High speed data networks in CMS

Petr Žejdl (Fermi National Accelerator Laboratory)
André Holzner (University of California, San Diego)

on behalf of the CMS DAQ group
Overview

- Ethernet data concentrator network
  - Fat tree topology
  - Long lasting TCP connections: importance of hash functions

- Infiniband
  - Custom forwarding tables for improved performance
  - Blocking counters
Data Concentration Ethernet Network in CMS

(Fat-Tree)
Data Concentration Network

Timing, Trigger and Control (TTC) front-end distribution system

Detector Front-End Drivers (FED x ~700)

Trigger Throttle System (TTS), Fast Merging Module (FMM)

Input: old FED copper 400 MBs Slink, new FED 4/10 Gbs optical

576 Front-End Readout Optical Link (FEROL-PCIx)

Patch panels

Data to Surface ~ (2 x) 576 x 10 GbE links (5.8 Tbs)

Data Concentration Network Ethernet Fat-Tree

Event Builder 108 x 64 (3.5 Tbs) & Data Backbone

InfiniBand-FDR CLOS-216 network (216 external ports)

Data backbone (10/40 GbE)

BU-FU appliance
- 1 BU (256 GB RAM, 2TB magnetic disks)
- 16 FU nodes
- FU: Dual E5-2670 8 core (2 x 1 GbE)
- FU: Dual Xeon 6 core (2 x 1 GbE)

BU-FU appliance
- 1 BU (256 GB RAM, 2TB magnetic disks)
- 8 FU nodes
- FU: Dual Haswell with 12 cores (10 GbE)

Surface Counting room

Surface

Underground
Data Concentration Network

- Plays a key part in data concentration
  - Sources 10GbE optical links from ~600 FEROLs (10 GbE TCP/IP)
  - Sinks 40GbE optical links to 108 Readout Units (RU)
  - Designed as a Fat-Tree Network with three switching layers

- FEROLs and RUs connected to leaf switches
- 360 Gbit/s between top and middle layer
- 200/280 Gbit/s between leaf and middle layer
Top layer: 1x 40GbE
Middle layer: 4x 40GbE SX1036
Leaf switches: 14x 10/40GbE, 2x 40 GbE
120 copper cables deployed
9x iperf running in parallel:
Connecting to host ru-c2e12-24-01.fbs0v0.cms, port 5201: [  4] 0.00-10.00  sec  39.5 Gbits/sec
Connecting to host ru-c2e12-25-01.fbs0v0.cms, port 5201: [  4] 0.00-10.00  sec  39.6 Gbits/sec
Connecting to host ru-c2e12-26-01.fbs0v0.cms, port 5201: [  4] 0.00-10.00  sec  39.6 Gbits/sec
Connecting to host ru-c2e14-24-01.fbs0v0.cms, port 5201: [  4] 0.00-10.00  sec  39.6 Gbits/sec
Connecting to host ru-c2e14-25-01.fbs0v0.cms, port 5201: [  4] 0.00-10.00  sec  39.6 Gbits/sec
Connecting to host ru-c2e14-26-01.fbs0v0.cms, port 5201: [  4] 0.00-10.00  sec  39.6 Gbits/sec
Connecting to host ru-c2e12-30-01.fbs0v0.cms, port 5201: [  4] 0.00-10.00  sec  39.6 Gbits/sec
Connecting to host ru-c2e12-34-01.fbs0v0.cms, port 5201: [  4] 0.00-10.00  sec  39.6 Gbits/sec
Connecting to host ru-c2e12-35-01.fbs0v0.cms, port 5201: [  4] 0.00-10.00  sec  39.5 Gbits/sec
Connecting to host ru-c2e14-30-01.fbs0v0.cms, port 5201: [  4] 0.00-10.00  sec  39.5 Gbits/sec
DONE?
A LAG (Link Aggregation Group) / port trunking

- Combines a number of physical ports together to make a single high-bandwidth data path

- Uses a load-balancing method (hash function) for packet distribution, usually a combination of
  
  - L2 (MAC Address)
  - L3 (IP address)
  - L4 (TCP/UDP port numbers)
Link Aggregation

- Hashing based on L2/L3/L4 is distributing network flows across the links in the LAG, hash collisions can happen!

