Patronus: High-Performance and Protective Remote Memory

Bin Yan, Youyou Lu, Qing Wang, Minhui Xie, Jiwu Shu

Tsinghua University
Remote Memory

Remote memory architecture

- Physically separate CPU and memory into network-attached components

many CPU cores, small local DRAM

Compute Nodes (CNs)

large DRAM, several wimpy cores

Memory Nodes (MNs)
Remote Memory

Remote memory architecture

- Physically separate CPU and memory into network-attached components

many CPU cores, small local DRAM
Compute Nodes (CNs)

large DRAM, several wimpy cores
Memory Nodes (MN)s

Low latency & high bandwidth
Bypass remote CPUs

RDMA network
Remote Memory

Remote memory architecture

- Physically separate CPU and memory into network-attached components.

- Compute Nodes (CNs): many CPU cores, small local DRAM.
- Memory Nodes (MNs): large DRAM, several wimpy cores.

RDMA-enabled remote memory is widely deployed to improve memory utilization. Low latency & high bandwidth Bypass remote CPUs.
Remote Memory

Many efforts to make remote memory **practical** on multiple fronts

- Efficient remote indexes
  - [ATC’21, SIGMOD’22, HotOS’19]

- Easy-to-use programming models
  - [OSDI’20, ATC’18, SoCC’17]

- Popular applications and more
  - [FAST22, OSDI’18, ATC’15]

However, **protection** for remote memory is not explored
Necessity: Protection for Remote Memory

Unprotected RM fails to avoid application anomalies

Example 1 buggy/malicious clients (access illegal address)

(client)

(protected)

(write(0x00-0xff))

(read(0x00-0xff))

(out-of-bound)

(unregulated)

(corrupt data!)

(privacy breaches!)
Necessity: Protection for Remote Memory

Unprotected RM fails to avoid application anomalies

Example I **buggy/malicious clients**
(access illegal address)

- **client**
- (out-of-bound)
  - write(0x00-0xff)
  - corrupt data!
- (unregulated)
  - read(0x00-0xff)
  - privacy breaches!

Example II **memory management race**
(access at illegal time)

- free(0xa0)
- (concurrent)
- get garbage data!
- read(0xa0)
Necessity: Protection for Remote Memory

Unprotected RM fails to avoid application anomalies

Example I buggy/malicious clients (access illegal address)

(client)

(out-of-bound)
write(0x00-0xff)
corrupt data!

(protected)
read(0x00-0xff)

Example II memory management race (access at illegal time)

(client)

(unregulated)
free(0xa0)
get garbage data!

(protected)
read(0xa0)

RM protection is necessary especially for workloads with shared access patterns
Difficulty: Protection + Performance (I)

It is difficult to achieve **high-performance protection** on the common path

- **Reason 1:** CPUs are weak on memory nodes
- **Reason 2:** Existing protection mechanisms are expensive
Difficulty: Protection + Performance (I)

It is difficult to achieve high-performance protection on the common path

Reason 1: CPUs are weak on memory nodes
Reason 2: Existing protection mechanisms are expensive
Difficulty: Protection + Performance (I)

It is difficult to achieve high-performance protection on the common path.

Reason 1: CPUs are weak on memory nodes.
Reason 2: Existing protection mechanisms are expensive.

- **Client Memory Region (MR)**
  - Queue Pair (QP)
  - RDMA op

- **Operation Latency**
  - RDMA op QP-base: 2 us
  - RDMA op MR-base: 100 us

- **Remote Memory**

- **70Mops/s for ConnectX-5 RNIC**

- **100 us**
  - X50

- **1ms (for 256 MB)**
  - X10
Difficulty: Protection + Performance (I)

It is difficult to achieve **high-performance protection** on the common path

**Reason 1**: CPUs are weak on memory nodes

**Reason 2**: Existing protection mechanisms are expensive

Existing mechanisms are **50X-500X slower** than RDMA data path.

