High Speed Object Transfer

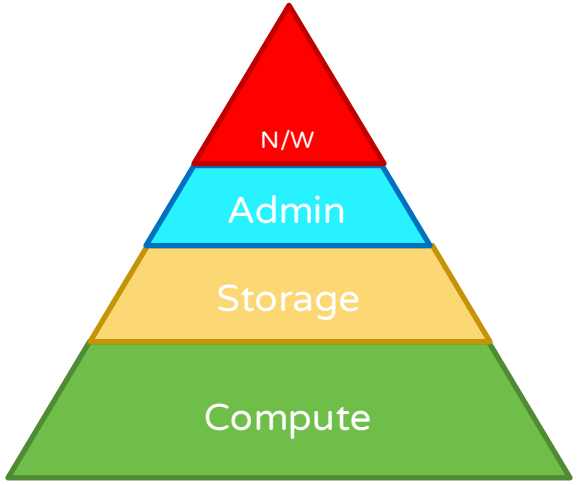
Motr Interfaces Workshop

5th Nov 2021

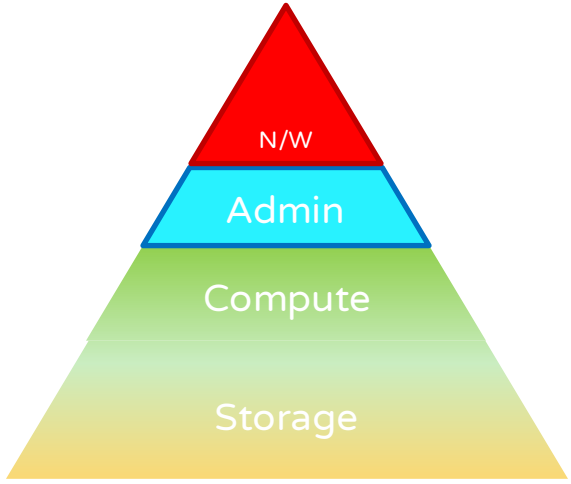
Andrew Lahiff, Shaun de Witt (UK Atomic Energy Authority)

Motivation

- ★ The data tsunami is here…
- ★ Use cases have changed with the blurring of the lines between traditional HPC and AI at scale
  - ★ CORTX is optimised to make use of storage both through IO tiering and making the storage part of the compute



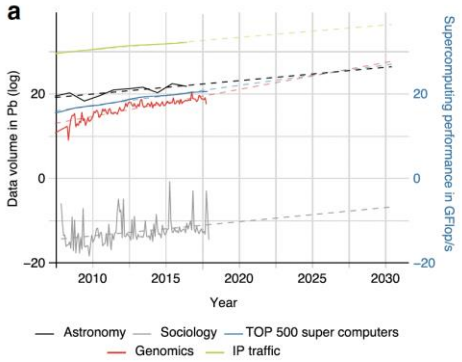
Traditional HPC

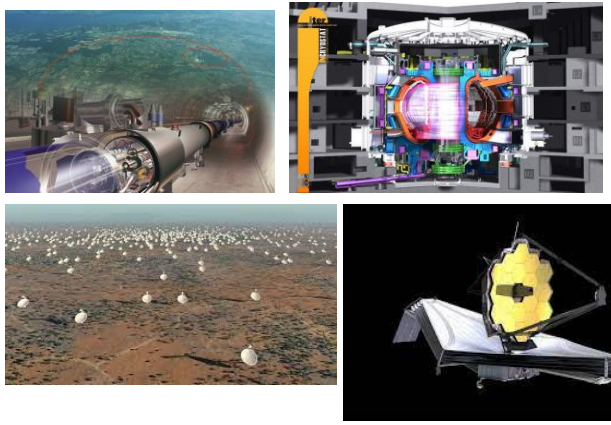


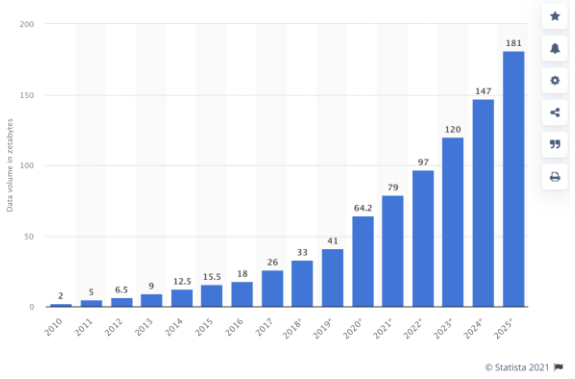
CORTX HPC

Data Volumes and the Rise of AI

- ★ Data Volume Growth outstripping “traditional” HPC growth
- ★ Several “Big Data” experiments coming online in the next decade
- ★ AI for science and commerce becoming a huge data magnet to extract information from un- or semi-structured data







