

Tyche: An efficient Ethernet-based protocol for converged networked storage

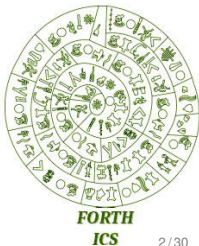
Pilar González-Férez and Angelos Bilas



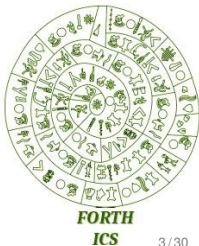
30th International Conference on Massive Storage Systems and Technology
MSST 2014

June 6, Santa Clara, California

- 1 Introduction
- 2 Design
- 3 Results
- 4 Conclusions and Future Directions



- 1 Introduction
- 2 Design
- 3 Results
- 4 Conclusions and Future Directions





Efficient access to networked storage

- Public clouds use shared storage $\Rightarrow$ lower cost
 - Easier to support migration and other operations
- *Converged storage* places low-latency storage devices in all servers
 - Storage requests exchanged between all compute servers
 - Network protocol $\Rightarrow$ important for achieving high I/O throughput
- Modern servers increase number of cores and NICs
 - Cost to access storage a concern as well
 - Cannot use custom NICs or controllers in all servers
- **Ethernet** $\Rightarrow$ dominant technology for datacenters
 - Lower cost and complexity $\Rightarrow$ single Ethernet network for storage and network data
- How to reduce protocol overheads for accessing remote storage over Ethernet?



Efficient access to networked storage (ii)

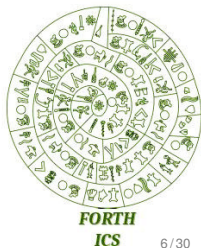
Challenges

- Synchronization from 10s of cores to a single link
- Link bundling for spatial parallelism
- NUMA affinity
- Dynamic assignment of links to cores

Our goal

Design a networked storage access protocol that dynamically manage cores, NICs, NUMA affinity

- 1 Introduction
- 2 Design
- 3 Results
- 4 Conclusions and Future Directions





Our Proposal

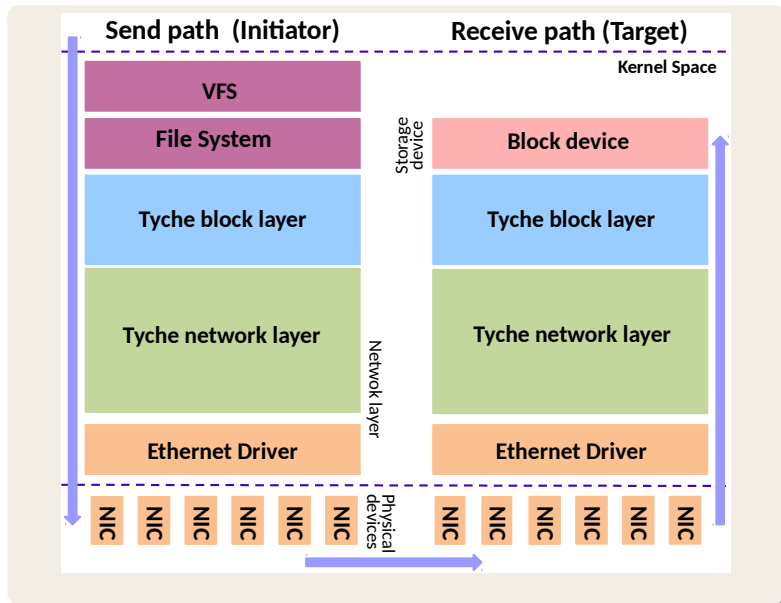
Tyche a network storage protocol that efficiently shares remote resources by using transparently several NICs and connections

Design goals

- Connection-oriented protocol
- Edge-based communication subsystem
- Use Ethernet
- Provide RDMA-type operations without any hardware support
- Can be deployed in existing infrastructures
- Create block device $\Rightarrow$ local view of a remote storage device
- Support any existing file system



Overview





Design Challenges

- Efficiently map I/O requests to network messages
- Memory management
- NUMA affinity
- Synchronization
- Allow high concurrency to saturate many NICs



Map I/O Requests to Network Messages

Network messages

- Request/completion messages $\Rightarrow$ I/O requests and completions
 - A request message corresponds to a single request packet
 - Request packet transferred as small Ethernet frames (< 100 bytes)
- Data messages $\Rightarrow$ data pages
 - RDMA operations $\Rightarrow$ scatter-gather list of memory pages
 - Data packets transferred as Jumbo Ethernet frames
 - Zero copy $\Rightarrow$ avoid data copy in receive path
 - For writes $\Rightarrow$ interchange NIC pages with Tyche pages
 - For reads, interchange cannot be applied
- ★ Ethernet header $\Rightarrow$ information about packets/messages
 - Provide end-to-end flow-control
 - Facilitate communication between block layer and network layer



