Effectively Prefetching Remote Memory with Leap

Hasan Al Maruf and Mosharaf Chowdhury
Memory-Intensive Applications

- powergraph
- NumPy
- Apache Spark
- VoltDB
- Redis
- Memcached
Perform Great!

**TPC-C on VoltDB**

In-Memory Working Set

38.61 TPS (Thousands)
Perform Great Until Memory Runs Out

TPC-C on VoltDB
Perform Great Until Memory Runs Out

TPC-C on VoltDB

TPS (Thousands)

<table>
<thead>
<tr>
<th>In-Memory Working Set</th>
<th>100%</th>
<th>75%</th>
<th>50%</th>
</tr>
</thead>
<tbody>
<tr>
<td>100%</td>
<td>38.61</td>
<td>6.61</td>
<td>1.01</td>
</tr>
</tbody>
</table>

PageRank on PowerGraph

Completion Time (s)

<table>
<thead>
<tr>
<th>In-Memory Working Set</th>
<th>100%</th>
<th>75%</th>
<th>50%</th>
</tr>
</thead>
<tbody>
<tr>
<td>100%</td>
<td>116.19</td>
<td>124.96</td>
<td>424.47</td>
</tr>
</tbody>
</table>
50% Less Memory Causes Slowdown of ...

**TPC-C on VoltDB**

![Chart showing TPS (Thousands) for TPC-C on VoltDB]

**PageRank on PowerGraph**

![Chart showing Completion Time (s) for PageRank on PowerGraph]
Between a Rock and a Hard Place

**Underallocation**
Leads to severe performance loss

**Overallocation**
Leads to underutilization
30-40% in Google, Alibaba, and Facebook
Remote Memory Access

User-space Applications

Memory Disaggregation Frameworks

- Infiniswap (NSDI’17)
  - Remote memory paging
- Remote Regions (ATC’18)
  - Remote file abstraction
- LegoOS (OSDI’18)
  - Disaggregated OS

Remote Memory

4KB page access latency
local vs. remote

100 ns vs. 4 µs
Remote Memory Access

User-space Applications

Memory Disaggregation Frameworks

- Infiniswap (NSDI’17)
  Remote memory paging

- Remote Regions (ATC’18)
  Remote file abstraction

- LegoOS (OSDI’18)
  Disaggregated OS

Remote Memory

4KB page access latency
local vs. remote

100 ns vs. 4 µs

Latency requirement for preferable performance[1]

3 µs

Existing frameworks can’t achieve!

Remote Memory Access

User-space Applications

Memory Disaggregation Frameworks

- Infiniswap (NSDI’17)
  - Remote memory paging

- Remote Regions (ATC’18)
  - Remote file abstraction

- LegoOS (OSDI’18)
  - Disaggregated OS

Remote Memory

4KB page access latency
local vs. remote

100 ns vs. 4 µs

Latency requirement for preferable performance[1]

3 µs

Existing frameworks can’t achieve!


Data path overhead

Variation in network latency

Existing frameworks can’t achieve!
Life of a Page

I/O Scheduler
Request queue processing: Insertion, Merging, Sorting, Staging and Dispatch

Device Mapping Layer

Generic Block Layer

I/O Scheduler
Request queue processing: Insertion, Merging, Sorting, Staging and Dispatch

Block Device Driver
Remote Memory

Memory Management Unit (MMU)

Page Cache

User Space
Kernel Space

Process 1
Process 2

Page Fault

0.27 us
2.1 us
10.04 us
21.88 us
2.1 us
10.04 us

Cache Hit
Cache Miss

RDMA: 4.3 us

1.04 us
2.1 us
Where Does the Time Go?

Flowchart:
- Page Request
  - In Page Cache?
    - Yes: 0.12 µs
    - No: Read Request?
      - Yes: Update Page Table & End I/O

Times:
- 0.12 µs
- 0.15 µs

Fast Path
Where Does the Time Go?

