Nottingham/Horizon

Seminar

27th January 2016

Cybersecurity and network measurement problematic in so many ways

http://www.cl.cam.ac.uk/~jac22

http://metrics-itn.eu/

Team Jon

outline

- Three parts

1. The Internet at large
2. Measurement data is big data – what's hard?
3. Measuring things is not neutral – why?

Part 1 – The Internet is Big

- Not just size but complexity
  - Graph data – has billions nodes & edges
    - Hypergraph – edges have multiple meanings
    - Sparse, and dynamic
    - Topology, topography, policy at IP level
    - Authorship, ownership, ACLs at Web level
  - So simple questions (clusters, cliques, hubs, etc)
    - Are computationally very expensive (O(m^3/2))

Part 2- Big Data

1. The “Big” in Big Data is relative
2. Big Social Data
3. Big Science Big Data
4. Big Private Data
5. Big Bad Data

Alan Turing Institute for Data Science

http://www.turing.ac.uk/

“Big” Data is Relative

- Social Sciences
- Natural Sciences
- Computational Sciences

Social Sciences

- Big > 12, or “Complete”
- E.g. all of a family, town, country, world
- 10 Billion is not really big
  - if you’re just counting
- Problem is Ground Truth
  - E.g. where did you get your data from?

Social Big Data problems

- Bias
  - Sample Bias
  - Recruitment Bias
  - Survivor Bias
- E.g. Data from smart phones
  - Who has smart phones?
    - What type? (MAC addr no longer tells ☺)
  - WEIRD
    - white educated industrialized rich democratic

Social Graph Data

- Even when you have “large” data
  - Beware, McSherry et al, results

    http://www.frankmcsherry.org/graph/scalability/cost/2015/01/15/COST.html
  - See annex 1 slides…
  - But also ground truth etc etc
- And aforesaid sample/ground truth questions

Natural Science Data

- Particle Physics: LHC/CERN
  - 600M events/sec
  - 10Gbps
  - Mostly noise☺
- Square Kilometer Array
  - 10^15 bps (petabit per sec)
  - Trickier = 100* the whole internet ☹
- Lesson - **they** *will* build big enough processing

Computational Science

- Complexity….Big Bad Data
- Genetics/Epigenetics/Phenomics
  - Interdependence within data –
  - poster child e.g. is protein folding
  - Complexity in model is exponential
  - What hope?
- Lesson:- people will do approximation algo

Physics/Chem/Bio

- Use HPC clusters/rack scale systems
  - Tighter memory interconnect (e.g. Cray)
  - Very very large, fast RAM
  - multiple terabytes today
  - Vector processor support
- Lesson: Not much use for us…or is it?

Private Data

- Much social data is PII
  - Even meta data is PII
  - Protect “big” data by AAA
  - Anonymize? Very hard, especially graphs
  - Inference on nodes easy
- Re-identification is almost trivial
  - E.g. fb, yellowcab, medicare
  - Via public diary, postcode, other sources
- DiffPriv – works, but care still needed
- Homomorphic Cryptography – tbd!!!
- Lesson:- Not a solved problem, access control vital

Public Health

- Aside from IoT, PH is biggest valid use of PII
  - On negative side, privacy crucal&legal
  - On positive side, few genuine researchers, so
  - AAA&Diff Priv work pretty well
- Quantified Self + Wellbeing/fitness already…
- Fitbit, food diaries etc
- Lesson – good motives but mission creep

Big Data processing tools

- Aside from Hadoop,
  - Apache's Spark Streaming and Graphx
  - R
  - Naiad (unsupported for now)
  - Write your own ☺
- Also care about data center network
  - Latency bounds improve performance
  - See annex 2 slides

Big Analytics companies

- Google, facebook
  - See OpenStack and Datacenter Networking (Yongguang Zhang) later….
- Run on specialized data centers
  - Non standard interconnects
    - (clos nets)
  - Non standard protocols
    - IP routing doesn't scale (l2 bridge+vpn++)
    - TCP hacks…
    - Rdma (microsoft)

google

The Google Stack

Source: Malte Schwarzkopf. "Operating system support for warehouse-scale computing". PhD thesis. University of Cambridge Computer Laboratory (to appear), 2015, Chapter 2.

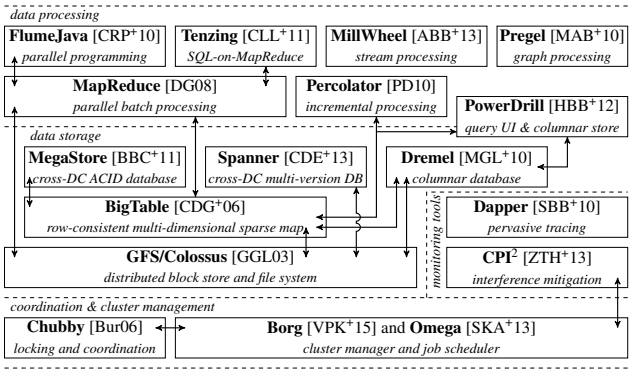


Figure 1: The Google infrastructure stack. I omit the F1 database [SOE+12] (the back-end of which was superseeded by Spanner), and unknown front-end serving systems. Arrows indicate data exchange and dependencies between systems; simple layering does *not* imply a dependency or relation.

In addition, there are also papers that do not directly cover systems in the Google stack:

- An early-days (2003) high-level overview of the Google architecture [BDH03].
- An extensive description of Google's General Configuration Language (GCL), sadly with some parts blackened [Bok08].
- A study focusing on tail latency effects in Google WSCs [DB13].
- Several papers characterising Google workloads from public traces [MHC+10; SCH+11; ZHB11; DKC12; LC12; RTG+12; DKC13; AA14].
- Papers analysing the impact of workload co-location [MTH+11; MT13], hyperthreading [ZZE+14], and job packing strategies on workloads [VKW14].

facebook

The Facebook Stack

Source: Malte Schwarzkopf. "Operating system support for warehouse-scale computing". PhD thesis. University of Cambridge Computer Laboratory (to appear), 2015, Chapter 2.

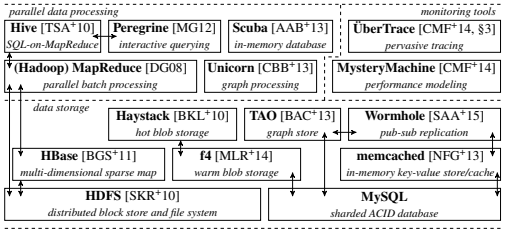


Figure 1: The Facebook infrastructure stack. I omit front-end serving systems about which details are unknown. Arrows indicate data exchange and dependencies between systems; simple layering does *not* imply a dependency or relation.

In addition, there are several papers that do not directly cover systems in the Facebook stack, but describe workloads, techniques or data centre hardware:

- Descriptions of the physical design of Facebook's server machines as of 2011 [FHL+11] and data centre network architecture as of 2013 [FA13].
- Another paper on the HBase back-end for Facebook messages [ABC+12] and a measurement paper looking at the HDFS-level usage patterns of this HBase deployment [HBD+14].
- Papers on the use of erasure codes in HDFS at Facebook [RSG+13; SAP+13; RSG+14].
- Several papers analysing the Facebook memcached workload [AXF+12] and evaluating new sampling strategies to improve hit rates in memcached [LLD+13].
- A study of Facebook's wide-area photo caching infrastructure [HBR+13].
- A description of how Facebook uses shared memory to persist in-memory state across restarts of Scuba server processes [GCG+14].
- The HipHop Virtual Machine (HHVM) is a JIT compiler and runtime for PHP code heavily used in front-end page generation [AEM+14]. Previously, Facebooke used a source-to-source compiler (also called "HipHop", HPHPc) to transform PHP into semantically equivalent C++ code that can be compiled into native code [ZPY+12].

More subtle stuff

- Deep learning
- ML using neural nets, etc
  - May be amenable to other non standard h/w
    - Not transparent or even explanatory?
  - Some say quantum computing
    - Others put that in doubt…
- Lesson:- AI is ML that doesn't work, ***yet***

Part 3 - Three Use Cases

In order of increasing badness:

- Maps
- FluPhone
- Censorship

Use Case #1: Crowd Sourced Net Atlas

- Carna Botnet
  - Used to measure net from 420,000 vantage points
  - Used default password exploit
  - Illegal in most countries
    - See “Internet census 2012: port scanning/0 using insecure embedded devices”
  - Pass “Does no harm” test?
    - Technically yes & no (bandwidth costs)
    - Reputationally no

Use Case #1 continued

- Was it useful?
  - A bit
  - But alternatives exist
  - CAIDA & Internet Atlas Projects
- Is it dangerous?

