ASPERA HIGH-SPEED TRANSFER

Moving the world's data at maximum speed

80 GBIT/S OVER IP USING DPDK

Performance, Code, and Architecture

Charles Shiflett

Developer of next-generation FASP

bear @ us . IBM . com

bear@asperasoft.com

Benefits of using DPDK

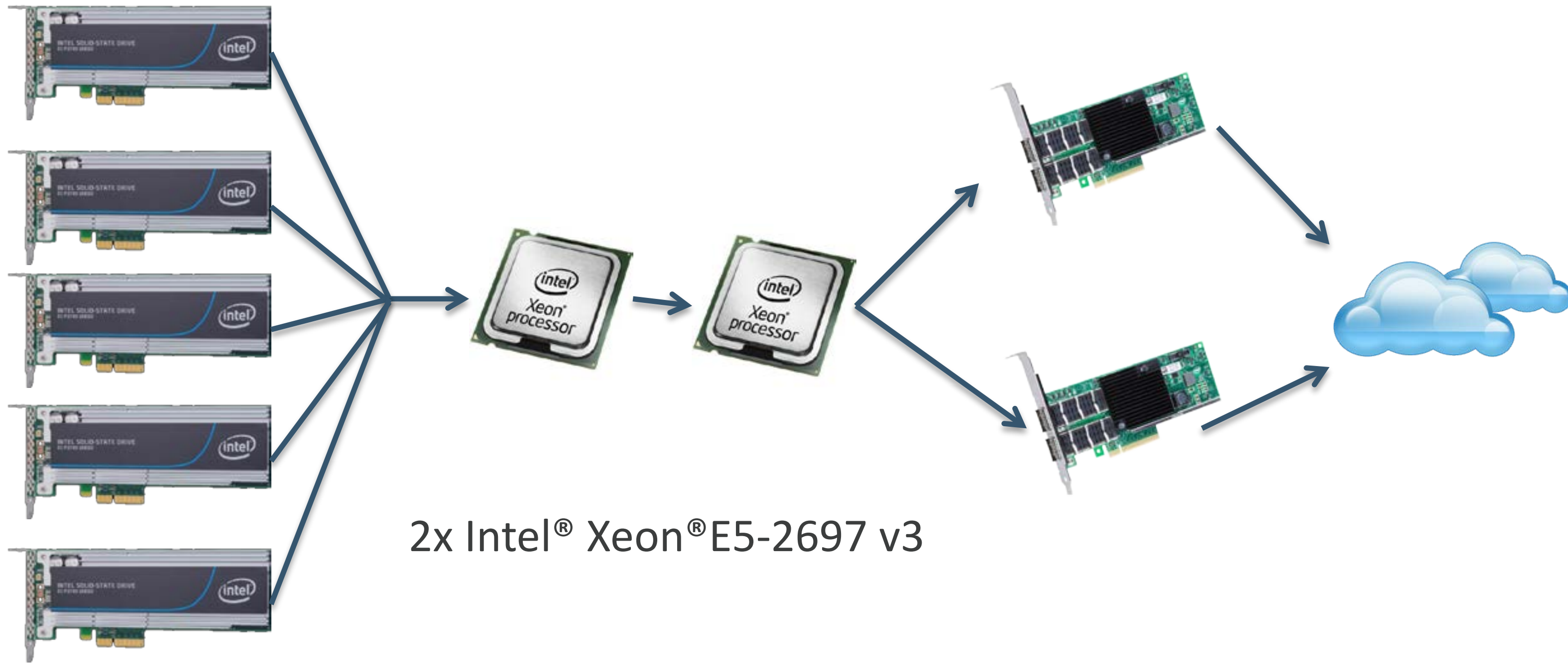
- Lightweight API that exposes Network and Acceleration Cards
- Rapid Iteration on high performance transport designs as compared to equivalent code developed as Kernel Module
- Set of performance best practices
  - Time, Locking, Memory Alignment, and Synchronization
  - Avoids libc and system call interaction when possible
- NUMA aware solution
- Framework for High Performance Userland Direct I/O

FASP NX DESIGN GOAL

Goal is to bring industry closer to the data they are using both within the data center and across the globe.

- Effectively utilize PCIe based IO on Intel Server platform.
- Solve storage bottleneck using Direct I/O w/ multiple controllers
- Solve network bottleneck using DPDK and link aggregation
- Provide a secure transport solution using Intel AES-NI GCM
- Significantly reduce time spent getting your data set to the CPU