**Page Request**

- **In Page Cache?**
  - Yes: Update Page Table & End I/O
  - No: Allocate Cache for Page

**Allocate Cache for Page**

- **Prepare for I/O**
  - No: Queue and Batch Requests
  - Yes: Execute I/O

**Queue and Batch Requests**

- **Execute I/O**

**RDMA:** 4.3 µs

**Fast Path**

- 0.12 µs

**Slow Path**

- 2.1 µs
- 10.04 µs
- 21.88 µs

**Fast Path**

- Yes: Update Page Table & End I/O

**Prepare for I/O**

- No: Queue and Batch Requests

**Queue and Batch Requests**

- **Execute I/O**

**RDMA:** 4.3 µs

**Fast Path**

- Yes: Update Page Table & End I/O

**Update Page Table & End I/O**

- Yes: Update Page Table & End I/O

**Update Page Table & End I/O**

- Yes: Update Page Table & End I/O
Design Goal

1. Increase cache hit
   • faster path serves more page faults

2. Reduce the latency of the slow path
   • remove unnecessary block-layer operations for RDMA
Leap

Identifies memory access patterns to prefetch pages in a
• fast,
• cache-efficient, and
• resilient manner

without modifying any
• applications, or
• hardware
Life of a Page

- **Cache Hit**: 0.27 us
- **Cache Miss**: 2.1 us
- **Page Fault**: 10.04 us
- **RDMA**: 4.3 us

**Device Mapping Layer**

**Generic Block Layer**
- I/O Scheduler
  - Request queue processing: Insertion, Merging, Sorting, Staging and Dispatch

**Block Device Driver**

**Memory Management Unit (MMU)**
- MMU Page Cache

**Request Queue Processing**:
- Insertion, Merging, Sorting, Staging and Dispatch

**Remote Memory**
Life of a Page w/ Leap

- Process 1
- Process 2
- …
- Process N

Memory Management Unit (MMU)

Page Cache

Remote Memory

Cache Hit:
- 0.27 us

Cache Miss:
- 2.1 us

RDMA: 4.3 us
Life of a Page w/ Leap

- Cache Hit: 0.27 us
- Cache Miss: 0.34 us
- RDMA: 4.3 us

Process 1, Process 2, ... to Process N

Memory Management Unit (MMU)

MMU Page Cache

Leap

- Process Specific Page Access Tracker
- Prefetcher Trend Detection
- Prefetch Candidate Generation
- Eager Cache Eviction

Remote Memory
Prefetching in Linux

Reads ahead pages sequentially

Based only on the last page access

\[
\begin{cases}
\text{too aggressive on seq: cache pollution} \\
\text{too conservative off seq: brings nothing}
\end{cases}
\]

Does not distinguish between processes

Cannot detect thread-level access irregularities
## Prefetching Techniques

<table>
<thead>
<tr>
<th>Approach</th>
<th>Low Computational Complexity</th>
<th>Low Memory Overhead</th>
<th>Unmodified Application</th>
<th>HW/SW Independence</th>
<th>Temporal Locality</th>
<th>Spatial Locality</th>
<th>Low Cache Pollution</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

- **Low Computational Complexity**: Yes
- **Low Memory Overhead**: Yes
- **Unmodified Application**: Yes
- **HW/SW Independence**: Yes
- **Temporal Locality**: Yes
- **Spatial Locality**: Yes
- **Low Cache Pollution**: Yes
## Prefetching Techniques

<table>
<thead>
<tr>
<th>Approach</th>
<th>Low Computational Complexity</th>
<th>Low Memory Overhead</th>
<th>Unmodified Application</th>
<th>HW/SW Independence</th>
<th>Temporal Locality</th>
<th>Spatial Locality</th>
<th>Low Cache Pollution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Next N-Line</td>
<td><strong>Yes</strong></td>
<td><strong>Yes</strong></td>
<td><strong>Yes</strong></td>
<td><strong>Yes</strong></td>
<td><strong>No</strong></td>
<td><strong>Yes</strong></td>
<td><strong>No</strong></td>
</tr>
<tr>
<td>Stride</td>
<td><strong>Yes</strong></td>
<td><strong>Yes</strong></td>
<td><strong>Yes</strong></td>
<td><strong>Yes</strong></td>
<td><strong>No</strong></td>
<td><strong>Yes</strong></td>
<td><strong>No</strong></td>
</tr>
</tbody>
</table>
## Prefetching Techniques