*Example* Existing applications (FaRM [NSDI’14], Octopus [ATC’17]) have to use 2GB coarse-grained MR, leaving RM no protection.
Difficulty: Protection + Robustness (II)

It is difficult to remain robust on the failure path:

Exception I  Client failures

Crashed client → Another client

owning (at 0xa0) exclusive permission…

Acquire Permission (also at 0xa0)

pending infinitely

Client failures impact system progress
Difficulty: Protection + Robustness (II)

It is difficult to remain robust on the failure path:

Exception I  Client failures

Crashed client
owning exclusive permission…

Another client
Acquire Permission (also at 0xa0)
pending infinitely

Exception II  QP failures

RDMA op
no ack… (more than 1ms)

Crashed QP

Client failures impact system progress

QP failures interrupt application execution
Goal - Protective System

- RM systems are performance-critical
- Client failures impact system progress
- QP failures interrupt application execution

- Fast protection management on par with data path
- React fast to client failures
- Retain performance under QP failures
Goal - Protective System

- RM systems are performance-critical
  - Fast protection management on par with data path

- Client failures impact system progress
  - React fast to client failures

- QP failures interrupt application execution
  - Retain performance under QP failures

Patronus: a protective RM system that is high-performance and robust under all situations
Outline

- Background & Motivation
- Patronus – High-Performance Protective Remote Memory
- Results
- Conclusion
Patronus Overview

CNs

Permission Management

<table>
<thead>
<tr>
<th>Address</th>
<th>Perm</th>
</tr>
</thead>
<tbody>
<tr>
<td>$&lt;0xa0,8&gt;$</td>
<td>R+W</td>
</tr>
<tr>
<td>$&lt;0xd0,64&gt;$</td>
<td>expired</td>
</tr>
</tbody>
</table>

MNs

Memory Management

remote memory

Permission requests (RPC)

data access (one-sided)

RDMA network
Patronus Overview

CNs

Permission starts

allocate(64) / acquire(0xb0,64)

Permission Management

<table>
<thead>
<tr>
<th>Address</th>
<th>Perm</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;0xa0,8</td>
<td>R+W</td>
</tr>
<tr>
<td>&lt;0xd0,64</td>
<td>expired</td>
</tr>
</tbody>
</table>

Memory Management

remote memory

requests
Patronus Overview

Permission starts:

allocate(64) / acquire(0xb0,64)

Data access:

read/write(Perm,0xa0,8)

Permission Management:

<table>
<thead>
<tr>
<th>Address</th>
<th>Perm</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;0xa0,8&gt;</td>
<td>R+W</td>
</tr>
<tr>
<td>&lt;0xd0,64&gt;</td>
<td>expired</td>
</tr>
</tbody>
</table>

Memory Management:

remote memory

Address: <0xa0,8>
Perm: R+W
Patronus Overview

Permission Management

<table>
<thead>
<tr>
<th>Address</th>
<th>Perm</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;0xa0, 8</td>
<td>R+W</td>
</tr>
<tr>
<td>&lt;0xd0, 64</td>
<td>expired</td>
</tr>
</tbody>
</table>

Memory Management

remote memory

--

1. Permission starts

allocate(64) / acquire(0xb0, 64)

2. Data access with check

read/write(Perm, 0xa0, 8)

ACK

NAK (out of bound)

read/write(Perm, 0xff, 8)

ACK

Permission requests (RPC)

data access (one-sided)

RDMA network

CNs

MNs
Patronus Overview

Permission starts
- allocate(64) / acquire(0xb0, 64)
- read/write(Perm, 0xa0, 8)
- ACK
- read/write(Perm, 0xff, 8)
- NAK (out of bound)
- revoke(Perm)

Data access with check
- ACK
- ACK

Permission ends
- ACK

Permission Management
- Address: <0xa0, 8>
  - Perm: R+W
- Address: <0xd0, 64>
  - Perm: expired

Memory Management
- remote memory
- blocks

Requests
- data access (one-sided)
- permission requests (RPC)

CNs
- RDMA network

MNs
Opportunity – Memory Window (MW)

**Memory Window:** an advanced & light-weight protection mechanism

- Patronus leverages MWs for fast permission management.
Opportunity – Memory Window (MW)

**Memory Window**: an advanced & light-weight protection mechanism

- Patronus leverages MWs for fast permission management.

