Systems (th)at Scale for L* Uni Seminars

Jon Crowcroft,
http://www.cl.cam.ac.uk/~jac22
1. **New Cloud OS to meet new workloads**
   - Includes programming language
   - Collabs incl REMS (w/ P. Gardner/Imperial)

2. **New Data Center structure**
   - Includes heterogeneous h/w
   - Collabs incl NaaS(Peter Pietzuch Imperial)
   - Trilogy (Mark Handley et al UCL)

3. **New Networks (for data centers&)**
   - To deal with above😊
What not talking about

- Security
  - (we do that – had another workshop)

- Data
  - Hope Ed folks will!

- Scaling Apps
  - Oxford

- Languages for Apps
  - Ed++
1. Cloud OS

- Unikernels (Mirage, SEL4, ClickOS)

---

(a) Containers, e.g., Docker.
(b) Picoprocesses, e.g., Drawbridge.
(c) Unikernels, e.g., MirageOS.

Figure 2: Contrasting approaches to application containment.
Unikernels in OCaml

- But also Go, Scala, Rust etc
- Type safety->security, reliability
- Apps can be legacy or in same languages

Figure 1: Jitsu architecture: external network connectivity is handled solely by memory-safe unikernels connected to general purpose VMs via shared memory.
Data Centers don’t just go fast

- They need to serve applications
  1. Latency, not just throughput
  2. Face users
     1. Web, video, ultrafast trade/gamers
     2. Face Analytics...
  3. Availability & Failure Detectors
  4. Application code within network
  5. NIC on host or switch – viz
Industry (see pm 😊)

Azure
http://conferences.sigcomm.org/
sigcomm/2015/pdf/papers/keynote.pdf

Facebook:
http://conferences.sigcomm.org/
sigcomm/2015/pdf/papers/p123.pdf

Google:
http://conferences.sigcomm.org/
sigcomm/2015/pdf/papers/p183.pdf
2. Deterministic latency bounding

- Learned what I was teaching wrong!
- I used to say:
  - Integrated Service too complex
    - Admission&scheduling hard
  - Priority Queue can’t do it
    - PGPS computation for latency?
- I present Qjump scheme, which
  - Uses intserv (PGPS) style admission ctl
  - Uses priority queues for service levels
  - http://www.cl.cam.ac.uk/research/srg/netos/qjump/
Data Center Latency Problem

- Tail of the distribution,
  - due to long/bursty flows interfering
- Need to separate classes of flow
  - Low latency are usually short flows (or RPCs)
  - Bulk transfers aren’t so latency/jitter sensitiv
Data Center Qjump Solution

- In Data Center, not general Internet!
  - can exploit topology &
  - traffic matrix &
  - source behaviour knowledge
- Regular, and simpler topology key
- But also largely “cooperative” world...
Hadoop perturbs time synch

- Clock offset [μs]
- Time since start [sec]
- PTPd, ptpd only, ptpd with Hadoop

Timeline of PTP synchronization offset.

- Median and 99th percentile latencies observed
- Table 1: Median and 99th percentile request latency degrades by 1.5×

Figures:
- Figure 1: Timeline of PTP synchronization offset.
- Figure 2: CDF of memcached request latencies.
- Figure 3: CDF of Naiad barrier sync. latencies.
Hadoop perturbs memcached

Motivating experiments: Hadoop traffic interferes with other latency-sensitive applications (PTPd, memcached, etc.).

We begin by showing that shared switch queues are the primary source of network interference. We then quantify the extent to which network interference impacts application-observable metrics of performance.

When running two instances of a throughput-intensive bulk transfer application by running Hadoop on a network shared with Hadoop, the 99th percentile latency increases by over 16× compared to the unshared case.

Table 1 shows the results of experiments ranging from one host, idle network, to two hosts, shared switch queue. With Hadoop running, the 99th percentile latency increases by over 500µs at the 99th percentile to perform a four-way barrier synchronization benchmark (provided by the Naiad authors) with and without Hadoop running. Figure 1a shows a timeline of PTP synchronization offset. One host, idle network, shows the distribution of Naiad synchronization latencies. Barrier synchronization causes PTPd to temporarily fall 200–500µs out of synchronization from a time server to machines on a local network. In Figure 1b, we show a timeline of PTPd synchronizing a host clock on both an idle network and a network shared with Hadoop. With Hadoop running, the median and 99th percentile latencies observed are worse than on an idle network. With interference, this grows to around 500µs at the 99th percentile to perform a four-way barrier synchronization. With interference, this grows to around 500µs at the 99th percentile to perform a four-way barrier synchronization. Even worse, approximately 1 in 6,000 requests take over 200ms to complete.

Figure 1c shows the CDF of memcached request latencies. Request latencies for memcached only are distributed from 779µs to 1196µs. Even worse, approximately 1 in 6,000 requests take over 200ms to complete. Likely because packet loss triggers the TCP minRTO timeout.