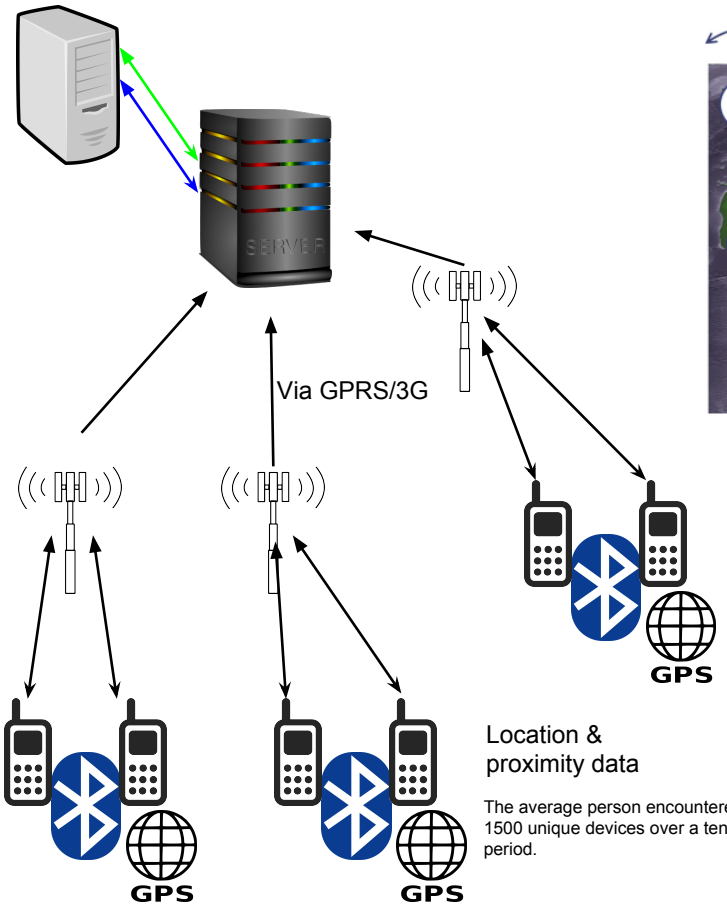
  Gives an open example of an exploit

  Possibly – shows where to attack net hubs

Use Case #2: FluPhone

- Goal to collect encounter data
  - during H1/N1 influenza epidemic
  - Get SIR parameters early
  - Find other features of epidemic
  - Vector, age/gender effects, herd immunity
  - http://www.cl.cam.ac.uk/research/srg/netos/projects/archive/fluphone/
- At start of epidemic, mortality was high
  - Privacy not an issue (notifiable disease)?
  - But medical ethics committee:
  - Weren't allowed to collect on children!
  - Bad, as they are a key mix part of flu spreading!

Use Case #2 continued



Location &
proximity data

The average person encountered over 1500 unique devices over a ten day period.

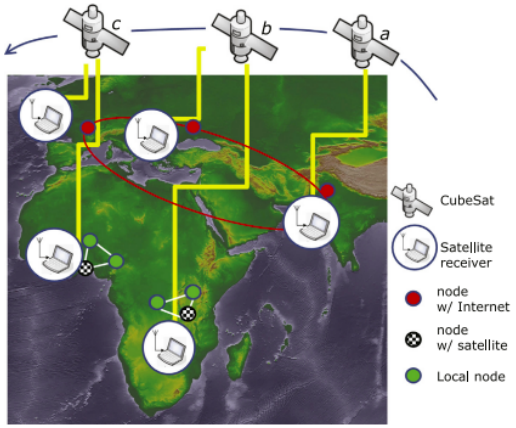


Fig. 6. Use of CubeSat for data collection from Africa.

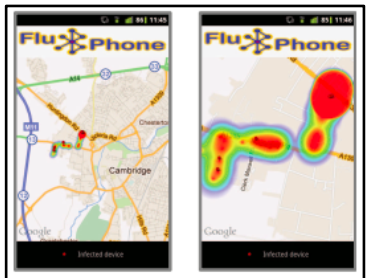


Figure 4: Virtual Disease showing Infection Map

Use Case #3: Censorship

- "Encore: Lightweight Measurement of Web Censorship with Cross-Origin Requests"
  http://conferences.sigcomm.org/sigcomm/2015/pdf/reviews/226pr.pdf
- lesson in Do's &Don'ts
  - of ethical measurement
  - Methodologically
- Idea: cause browser visiting innocent site A
- To be redirected to "is it censored, site B"

Use case #3 continued

What could possibly go wrong, part 1?

1. If you are in a dangerous country and your browser visits a censored site, the excuse
2. “I didn’t click on that” doesn’t help you from being arrested and tortured
3. We know dangerous countries have logging firewalls to implement censorship
4. E.g. Bluecoat technology illegally shipped to Syria, Iran, Russia etc etc

Use case #3 continued

What could possibly go wrong, part 2?

- The ACLs will rapidly be updated
  - To block the site A (redirector script site)
  - Or the script pattern itself
  - Rendering the experiment useless.
- Meanwhile, other people have already done successful experiments in any case, e.g.
  - Censorship in the Wild: Analyzing Internet Filtering in Syria, doi>10.1145/2663716.2663720
  - *And did no harm*

And another thing

- Interference is a bad thing
  - In today's internet (of things), s/w is fragile
  - You don't know what a device is (for)
  - E.g. ipad for reading email
  - Might also be car dashboard (Tesla)
  - You change library (e.g. random # gen)
  - Might crash car…or open it up to hackers
  - Who crash car. loss of privacy -> loss of life

Future is interesting

- Lots to do, lots ***not*** to do.
- Interesting/diverse and useful
  - But also care needed
  - Making more haystacks to find less needles…
  - Medical ethics overly strict
  - Advertising ethics underly strict
- Cybersecurity? You work it out…
  - Better not have sample bias or inexplicable ML
  - Please map the examples I gave onto cybersec questions
  - And see what would be useful,
  - and what would ***be counter-productive***
- Excellent careers right now for CS+X
  - For X=science, commerce, math/stats

Questions?

- I'm happy to repond to followups.

jon.crowcroft@cl.cam.ac.uk

Annex 2 slides on Graph Processing

By

J Crowcroft from work by Malte Schqarzkopf & Frank McSherry

Computer Laboratory & Unafilliated

University of Cambridge

tl;dr #1

- Network speed may not matter with a Spark based stack, but it does matter to higher performance analytics stacks, and for graph processing especially.
- By moving from a 1G to a 10G network, we see a 2x-3x improvement in performance for timely dataflow.

tl;dr #2

- A well balanced distributed system offers performance improvements even for graph processing problems that fit into a single machine;
- running things locally isn't always the best strategy

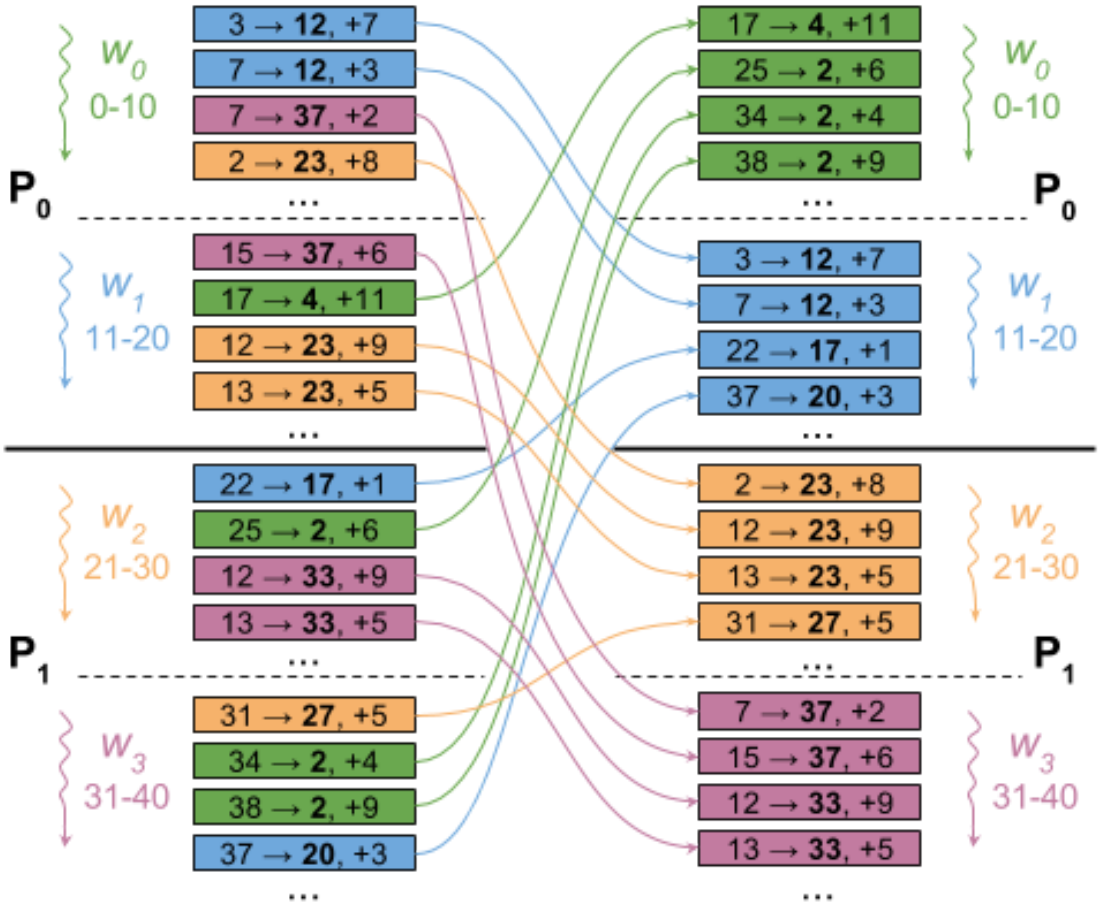
tl;dr #3

- PageRank performance on GraphX is primarily system bound. We see a 4x-16x performance increase when using timely dataflow on the same hardware, which suggests that GraphX (and other graph processing systems) leave an alarming amount of performance on the table