![Operation Latency Diagram]

- RDMA op: 2 us
- MW op: 1.1 us
- QP op: 100 us
- MR op: 1 ms

**Same-level latency**
Opportunity – Memory Window (MW)

**Memory Window**: an advanced & light-weight protection mechanism

- Patronus leverages MWs for fast permission management.

---

**Action** | **MW** | **Effect**
--- | --- | ---
Permission starts | bind | $\rightarrow \checkmark$
Permission ends | unbind | $\times$

$(\text{rebind} = \text{unbind} + \text{bind})$

---

![Diagram showing operation latencies for RDMA, MW, QP, and MR operations with corresponding times: 2 us for RDMA, 1.1 us for MW, and 100 us for QP. MR operation spans 1 ms.](image)
Opportunity – Memory Window (MW)

**Memory Window**: an advanced & light-weight protection mechanism

- Patronus leverages MWs for fast permission management.

<table>
<thead>
<tr>
<th>Operation</th>
<th>MW Latency</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDMA op</td>
<td>2 us</td>
<td></td>
</tr>
<tr>
<td>MW op</td>
<td>1.1 us</td>
<td></td>
</tr>
<tr>
<td>QP op</td>
<td></td>
<td>bind</td>
</tr>
<tr>
<td>MR op</td>
<td></td>
<td>unbind</td>
</tr>
</tbody>
</table>

![Diagram](diagram.png)

*NOT ENOUGH*

(rebind = unbind + bind)
Opportunity – Memory Window (MW)

**Memory Window**: an advanced & light-weight protection mechanism

- Patronus leverages MWs for fast permission management.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDMA op</td>
<td>2 us</td>
</tr>
<tr>
<td>MW op</td>
<td>1.1 us</td>
</tr>
<tr>
<td>QP op</td>
<td></td>
</tr>
<tr>
<td>MR op</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operation</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>bind</td>
<td>-&gt; 0</td>
</tr>
<tr>
<td>unbind</td>
<td>×</td>
</tr>
</tbody>
</table>

(rebind = unbind + bind)

**NOT ENOUGH**
Opportunity – Memory Window (MW)

**Memory Window**: an advanced & light-weight protection mechanism

- Patronus leverages MWs for fast permission management.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDMA op</td>
<td>2 us</td>
</tr>
<tr>
<td>MW op</td>
<td>1.1 us</td>
</tr>
<tr>
<td>QP op</td>
<td></td>
</tr>
<tr>
<td>MR op</td>
<td></td>
</tr>
</tbody>
</table>

**Operation Latency**

![Diagram]

### NOT ENOUGH

<table>
<thead>
<tr>
<th>MW Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>bind</td>
</tr>
<tr>
<td>unbind</td>
</tr>
</tbody>
</table>

(\text{rebind} = \text{unbind} + \text{bind})

**Weak CPUs?**
Opportunity – Memory Window (MW)

**Memory Window**: an advanced & light-weight protection mechanism

- Patronus leverages MWs for fast permission management.

---

<table>
<thead>
<tr>
<th>Operation</th>
<th>Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDMA op</td>
<td>2 us</td>
</tr>
<tr>
<td>MW op</td>
<td>1.1 us</td>
</tr>
<tr>
<td>QP op</td>
<td></td>
</tr>
<tr>
<td>MR op</td>
<td></td>
</tr>
</tbody>
</table>

---

**MW**

<table>
<thead>
<tr>
<th>Operation</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>bind</td>
<td>-&gt; ✅</td>
</tr>
<tr>
<td>unbind</td>
<td>✗</td>
</tr>
</tbody>
</table>

(rebind = unbind + bind)

---

**Client failure?**

Weak CPUs?
Opportunity – Memory Window (MW)

**Memory Window**: an advanced & light-weight protection mechanism

- Patronus leverages MWs for fast permission management.