- **Without explicit load balancing some links can be congested!**
  - Not good for long-lived data flows/streams

- **Idea: Since DAQ network is static and the actual traffic pattern is known, we can distribute the expected traffic evenly**
  - This will make use of full LAG bandwidth and eliminate any possible collisions
  - We need to know the HASH function! Can we?
• Details of hash functions are usually not disclosed

• A LAG simulation software can be available
  – For a given network flow parameters (MAC, src/dst IP address and/or port number) it produces a LAG logical output port/link

• By using a parameter which can be easily changed (port numbers) a reverse hash table can be created

• Example: LAG with 3 links
  – Link 1: \{1, 3, 4, 5, 6, 11, \ldots\}
  – Link 2: \{7, 8, 9, 10, 18, 20, \ldots\}
  – Link 3: \{2, 12, 13, 15, 16, 17, \ldots\}

• We can force/bind a TCP/IP stream to a particular LAG port with some clever algorithm
Algorithm

- Problem is two fold (Place and Route)
  - Place part
    - Find a good destination (RU) for a given set of FEROL sources (FEDBuilder)
    - Minimize switch crossing
    - Do not exceed capacity of any physical link
  - Route part
    - Find a good source port assignments such as the hash function doesn't have collisions / network traffic don't overlap between LAG links

- Combinatorial problem with factorial time complexity
  - Greedy heuristics
    - for each destination (RU) a rank is calculated based on the network throughput it receives, number of FEROLs and number of hops
    - RU with the lowest rank is selected (placed)
Implementation: DAQ Configurator

- Algorithms implemented in Java, part of DAQ Configurator tool:

<table>
<thead>
<tr>
<th>Switch to switch link occupancy:</th>
</tr>
</thead>
<tbody>
<tr>
<td>sw-eth-c2e23-08-01 -&gt; sw-eth-c2e23-41-01: Sending 280 Gb/s over a 280 Gb/s link, i.e. 100% of the link bandwidth</td>
</tr>
<tr>
<td>sw-eth-c2e24-17-01 -&gt; sw-eth-c2e23-11-01: Sending 200 Gb/s over a 360 Gb/s link, i.e. 56% of the link bandwidth</td>
</tr>
<tr>
<td>sw-eth-c2e24-08-01 -&gt; sw-eth-c2e24-41-01: Sending 200 Gb/s over a 280 Gb/s link, i.e. 71% of the link bandwidth</td>
</tr>
<tr>
<td>sw-eth-c2e23-38-01 -&gt; sw-eth-c2e23-08-01: Sending 200 Gb/s over a 200 Gb/s link, i.e. 100% of the link bandwidth</td>
</tr>
<tr>
<td>sw-eth-c2e24-38-01 -&gt; sw-eth-c2e4-08-01: Sending 160 Gb/s over a 200 Gb/s link, i.e. 80% of the link bandwidth</td>
</tr>
<tr>
<td>sw-eth-c2e23-11-01 -&gt; sw-eth-c2e23-29-01: Sending 80 Gb/s over a 200 Gb/s link, i.e. 40% of the link bandwidth</td>
</tr>
</tbody>
</table>

...
Infiniband Event Building Network
**CMS Infiniband EVB network**

- central part of event builder
  - assembles full events
  - all destinations need to receive from all sources
    - full NxM connectivity needed

- 12 leaf, 6 spine switches
- 36 FDR (56 GBit/s) ports per switch
- 3 links between each leaf/spine pair
- 18 x 12 = 216 external ports
  - ~ 6 Tbit/s bandwidth
Infiniband forwarding

- Infiniband uses **Local Identifiers (LIDs)** for addressing
  - 16 bits
  - assigned by subnet manager (SM) when nodes or SM comes up
  - used in the Local Route Header:

  ![Infiniband Forwarding Diagram](image)