HARDWARE CONFIGURATION



5x Intel® DC P3700 NVMe SSD

2x Intel® XL710 40 GbE Ethernet QSFP+

aspera
an IBM® company

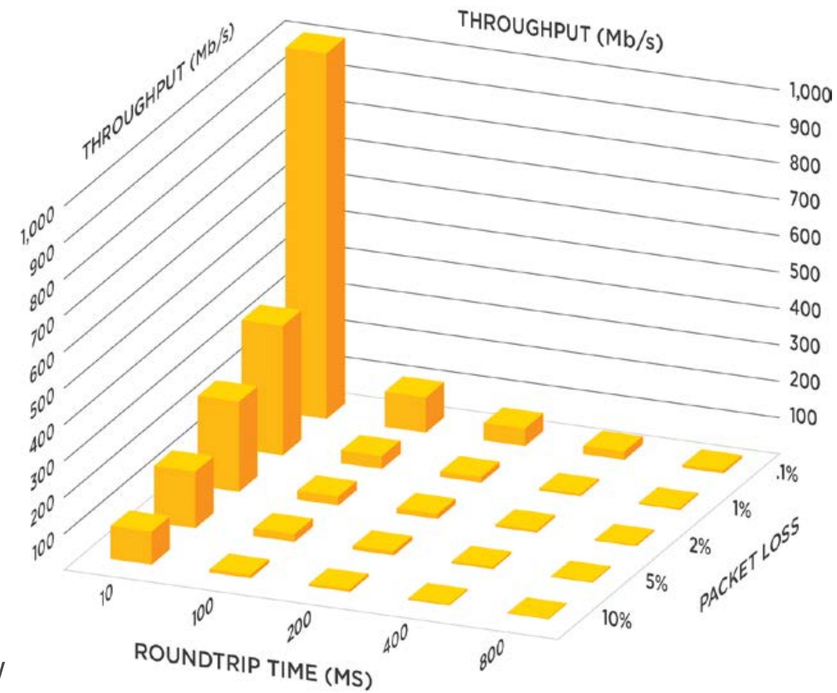
ASPERA'S MISSION

Creating next-generation transport technologies

that move the world's digital assets at maximum speed,

regardless of file size, transfer distance and network conditions.

CHALLENGES WITH TCP AND ALTERNATIVE TECHNOLOGIES



Distance degrades conditions on all networks

- Latency (or Round Trip Times) increase
- Packet losses increase
- Fast networks just as prone to degradation

TCP performance degrades with distance

- Throughput bottleneck becomes more severe with increased latency and packet loss

TCP does not scale with bandwidth

- TCP designed for low bandwidth
- Adding more bandwidth does not improve throughput

Alternative Technologies

- TCP-based - Network latency and packet loss must be low
- UDP traffic blasters - Inefficient and waste bandwidth
- Data caching - Inappropriate for many large file transfer workflows
- Modified TCP - Improves on TCP performance but insufficient for fast networks
- Data compression - Time consuming and impractical for certain file types
- CDNs & co-lo build outs - High overhead and expensive to scale

FASP™: HIGH-PERFORMANCE TRANSPORT

Maximum transfer speed

- Optimal end-to-end throughput efficiency
- Transfer performance scales with bandwidth independent of transfer distance and resilient to packet loss

Congestion Avoidance and Policy Control

- Automatic, full utilization of available bandwidth
- On-the-fly prioritization and bandwidth allocation

Uncompromising security and reliability

- Secure, user/endpoint authentication
- AES-128 cryptography in transit and at-rest

Scalable management, monitoring and control

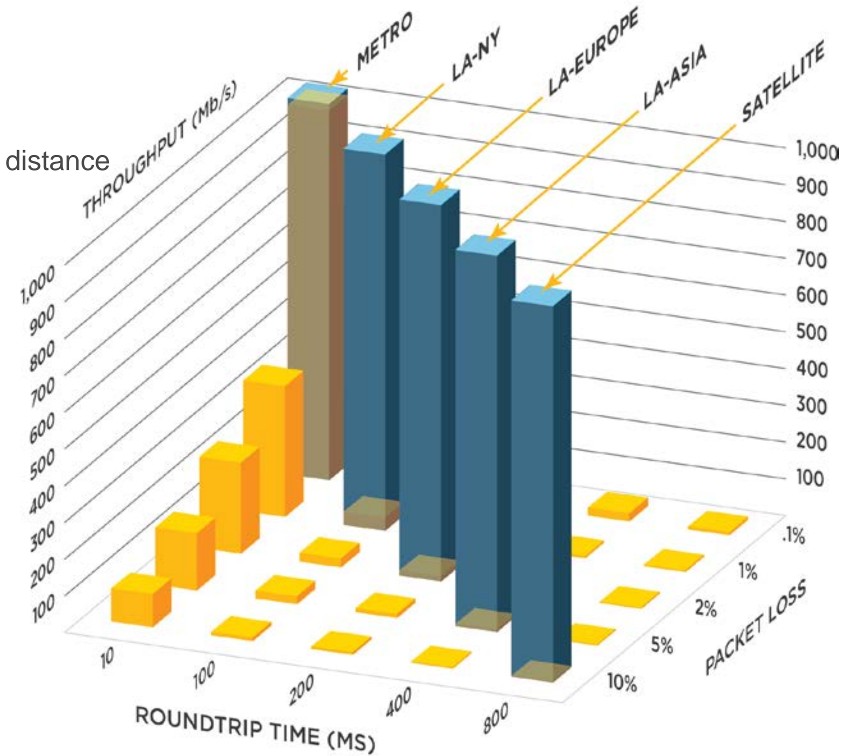
- Real-time progress, performance and bandwidth utilization
- Detailed transfer history, logging, and manifest

