Reducing CPU and network overhead for small I/O requests in network storage protocols over raw Ethernet

Pilar González-Férez and Angelos Bilas

FORTH
Institute of Computer Science

31^{th} International Conference on Massive Storage Systems and Technology
MSST 2015
June 5^{th}, Santa Clara, California

Motivation

Small I/O requests limited by host I/O path overhead

- Small I/O requests are important for a large number of workloads
  - Most files are 4kB or smaller
  - Metadata requests are typically small and $\simeq 50\%$ of the I/Os
- Storage devices, specially HDDs, dominate all I/O overheads
  - Techniques for improving throughput focused on large requests
- NVM technologies exhibit performance similar to DRAM
  - Device overhead of small I/Os a few microseconds
  - Bottleneck shifts from device to host I/O stack
- Most systems today use some form of networked storage
  - Ethernet NICs have improved significantly latency as well

Our goal

Reduce host-level overhead for small I/O requests

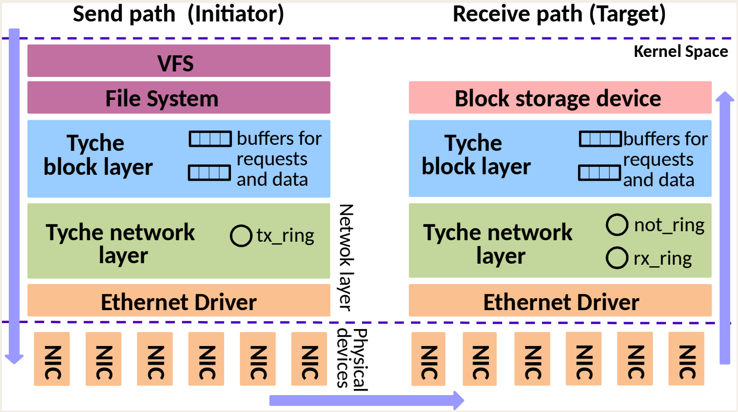
Tyche Semantics

Tyche is an in-house network storage protocol over raw Ethernet that achieves high throughput without any hardware support [MSST14]

- Connection-oriented protocol over raw Ethernet that can be deployed in existing infrastructures
- Create and present a local view of a remote storage device (NBD-like)
- Support any existing file system
- Transparent bundling of multiple NICs (tested up to 6)
- Reliable delivery
- Provide RDMA-like operations without any hardware support
- Copy reduction via page remapping in the I/O path
- NUMA affinity management
- Storage-specific packet processing
- Pre-allocation of memory

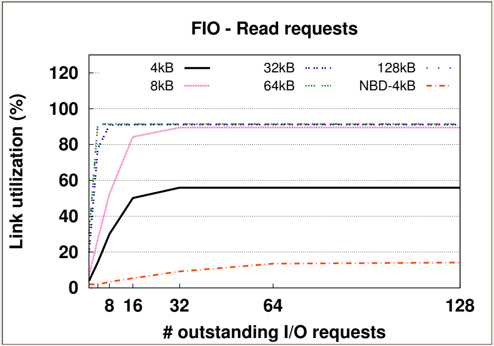
Tyche Design

Tyche achieves up to 6.7GB/s by using six 10Gbit/s NICS without hardware support [MSST14]



However . . .

Tyche still exhibits low throughput and low network link utilization for small I/O requests



Link utilization

Tyche provides up to 5x the link utilization of NBD, but

Tyche achieves only up to 56% for 4kB requests, while up to 90% for 8kB requests

Therefore, our goal is . . .

- We analyze the host CPU overheads in the networked I/O path
- We find that small I/O requests are limited by:
  - Context switch overhead
  - Network packet processing
- We reduce overheads and increase throughput for small I/O requests:
  - Low I/O concurrency ⇒ Context switches
  - High I/O concurrency ⇒ Dynamic batching of requests

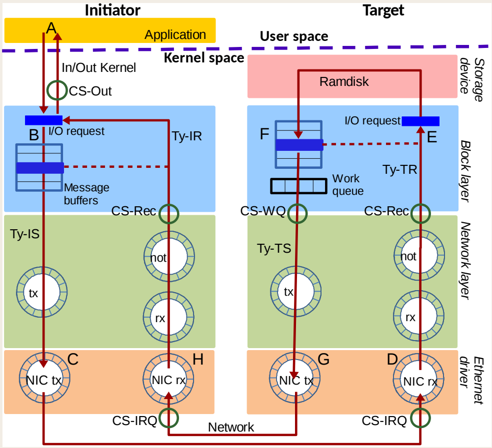
1. Motivation
2. Overhead Analysis
3. Reducing Context Switches
4. Adaptive Batching
5. Conclusions

