CS434/534: Topics in Network Systems

Network Telemetry;
Course Summary

Yang (Richard) Yang
Computer Science Department
Yale University
208A Watson
Email: yry@cs.yale.edu

http://zoo.cs.yale.edu/classes/cs434/
Outline

- Admin and recap
- Network telemetry
- Course summary
Admin

- Instructor office hours
  - Tuesday: 1:30-2:30
  - Thursday: 3:00-4:00
  - This Friday: 1:30-2:30 pm -> 3:30-4:30 pm

- Projects
  - Milestones (exactly 4 weeks left)
    - 4/17 (T+1 week): Finish reading major related work; a google doc listing related papers (at least 4 papers)
    - 4/24 (T+2 weeks): Finish architecture design (slides/write up of architecture, including all key components)
    - 5/1 (T+3 weeks): Initial, preliminary evaluations (slides/write up, about experiment/analysis setup)
    - 5/8 (T+4 weeks) 5:30 pm, final report due
Recap: Network Verification

- **Goal:** verify that a network satisfies properties such as reachability
- **Two basic types of approaches**
  - Control plane verification
  - Data plane verification
    - Anteater
    - Header space analysis

```plaintext
function reach(s, t, k, G)
  r[e][0] ← true
  r[v][0] ← false for all v ∈ V(G) \ t
  for i = 1 to k do
    for all v ∈ V(G) \ t do
      r[v][i] ← ∨
      (P(v, u) ∧ r[u][i - 1])
    end for
  end for
  return ∨ r[s][i]
```

- **DP** → **Boolean expr**;
- **SAT** solver
- "Symbolic" exec wild card packets
Recap: Real-time Network Verification

- Real-time HSA
- Veriflow
- Atomic predicates verifier
Network Telemetry

- Huge progress in network telemetry - now designers are even thinking of the possibility of seeing the behavior of each, every packet

- Key questions:
  - What is the programming model?
  - How to make it possible at such a high speed?
Outline

- Admin and recap
- Network telemetry
- Course summary
Revisit some slides of first class
Lecture 1: Network Systems == Boxes?

Router  
Label Switched Router  
Load balancer  
Serving Gateway  
Switch  
Repeater  
Gateway  
Intrusion Detection System  
Bridge  
Route Reflector  
Deep Packet Inspection  
Firewall  
Hub  
DHCP server  
Packet shaper  
NAT  
WAN accelerator  
DNS server  
PDN Gateway  
eNodeB  
Packet sniffer  
Proxy
Course Topic 1: Network Foundation and Software-Defined Network

- Unifying network compute abstraction and OpenFlow as a standard network device programming interface
- Network operating systems: NOX, ODL, ONOS
- Network programming: Maple’13, Frenetic’13, Magellan’18
- Distributed NOS foundation: Paxos, Raft
- General programmable ASIC: FB OCP Wedge, RMT’13, dRMT’17, Intel FlexPipe
- Network device programming: Click’00, FBOSS’15, SONiC’17
- Basic SDN use cases: ACL, telemetry, network compute
Course Topic 2: Cloud Networking: Generic Mechanisms

- Basic design: VL2’09
- Topology: Fat tree, Juniper rising’15
- Layer 2/3: NVP’14 DC virtualization
- Layer 4/app load balancing (LB) in CDC: software LB (Maglev’16), hardware LB (Duet’15, Silkroad’17)
- Layer 4 transport in CDC: DCTCP’10, MPTCP’14, CC framework’17
- Routing and traffic engineering in CDC: COPE, R3, CONGA’14, Hedera’15
Course Topic 3: Data Center Interconnect, WAN Connection

- Software defined WAN: SWAN'13, B4'13/Espresso'17/B4 and after'18, Edge Fabric'17
- Software defined exchange (SDX'14)
- SD-WAN programming: Propane'16
Course Topic 4: Cloud Up the Stack and into the Host