Memory Management Overhead

Block layer

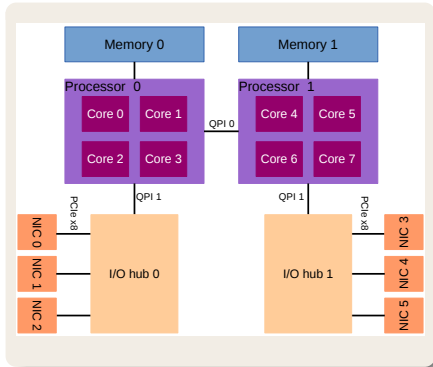
- **remq** $\Rightarrow$ Queue of pre-allocated request messages
 - Request and completion use the same message buffers
- **damq** $\Rightarrow$ Queue of pre-allocated descriptors for data messages
 - Target uses pre-allocated pages – avoids alloc/free
 - Initiator uses pages of “regular” I/O requests



NUMA Affinity

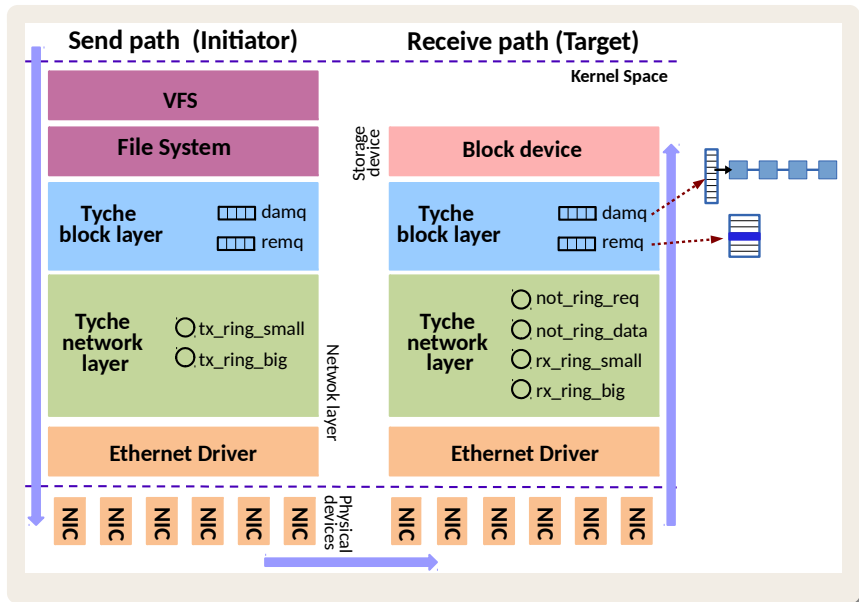
Maximum throughput only with right placement

- Logical connection per NIC
- Resources allocated on NUMA node where NIC is attached
 - `remq`
 - `damq`
 - `tx_ring`
 - `rx_ring`
 - `not_ring`
 - Private NIC rings
- Connection selected depending on location of buffers of users I/O requests