FORTH
ICS

Experimental Testbed

- Two nodes 4-core Intel Xeon E5520 @2.7GHz
  - **Initiator**: 12GB DDR-III DRAM
  - **Target**: 48GB DDR-III DRAM, but 36GB used as ramdisk
- 1 Myri10ge card each node, connected back to back
- CentOS 6.3
- Tyche implemented in Linux kernel 2.6.32
- **Benchmark**: FIO
  - Direct I/O, 4kB requests
  - Queue depth: single thread with 1 outstanding request

End-to-end Overhead Analysis



Overheads measured

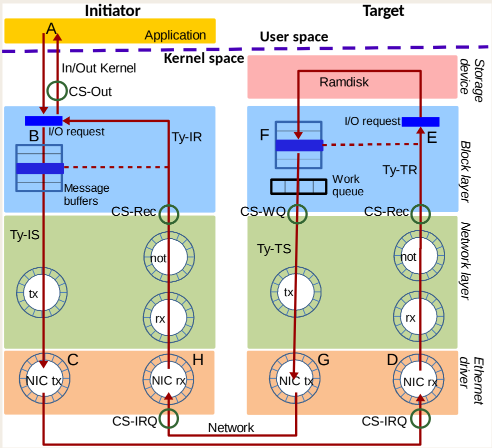
- Total
- Tyche Send Path: Ty-IS, Ty-TS
- Tyche Receive Path: Ty-IR, Ty-TR
- Context switches: CS-WQ, CS-Rec, CS-IRQ
- Ramdisk

Overheads computed

- In/Out Kernel
- Link+NIC

4kB Read Requests At Low Concurrency

Total: $73.69\mu s$ and **Throughput:** 52.50MB/s



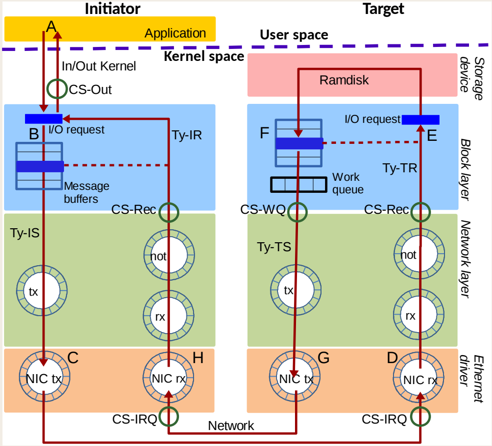
(queue depth=1)

Overhead

	μs	%
I/O kernel	13.19	17.9
Ty-IS	2.75	20.0
Ty-TR	3.00	
Ty-TS	4.00	
Ty-IR	5.00	
CS-WQ	4.00	27.3
CS-Rec	8.00	
CS-IRQ	8.15	
Ramdisk	1.00	1.4
Link+NIC	24.60	33.4

4kB Write Requests At Low Concurrency

Total: $73.80\mu s$ and **Throughput:** 52.50MB/s



(queue depth=1)

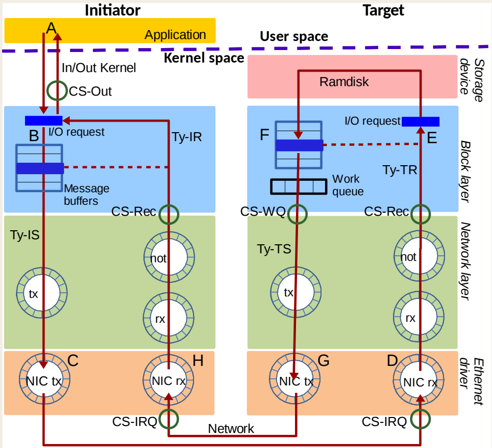
Overhead

	μs	%
I/O kernel	12.80	17.3
Ty-IS	4.75	
Ty-TR	5.00	20.3
Ty-TS	3.00	
Ty-IR	2.25	
CS-WQ	4.00	
CS-Rec	8.00	27.3
CS-IRQ	8.13	
Ramdisk	1.00	1.4
Link+NIC	24.87	33.7

FORTH
ICS

Overhead of Context Switches

Each context switch costs $\simeq 4\mu s$ and up to 27.5% of total overhead



Two threads just performing context switches:

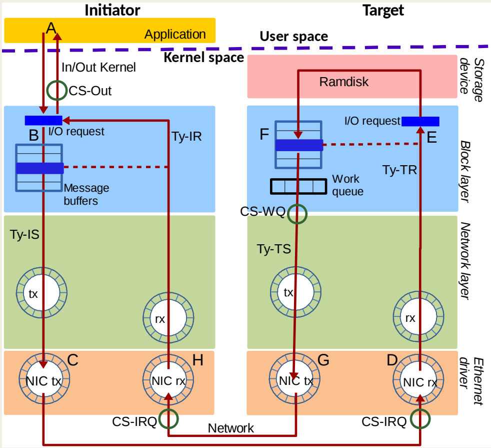
- Same NUMA node: $2.5\mu s$
- Different node: $5\mu s$

Overhead from saving, restoring processor registers, pipeline flush, caches ...