---

**MW** | **Effect**
--- | ---
bind | →
unbind | 

(\textit{rebind} = unbind + bind)

---

NOT ENOUGH

---

**Operation Latency**

- RDMA op: 2 us
- MW op: 1.1 us
- QP op
- MR op

---

**Clients**

- **QP failure?**

---

**Memory Nodes**

- **Weak CPUs?**

---

**Permission starts**

bind

**Permission ends**

unbind

(\textit{rebind} = unbind + bind)
Technique (1) – MW Operation Reduction

- **MW handover**
  - Observation: binding and unbinding ops co-exist
  - Bind op + unbind op => rebind op
  - Hand over MWs between requests
  - Result: reduce *half* of MW ops

$\text{bind} + \text{unbind} = 2 \text{ op}$
$\text{rebind} = 1 \text{ op}$

Same semantics can be achieved with fewer operations
Technique (1) – MW Operation Reduction

- **MW handover**
  - **Observation**: binding and unbinding ops co-exist
  - **Bind op + unbind op** => **rebind op**
  - **Hand over MWs between requests**
  - **Result**: reduce half of MW ops

- **Exploit locality**
  - **Observation**: space and time locality
  - Multiple **bind ops** (w/ locality) => one **bind op**
  - **Result**: reduce binding MW ops

*Same semantics can be achieved with fewer operations*
Technique (2) – Lease for Client Failures

Use *leases* to handle client failures
Use *leases* to handle *client failures*

- Equip MWs with automatic reclamation
- Memory nodes poll for expiration periodically (1)
Technique (2) – Lease for Client Failures

Use leases to handle client failures

- Equip MWs with automatic reclamation
- Memory nodes poll for expiration periodically (1)

Offloads lease extension overhead to compute nodes
Technique (2) – Lease for Client Failures

Use *leases* to handle *client failures*

- Equip MWs with automatic reclamation
- Memory nodes poll for expiration periodically (1)

---

**Offloads lease extension overhead to compute nodes**

---

**Client-collaborated lease extension**

- **Enabler:** MWs are byte-granularity to expose variables
- Expose the *lease_time* variable (2) to clients
- Clients extend permission via RDMA_CAS (2)
- **Result:** Extension only costs one in-bound RDMA op
Technique (3) – Over-Provisioning for QP failures

**Over-provision QPs** to hide interruption from QP failures

- On QP failures: promote a healthy QP as substitution
Technique (3) – Over-Provisioning for QP failures

Over-provision QPs to hide interruption from QP failures

- On QP failures: promote a healthy QP as substitution

![Diagram showing over-provisioning of Compute Nodes and Memory Nodes with arrows indicating promotion and over-provisioned state.](image-url)
Technique (3) – Over-Provisioning for QP failures

**Over-provision QPs** to hide interruption from **QP failures**

- On QP failures: promote a healthy QP as substitution

![Diagram showing over-provisioning for QP failures](image)
Technique (3) – Over-Provisioning for QP failures

**Over-provision QPs to hide interruption from QP failures**

- On QP failures: promote a healthy QP as substitution
- *Enabler:* MWs can remain valid across QPs => previous permission still works

![Diagram showing over-provisioned nodes for Compute and Memory Nodes, with arrows indicating promotion and still valid state.]
Technique (3) – Over-Provisioning for QP failures

**Over-provision QPs to hide interruption from QP failures**

- On QP failures: promote a healthy QP as substitution
- **Enabler:** MWs can remain valid across QPs => previous permission still works
- **Result:** low downtime under QP failures