- **Data analytics (DA) in CDC**
  - MapReduce, Spark, SparkStream
  - *Generic DA scheduling: YARN, Mesos, GoogleBorg, DRF, Appollo*
  - Network scheduling and impacts, CoFlow'14, Sincronia'18

- **Realtime apps in CDC**: CloudMirror'16, ElasticSwitch

- **RDMA'18**: Smart NIC, PCIe'18, NVMe'18
Course Topic 5: Reliability/Availability Mechanisms

- **Testing**
  - Shadow configuration and streaming

- **Analysis and verification**
  - Header space analysis'12,'13, Veriflow
  - Minesweeper'17
  - Verified NAT'17, P4V'18
Course Topic 6: Beyond Simple Switching (5G and IoT)

- Network function virtualization, Click OS’00, NFP’17, Trident’18
- 5G network architecture PEPC’17
- IoT protocols, COAP, MQTT
- Non-conventional links, body wireless networks
- Smart cities
What are the Goals/Topics of this Course?

- Learn protocols, boxes, tools, theory, and systems in the context of real, important, less well-understood network systems
  - Data centers (cloud), (big) data analytics, Internet of things

- Go beyond pieces, to derive higher-level principles, insights
Lecture 1: Example “Higher” Principles

- How to
  - Design and manage protocols/boxes
  - That can be used and combined in many ways
  - To do many things

- How to
  - Unify diverse protocols/Boxes (to reduce their numbers)
  - But provide more flexibility and achieve more
Outline

- Admin and recap
- Network telemetry
- Course summary
  - Network system programming models
Exercise

- What programming models did we cover?
Programming Models

- Programming models
  - SDN programming models
  - Network function programming models
  - Network monitoring programming models
  - Datacenter application programming models
    - Web programming model
    - Distributed learning programming model
    - Data analytics programming models
Maple SDN
Programming Model

Route f(Packet p) {
    if (p.tcpDstIs(22)) return null();
    else {
        Location sloc = hostTable(p.ethSrc());
        Location dloc = hostTable(p.ethDst());
        Route path = myRoutingAlg(topology(), sloc, dloc);
        return path;
    }
}

Route myRoutingAlg(Topology topo, Location sLoc, Location dLoc) {
    if (isSensitive(sLoc) || isSensitive(dLoc))
        return secureRoutingAlg(topo, sloc, dloc);
    else
        return standardRoutingAlg(topo, sloc, dloc);
}
Click NF/Middlebox Programming Model

// Declare three elements...
src :: FromDevice(eth0);
ctr :: Counter;
sink :: Discard;
// ...and connect them together
src -> ctr;
ctr -> sink;

// Alternate definition using syntactic sugar
FromDevice(eth0) -> Counter -> Discard;

class NullElement: public Element { public:
  NullElement() { add_input(); add_output(); }
  const char *class_name() const { return "Null"; }
  NullElement *clone() const { return new NullElement; }
  const char *processing() const { return AGNOSTIC; }
  void push(int port, Packet *p) { output(0).push(p); }
  Packet *pull(int port) { return input(0).pull(); }
};

// Diagram:
FromDevice(eth0) -> Counter -> Discard

FromDevice
  |receive packet p|
  |   push(p)    |
  |   return     |
  |   dequeue p
  |   and return it|

Null
  |push(p)    |
  |return     |
  |enqueue p  |
  |pull()     |

Null
  |push(p)    |
  |return p   |
  |return p   |
  |return p   |

ToDevice
  |ready to transmit|
  |send p|
ClickNP NF/Middlebox Programming Model

```plaintext
.element Count <1, 1> {
  .state{
    ulong count;
  }
  .init{
    count = 0;
  }
  .handler{
    if (get_input_port() != PORT_1) {
      return (PORT_1);
    }
    flit x;
    x = read_input_port(PORT_1);
    if (x.fd.sop) count = count + 1;
    set_output_port(PORT_1, x);
    return (PORT_1);
  }
  .signal{
    CLSignal p;
    p.Sig.LParam[0] = count;
    set_signal(p);
  }
}
```