Tyche Overview





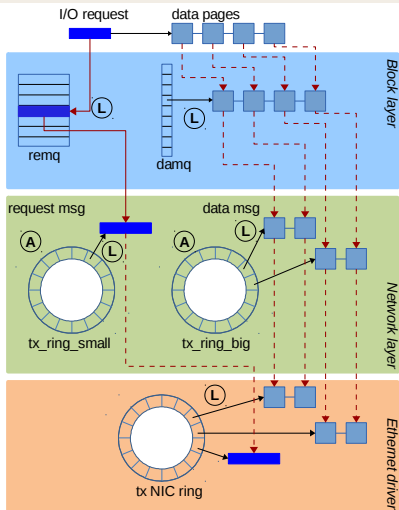
Synchronization Overhead

Context synchronization reduced for shared structures

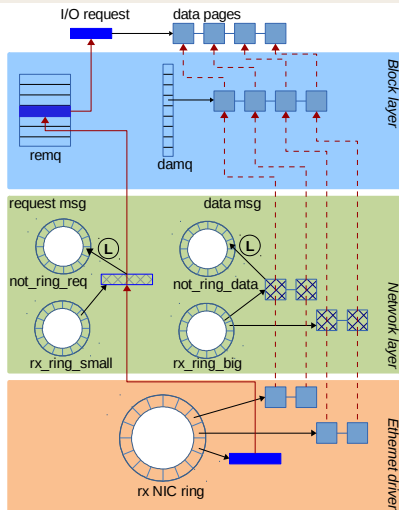
- Each connection has its own private resources
- Network layer
 - Three logical rings
 - **tx_ring** $\Rightarrow$ Transmission ring
 - **rx_ring** $\Rightarrow$ Receive ring
 - **not_ring** $\Rightarrow$ Notification ring
 - For each logical ring $\Rightarrow$ 2 different physical rings
 - A “small” ring $\Rightarrow$ request packets
 - A “large” ring $\Rightarrow$ data packets
 - Each physical ring has only two sync variables: head and tail
- Initiator specifies fixed positions at **remq** and **damq**
- For each packet, the sender specifies its position in **rx_ring**'s



Synchronization Overhead (ii)



Send path



Receive path



Synchronization Overhead (iii)

Many threads simultaneously issuing write requests cause lock synchronization overhead and lock contention at the NIC level

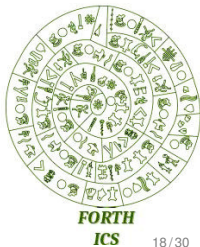
- Two modes of operation
 - **Inline mode:**
 - Application context issues requests with no context switch
 - **Queue mode:**
 - Applications insert I/O requests in a Tyche queue
 - Several threads submit network requests



Allow High Concurrency to Saturate Many NICs

- Tyche scales with load at initiator and target
- **Send path**
 - Initiator uses queue mode
 - Multiple threads place requests in a queue
 - Tyche controls the number of threads accessing each link
 - Target uses work queues to send I/O completions back
 - One work queue thread per physical core
- **Receive path**
 - Network layer $\Rightarrow$ one thread/NIC processes incoming data
 - Block layer $\Rightarrow$ several threads per NIC issue/complete requests
- Tested up to 6 x 10 Gbits/s

- 1 Introduction
- 2 Design
- 3 Results
- 4 Conclusions and Future Directions





Experimental Testbed

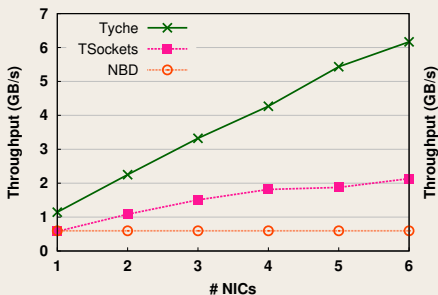
Hardware & Software

- Two nodes 4-core Intel Xeon E5520 @2.7GHz
 - **Initiator:** 12 GB DDR-III DRAM
 - **Target:** 48 GB DDR-III DRAM
 - 36 GB used as ramdisk
- 6 Myri10ge cards each node $\Rightarrow$ connected back to back
- CentOS 6.3
- Linux kernel 2.6.32
- **Benchmarks:** zmIO, FIO, Hbase+YCSB, Psearchy, Blast,...
- Tyche compared to:
 - Linux Network Block Device – NBD (today)
 - TSocket $\Rightarrow$ Tyche block layer using TCP/IP protocol

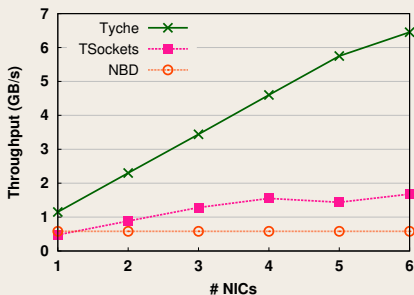


Baseline Performance

- zmIO, 32 threads, raw device (no file system), 1 MB request size
- Tyche throughput scales with the number of NICs
- Tyche achieves between 82% and 92 % of NIC throughput
- Tyche improves around 10x the throughput of NBD



Read requests

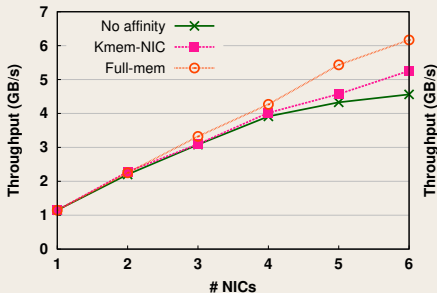


Write requests

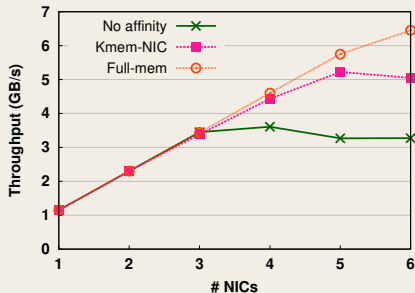


Impact of Affinity

- zmIO, 32 threads, raw device (no file system), 1 MB request size
- Tyche achieves maximum throughput only with right placement:
 - Full-mem placement improves no affinity performance up to 97%
 - Kmem-NIC placement improves no affinity performance up to 54%