Over-provisioning improves robustness
Outline

- Background & Motivation
- Patronus – High-Performance Protective Remote Memory
- Results
- Conclusion
Experimental Setup

Hardware Platform

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>Xeon Gold 6240M CPUs, 32 cores per node</td>
</tr>
<tr>
<td>DRAM</td>
<td>186GB DDR4</td>
</tr>
<tr>
<td>NIC</td>
<td>Mellanox MT27800 ConnectX-5 Family</td>
</tr>
</tbody>
</table>

Cluster

- 3 Compute node (no cache)
- 1 Memory node (<= 4 CPU cores)
Experimental Setup

Hardware Platform

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>Xeon Gold 6240M CPUs, 32 cores per node</td>
</tr>
<tr>
<td>DRAM</td>
<td>186GB DDR4</td>
</tr>
<tr>
<td>NIC</td>
<td>Mellanox MT27800 ConnectX-5 Family</td>
</tr>
</tbody>
</table>

Cluster

- 3 Compute node (no cache)
- 1 Memory node (<= 4 CPU cores)

Evaluated Cases

- Common path: 2 one-sided data structures and 2 function-as-a-service workloads.
- Exception path: client failures and QP failures.
Evaluation (1): Overall Performance

RACE Hashing

Zipfian 0.99
KV = 4KB

Concurrent Queue

Function as a Service
Evaluation (1): Overall Performance

RACE Hashing

Zipfian 0.99
KV = 4KB

Concurrent Queue

Function as a Service

Vanilla unprotected impl
Use Patronus for permission
MR for permission
Use RPC in data path

Evaluation (1): Overall Performance

Throughput (Mops)

Throughput (Mops)

Throughput (Mops)

Unprot
Patronus
RPC
MR
(a) RO
(b) RW
(c) WO

Vanilla unprotected impl
Use Patronus for permission
MR for permission
Use RPC in data path

Evaluation (1): Overall Performance

Throughput (Mops)

Throughput (Mops)

Throughput (Mops)

Unprot
Patronus
RPC
MR
(a) Image processing
(b) Data analysis

Evaluation (1): Overall Performance

Evaluation (1): Overall Performance
Evaluation (1): Overall Performance

RACE Hashing
Zipfian 0.99
KV = 4KB

Concurrent Queue

Function as a Service

Patronus performs up to X5.2 better than the competitors and has <= 28% overhead than vanilla implementation
Evaluation (2): Failure Handling

Handling client failures

- **Vanilla**: not handling client failures
- **Patronus**: resumes progress after 80 epochs.
Evaluation (2): Failure Handling

Handling client failures

- **Vanilla**: not handling client failures
- **Patronus**: resumes progress after 80 epochs.

Handling QP failures

- Trigger out-of-bound access manually

<table>
<thead>
<tr>
<th>Category</th>
<th>Vanilla</th>
<th>Patronus</th>
</tr>
</thead>
<tbody>
<tr>
<td>Promote QP</td>
<td>-</td>
<td>78 us</td>
</tr>
<tr>
<td>Notify QP Failure</td>
<td>8 us</td>
<td>-</td>
</tr>
<tr>
<td>Recover QP</td>
<td>1004 us</td>
<td>-</td>
</tr>
<tr>
<td>Summary</td>
<td>1012 us</td>
<td>78 us (8%)</td>
</tr>
</tbody>
</table>

Patronus is robust to handle client failures and QP failures quickly
Conclusion

- We propose **Patronus**, a high-performance protective remote memory system for RM protection.

- Three techniques for **performance & robustness**: MW operations reduction, client-collaborated lease, and QP over-provisioning.

- Patronus takes less than **28%** overhead and performs at most **x5.2** than the competitors in various real-world workloads.

- More analysis and evaluation results in the paper.
Thanks

Q&A

Patronus – High-Performance and Protective Remote Memory

Bin Yan, Youyou Lu, Qing Wang, Minhui Xie, Jiwu Shu
yanb20@mails.tsinghua.edu.cn