Figure 2 shows the same queue in the same network switch. To assess the impact of interference in each of these situations, we enumerate two instances of a latency-sensitive RPC application using `iperf` with and without Hadoop running.

In iterative computations, Naiad’s performance depends on low-latency barrier synchronization between worker nodes. To test Naiad’s sensitivity to network interference, we execute a state synchronization benchmark (provided by the Naiad authors) with and without Hadoop running. Figure 1a shows a timeline of Naiad state synchronization from a time server to machines on a local network. In Figure 1b, we show a timeline of Naiad state synchronization; the distribution of request latencies on an idle network and a network shared with Hadoop. With Hadoop running, the median and 99th percentile latencies observed are worse than on an idle network. Hadoop’s shuffle phases causes queue congestion; the effect is worst when applications share a work sharing. Although any sharing situation results in interference, the effect is worst when applications share a work sharing.

Different applications use the network in different ways. We begin by showing that shared switch queues are the primary source of network interference. We then quantify the extent to which network interference impacts application-observable metrics of performance.

Clock offset [µs]

Timeline of PTP synchronization offset.

Request latency [µs]

CDF of memcached request latencies.
Network interference may occur at various places on the network path. Applications may share ingress or egress to a network switch, share common egress ports, or share sources, as demonstrated in Figure 1.

Table 1: percentiles of request latencies observed with various degrees of network interference. Table 2 shows the results of experiments running two instances of a throughput-intensive bulk transfer application by running ping and memcached and Naiad) on a network shared with Hadoop (details in §1.5).

To demonstrate the degree to which network interference affects different applications, we run three representative latency-sensitive applications (PTPd, memcached and PTPd with Hadoop) on a network shared with Hadoop. With Hadoop running, the clock offset grows by over 16 ms compared to the unshared case.

Motivating experiments: Hadoop traffic interferes with Naiad's sensitivity to network interference, we execute a computation, Naiad's performance depends on low-latency barrier synchronization. With interference, this grows to around 500 µs at the 99th percentile to perform a four-way barrier synchronization. With perturbs Naiad's performance.

Figure 1: Timeline of PTPd synchronization offset. Table 1: CDF of memcached request latencies. Figure 2: Bar synch. latency CDF. We begin by showing that shared switch queues are the primary source of network interference. We then quantify the extent to which network interference impacts application-observable metrics of performance.

Hadoop perturbs Naiad

![Graph showing the comparison between Naiad only and Naiad with Hadoop](chart.png)
Qjump - two pieces

1. At network config time
   - Compute a set of (8*) rates based on
   - Traffic matrix & hops => fan in (f)

2. At run time
   - Flow assigns itself a priority/rate class
   - Subject it to (per hypervisor) rate limit

* 8 arbitrary - but often h/w supported😊
Memcached latency redux w/ QJ
QJ naiad barrier synch latency redux

![Graph showing latency distribution](image)

- QJ fixes Naiad barrier synchronization

---

**Figure 4:** Network topology of our test-bed.
In the data-mining workload, 85% of all flows are transferred in a time window of less than 100kB. As described in §6.5, we use a combination of two approaches: one is to tune the levels to different applications, and the other is to apply any rate-limiting (scheduling) to them. This approach results in sub-optimal behavior. A combined approach, where we apply rate-limiting to flows in a specific mix of applications. For some applications, their performance at different throughput factors is less straightforward. Memcached is an example of such an application. It needs low request latency variations as well as reasonable request throughput. Figure 9 shows memcached's request throughput and latency as a function of rate-limiting. Peak throughput is reached when the request latency also stabilizes. Hence, a rate-limit of 5Gb/s yields a throughput factor of 1.9 at a rate allocation of around 5Gb/s. At the same point, DCTCP offers a bounded latency level at throughput restrictions. To convert this to a throughput factor, we get a throughput factor of 4.6. We can therefore choose a throughput factor of 5Gb/s, which is less than 15% of the rate allocation limit. Figure 10 shows the normalized flow completion times in a 144-host simulation (1 is ideal): QJump, DCTCP, and TCP for small flows, while DCTCP regulates the interactions between large flows. It is clear that they perform best at guaranteed latency and small flows, while DCTCP is 63% worse than pFabric's (Fig. 9f). On the data-mining workload, 85% of all flows are transferred in a time window of less than 100kB. It is interesting to note that DCTCP outperforms pFabric by up to 20% at high load, but loses 15% at low load (Fig. 9c).
### Big Picture Comparison - Related work...