- **differences to other protocols:**
  - does **not** allow strict source routing like Myrinet
    - all packets for a given destination LID go out on the same port on a given switch
    - no hash functions for trunks like for Ethernet
    - but can have **multiple LIDs per network card**
  - 16 bit LID can be seen as having the **same function as the 64 bit Ethernet MAC address**
    - instead of using a content addressable memory like in Ethernet switches, the **linear forwarding table (LFT)** is an array of 12 kBytes
      - address is destination LID
      - content is **output port**
Routing engines

- Subnet manager generates forwarding tables
  - Mellanox InfiniBand switches have an embedded OpenSM
  - OpenSM can also be run on any server connected to InfiniBand network
    - allows for more flexibility
  - each subnet manager is assigned a priority
    - highest priority manages the network
    - lower priority SM are in standby

- Available routing engines in OpenSM:
  - minhop (default)
  - updn
  - dnup
  - ftree
  - file
  - lash
  - dor
  - torus-2QOS
  - dfsssp
  - sssp

  our default for folded Clos network

  our gateway to our own static routing tables
Routing algorithm

- The default fat tree routing algorithm does **not assume anything about the actual traffic pattern**
  - should not matter in theory as long as one does send more than from any of the N sources to any of the M destinations
  - in practice however:
    - our links are **temporarily oversubscribed** (‘bursty traffic’)  
    - we must **wait for the slowest source**
    - **head of line blocking** is most likely an issue

- Idea: since we **know the actual traffic pattern**, we can **distribute the expected traffic evenly**
  - this should **eliminate bottlenecks**
  - make use of links which otherwise would have low utilization
Algorithm

- Problem formulation:
  - **minimize spread** of
    - number of communications over the **spine switches** or
    - number of communications over **the links between leaf and spine switches**
  - etc.

- subject to:
  - do not exceed **capacity** of any link
  - communications to the same destination on a switch must continue over the same links

- Looks like a **combinatorial problem** with factorial time complexity
  - use a **heuristic**:
    - go through all (source → destination) pairs
      - assign route over leaf switch with least occupancy so far
    - if not already constrained by previous routing table entries
Implementation

- Algorithm is implemented in python
  - **fast turnaround** to try out new algorithms
  - algorithm core separated from ‘framework’ code
  - independent of OpenSM
  - **only static** forwarding tables possible
    - no adaptation to **actual throughput** as luminosity decreases with time

- Here be dragons:
  - during early stages of development, managed to bring down the entire Infiniband network, power cycle needed
  - need to assign all forwarding table entries, including those to LIDs of switches
    - diagnostics such as ibqueryerrors will not work otherwise
Performance

from Remi Mommsen’s talk

CMS Preliminary

8 streams per RU
- 48x48 (custom)
- 72x72 (custom)
- 48x48 (default)
- 72x72 (default)

02 Mar 2016

16 (40) % improvement for 72x72 (48x48) system
generation insensitive to actual list of sources and destinations
Finding bottlenecks in the IB network

- Infiniband supports many diagnostic counters
  - PortXmitWait. This provides information on a potential victim of congestion. It is the number of ticks during which the port had data to transmit but was unable to because of a lack of credits or a lack of arbitration.

- Some confusion how to interpret the value
  - initially did not know about the ‘multiplier’ register (factor 32…)
  - checked with manufacturer about the exact length of a ‘tick’ (depends on the signalling speed)

- These counters can be accessed over the network
  - `ibqueryerrors` allows to retrieve them from all ports in the network in one go

- we now periodically store PortXmitWait counter values in a database
blocking counters example (72x72 setup)

- **source PCs output**
- **spine switch back to leaf switch**
- **leaf switch to spine switch**
- **leaf switch to destination PC**
Summary / Outlook

- **Ethernet:**
  - installed a **fat tree** topology for **data aggregation**
    - gives us **additional flexibility** in case of PC failure
  - implemented a **configuration dependent routing**
    - knowledge of LAG hash functions vital

- **Infiniband:**
  - custom routing gives us **better use of the available capacity**
  - **diagnostic counters** are a useful tool to **identify bottlenecks** in the network
  - routing can potentially be improved by taking into account **actual fragment sizes**
  - use of **multiple virtual lanes** may reduce head of line blocking