<table>
<thead>
<tr>
<th>Approach</th>
<th>Low Computational Complexity</th>
<th>Low Memory Overhead</th>
<th>Unmodified Application</th>
<th>HW/SW Independence</th>
<th>Temporal Locality</th>
<th>Spatial Locality</th>
<th>Low Cache Pollution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Next N-Line</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Stride</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Instruction Prefetch</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>
# Prefetching Techniques

<table>
<thead>
<tr>
<th>Approach</th>
<th>Low Computational Complexity</th>
<th>Low Memory Overhead</th>
<th>Unmodified Application</th>
<th>HW/SW Independence</th>
<th>Temporal Locality</th>
<th>Spatial Locality</th>
<th>Low Cache Pollution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Next N-Line</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Stride</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Instruction Prefetch</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Linux Read-Ahead</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>
## Prefetching Techniques

<table>
<thead>
<tr>
<th>Approach</th>
<th>Low Computational Complexity</th>
<th>Low Memory Overhead</th>
<th>Unmodified Application</th>
<th>HW/SW Independence</th>
<th>Temporal Locality</th>
<th>Spatial Locality</th>
<th>Low Cache Pollution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Next N-Line</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Stride</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Instruction Prefetch</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Linux Read-Ahead</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Leap</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

25
Leap Prefetcher

Linear-time and constant memory space

Two main components:
- Trend detection
- Prefetch window size detection

Get Prefetch Window Size

Window Size = 0?

Yes

Read only the requested page

No

Trend Found?

No

Prefetch with Previous Trend

Yes

Prefetch with Current Trend
Trend Detection

Flexible to short term irregularity

Identifies the majority element in access history

Regular trends can be found within recent accesses

Start with a smaller window of Access History

Run Boyer-Moore on the window

Majority found?

Yes

Return Majority $\Delta_{maj}$

No

Doubles the window size

Max. window size?

Yes

No trend found

No
Trend Detection Example

(a) at time $t_3$
- Trend of -3
- $0x48 \rightarrow 0x45 \rightarrow 0x42 \rightarrow 0x3F$
- $+72 \rightarrow -3 \rightarrow -3 \rightarrow -3$

(b) at time $t_7$
- Trend of -3 disappears, no major new trend
- $0x48 \rightarrow 0x45 \rightarrow 0x42 \rightarrow 0x3F$
- $+72 \rightarrow -3 \rightarrow -3 \rightarrow -3$
- $t_0 \rightarrow t_1 \rightarrow t_2 \rightarrow t_3$
- Trend of +2 detected
- $0x08 \rightarrow 0x45 \rightarrow 0x42 \rightarrow 0x3F$
- $t_8 \rightarrow t_1 \rightarrow t_2 \rightarrow t_3$
- $0x3C \rightarrow 0x02 \rightarrow 0x04 \rightarrow 0x06$
- $t_4 \rightarrow t_5 \rightarrow t_6 \rightarrow t_7$
- $-58 \rightarrow +2 \rightarrow +2$

(c) at time $t_8$
- Trend of +2 detected among irregularities
- $0x08 \rightarrow 0x0A \rightarrow 0x0C \rightarrow 0x10$
- $t_8 \rightarrow t_9 \rightarrow t_{10} \rightarrow t_{11}$
- $0x39 \rightarrow 0x12 \rightarrow 0x14 \rightarrow 0x16$
- $t_{12} \rightarrow t_{13} \rightarrow t_{14} \rightarrow t_{15}$
- $+2 \rightarrow +2 \rightarrow +4 \rightarrow +2$

(d) at time $t_{15}$
- Trend of +2 detected
- $0x08 \rightarrow 0x0A \rightarrow 0x0C \rightarrow 0x10$
- $t_8 \rightarrow t_9 \rightarrow t_{10} \rightarrow t_{11}$
- $0x39 \rightarrow 0x12 \rightarrow 0x14 \rightarrow 0x16$
- $t_{12} \rightarrow t_{13} \rightarrow t_{14} \rightarrow t_{15}$
- $+2 \rightarrow +2 \rightarrow +4 \rightarrow +2$
Prefetch Window Size Detection