The ClickNP NF/Middlebox Programming Model uses channels to communicate between components, rather than shared memory. This allows for efficient and atomic communication. The code snippet demonstrates the use of `element`, `state`, `init`, `handler`, and `signal` to define the behavior of a component. The diagram illustrates the flow of data and control signals, with `states` and `signal handler` indicating the main thread and interrupt handler, respectively. The input and output channels are clearly depicted, showing how data flows between components.
Bro NF/Middlebox Programming Model

```c
int FingerConn::NewLine(TCP_Endpoint* /* s */, double /* t */, char* line)
{
    line = skip_whitespace(line);

    // Check for /W.
    int is_long = (line[0] == '/' && toupper(line[1]) == 'W');

    if (is_long)
        line = skip_whitespace(line+2);
    val_list* vl = new val_list;
    vl->append(BuildConnVal());
    vl->append(new StringVal(line));
    vl->append(new Val(is_long, TYPE_BOOL));
    mgr.QueueEvent(finger_request, vl);
    return 0;
}
```

```
global hot_names = { "root", "lp", "uucp" };
global finger_log =
    open(getenv("BRO_ID") == "" ?
    "finger.log",
    fmt("finger.%s", getenv("BRO_ID"));

event finger_request(c::connection,
    request: string,
    full: bool)
{
    if (byte_len(request) > 80) {
        request = fmt("%s...",
            sub_bytes(request, 1, 80));
        ++c$shot;
    }
    if (request in hot_names)
        ++c$shot;

    local req = request == "" ?
        "ANY" : fmt("%s\%s", request);
    if (c$addl != "")
        # This is an additional request.
        req = fmt("%s", req);
    if (full)
        req = fmt("%s (/W)", req);

    local msg = fmt("%s > %s %s",
        c$addorig_h,
        c$addresp_h,
        req);
    if (c$shot > 0)
        log fmt("finger: %s", msg);
    print finger_log,
        fmt("%.6f %s", c$start_time, msg);
    c$addl = c$addl == "" ?
        req : fmt("%s, %s", c$addl, req);
}
```
NetBricks NF/Middlebox Programming Model

Listing 1: NetBricks NF that decrements TTL, dropping packets with TTL=0.

```rust
pub fn ttl_nf<T: 'static + NbNode>(input: T) -> CompositionNode {
    input.parse::<MacHeader>()
        .parse::<IpHeader>()
        .transform(box |pkt| {
            let ttl = pkt.hdr().ttl() - 1;
            pkt.mut_hdr().set_ttl(ttl);
        })
        .filter(box |pkt| (pkt.hdr().ttl() != 0))
        .compose()
}
```

Listing 2: Operator code for using the NF in Listing 1

```rust
// cfg is configuration including
// the set of ports to use.
let ctx = NetBricksContext::from_cfg(cfg);
ctx.queue.map(|p| ttl_nf(p).send(p));
```


```rust
pub fn maglev_nf<T: 'static + NbNode>(input: T, backends: &str,
                                      ctx: nb_ctx,
                                      lut_size: usize)
    -> Vec<CompositionNode> {
    let backend_ct = backends.len();
    let lookup_table = Maglev::new_lut(ctx,
                                        backends,
                                        lut_size);
    let mut flow_cache = BoundedConsistencyMap::<usize, usize>::new();

    let groups = input.shuffle(BuiltinShuffle::flow)
        .parse::<MacHeader>()
        .group_by(backend_ct, ctx,
                  box move |pkt| {
                      let hash = ipv4_flow_hash(pkt, 0);
                      let backend_group = flow_cache.entry(hash)
                          .or_insert_with(|| {
                              lookup_table.lookup(hash)
                          });
                      backend_group
                  });
    groups.iter().map(|g| g.compose()).collect()
}
```
Noria DC/Web Programming Model

```sql
/* base tables */
CREATE TABLE stories
  (id int, author int, title text, url text);
CREATE TABLE votes (user int, story_id int);
CREATE TABLE users (id int, username text);
/* internal view: vote count per story */
CREATE INTERNAL VIEW VoteCount AS
  SELECT story_id, COUNT(*) AS vcount
  FROM votes GROUP BY story_id;
/* external view: story details */
CREATE VIEW StoriesWithVC AS
  SELECT id, author, title, url, vcount
  FROM stories
    JOIN VoteCount ON VoteCount.story_id = stories.id
WHERE stories.id = ?;
```
MapReduce Data Analytics
Programming Model