Read requests

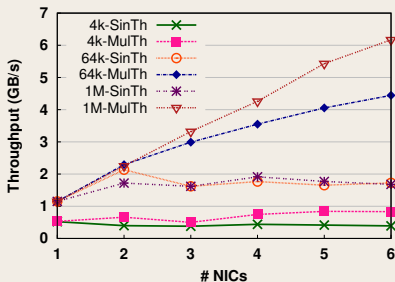


Write requests

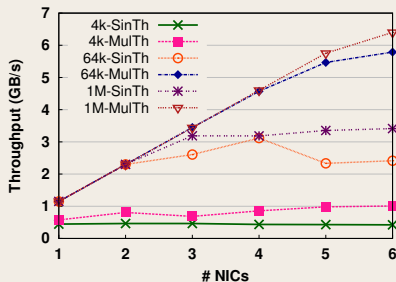


Receive Path Scaling

- zmIO, 32 threads, raw device, 4 kB, 64 kB, and 1 MB request sizes
- A single thread can process requests for three NICs: $\simeq 30$ GBits/s
- By using a thread per NIC:
 - Can achieve maximum throughput
 - Reduce receive path synchronization



Read requests

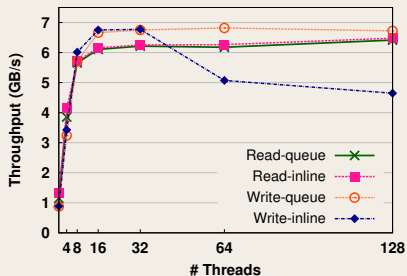
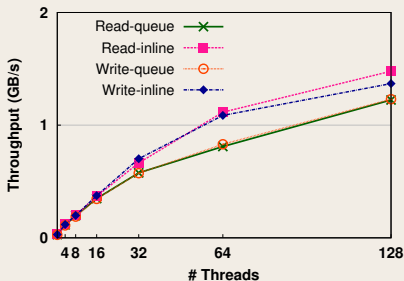


Write requests



Send Path Scaling

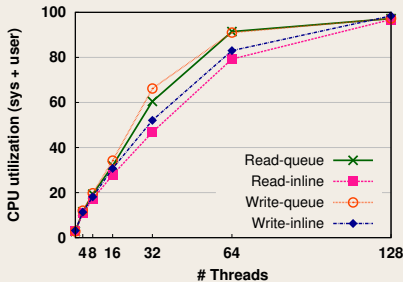
- FIO, XFS, 256 MB file size, several threads, each one its own file
- 4 kB requests: queue mode makes context switch
 - Inline mode outperforms queue mode up to 31%
- 512 kB requests: inline mode $\Rightarrow$ synchronization overhead and lock contention
 - Writes: queue mode outperforms inline mode up to 45%



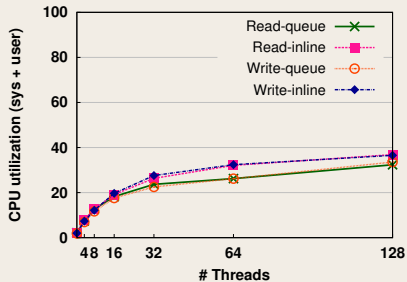


Queue vs. Inline Mode Overhead: 4 kB

- Queue mode pays context switch overhead
 - **Initiator:** CPU utilization increases up to 29%
 - **Target:** lower throughput $\Rightarrow$ CPU utilization drops up to 19%



Initiator, 4 kB request size

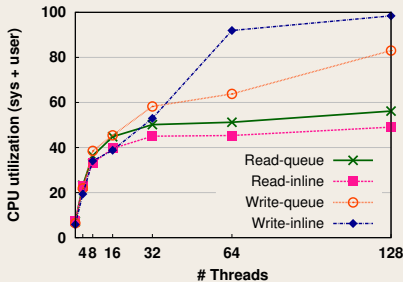


Target, 4 kB request size

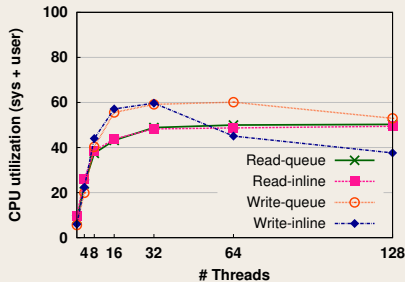


Queue vs. Inline Mode Overhead: 512 kB

- Writes: inline mode $\Rightarrow$ synchronization overhead and lock contention
 - Initiator:** CPU utilization increases up to 30%
 - Target:** lower throughput $\Rightarrow$ CPU utilization drops up to 40%