Navarro et al. Genome Biology (2019) 20:109
https://doi.org/10.1186/s13059-019-1724-1

Science Data

- Large experiments will not store data in single “silos”
  - Most experiments are global and data is distributed globally
  - Global data movement is inefficient
- CORTX systems will have similar issue to traditional HPC…
  - Getting data to/from the HPC system is time consuming and need to be done BEFORE execution can happen
- … but will be magnified
  - The movement from compute centric to data centric means more and larger data will need to be imported efficiently
  - Sometimes will be unable to run “back-fill” jobs since most storage will be needed by data heavy applications

Traditional HPC
Small Data In, Big Data Out

Converged HPC
Big Data In, Small Data Out

CORTX HPC
Big Data In, Big Data Out

Impact of packet loss over WAN

- ★ Packet loss has a massive impact on WAN transfers
- ★ Theoretical TCP bandwidth governed by Mathis Equation

$$T_{TCP} = \frac{MSS}{RTT} * \frac{1}{\sqrt{p}}$$

(MSS = Max Segment Size, RTT is round trip time and p is probability of packet loss)

- ★ Some examples with a storage system in Paris and fixed segment size of 1 kByte
  - ☆ Fixed packet loss of 0.01%
    - ✦ Throughput (London; ~10ms RTT); T_{TCP} ~ 10MB/s
    - ✦ Throughput (New York; ~75ms RTT); T_{TCP} ~ 1.3 MB/s
    - ✦ Throughput (Tokyo; ~260ms RTT); T_{TCP} ~ 384 kB/s
  - ☆ Fixed RTT (Helsinki->Paris ~ 44ms)
    - ✦ Packet loss 0.001%; T_{TCP} ~ 718 kB/s
    - ✦ Packet loss 0.01%; T_{TCP} ~ 227 kB/s
    - ✦ Packet loss 0.1%; T_{TCP} ~ 72 MB/s
- ★ UDP largely ignore packet loss so just governed by MSS and RTT
  - ☆ But non-negligible risk of data corruption/error

Existing High Speed Protocols: Globus GridFTP

- ★ By default uses TCP
- ★ Allows parallel transfers
  - ☆ Multiple streams for a single file for large files
  - ☆ Multiple channels for transferring many small files
- ★ Was very commonly used in WLCG, but being deprecated
  - ☆ Open source version no longer available
  - ☆ Move away from certificate based transfers

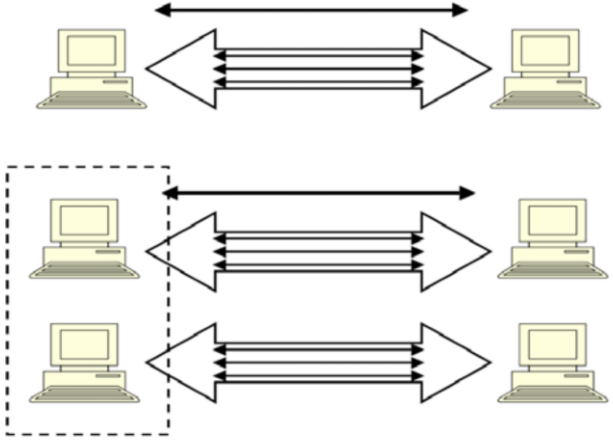


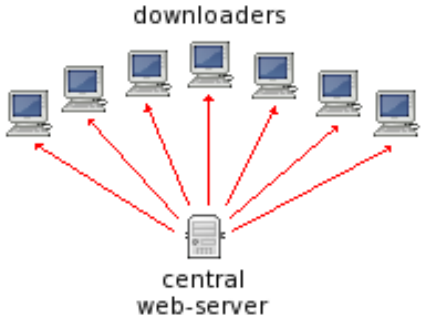
Figure 4. GridFTP Striped Data Transfer and Parallel and Striped Data Transfer.

W. Allcock, J. Bresnahan, R. Kettimuthu, M. Link, C. Dumitrescu, I. Raicu, I. Foster, The Globus Striped GridFTP Framework and Server, SC'05, ACM Press, 2005.