Low Overhead

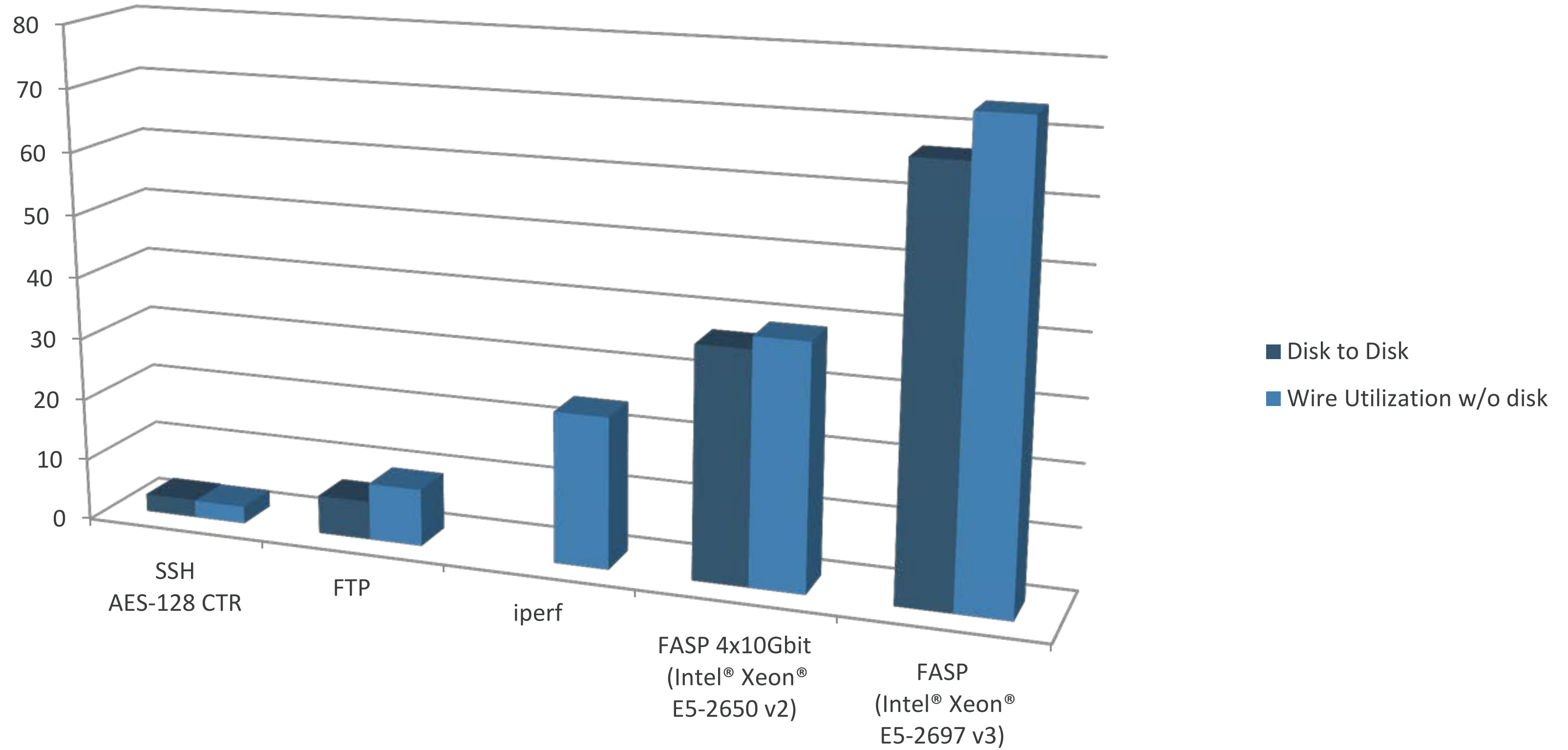
- Less than 0.1% overhead on 30% packet loss
- High performance with large files or large sets of small files

Resulting in

- Transfers up to thousands of times faster than FTP
- Precise and predictable transfer times
- Extreme scalability (concurrency and throughput)