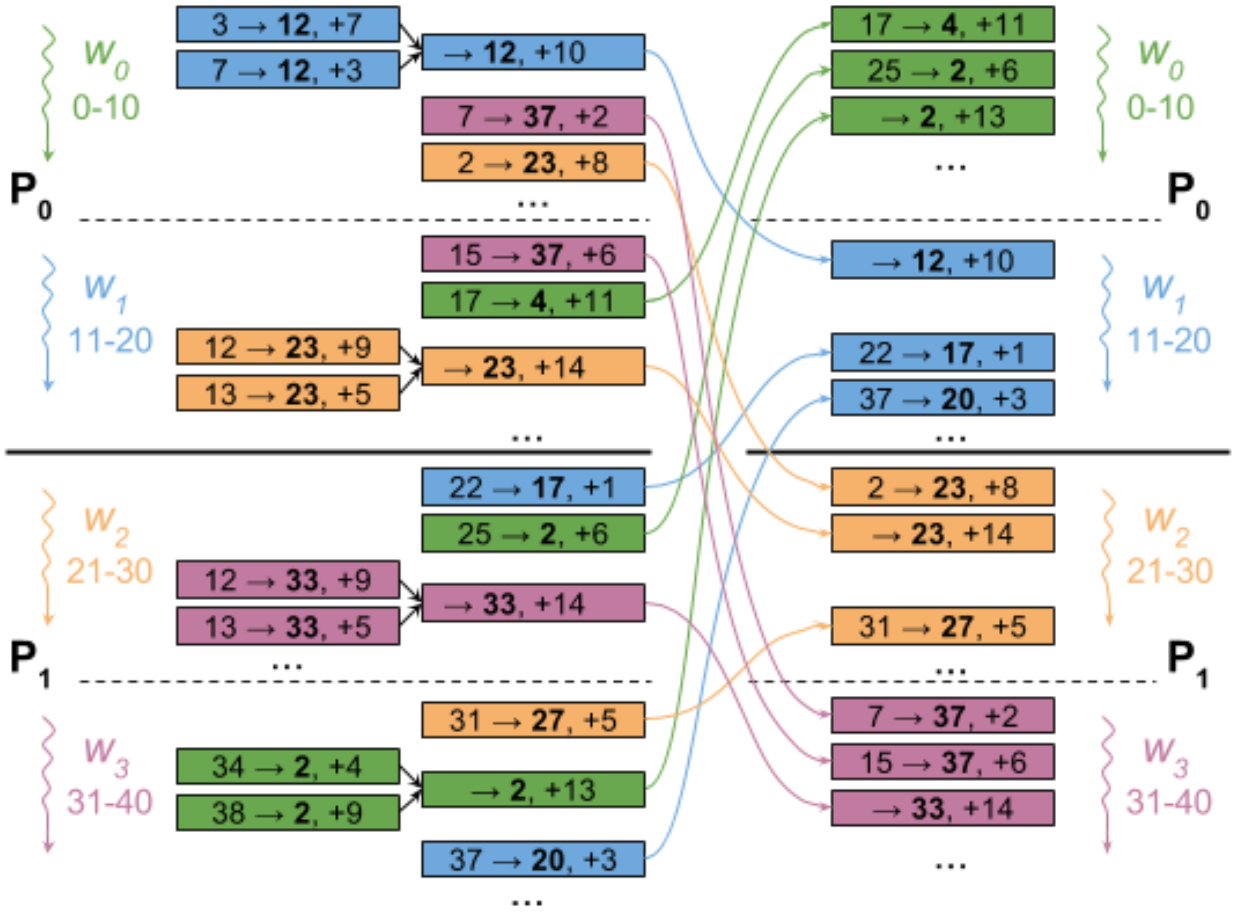
PageRank in Rust

```
fn pagerank(graph: &G: Graph, vertices: usize, alpha: f32)
{
    // mutable per-vertex state
    let mut src = vec![0f32; vertices];
    let mut dst = vec![0f32; vertices];
    let mut deg = vec![0f32; vertices];


    // determine vertex degrees
    for (x,_) in graph.edges() { deg[x] += 1f32; }

    // perform 20 iterations
    for _iteration in (0 .. 20) {

        // prepare src ranks
        for vertex in (0 .. vertices) {
            src[vertex] = alpha * dst[vertex] / deg[vertex];
            dst[vertex] = 1f32 - alpha;
        }

        // do the expensive part
        for (x,y) in graph.edges() { dst[y] += src[x]; }
    }
}
```