Existing High Speed Protocols: Bit-Torrent

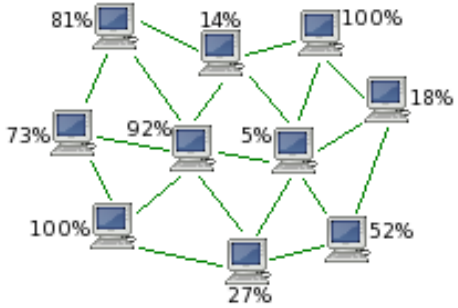
- ★ Widely used in file sharing and streaming services
- ★ Based on P2P Networking, by default using TCP
- ★ Not widely used in science where P2P sharing is currently not the norm…

Traditional Centralized Downloading



- Slow
- Single point of failure
- High bandwidth usage for server

Decentralized Peer-to-Peer Downloading



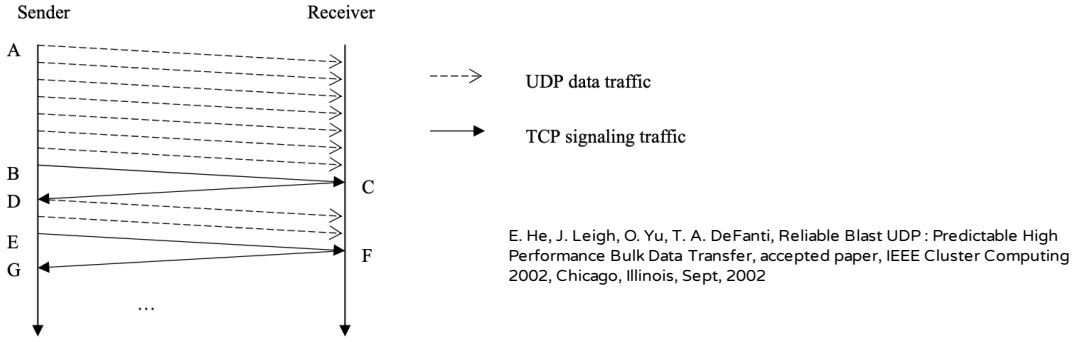
- Fast
- No single point of failure
- All downloaders are also uploaders

Existing High Speed Protocols: HSCP and UDR

- ★ Provide UDT based replacements for scp & rsync
- ★ UDT = UDP based Data Transfer Library
  - ☆ Hybrid scp (HSCP): UDP (uses UDT) for the file transfer part of scp
  - ☆ UDR: UDT wrapper around rsync
- ★ Discovered late in the project – subject for future community work

Existing High Speed Protocols: Blast-UDP

- What it is
  - Designed for moving very large files over wide area networks
  - Simple replacement for FTP
  - UDP for data transport, TCP for control channel
  - Supports congestion control
- Results
  - Experimentally able to get 680 MBs between Amsterdam and Chicago down a 1Gbps network
- BUT… According to own documentation only any good on private or dedicated networks



E. He, J. Leigh, O. Yu, T. A. DeFanti, Reliable Blast UDP : Predictable High Performance Bulk Data Transfer, accepted paper, IEEE Cluster Computing 2002, Chicago, Illinois, Sept, 2002

Figure 1. The Time Sequence Diagram of RBUDP

Existing High Speed Protocols: Tsunami

- What it is
  - Similar to Blast-UDP but less aggressive
  - Needs less tuning and should be more robust
  - Standard client/server architecture
  - TCP control stream & UDP data stream
  - User-space application
- Example
  - Copying files between VMs in the AWS US-East & Singapore regions

File size	Tsunami UDP (time)	SCP (time)
5 GB	51.56s	11m 50s
50 GB	19m 33s	1hr 50m

http://harish11g.blogspot.com/2013/05/How-to-transfer-huge-data-to-Amazon-Web-Services-AWS-using-Tsunami-UDP-Configuration-Comparison.html