Jongmin *et.al.*, Transactions on Storage 11(2), 2015

Avoiding Context Switch at the Receive Path

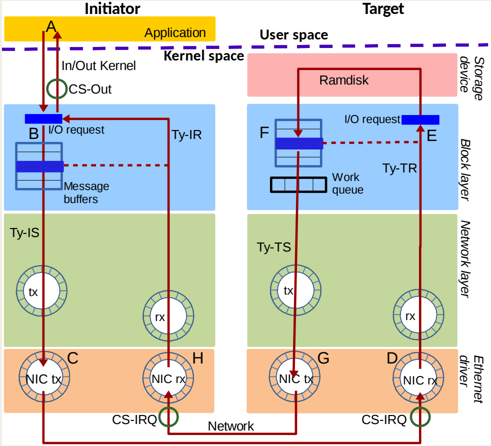
A single thread runs network layer tasks and block layer tasks



- Shared structures to communicate both layers are not needed
- Overhead of receive path is significantly reduced
- Total overhead is reduced by 18%
- Throughput increases by 22%

Avoiding Context Switch at the Target Send Path

No work queue to send the completion back at the target



- No sync with the work queue threads is needed
- Overhead of target send path is also reduced
- Total overhead is reduced by 22%
- Throughput increases by 45%

Summary

For 4kB requests, at low concurrency:

- Total I/O overhead per I/O request is reduced by 27%
- Throughput improves by 45%
- Context switch overhead is reduced by up to 60%
- Tyche processing is reduced by 56%/61% for reads/writes

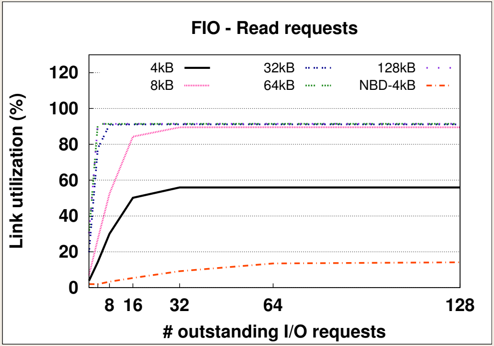
1. Motivation
2. Overhead Analysis
3. Reducing Context Switches
4. **Adaptive Batching**
5. Conclusions

FORTH
ICS

Adaptive Batching

Problem

Under high concurrency, Tyche still cannot achieve high throughput for small I/O requests



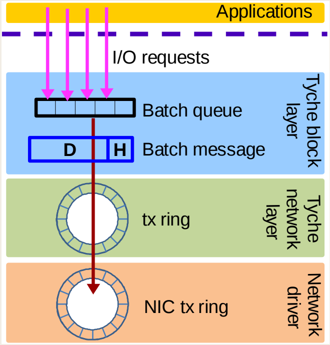
Batch several requests into a single request message

Batching reduces

- Number of messages
- Message processing
- Number of context switches

Batching Request Messages

A batch request message includes I/O requests or I/O completions



★ Initiator has a batch queue and thread

Initiator:

- Applications enqueue I/O requests into the batch queue
- A thread dequeues these requests and inserts into a batch message

Target:

- Issues a regular I/O requests per request in the batch message
- Batches completions as well

Batching Data Messages

- Data of a 4kB request is sent by using a single data packet in a Jumbo Ethernet frame of 4kB
- We batch data messages:
  - Data for 4kB request messages is sent together in a single data message
  - We use data packets of 8kB
  - Jumbo Ethernet frame of 8kB is used

Dynamic Batching

Problem

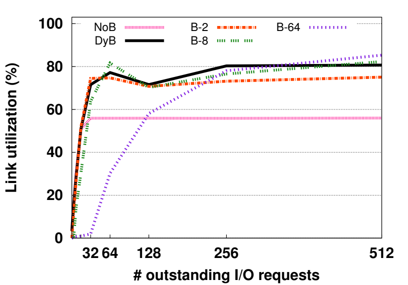
Should we send a batch message or wait for more I/O requests?