Map(input_key, input_value) {
    foreach word w in input_value:
        emit(w, 1);
}

Reduce(output_key, intermediate_vals) {
    set count = 0;
    foreach v in intermediate_vals:
        count += v;
    emit(output_key, count);
}
val points = spark.textFile(...) .map(parsePoint).persist()

var w = // random initial vector
for (i <- 1 to ITERATIONS) {
  val gradient = points.map{ p =>
    p.x * (1/(1+exp(-p.y*(w dot p.x)))-1)*p.y
  }.reduce((a,b) => a+b)
  w -= gradient
}
Discussion

- What do those programming models achieve?
Hide Lower Layer Complexity or Raise Operation Abstraction

Higher-level programming model

Standard programming model

Lower-level programming model

Hide lower layer limited datapath

Hide lower layer distributed computing

Hide lower layer dynamic update

Assert: TcpDsd==22
false
true

Read: EthDst

Read: EthDst

(hostTable, 1)

(hostTable, 2)
(topology())

(path1)

(path2)
Problems of current SDN/NFV programming models

- SDN can program only L2-L4 headers (info must be contained in each and every packet)
  - NFV extracts L4-L7 info, but SDN program cannot naturally use (info not in a specific packet)
- A key function of SDN is route control (for both network and NF composition), but current SDN route control programming is low level
Trident Integrating NF: Stream Attribute

Define a stream attribute:

```scala
val http_uri = StreamAttribute[String]("HTTP_URI", TCP5TUPLE)
```

- The type information, e.g., String, Int
- A descriptive name, e.g., HTTP_URI, authenticated
- The **stream type** (bit masks on packet header fields), e.g., TCP5TUPLE, SRC_IPADDR, DST_IPADDR
Trident Integrating NF: Stream Attribute

Use a stream attribute just like a packet header

pkt.http_uri, pkt.authenticated, pkt.heavy_hitter, ...

Stream attribute MAY have an unknown value

Trident treats unknown values as valid and uses Kleene’s 3-valued logic to select packets based on stream attribute.

```c
// 3-way branch
if ((pkt.authenticated) && (pkt.http_uri == "www.xyz.com")) {
   // true branch
} else {
   // else branch
} unknown {
   // unknown branch
}

// fallback branch
iff ((pkt.authenticated) && (pkt.http_uri == "www.xyz.com")) {
   // true branch
} else {
   // else and unknown branch
}
```
**Objectives:** Use well-structured, declarative expressions to specify the construction of consistent, correlated routes

- The basic unit of route algebra is **route set**.
- Each route set has a **network function indicator** to specify the symmetry requirements of network functions.

---

**Union (∪)/Intersection (∩)/Difference (∖)**

Given two route sets $\Delta_1$ and $\Delta_2$, return the union/intersection/difference of $\Delta_1$ and $\Delta_2$:

- $\Delta_1 \cup \Delta_2 = \{ r \mid r \in \Delta_1 \lor r \in \Delta_2 \}$
- $\Delta_1 \cap \Delta_2 = \{ r \mid r \in \Delta_1 \land r \in \Delta_2 \}$
- $\Delta_1 \setminus \Delta_2 = \{ r \mid r \in \Delta_1 \land r \notin \Delta_2 \}$

---

**Concatenation (+)**

Given two route sets $\Delta_1$ and $\Delta_2$, return a new route set by concatenating all route pairs $(r_1, r_2)$ in $\Delta_1 \times \Delta_2$ and removing the invalid ones:

$\Delta_1 + \Delta_2 = \{ r_1 + r_2 \mid r_1 \in \Delta_1, r_2 \in \Delta_2, \text{dist}_{r_1} = \text{src}_{r_2} \}$.