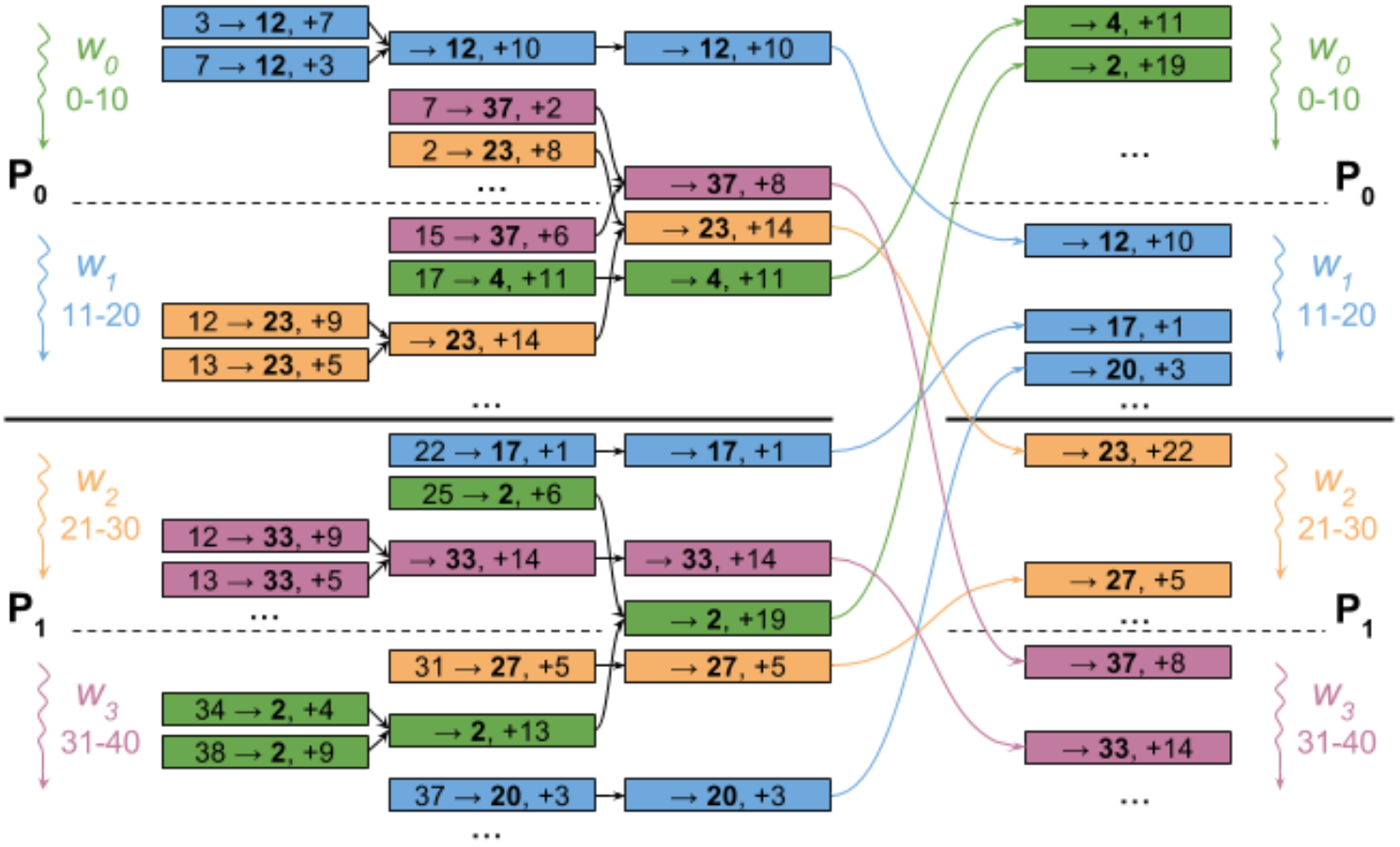
Impl #1: Send everything



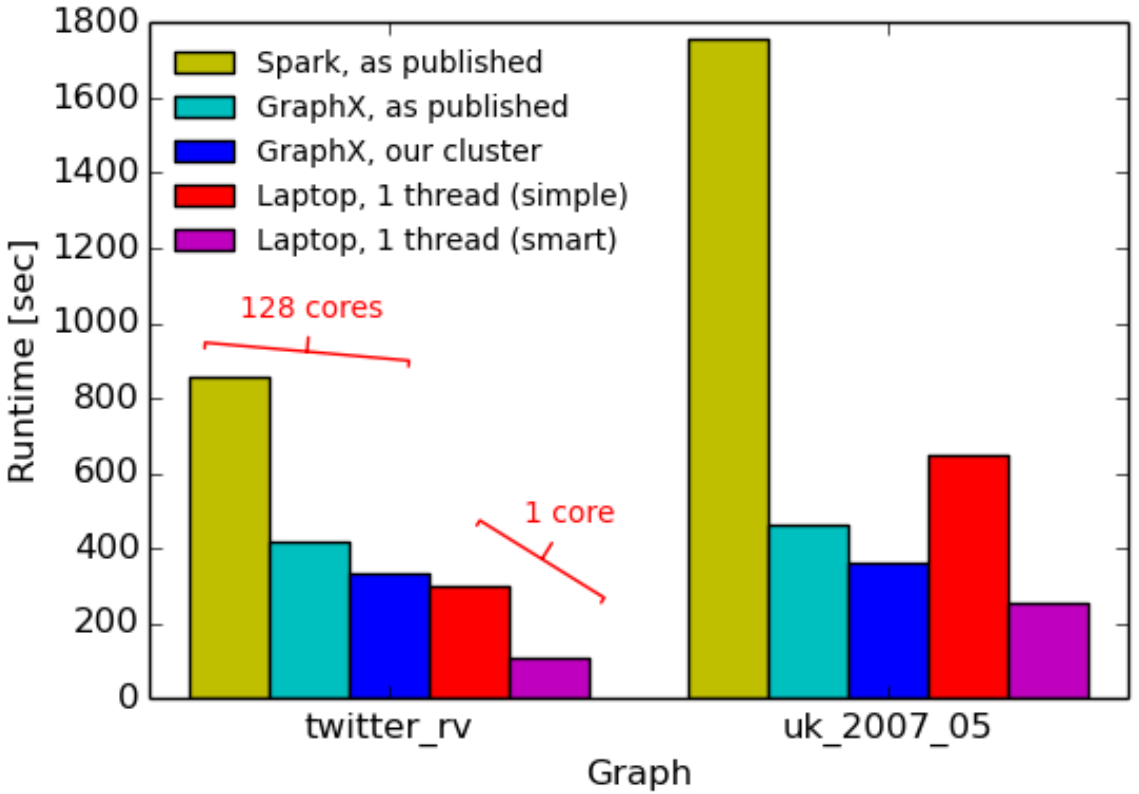
Impl #2: Worker-level aggregation



Impl #3: Process-level aggregation



Some Baseline figures

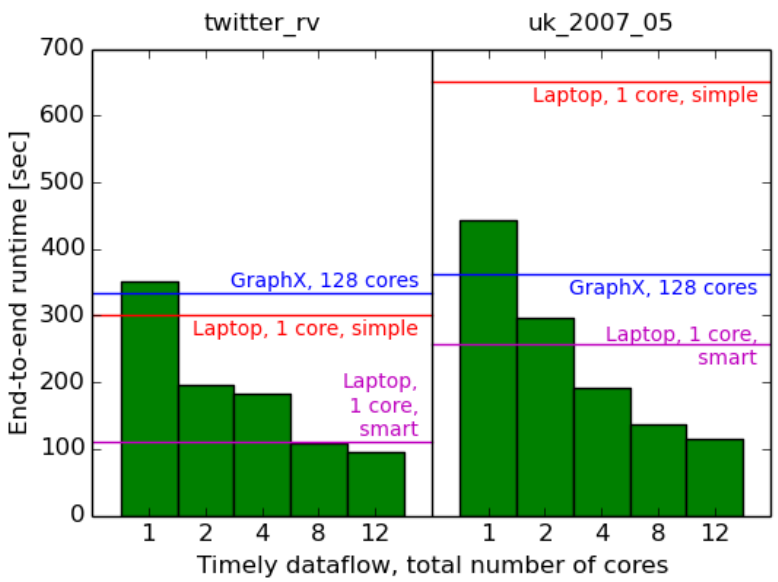


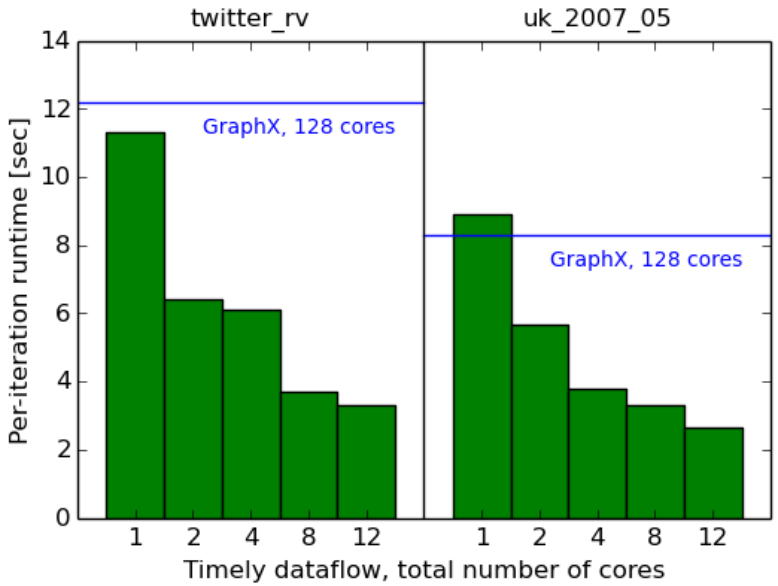
Twenty pagerank iterations, baseline measurements.

System

System	source	cores	twitter_rv	uk_2007_05
Spark	GraphX paper (https://www.usenix.org/system/files/conference/osdi14/osdi14-paper-gonzalez.pdf)	16x8	857s	1759s
GraphX	GraphX paper (https://www.usenix.org/system/files/conference/osdi14/osdi14-paper-gonzalez.pdf)	16x8	419s	462s
GraphX	measured on our cluster	16x8	334s	362s
Single thread (simpler)	COST paper (https://www.usenix.org/conference/hotos15/workshop-program/presentation/mcsherry)	1	300s	651s
Single thread (smarter)	COST paper (https://www.usenix.org/conference/hotos15/workshop-program/presentation/mcsherry)	1	110s	256s

Timely dataflow impl

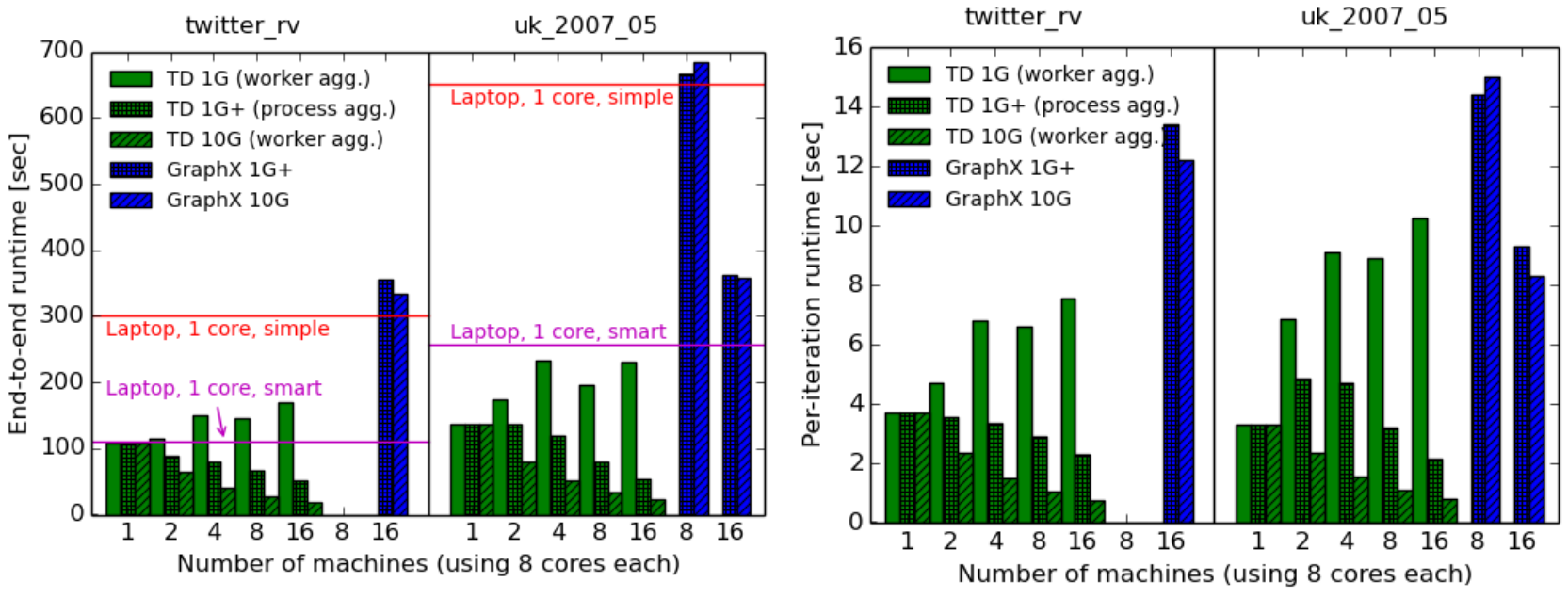




Twenty pagerank iterations on one machine, multiple threads.

System	cores	twitter_rv	uk_2007_05
Timely dataflow	1	350.7s (11.33s)	442.2s (8.90s)
Timely dataflow	2	196.5s (6.39s)	297.3s (5.67s)
Timely dataflow	4	182.4s (6.12s)	192.0s (3.78s)
Timely dataflow	8	107.6s (3.70s)	137.1s (3.29s)
Timely dataflow	12	95.0s (3.32s)	114.5s (2.65s)

Now you can have multiple …



Conclusions 1

- As we have seen, the three implementations (GraphX and the two timely dataflow ones) have *different bottleneck resources*.
- GraphX does more compute and is CPU-bound even on the 1G network, whereas the leaner timely dataflow implementations become CPU-bound only on the 10G network.
- Drawing conclusions about the scalability or limitations of either system based on the performance of the other is likely misguided.

Conclusions 2

- Fast 10G networks *do* help reduce reduce the runtime of parallel computations by significantly more than 2-10%: we've seen **speedups up to 3x going from 1G to 10G**.
- However, the structure of the computation and the implementation of the data processing system must be suited to fast networks, and different strategies are appropriate for 1G and 10G networks.
- For the latter, being less clever and

Conclusions 3

- Distributed data processing makes sense even for graph computations where the graph fits into one machine.
- When computation and communication are overlapped sufficiently, **using multiple machines yields speedups up to 5x** (e.g., on twitter_rv, 1x8 vs. 16x8). Running everything locally isn't necessarily faster.