Proposal

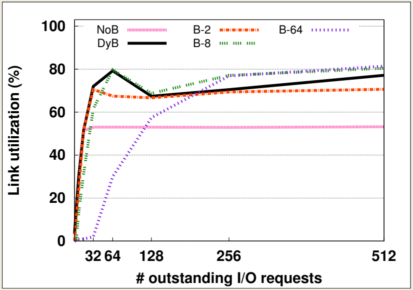
- We define N as the number of requests to put in a message
  - We calculate N using feedback from achieved (measured) throughput and available concurrency (current queue depth)
  - N is calculated constantly (every second)
  - We send a message every N requests or when a timeout occurs
  - To avoid local minimum we artificially increase/decrease N if it stays constant for some time

Batching Requests/Responses (no data)

- Dynamic Tyche-Batch improves link utilization by up to 49.5%
- Batching achieves up to 80.7% of link utilization



Read requests

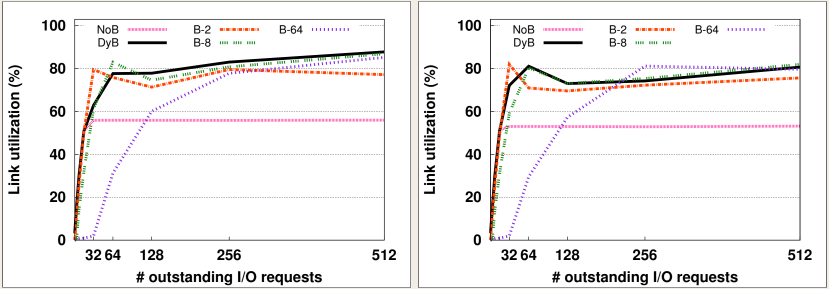


Write requests

FIO, 4kB requests, 1, 2, . . . , and 128 threads, and 4 outstanding I/O per thread

Batching Data + Requests/Responses

- Dynamic Tyche-Batch improves link utilization by up to 56.9%
- Batching achieves up to 88.0% of link utilization



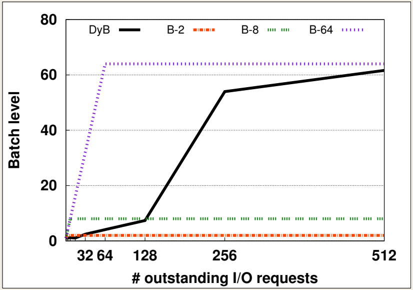
Read requests

Write requests

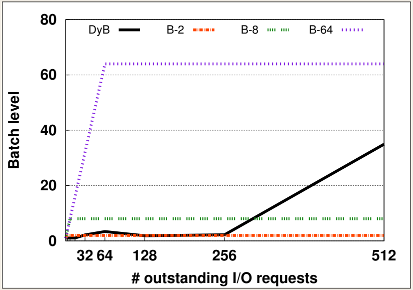
FIO, 4kB requests, 1, 2, . . . , and 128 threads, and 4 outstanding I/O per thread

Batching Requests/Responses (no data): Batch Level

- Reads: batch level increases as number of requests does
- Writes: batch level is kept constant up to more than 256 requests



Read requests

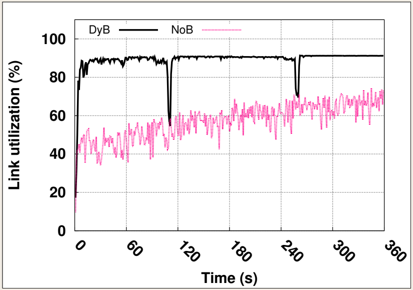


Write requests

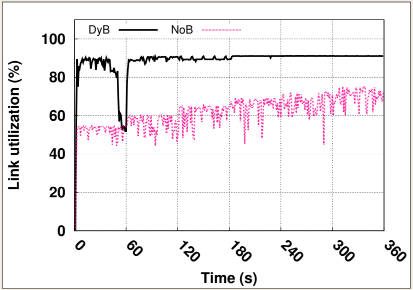
FIO, 4kB requests, 1, 2, . . . , and 128 threads, and 4 outstanding I/O per thread

Mixed Request Sizes

- 32 threads issue 4kB, every 60s a new thread issues 128kB
- Batching achieves up to 91.3% of link utilization



Read requests



Write requests

Conclusions

- We analyze overhead of Tyche [MSST14]
  - CPU overhead is up to 65% of total overhead (including OS etc)
  - I/O protocol is up to 47% of total overhead (excluding OS etc)
- At low concurrency: we reduce I/O protocol overhead by up to 61% via reducing context switches
- At high concurrency: we improve link utilization by up to 57% via adaptive batching
- We achieve:
  - 14μs overhead per I/O request, about 7μs on each of initiator/target (excluding network link)
  - 91% of theoretical maximum of link utilization: 287K out of 315K IOPS on 10GigE
  - No hardware support required

Reducing CPU and network overhead for small I/O requests in network storage protocols over raw Ethernet

Pilar González-Férez and Angelos Bilas

pilar@ditec.um.es bilas@ics.forth.gr

FORTH Institute of Computer Science

nanostreams

NESUS Network for Sustainable Ultrascale Computing

cost IC1305