<table>
<thead>
<tr>
<th>System</th>
<th>Commodity</th>
<th>Unmodified</th>
<th>Coord.</th>
<th>Flow</th>
<th>Bounded</th>
<th>Implemented</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>hardware</td>
<td>protocols</td>
<td>free</td>
<td>deadlines</td>
<td>latency</td>
<td></td>
</tr>
<tr>
<td>Deployable</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pause frames</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ECN</td>
<td>✓*, ECN</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>DCTCP [1]</td>
<td>✓*, ECN</td>
<td>✓*, x</td>
<td>✓</td>
<td>x</td>
<td>x</td>
<td>✓</td>
</tr>
<tr>
<td>Fastpass [29]</td>
<td>✓</td>
<td>✓, module</td>
<td>✓</td>
<td>x</td>
<td>x</td>
<td>✓</td>
</tr>
<tr>
<td>EyeQ [22]</td>
<td>✓*, ECN</td>
<td>✓, x</td>
<td>✓</td>
<td>x</td>
<td>x</td>
<td>✓</td>
</tr>
<tr>
<td>QJUMP</td>
<td>✓</td>
<td>✓</td>
<td>✓, module</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Not deployable</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D²TCP [33]</td>
<td>✓*, ECN</td>
<td>✓*, x</td>
<td>✓</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>HULL [2]</td>
<td>x</td>
<td>✓*</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>✓</td>
</tr>
<tr>
<td>D³ [35]</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>PDQ [17]</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>✓</td>
<td>x</td>
</tr>
<tr>
<td>pFabric [3]</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>✓</td>
<td>✓*</td>
<td>x</td>
</tr>
<tr>
<td>DeTail [37]</td>
<td>x</td>
<td>✓</td>
<td>x</td>
<td>x</td>
<td>✓</td>
<td>x</td>
</tr>
<tr>
<td>Silo [21]</td>
<td>✓</td>
<td>✓</td>
<td>✓*</td>
<td>✓</td>
<td>✓, SLAs</td>
<td>✓</td>
</tr>
<tr>
<td>TDMA Eth. [34]</td>
<td>✓<em>, ✓</em></td>
<td>✓</td>
<td>✓*</td>
<td>✓</td>
<td>x</td>
<td>✓</td>
</tr>
</tbody>
</table>
Failure Detectors

- 2PC & CAP theorem
- Recall CAP (Brewer’s Hypothesis)
  - Consistency, Availability, Partitions
  - Strong & weak versions!
  - If have net & node deterministic failure detector, isn’t necessarily so!
- What can we use CAP-able system for?
2b 2PC throughput with and without QJump

![2b 2PC throughput with and without QJump](image-url)
Consistent, partition tolerant app?

- Software Defined Net update!
  - Distributed controllers have distributed rules
  - Rules change from time to time
  - Need to update, consistently
  - Need update to work in presence of partitions
    - By definition!
  - So Qjump may let us do this too!
3. Application code -> Network

- Last piece of data center working for application
- Switch and Host NICs have a lot of smarts
  - Network processors,
  - like GPUs or (net)FPGAs
  - Can they help applications?
  - In particular, avoid pathological traffic patterns (e.g. TCP incast)
E.g. shuffle phase in map/reduce
- Does a bunch of aggregation
- \((\text{min, max, ave})\) on a row of results
- And is cause of traffic “implosion”
- So do work in stages in the switches in the net (like merge sort!)

- Code very simple
- Cross-compile into switch NIC cpus
Other application examples

- Are many ...
- Arose in Active Network research
  - Transcoding
  - Encryption
  - Compression
  - Index/Search
- Etc etc
Need language to express these

- Finite iteration
- (not Turing-complete language)
- So design python- with strong types!
- Work in progress in NaaS project at Imperial and Cambridge...
Cloud Computing Isn’t For Everything!

Latency effect on facial recognition

Remote Processing
• “being fast really matters...half a second delay caused a 20% drop in traffic and it killed user satisfaction” - Marissa Mayer @ Web 2.0 (2008)
• “A millisecond decrease in a trade delay may boost a high-speed firm's earnings by about 100 million per year” – SAP, 2012
• “It’s simply not appropriate to just drag and drop our databases into a cloud platform” – Thomas Kadlec, Tesco, 2015

Local Processing

Source: Glimpse project, MIT, 2014
Tiny Terabit Datacentre
An End-Host Networked-Server Architecture

- High Performance
- Resource Isolation
- Flexible Implementation
- Predictable Latency
- Low Latency Interconnect
- Affordable

Centralized network fabric
Distributed network fabric
Multi-controller memory
Centralized memory
Networks, Interfaces and Transports for Rack-Scale Operating Systems
Conclusions/Discussion

- Data Center is a special case!
- It's important enough to tackle
  - We can hard bound latency easily
  - We can detect failures and therefore solve some nice distributed consensus problems
  - We can optimise applications pathological traffic patterns
  - Integrate programming of net&hosts
  - Weird new h/w...
- Plenty more to do...