Conclusions 4

- Can make PageRank run **16x faster per iteration using distributed timely dataflow** than using GraphX (from 12.2s to 0.75s per iteration).
- This tells us something about how much scope for improvement there is even over numbers currently considered state-of-the-art in research!

Annex 2 - Systems (th)at Scale – reducing latency in data center network

Jon Crowcroft,

http://www.cl.cam.ac.uk/~jac22

Cloud, Data Center, Networks

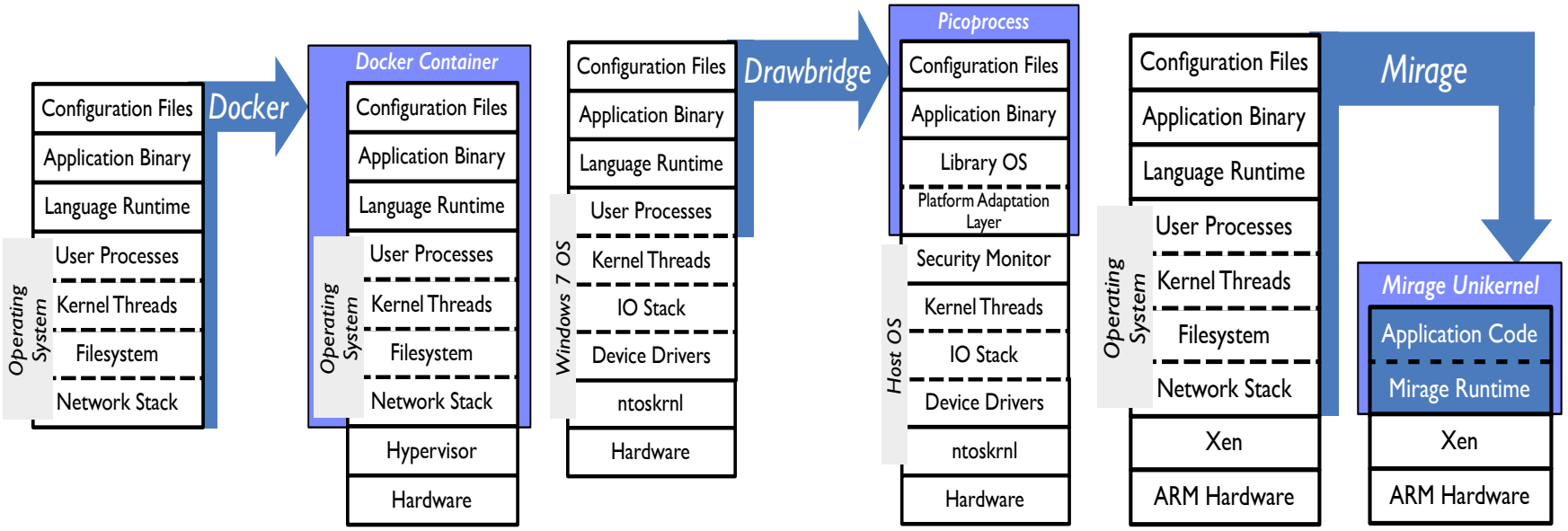
1. New Cloud OS to meet new workloads
   - Includes programming language
   - Collabs incl REMS (w/ P.Gardner/Imperial)
2. New Data Center structure
   - Includes heterogeneous h/w
   - Collabs incl NaaS(Peter Pietzuch Imperial)
   - Trilogy (Mark Handley et al UCL)
3. New Networks (for data centers&)
   - 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

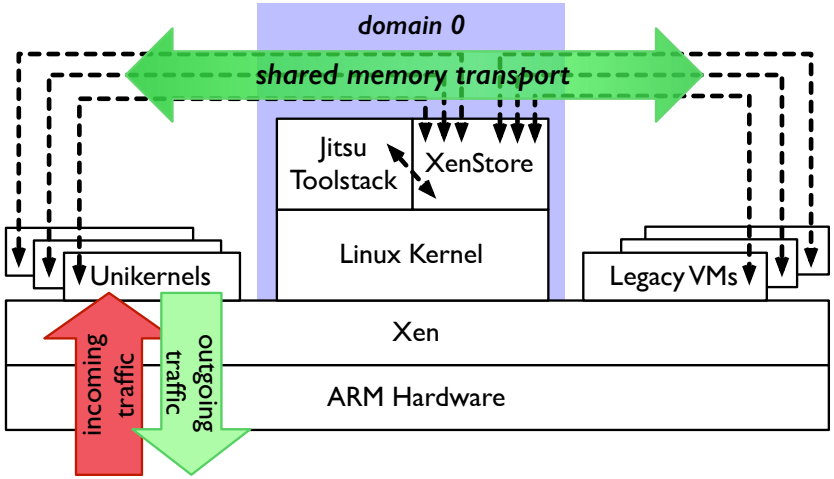


Figure 1: Jitsu architecture: external network connectivity is handled solely by memory-safe unikernels connected to general purpose VMs via shared memory.

