asymmetry and Concurrency.

<table>
<thead>
<tr>
<th>Device</th>
<th>4KB</th>
<th>8KB</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\alpha$</td>
<td>$k_r$</td>
</tr>
<tr>
<td>Optane SSD</td>
<td>1.1</td>
<td>6</td>
</tr>
<tr>
<td>PCiLe SSD (with FS)</td>
<td>2.8</td>
<td>80</td>
</tr>
<tr>
<td>PCiLe SSD (w/o FS)</td>
<td>3.0</td>
<td>16</td>
</tr>
<tr>
<td>SATA SSD</td>
<td>1.5</td>
<td>25</td>
</tr>
<tr>
<td>Virtual SSD</td>
<td>2.0</td>
<td>11</td>
</tr>
</tbody>
</table>
Modern Storage Devices

- **Asymmetry (Random R/W)**
- **Concurrency (#channels)**

**Plot Description:**
- Green circles: Non-Optane Series
- Orange circles: Optane Series
- Black circles: Used Devices

- Sequential Asymmetry
- Symmetry 5
- Symmetry 1

**Observation:**
- Most devices have high asymmetry.
How should the I/O model be adapted in light of read/write asymmetry and concurrency?
PIO($M, k_r, k_w, \alpha$) assumes a fast main memory with capacity $M$, and storage of unbounded capacity that has read/write asymmetry $\alpha$, and read (write) concurrency $k_r$ ($k_w$).
Performance Analysis

We classify storage-intensive applications into four classes:

- Unbatchable Reads, Unbatchable Writes
- Unbatchable Reads, Batchable Writes
- Batchable Reads, Unbatchable Writes
- Batchable Reads, Batchable Writes
Unbatchable Reads, Batchable Writes

• Can exploit write concurrency \((k_w)\) by batching writes

• Amortized cost per write following PIO is \(\frac{\alpha}{k_w}\)

• Example: DBMS bufferpool
Unbatchable Reads, Batchable Writes

Speedup increases with increasing concurrent I/Os

Speedup depends on asymmetry – gain is higher for a device with higher asymmetry
Batchable Reads, Unbatchable Writes

- Can exploit read concurrency ($k_r$) by batching read

- Amortized cost per read following PIO is $\frac{1}{k_r}$

- Example: Graph traversal
Batchable Reads, Unbatchable Writes

Speedup increases with increasing concurrent I/Os

Speedup depends on asymmetry – gain is higher for a device with lower asymmetry
Batchable Reads, Unbatchable Writes

• Can exploit both read and write concurrency ($k_r$, $k_w$)

• Amortized cost per read following PIO is $\frac{1}{k_r}$

• Amortized cost per write following PIO is $\frac{\alpha}{k_w}$

• Example: LSM compaction
Batchable Reads, Batchable Writes

Speedup increases with increasing concurrent I/Os, and depends on asymmetry

Impact of utilizing write concurrency is higher than utilizing read concurrency
Importance of using Proper $k$

A sample application with unbatchable reads and batchable writes

We use PCIe SSD ($k_w = 8$) to run this concurrency-aware application

Optimal speedup at the device concurrency.
Guidelines for Algorithm Design

Know Thy Device

Exploit Device Concurrency

Use Concurrency with Care

It is suboptimal to treat a read and a write equally for a device with asymmetry

Asymmetry Controls Performance
Conclusion

Modern Storage Devices

Read/Write Asymmetry

Concurrency

Need for a new parametric I/O model

PIO (M, k, \( \alpha \))

Benefits of PIO (M, k, \( \alpha \))

- algorithms tailored to new devices
- Can capture any new device

Prerequisite: quantify \( k \) and \( \alpha \)
Make *asymmetry and concurrency* part of *algorithm design*

... not simply an engineering optimization

Build algorithms/data structures for storage devices with *asymmetry $\alpha$* and *concurrency $k$*
Thank You!

disc.bu.edu/pio
Backup Slides
Expected Parallelism

I/O Requests

SSD

SSD Controller

ch-1

ch-2

ch-3

ch-n

How many of the channels will be occupied for uniform distribution?
Expected Parallelism

\[ E_n = \text{expected number of empty channels after } n \text{ I/Os are uniformly distributed} \]

We use \( E_{n-1} \) and consider where the \( n^{th} \) I/O will be routed

\[
E_n = \frac{E_{n-1}}{n} (E_{n-1} - 1) + \left(1 - \frac{E_{n-1}}{n}\right) E_{n-1} = \frac{n - 1}{n} E_{n-1}
\]

Since \( E_0 = n; \quad E_n = \left(\frac{n-1}{n}\right)^n n \)

Fraction of empty channels \(= \frac{E_n}{n} = \left(1 - \frac{1}{n}\right)^n \approx \frac{1}{e} \)

So, on average \( \left(1 - \frac{1}{e}\right) = 63.2\% \) channels will be accessed in parallel
Unbatchable Reads, Batchable Writes

1% reads

10% reads

50% reads

90% reads

99% reads

\[
\alpha = 1 \quad \alpha = 4 \quad \alpha = 8
\]

\[
\alpha = 2
\]

\[
\text{Speedup}
\]

\[
\text{Conc. Write I/Os}
\]