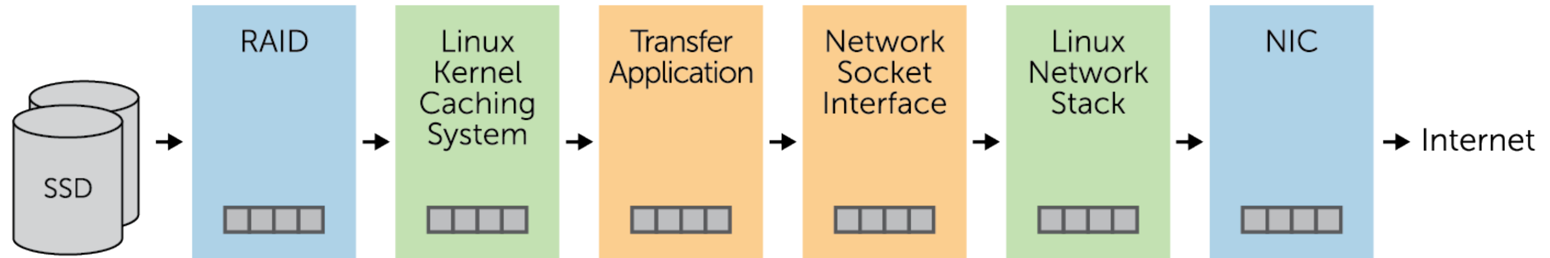
LAN TRANSFER RESULTS IN GBIT PER SECOND



Results from SC 14 showing the relative performance of Network Transfer Technologies

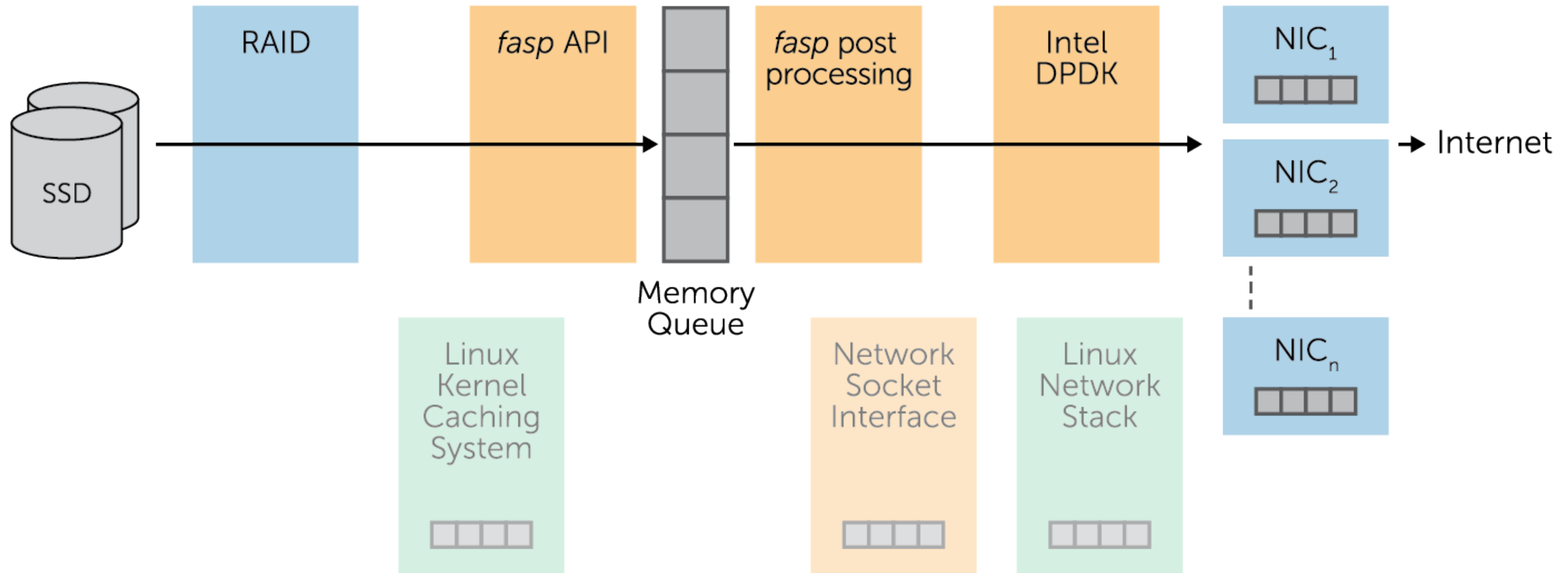
TYPICAL TRANSFER APPLICATION

BEFORE



Hardware Kernel Transfer Application

AFTER



AVOIDING STALLS

- Synchronization primitives aligned to cache-line, lockless. Example for circular buffer queues;

  Dequeue, claim block: `block_id = __sync_fetch_and_add ( &sess->block_tail, 1 ) % block_count;`
  Dequeue, consume block: `while (  !__sync_val_compare_and_swap( &sess->block[block_id].state, 0, 1) ) PAUSE;`

- Zero copy paradigm critical to good memory performance. Memory copies are VERY expensive!
  - Memory copies should be avoided at all costs, writes to Memory must be page aligned to avoid read of destination cacheline
  - Solution is to operate on datasets which fit into L3 cache. New tools provide better insight into L3 cache utilization such that code can be optimized.

- Low latency direct I/O.
  - Intel DDIO critical in bringing data into L3 from Storage, Network Adapters, and PCIe Accelerators.

- DPDK provides framework for operating on data streamed into L3 cache
  - Avoid TLB misses by using Hugepages
  - Avoid NUMA issues and minimize thread preemption.
  - Userland IO drivers where possible.