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/

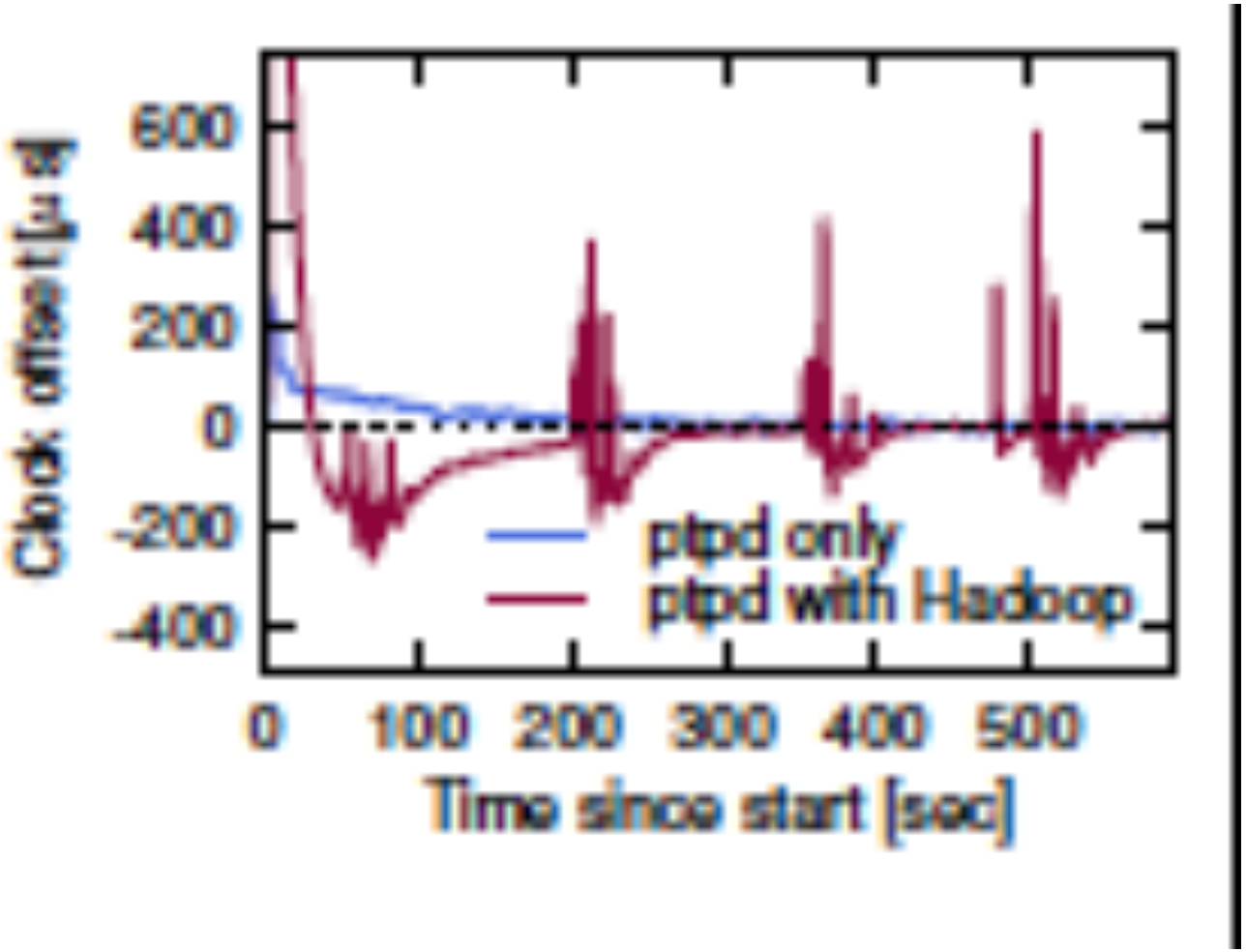
Data Center Latency Problem

- 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

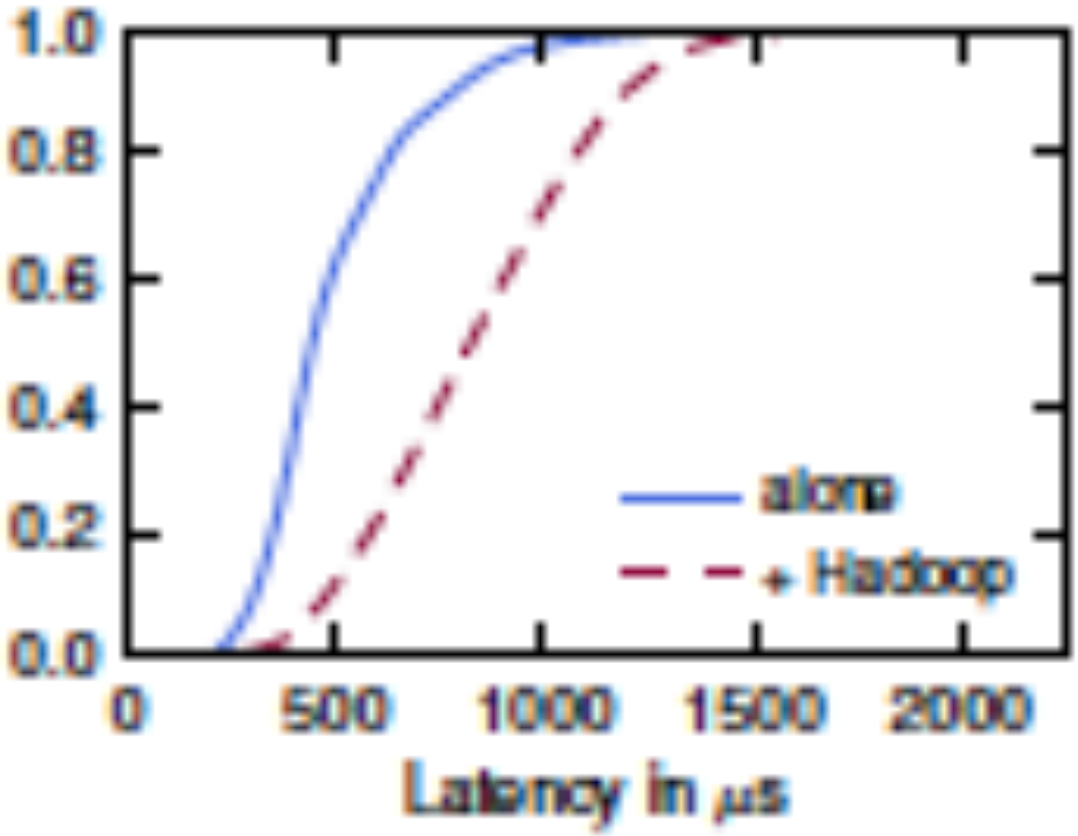
Data Center Qjump Solution

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

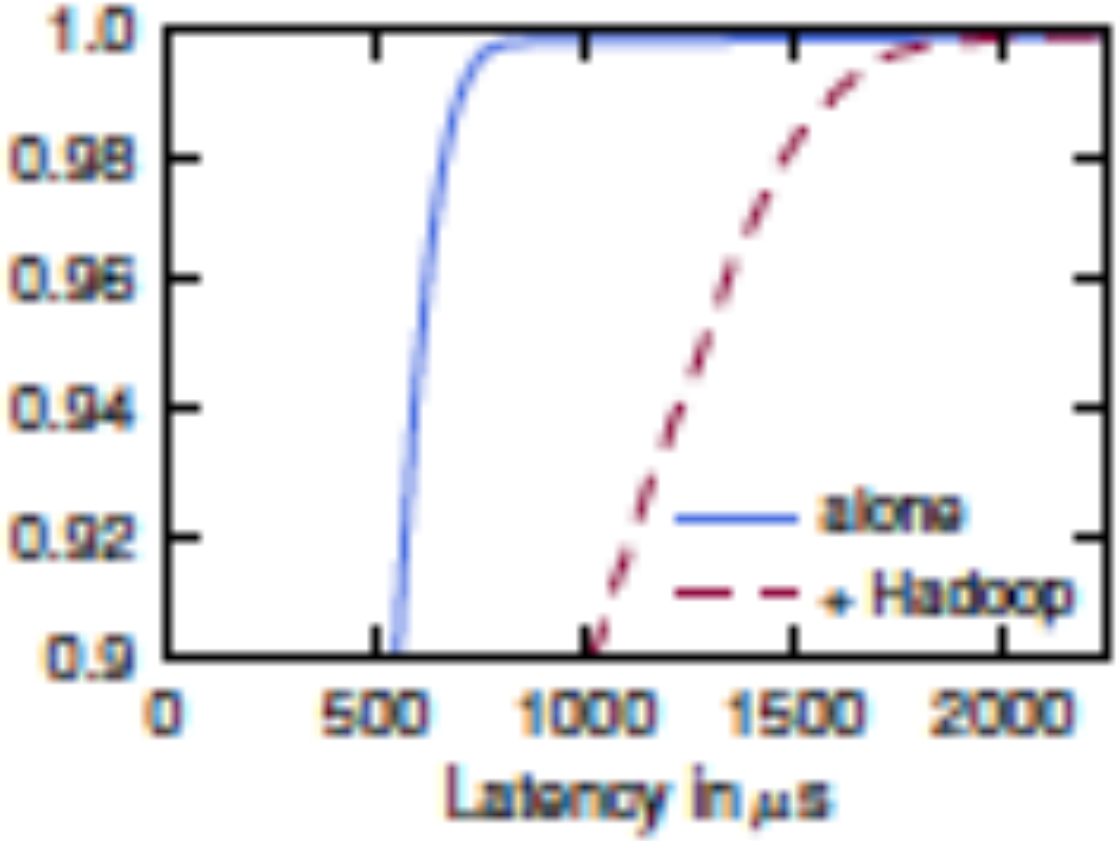
Hadoop perturbs time synch



Hadoop perturbs memcached



Hadoop perturbs Naiad

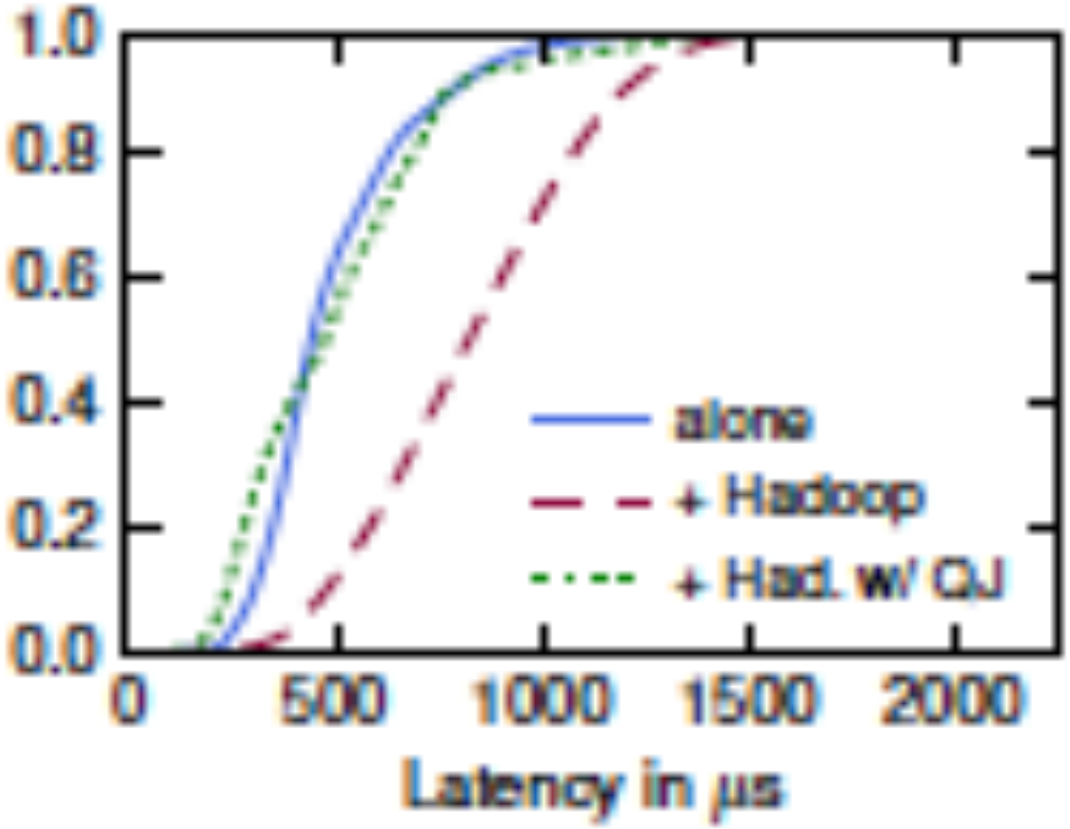


Qjump – two pieces

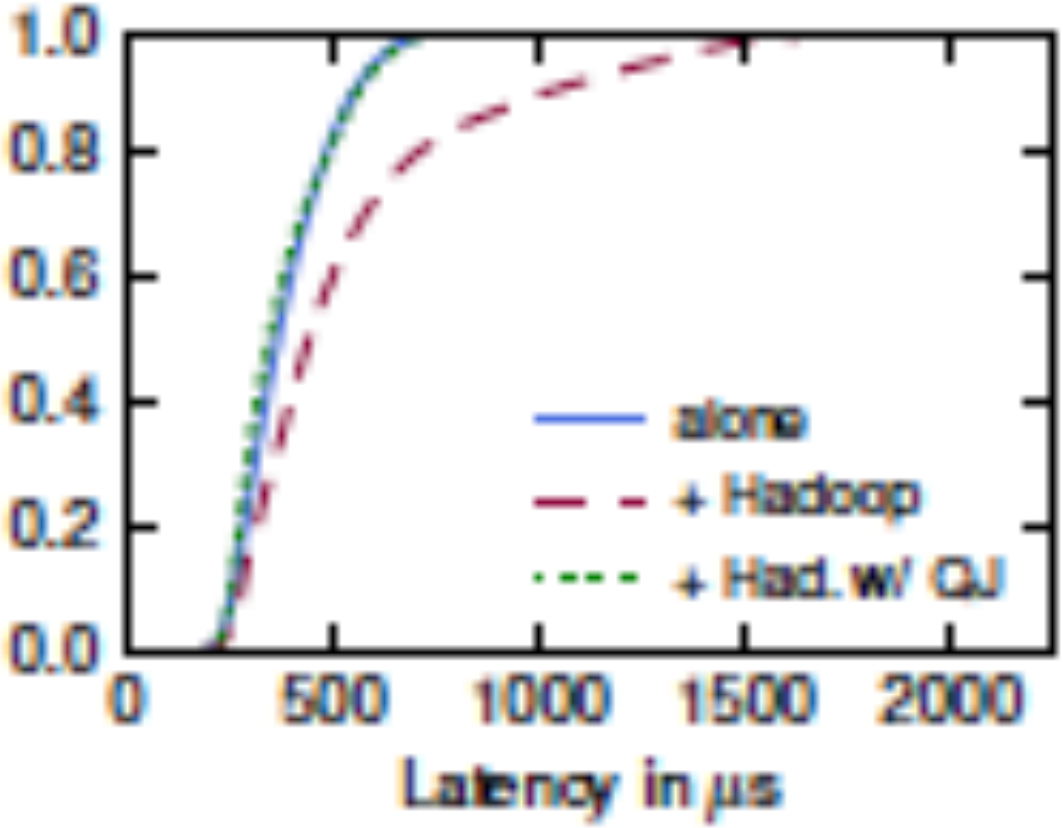
1. At network config time
   - Compute a set of (8*) rates based on
   - Traffic matric & hops => fan in (f)
2. At run time
   - Flow assigns itself a priority/rate class
   - subject it to (per hypervisor) rate limit

* 8 arbitrary – but often h/w supported☺

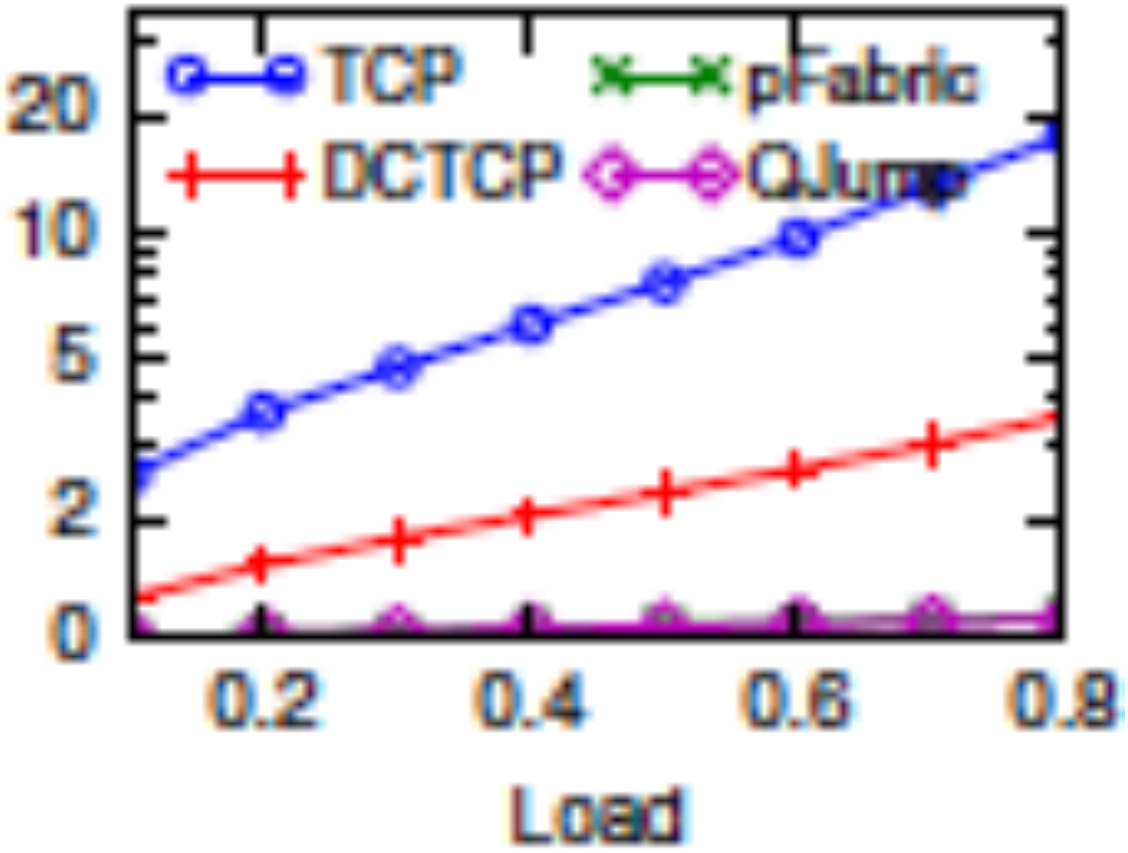