Existing High Speed Protocols: Tsunami

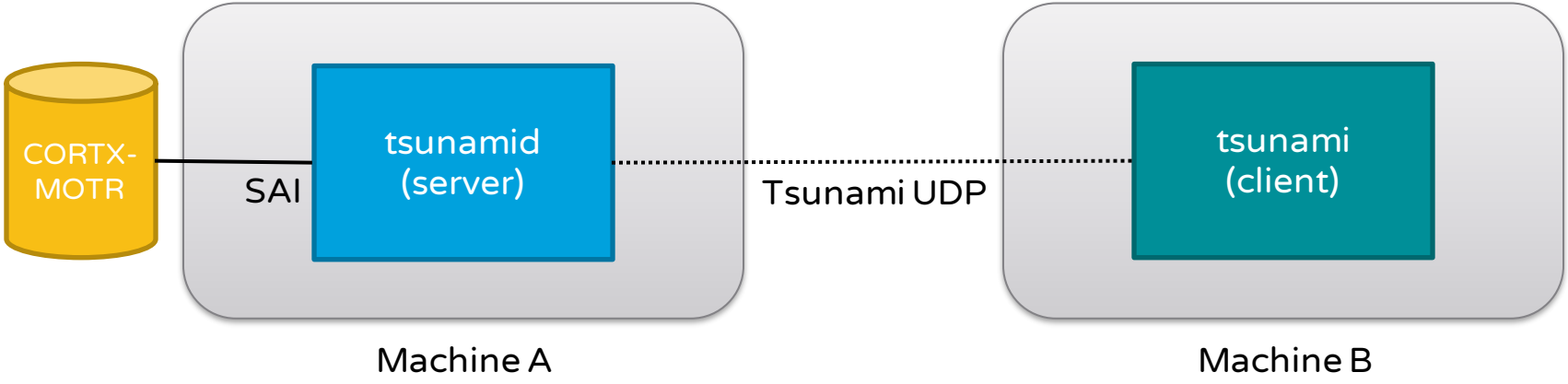
- TCP control stream
  - Client
    - requests file from server
    - client & server negotiate parameters
    - sends re-transmission requests to server & error rate info
  - Server
    - Polls control connection for re-transmission requests before sending new blocks
      - Adjusts inter-packet delay based on reported error rates
      - Transfer rate is controlled through the inter-packet delay
- UDP
  - Server sends data blocks over UDP

Tsunami & CORTX

- There is a Tsunami server (tsunamid) & client (tsunami)
- Client has FTP-like commands

```
tsunami> connect server.domain.com
tsunami> get filename
```

- Note there is no PUT command
- Using Simple Access Interface to integrate with CORTX



Example

- Copying a file out of CORTX

Client

```
[lahiff1@client-21 client]$ ./tsunami
Tsunami Client for protocol rev 20061025
Revision: v1.1 devel cvsbuild 43
Compiled: Sep  3 2021 21:57:51
tsunami> connect client-22
Connected.

tsunami> get aac6c3c3
Receiving data on UDP port 46224
Transfer complete. Flushing to disk and signaling server to stop...
!!!!
PC performance figure : 0 packets dropped (if high this indicates
receiving PC overload)
Transfer duration     : 0.06 seconds
Total packet data     : 0.12 Mbit
Goodput data          : 0.12 Mbit
File data             : 0.12 Mbit
Throughput            : 2.21 Mbps
Goodput w/ restarts   : 2.21 Mbps
Final file rate       : 2.21 Mbps
Transfer mode         : lossless
```

Server

```
[lahiff1@client-22 server]$ ./tsunamid
Block size: 1024
Buffer size: 20000000
Port: 46224
Tsunami Server for protocol rev 20061025
Revision: v1.1 devel cvsbuild 43
Compiled: Oct 29 2021 14:31:51
Waiting for clients to connect.
New client connecting from 172.16.1.21...
Client authenticated. Negotiated parameters are:
Block size: 1024
Buffer size: 20000000
Port: 46224
Request for object: 'aac6c3c3'
Sending to client port 46224
Transmission of aac6c3c3 complete.
Server 1 transferred 16384 bytes in 0.06 seconds (2.1 Mbps)
```

Example

★ Copying a file out of CORTX: specifying different block sizes (default 1 kB)

Client

```
[lahiff1@client-21 client]$ ./tsunami
Tsunami Client for protocol rev 20061025
Revision: v1.1 devel cvsbuild 43
Compiled: Sep  3 2021 21:57:51
tsunami> connect client-22
Connected.

tsunami> set blocksize 16384
blocksize = 16384

tsunami> get b0839a8c
Receiving data on UDP port 46224
            last_interval                    transfer_total
    buffers      transfer_remaining  OS UDP
time          blk    data       rate rexmit     blk    data       rate
rexmit queue  ring     blk   rt_len      err
00:00:00.845  100   0.00M   14.8Mbps   0.0%     100    0.0G   14.8Mbps
0.0%     0     0      28        0        0 --
Transfer complete. Flushing to disk and signaling server to stop...
!!!!
PC performance figure : 0 packets dropped (if high this indicates
receiving PC overload)
Transfer duration     : 0.98 seconds
Total packet data     : 16.00 Mbit
Goodput data          : 16.00 Mbit
File data             : 16.00 Mbit
Throughput            : 16.34 Mbps
Goodput w/ restarts   : 16.34 Mbps
Final file rate       : 16.34 Mbps
```

Tsunami has many other free parameters

- target transfer rate
- max error rate to obtain by rate throttling
- how fast to start throttling the rate
- how fast to recover
- weighting of past error rates in the throttling algorithm
- …

Thank You

andrew.lahiff@ukaea.uk