EXAMPLE OF ALIGNED/UNALIGNED MEMORY COPY

```
static inline void int_copy_1408 (uint64_t* dst, uint64_t* src) {
  int i=0;
  for (i=0; i<1408/8; i++)
    dst[i] = src[i];
}

static inline int naive_memcpy( uint8_t* dst, uint8_t* src, int len ) {
  int i=0;
  for (i=0; i < len; i+=1408)
    int_copy_1408( (uint64_t*)(dst+i), (uint64_t*)(src+i) );
  return 0;
}

/*
---------------------------------------||---------------------------------------
Time elapsed: 1000 ms
Called sleep function for 1000 ms
---------------------------------------||---------------------------------------
--             Socket 0              --||--             Socket 1              --
---------------------------------------||---------------------------------------
---------------------------------------||---------------------------------------
---------------------------------------||---------------------------------------
--   Memory Performance Monitoring   --||--   Memory Performance Monitoring   --
---------------------------------------||---------------------------------------
--  Mem Ch 0: Reads (MB/s): 3035.80  --||--  Mem Ch 0: Reads (MB/s):    0.30  --
--            Writes(MB/s): 1515.10  --||--            Writes(MB/s):    0.03  --
--  Mem Ch 1: Reads (MB/s): 3035.95  --||--  Mem Ch 1: Reads (MB/s):    0.28  --
--            Writes(MB/s): 1515.09  --||--            Writes(MB/s):    0.03  --
--  Mem Ch 2: Reads (MB/s): 3035.71  --||--  Mem Ch 2: Reads (MB/s):    0.25  --
--            Writes(MB/s): 1515.10  --||--            Writes(MB/s):    0.03  --
--  Mem Ch 3: Reads (MB/s): 3035.66  --||--  Mem Ch 3: Reads (MB/s):    0.27  --
--            Writes(MB/s): 1515.10  --||--            Writes(MB/s):    0.03  --
-- NODE0 Mem Read (MB/s):  12143.12  --||-- NODE1 Mem Read (MB/s):     1.10  --
-- NODE0 Mem Write (MB/s):  6060.39  --||-- NODE1 Mem Write (MB/s):    0.12  --
-- NODE0 P. Write (T/s) :        12  --||-- NODE1 P. Write (T/s):         0  --
-- NODE0 Memory (MB/s):    18203.51  --||-- NODE1 Memory (MB/s):       1.22  --
---------------------------------------||---------------------------------------
--                System Read Throughput(MB/s):  12144.22                     --
--               System Write Throughput(MB/s):   6060.51                     --
--              System Memory Throughput(MB/s):  18204.72                     --
---------------------------------------||---------------------------------------
INT COPY Time: 0.248 Written: 1408 MB, 5676.8MB/S
*/
```

```
static inline void copy128( uint8_t *dst, uint8_t *src ) {
#ifdef AVX2
  __m256i m0;
  __m256i m1;
  __m256i m2;
  __m256i m3;
  asm volatile (
    "VMOVDQA (%[src]), %[m0]\n\t"
    "VMOVDQA 32(%[src]), %[m1]\n\t"
    "VMOVDQA 64(%[src]), %[m2]\n\t"
    "VMOVDQA 96(%[src]), %[m3]\n\t"
    "VMOVNTDQ %[m0], (%[dst])\n\t"
    "VMOVNTDQ %[m1], 32(%[dst])\n\t"
    "VMOVNTDQ %[m2], 64(%[dst])\n\t"
    "VMOVNTDQ %[m3], 96(%[dst])\n\t"
    :  [m0] "=x" (m0) ,
       [m1] "=x" (m1) ,
       [m2] "=x" (m2) ,
       [m3] "=x" (m3)
    : [src] "r" (src),
      [dst] "r" (dst)
    : "memory"
  );
};

static inline int alimemcpy( uint8_t* dst, uint8_t* src, int len ) {
  int i=0;
  for (i=0; i < len; i+=1408)
    copy1408( dst+i, src+i );
  return 0;
}

/*
---------------------------------------||---------------------------------------
--             Socket 0              --||--             Socket 1              --
---------------------------------------||---------------------------------------
---------------------------------------||---------------------------------------
---------------------------------------||---------------------------------------
--   Memory Performance Monitoring   --||--   Memory Performance Monitoring   --
---------------------------------------||---------------------------------------
--  Mem Ch 0: Reads (MB/s): 2467.78  --||--  Mem Ch 0: Reads (MB/s):    0.42  --
--            Writes(MB/s): 2390.23  --||--            Writes(MB/s):    0.03  --
--  Mem Ch 1: Reads (MB/s): 2467.45  --||--  Mem Ch 1: Reads (MB/s):    0.46  --
--            Writes(MB/s): 2390.38  --||--            Writes(MB/s):    0.04  --
--  Mem Ch 2: Reads (MB/s): 2467.65  --||--  Mem Ch 2: Reads (MB/s):    0.34  --
--            Writes(MB/s): 2390.34  --||--            Writes(MB/s):    0.03  --
--  Mem Ch 3: Reads (MB/s): 2467.22  --||--  Mem Ch 3: Reads (MB/s):    0.39  --
--            Writes(MB/s): 2390.28  --||--            Writes(MB/s):    0.03  --
-- NODE0 Mem Read (MB/s):   9870.11  --||-- NODE1 Mem Read (MB/s):     1.61  --
-- NODE0 Mem Write (MB/s):  9561.24  --||-- NODE1 Mem Write (MB/s):    0.13  --
-- NODE0 P. Write (T/s) :   5956090  --||-- NODE1 P. Write (T/s):         0  --
-- NODE0 Memory (MB/s):    19431.35  --||-- NODE1 Memory (MB/s):       1.74  --
---------------------------------------||---------------------------------------
--                System Read Throughput(MB/s):   9871.72                     --
--               System Write Throughput(MB/s):   9561.37                     --
--              System Memory Throughput(MB/s):  19433.09                     --
---------------------------------------||---------------------------------------
VECTOR ALIGNED Time: 0.156 Written: 1408 MB, 9032.9MB/S
*/
```