Memcached latency redux w/ QJ



QJ naiad barrier synch latency redux



Web search FCT100Kb ave



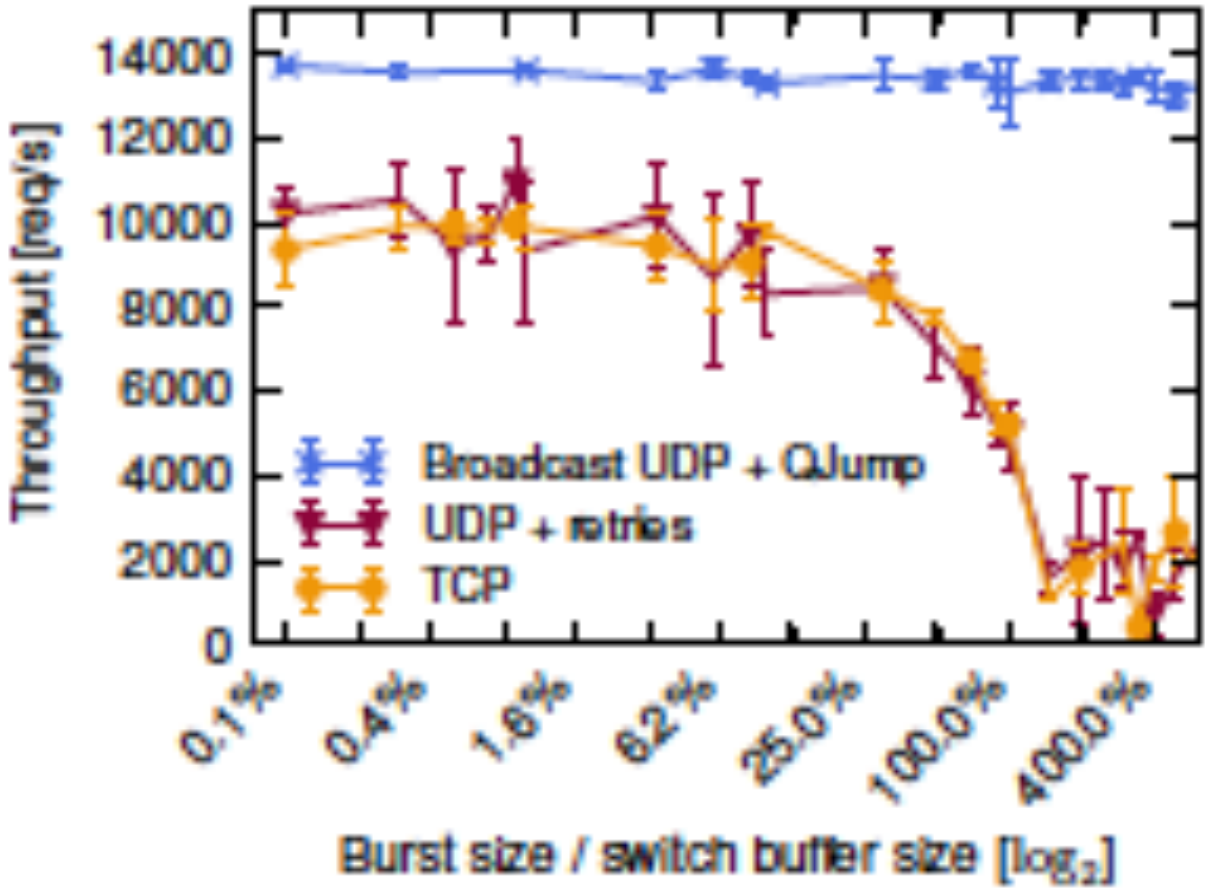
Big Picture Comparison – Related work…

System	Commodity hardware	Unmodified protocols	Unmodified OS kernel	Unmodified applications	Coord. free	Flow deadlines	Guaranteed latency	Implemented
Pause frames	✓	✓	✓	✓	✓	✗	✗	✓‡
ECN	✓*, ECN	✓	✓	✓	✓	✗	✗	✓‡
DCTCP [1]	✓*, ECN	✓*	✗	✓	✓	✗	✗	✓‡
Fastpass [23]	✓	✓	✓, module	✓	✗	✗	✗	✓‡
EyeQ [16]	✓*, ECN	✓	✗	✓	✗	✗	✗	✓‡
QJUMP	✓	✓	✓, module	✓	✓	✓*	✓	✓‡
D^{2}TCP [27]	✓*, ECN	✓*	✗	✓	✗*	✓	✗	✓
HULL [2]	✗	✓*	✗	✓	✓	✗	✗	✓*
D^3 [29]	✗	✗	✗	✗	✓	✓	✗	✗*, softw.
PDQ [13]	✗	✗	✗	✗	✗	✓	✗	✗
pFabric [3]	✗	✗	✗	✓	✓	✓*	✗	✗
DeTail [31]	✗	✓	✓	✗	✗*	✗	✗	✗*, softw.
Silo [15]	✓	✓	✗	✗*, hyperv.	✗*	✓*, SLAs	✗	✓
TDMA Eth. [28]	✓*	✓*	✗	✓*	✗	✗	✓	✓