---

**Inversion (×)**

Given a route set $\Delta$, return the inverse of $r \in \Delta$:

$\times \Delta = \{ \times r \mid r \in \Delta \}$.

---

**Preference (>)**

Given two route sets $\Delta_1$ and $\Delta_2$, return the preferred route. (If there is an equivalent route in $\Delta_1$, do not use the ones in $\Delta_2$):

$\Delta_1 \triangleright \Delta_2 = \{ r \mid r \in \Delta_1 \lor (r \in \Delta_2 \land \nexists r' \in \Delta_1, r \sim r') \}$.

---

**Selection (σ)**

Given a route set $\Delta$ and an evaluation function $f : R^* \mapsto \{0, 1\}$, return all routes in $\Delta$ that are evaluated as 1:

$\sigma_f(\Delta) = \{ r \in \Delta \mid f(r) = 1 \}$.

---

**Optimal selection (∅)**

Given one route set $\Delta$ and a routing cost function $d : R^* \mapsto \mathbb{R}$, return any route with the minimum value:

$\emptyset_d(\Delta) = \arg\min_{r \in \Delta} d(r)$.

---

**Arbitrary selection (*)**

Given one route set $\Delta$, return a route set containing exactly one route $r$ in $\Delta$:

$\ast \Delta = \emptyset_1(\Delta)$. 

---

34
Trident Realization: Live Variables

Objectives: Make dependency tracking and updates transparent to programmers (motivated by functional reactive programming).

- Live variable is a traceable data type which stores the value and the computation process (i.e., dependencies and computation methods).
- Stream attribute and route algebra are implemented as live variables.

```c
iff (pkt.is_endhost_infected) {
  drop(pkt)
} else {
  bind(pkt, r_1 + r_2)
}
```

- `sip: 10.0.1.5`
- `r_1`
- `r_2`
- `infected`
- `otherwise`
Interesting Network/Distributed System Programming Model: Privacy using Public Data

- Problem of current system: privacy leak
Splinter Architecture

- Have multiple providers hosting public data
- Query multiple providers using function secret sharing (FSS)
FSS Properties

Divides a function $f$ into $k$ shares, $f_i$, such that:

- $f_i$ can be evaluated quickly

$$\sum_{i=1}^{k} f_i(x) = f(x)$$

Given $k-1$ shares, cannot recover $f$

Efficient constructions of $f$ exist for two cases:

Point functions: $f(x) = 1$ if $x = a$; 0 otherwise

Interval functions: $f(x) = 1$ if $a \leq x \leq b$; 0 otherwise
FSS Basic Construction

FSS

\[ f(x) = 1 \text{ if } x = 5 \text{ and } 0 \text{ otherwise} \]

function shares: \( f_1, f_2 \)

\[ f_1(x) + f_2(x) = f(x) \]

<table>
<thead>
<tr>
<th>route</th>
<th>price</th>
<th>( f_1(\text{route}) )</th>
<th>( f_2(\text{route}) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>8</td>
<td>10</td>
<td>-9</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>-3</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>9</td>
<td>10</td>
<td>-9</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>7</td>
<td>-7</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>-3</td>
<td>3</td>
</tr>
</tbody>
</table>

Having either \( f_1 \) or \( f_2 \) does not reveal any information about \( f \)
Offline Exercise

- Given efficient point and interval constructions, what is the programming model that you can design for privacy preserving information retrieval?

- Hint: see Splinter’17
A key programming model of IoT appears to be pub/sub systems, but the details can be quite different.