5.7 GB/S copy performance constrained by RFO

9.0 GB/s copy avoids RFO, close to per core limit

* Per channel results via Intel Performance Counter Monitor

OPTIMIZING NETWORK THROUGHPUT

```
int w=0;
static const char names[] = "abcdefghijklmnopqrstuvwxyz123456789";
for ( each queue )
  // rte_mempool_create( name, # pkts, sz of pkts, # pkts in cache, ...
  queue = rte_mempool_create(names+w++, 0x1ff,
    1536+sizeof(struct rte_mbuf) + RTE_PKTMBUF_HEADROOM, 128, 128,
    rte_pktmbuf_pool_init, NULL,
    rte_pktmbuf_init, NULL,
    ai->p[i].socket, MEMPOOL_F_SP_PUT|MEMPOOL_F_SC_GET);
```

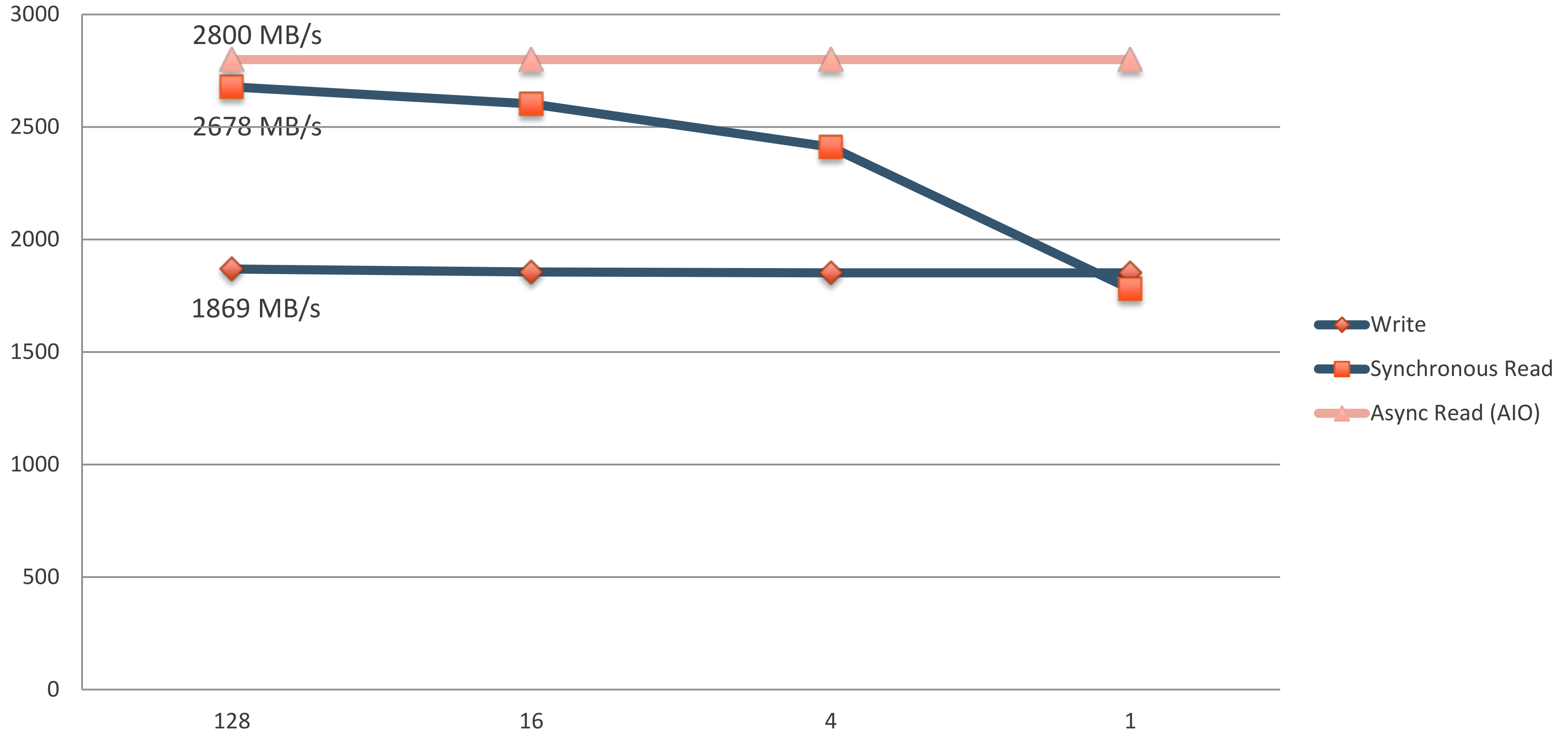
- Create network queues which fit into L3. Each queue tied to a specific core.
- Experiment with values to minimize memory, while retaining useful properties:
  - Should support bulk allocation
    - `MAX_RX_BURST < rx_free_thresh < nb_rx_desc (set when assigning queue to port)`
    - `Nb_rx_desc % rx_free_thresh == 0`
  - Ideally `pkts_in_cache % nb_pkts == 0` and `nb_pkts == 2^n-1`
- PTHRESH/HTHRESH/WTHRESH values feel like black magic, but can have a huge effect on performance. Start by copying values from example applications.