Initiator, 512 kB request size



Target, 512 kB request size



Other benchmarks

- Tyche always performs better than NBD and TSocketS

# NICs	Throughput (MB/s)				
	Tyche		NBD	TSockets	
	1	6	1	1	6
Psearchy	1,154	4,117	499	488	1,724
Blast	775	882	438	391	564
IOR-R 512k	573	1,670	212	226	745
IOR-W 512k	603	1,670	230	243	751
HBase-Read	303	295	154	168	229
HBase-Insert	106	112	99	54	92



Conclusions and Future Work

Conclusions

- **Tyche** $\Rightarrow$ networked storage protocol
- Transparently use multiple NICs and multiple connections
- Address contention, memory mgmt, and network ordering
- Address NUMA affinity issues
- Achieve scalable throughput
 - Reads: up to 6.4 GBytes/s (7 max)
 - Writes: up to 6.7 GBytes/s (7 max)
- Significantly outperform NBD and TSOckets

Future Directions

- Consider how can co-exist with other network protocols over Ethernet



Tyche: An efficient Ethernet-based protocol for converged networked storage

Pilar González-Férez and Angelos Bilas

pilar@ditec.um.es

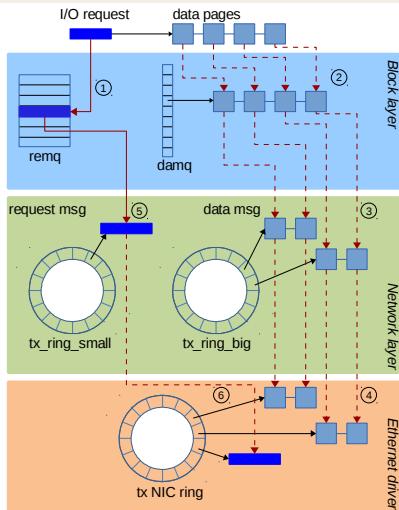
bilas@ics.forth.gr



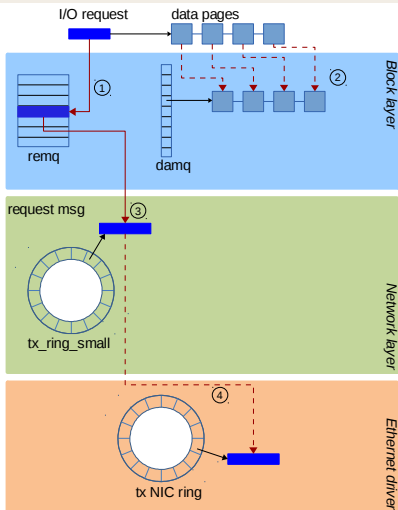
FP7-ICT-610509



Send Path Overview



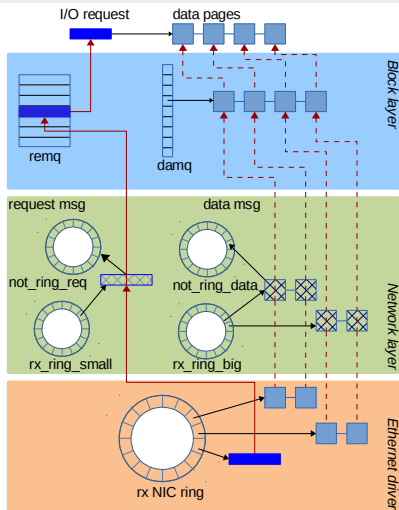
Write requests



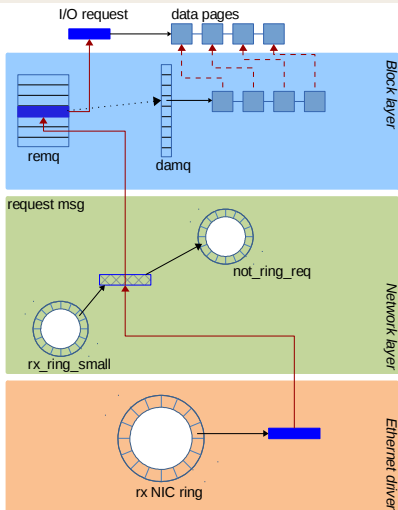
Read requests



Receive Path Overview



Write requests



Read requests