Cache hit indicates prefetch utilization

High cache hit: *increase* prefetch window *aggressively*

*no trend: decrease prefetch window gradually*

Gradual slow down helps during sudden changes
Evaluation

Deploy and evaluate over 56 Gbps InfiniBand network

Memory Disaggregation Frameworks

Disaggregated VMM: Infiniswap
Disaggregated VFS: Remote Regions
Lowers Remote Page Access Latency by…

### Sequential Access

- **CDF**: Cumulative Distribution Function
- **Latency (us)**: microseconds
- **Infiniswap**
- **Infiniswap+Leap**

### Stride Access

- **CDF**: Cumulative Distribution Function
- **Latency (us)**: microseconds

**4X**

**104X**
Efficient Pattern Detection

- Detects 29.70% more sequential accesses
- Detects most of the irregularity
Efficient Pattern Detection

Detects 29.70% more sequential accesses

Detects most of the irregularity

During irregularities, doing nothing helps the most
Perform Great Even After Memory Runs Out

**Disk**

TPC-C on VoltDB

<table>
<thead>
<tr>
<th>In-Memory Working Set</th>
<th>TPS (Thousands)</th>
</tr>
</thead>
<tbody>
<tr>
<td>100%</td>
<td>38.61</td>
</tr>
<tr>
<td>75%</td>
<td>6.61</td>
</tr>
<tr>
<td>50%</td>
<td>1.01</td>
</tr>
</tbody>
</table>

**Infiniswap**

<table>
<thead>
<tr>
<th>In-Memory Working Set</th>
<th>TPS (Thousands)</th>
</tr>
</thead>
<tbody>
<tr>
<td>100%</td>
<td>37.00</td>
</tr>
<tr>
<td>75%</td>
<td>27.74</td>
</tr>
<tr>
<td>50%</td>
<td>19.33</td>
</tr>
</tbody>
</table>

**Infiniswap + Leap**

<table>
<thead>
<tr>
<th>In-Memory Working Set</th>
<th>TPS (Thousands)</th>
</tr>
</thead>
<tbody>
<tr>
<td>100%</td>
<td>37</td>
</tr>
<tr>
<td>75%</td>
<td>36.3</td>
</tr>
<tr>
<td>50%</td>
<td>35.6</td>
</tr>
</tbody>
</table>

TPC-C on VoltDB

38X

2X

≈1X
Perform Great Even After Memory Runs Out

**TPC-C on VoltDB**

**Disk**

- 100% In-Memory Working Set: 38.61 TPS
- 75% In-Memory Working Set: 6.62 TPS
- 50% In-Memory Working Set: 1.01 TPS
- 25% In-Memory Working Set: Fails

**Infiniswap**

- 100% In-Memory Working Set: 37.00 TPS
- 75% In-Memory Working Set: 27.74 TPS
- 50% In-Memory Working Set: 19.33 TPS
- 25% In-Memory Working Set: 1.5 TPS

**Infiniswap + Leap**

- 100% In-Memory Working Set: 37 TPS
- 75% In-Memory Working Set: 36.3 TPS
- 50% In-Memory Working Set: 35.6 TPS
- 25% In-Memory Working Set: 15.6 TPS

**TPC-C on VoltDB**

- **Fails 24X**
- **Fails 2.4X**
Benefit Breakdown of Leap’s Components

Data path optimizations: single-µs latency till 95\textsuperscript{th} percentile

Prefetcher: sub-µs latency till 85\textsuperscript{th} percentile

Eager cache eviction: improves the 99\textsuperscript{th} percentile latency by 22%
Future Work

1. Thread-specific prefetching for multiple concurrent streams
   • memory is managed at the process level
   • this requires significant changes in virtual memory subsystem

2. Optimized remote I/O interface
   • load balancing,
   • fault-tolerance,
   • data locality, and
   • application-specific isolation in remote memory
Leap

Lightweight and efficient data path for remote memory

source code available at https://github.com/SymbioticLab/leap

Online prefetcher with a leaner data path and eager cache eviction policy to improve
• cache hit,
• remote I/O latency, and
• application-level performance

without modifying any
• application, or
• hardware
Thank You!

source code available at https://github.com/SymbioticLab/leap