IMPROVING STORAGE THROUGHPUT

- Traditional storage is built around the idea of moving slow data from disk to memory and then from memory to application. Memory is used to cache data to improve access speeds.
  - Cache structure quickly becomes a bottleneck as transfer speeds exceed 10 gbit/s.
  - While individual spinning disks are slow, JBOD’s of 100’s of disk have very high aggregate bandwidth
  - Modern SSDs (especially NVMe) is very fast.

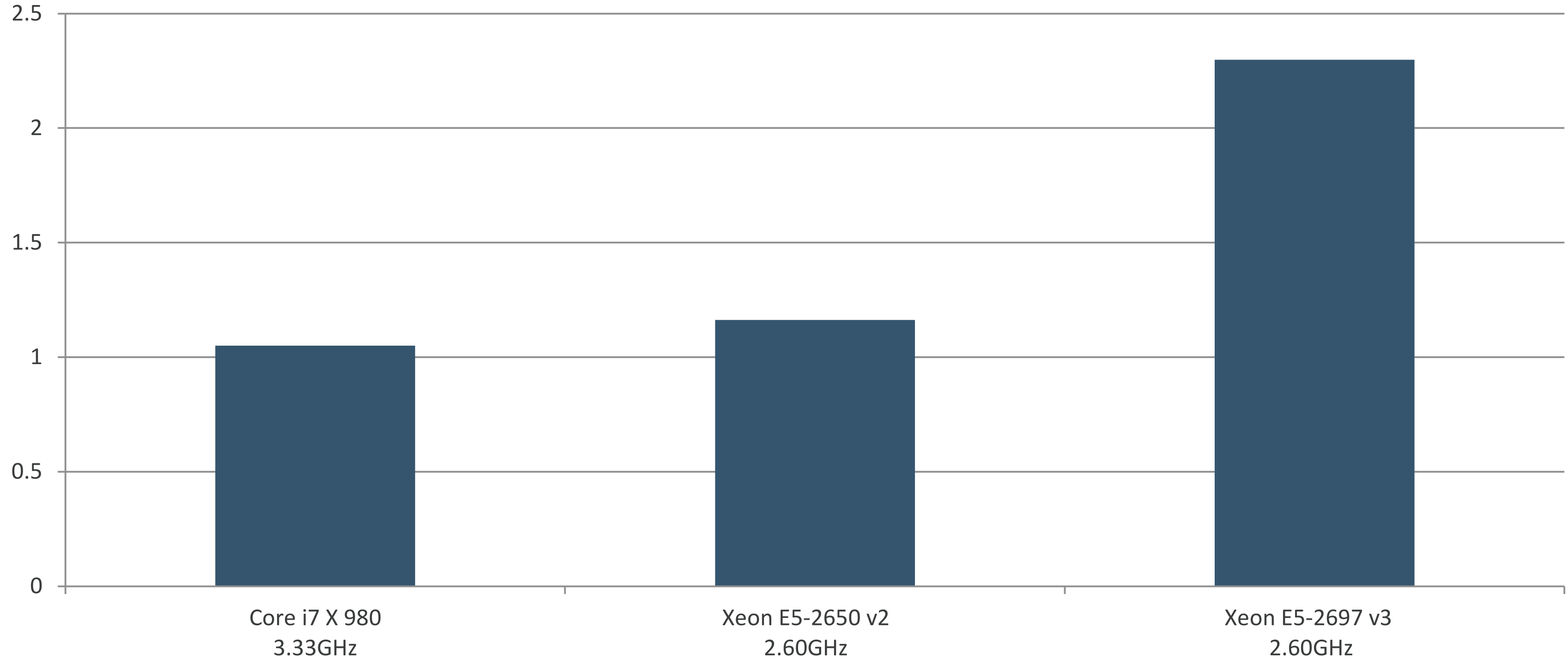
- Two solutions for fast data
  - Use XFS with direct IO (and/or MMAP)
    - Have shown performance at about 40gbit/s with hardware raid and direct attach disks.
    - Have shown performance at about 70gbit/s with NVMe SSD.
    - Limited by how many devices you can connect to PCIe data link.
  - Clustered parallel storage
    - Aspera is targeting IBM GPFS (Spectrum) or Lustre based solutions
    - Individual nodes can be slow, but by aggregating nodes high performance is achieved
    - Both offer direct I/O solutions