- **Offline read: DDS**
The Programming Model for LTE

- **Performance overheads + high complexity**
  - Frequent cross component synchronization

- **Migration is hard**
  - Distributed user state

- **Customization is hard**
  - Distributed user state + distributed computation
State-of-Art Cellular Programming Model

Existing EPCs

PEPC

user signaling traffic

user data traffic

user state

user state

MM E

S-GW

P-GW

signaling traffic

data traffic

signaling traffic

data traffic

Slice

Slice
Partition shared state at two levels → reduces contention

- By user
- Per-user state whether control or data thread writes to it
- Use fine grained locks → up to 5X improvement over coarse grained locks
State-of-Art Cellular Programming Model - PEPC Server

- Pause + snapshot user state —> simplifies state migration
- Modify slice data/control flow —> simplifies customization
Q: Do you need to introduce a programming model in your project? If so, what does it achieve?
Outline

- Admin and recap
- Network telemetry
- Course summary
  - Network system programming models
  - Abstractions (illusions)
Exercise

- What are some abstractions (illusions) that we saw introduced in the class?
The “One-Big-Switch” Abstraction

Provide the illusion of “One Big Switch”

10,000s of ports

Compute

Storage (Disk, Flash, ...)*
Benefits of the “Illusion”

- Turn the servers into a single large fungible pool
  - Dynamically expand and contract service footprint as needed

- Benefits
  - Lower server component cost
  - Achieve high performance and reliability
  - Increase service developer productivity

Agility: The same of most infrastructure projects
The OpenFlow Table Abstraction

- **match fields**: to match against packets. These consist of the ingress port and packet headers, and optionally other pipeline fields such as metadata specified by a previous table.
- **priority**: matching precedence of the flow entry.
- **counters**: updated when packets are matched.
- **instructions**: to modify the action set or pipeline processing.
- **timeouts**: maximum amount of time or idle time before flow is expired by the switch.
- **cookie**: opaque data value chosen by the controller. May be used by the controller to filter flow entries affected by flow statistics, flow modification and flow deletion requests. Not used when processing packets.
- **flags**: flags alter the way flow entries are managed, for example the flag OFPFF_SEND_FLOW_REM triggers flow removed messages for that flow entry.
The “Oracel” SDN Programming Abstraction

- Conceptually, programmer’s network control function $f$ is invoked on every packet entering the network.
- $f$ expressed in an existing, general purpose language (e.g., Java, Python), describing how a packet should be routed, not how data path flow tables are configured.

$f$: packet $\rightarrow$ route
The Single Data Flow Abstraction

```scala
val points = spark.textFile(...) .map(parsePoint).persist()

var w = // random initial vector
for (i <- 1 to ITERATIONS) {
  val gradient = points.map{ p =>
    p.x * (1/(1+exp(-p.y*(w dot p.x))))-1)*p.y
  }.reduce((a,b) => a+b)
  w -= gradient
}
```

### Transformations

- `map(f : T => U)` : `RDD[T] => RDD[U]`
- `filter(f : T => Boolean)` : `RDD[T] => RDD[T]`
- `flatMap(f : T => Seq[U])` : `RDD[T] => RDD[U]`
- `groupByKey()` : `RDD[(K, V)] => RDD[(K, Seq[V])]`
- `reduceByKey(f : (V, V) => V)` : `RDD[(K, V)] => RDD[(K, V)]`
- `union()` : `RDD[(K, V)], RDD[(K, W)] => RDD[(K, V, W)]`
- `intersect()` : `RDD[(K, V)], RDD[(K, W)] => RDD[(K, V & W)]`
- `crossPartitions()` : `RDD[(T, U)] => RDD[T, U]`
- `mapValues(f : V => W)` : `RDD[(K, V)] => RDD[(K, W)]` (Preserves partitioning)
- `sortBy(c : Comparator[K])` : `RDD[(K, V)] => RDD[(K, V)]`
- `partitionBy(p : Partitioner[K])` : `RDD[(K, V)] => RDD[(K, V)]`