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



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)

Application code

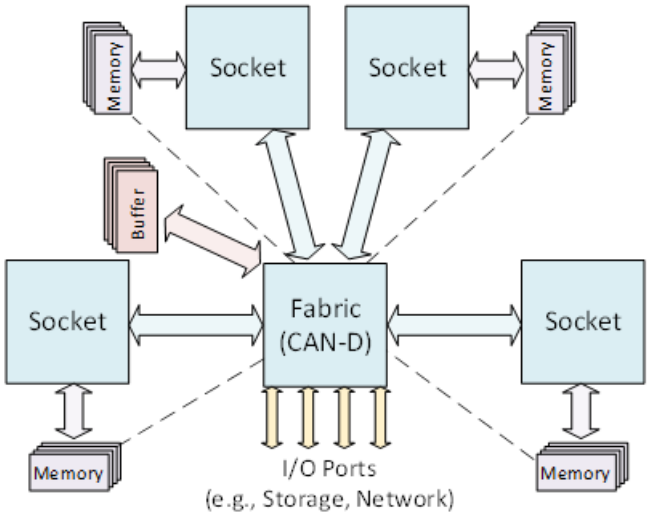
- E.g. shuffle phase in map/reduce
  - Does a bunch of aggregation
  - (min, max, ave) on a row of results
  - And is cause of traffic “implosion”
  - So do work in stages in the switches in the net (like merge sort!)
- Code very simple
- Cross-compile into switch NIC cpus

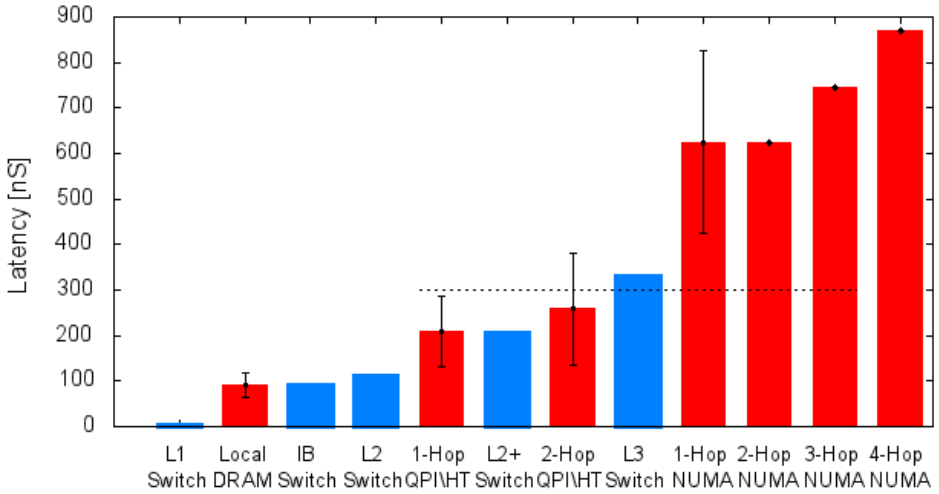
Other application examples

- Are many …
- Arose in Active Network research
  - Transcoding
  - Encryption
  - Compression
  - Index/Search
- Etc etc

Need language to express these

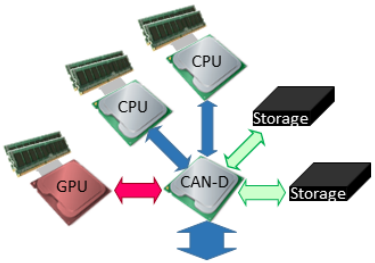
- Finite iteration
- (not Turing-complete language)
- So design python– with strong types!
- Work in progress in NaaS project at Imperial and Cambridge…



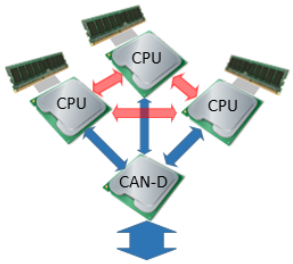


- ✓ **High Performance**
- ✓ **Resource Isolation**
- ✓ **Flexible Implementation**

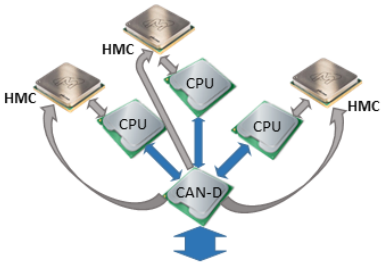
- ✓ **Predictable Latency**
- ✓ **Low Latency Interconnect**
- ✓ **Affordable**



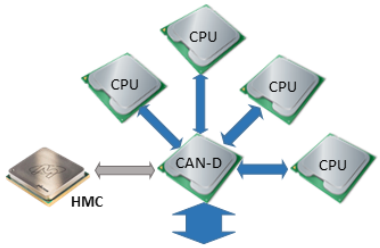
Centralized network fabric



Distributed network fabric

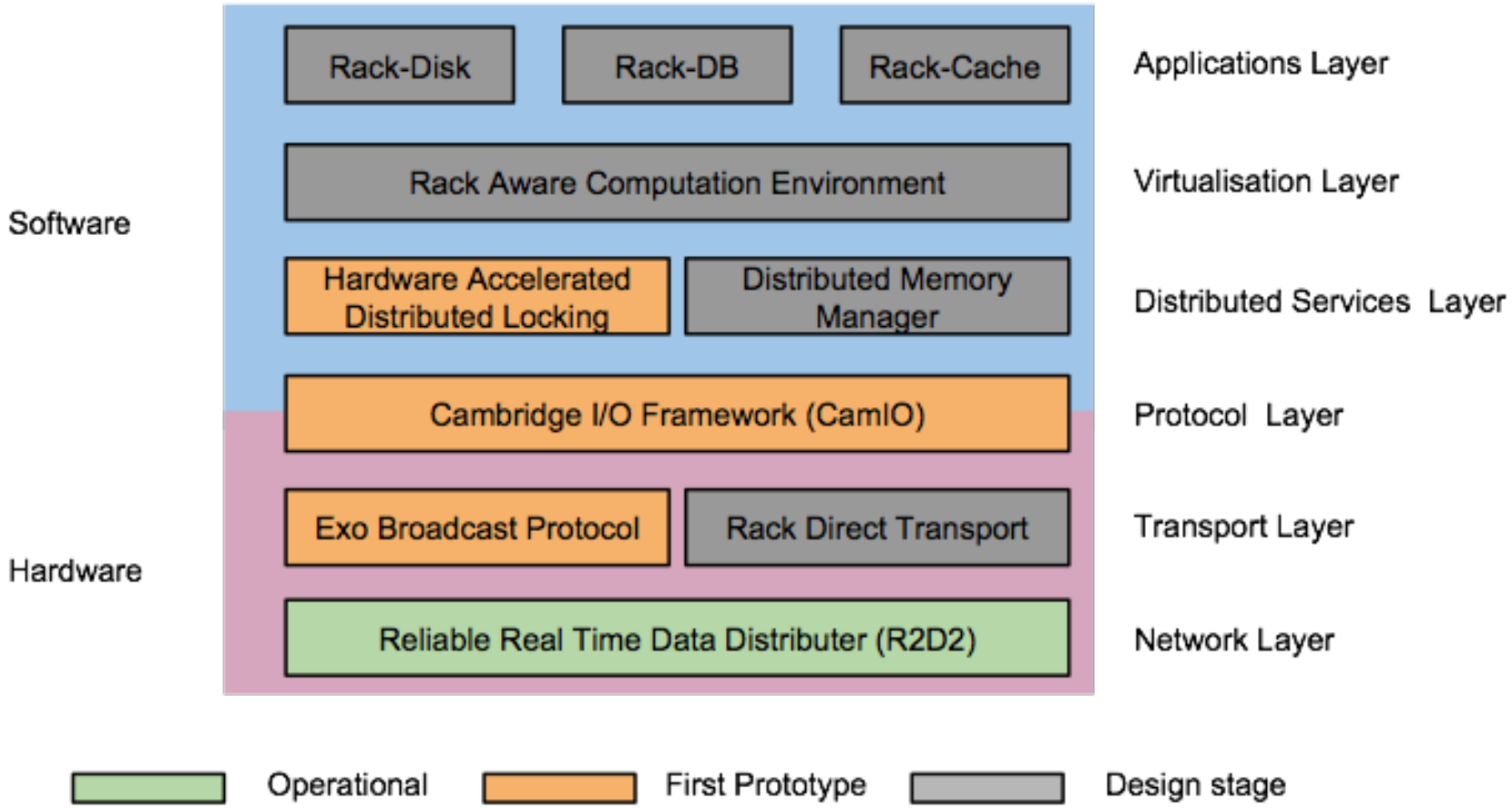


Multi-controller memory



Centralized memory

Networks, Interfaces and Transports for Rack-Scale Operating Systems



Conclusions/Discussion

- Data Center is a special case!
- Its 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…