DC P3700 PERFORMANCE (SINGLE DRIVE)



Performance relative to block size (SI Units in MB)

ENCRYPTION IS NOT A BOTTLENECK WITH HW AES-GCM



NEW ON THE WIRE PROTOCOL

- Built around AES-128 GCM, which is similar to DTLS (Datagram TLS).

```
AES-128 GCM {
  uint64 IV;
  uint8[] payload;
  uint8[16] MAC;
}
```

```
DTLSRecord {
  ContentType type;
  ProtocolVersion version;
  uint16 epoch;
  uint48 sequence_number;
  uint16 length;
  uint8_t[length] payload;
}
MAC {
  uint8[16] hash;
}
```

```
FASP4 {
  uint16 stream_id;
  uint8  ver_and_pkt_type;
  uint40 sequence_number;
  uint8[] payload;
  uint8[16] hash; // optional
}
```

- In the same way it is possible to encapsulate FASP NX header in UDP, will also support Encapsulating FASP4 in DTLS.

aspera
an IBM® company

EXAMPLE TRANSFER SESSION USING FASP NX API

```c
#include <stdio.h>
#include <stdint.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>


#include "fasp4.h"

#define BLK_SZ (512*1024)

int main ( int argc, char** argv ) {
  uint8_t* hugepage, buf;
  int res;

  hugepage = fasp4_claim_page( 4ULL * 1024 * 1024 * 1024 );

  if (! hugepage ) {
    perror("Claiming hugepage region");
    return res;
  }


  struct app_info* ai;
  ai = (struct app_info*) hugepage;

  // Set encryption key, and per port destination IPs

  memcpy ( ai -> session->key,  "0123456789abcdef", 16 );
  ai -> session->dst_ip[0] = IPv4 ( 192,168,1,1 );
  ai -> session->dst_ip[1] = IPv4 ( 192,168,1,2 );

  // Connect to remote host

  res= fasp4_connect( ai );
  if ( res ) {
    perror("Connection failed");
    return res;
  }

  int fd = open ( argv[1], O_RDONLY );
  int bytes_read = 1, i;
  struct block_info* bi;
```

```c
if ( !fd ) {
  perror("Opening file");
  return -1;
}

fasp4_send_meta( ai, fasp4_file_meta_simple( fd ) );

// begin transferring data.
while ( 1 ) {
  bi = fasp4_claim_block( ai, 16 );
  if (!bi) {
    // pause
    continue;
  }

  buf = fasp4_get_buf ( bi ) ;
  bytes_read = read( fd, buf, 16 * BLK_SZ );

  if ( !bytes_read )
    break;

  for ( i=0; i < 16; i++ ) {
    bi[i].sz = (bytes_read-(i*BLK_SZ)) > BLK_SZ ? BLK_SZ : bytes_read-(i*BLK_SZ);
    // sz < 0 signifies an empty block.
    fasp4_put_block ( &bi[i] );
  }

}

fasp4_send_meta( ai, fasp4_meta_transfer_complete() );
res = fasp4_wait_for_receiver( ai );

if ( res ) {
  // Transfer encountered an error, display it.
  return res;
}

fasp4_free( ai );
printf("Success\n");
exit(0);
```

FASP NX API'S

- Native FASP NX API
  - Share hugepage sized regions of memory with client application
  - Usable from VM or Native machine
  - Enables Zero Copy transfer solutions of non file data
  - Dynamically set rate policy, destination IP, and so on.
- Traditional Aspera (ascp) command line tools
  - Similar command line to SCP, with FASP NX performance.
- Existing Aspera API
  - Faspmgmt
  - Web Services
  - Rest based Queries

FUTURE DIRECTION

- Aspera's Goal: Transfer Solution to Petascale Datasets
  - 1 SI Terabyte in 2 Minutes
  - 1 SI Petabyte in 1⅜ Days
  - Performance improvements expected to scale relative to PCIe interconnect.
- Better integration with storage systems
  - Take advantage of native APIs to avoid kernel cache.
- Better integration with network hardware
  - Expected to show 100gbit/s transfers using Mellanox ConnectX®-4
  - Query network switch and routers?
- Support wider use cases
  - Compression, Forward Error Correcting Codecs, Berkeley Sockets API

THANK YOU

CHARLES SHIFLETT

bear @ us.ibm.com