### Actions

- `count()` : `RDD[T] => Long`
- `collect()` : `RDD[T] => Seq[T]`
- `reduce(f : (T, T) => T)` : `RDD[T] => T`
- `lookup(k : K)` : `RDD[(K, V)] => Seq[V]` (On hash/range partitioned RDDs)
- `save(path : String)` : Outputs RDD to a storage system, e.g., HDFS
Networking System Design Guide

The most profound technologies are those that disappear.
– Mark Weiser

*Scientific American, Vol. 265, No. 3 (September 1991), pp. 94-104*
Q: Do you need to introduce some illusions in your project?
Outline

- Admin and recap
- Network telemetry
- Course summary
  - Network system programming models
  - Abstractions (illusions)
  - Control architecture
Control Architecture

- Almost all modern network systems are large-scale, and hence control architectures are crucial
  - Scalability
  - Robustness
  - Efficiency
  - Flexibility
- Typical what being controlled is resources (e.g., bw, compute, storage)
Discussion

- Different systems can use different entities to control resource allocation (e.g., rate, path, slot)

- Exercise:
  - TCP CC
  - MP TCP
  - Hedora
  - DC TCP
  - Fastpass
  - Spark slot
  - Coflow Varys
Consistency Control Structure: Replicated Log

- Replicated operations log => replicated state machine
  - All servers execute same commands in same order
- Consensus module ensures proper log replication
- System makes progress as long as any majority of servers are up
Coflow Control Architecture

Centralized master-slave architecture
- Applications use a client library to communicate with the master

Coflow transport timing and rates are determined by the coflow scheduler

- Sender
  - Put
  - Varys Daemon

- Receiver
  - Get
  - Varys Daemon

- Driver
  - Reg
  - Varys Daemon

Network Interface
- (Distributed) File System
- TaskName
- Comp. Tasks calling
- Varys Client Library

Topology Monitor
Usage Estimator
Coflow Scheduler
Varys Master
YARN Control Architecture

Application Layer

Resource Layer
Mesos Control Architecture

MPI job

MPI scheduler

Hadoop job

Hadoop scheduler

Mesos master

Allocation

Resource offer

Mesos slave

MPI executor

task

Mesos slave

MPI executor

Hadoop executor

task

Framework-specific scheduling

Pick framework to offer resources to

Launches and isolates executors
Omega Control Architecture

- Multiple controllers, shared state
  - Each controller can independently read/write shared state
Q: What is the control architecture of your system? What about scalability, reliability, efficiency, flexibility?
“Gold Diggers” vs Tools Builders
Have a Great Summer!

https://www.dreamstime.com/stock-photo-have-great-summer-banner-beach-image54370051
Physical Infrastructures
Conventional DC Architecture

**Key**
- **CR** = Core Router (L3)
- **AR** = Access Router (L3)
- **S** = Ethernet Switch (L2)
- **A** = Rack of app. servers

~ 1,000 servers/pod == IP subnet

Reference

– “Data Center: Load Balancing Data Center Services”, Cisco 2004
Generic K-ary Fat Tree

- K-ary fat tree: three-layer topology (edge, aggregation and core)
  - k pods w/ each pod consisting of 2 layers of k/2 k-port switches
    - each edge switch connects to k/2 servers & k/2 aggr. switches
    - each aggr. switch connects to k/2 edge & k/2 core switches
  - each core switch connects to k pods

Assume
- Each Int switch has $D_I$ ports;
- Each Aggr has $D_A$ ports

Each Aggr switch uses half ports to connect to TOR switches, half to each Intermediate switch.

Each TOR connects to two Aggr switches.

Each Int switch connects to each Aggr switch.

Each TOR connects 20 servers.

$20 \times \left( \frac{D_I}{4} \right)$ servers

$D_I$ $D_A/4$ TOR

$D_A/2$ Int switches

$D_I$ Aggr switches

20 Servers
WAN Infrastructure

- Wide-area links are typically more expensive, less reliable and hence need better resource scheduling.
5G INfrastructure