CArDS: Traffic Steering at L3 for Reducing Service Request Completion Times

Karima Saif Khandaker, Dirk Trossen, Ramin Khalili, Zoran Despotovic, Artur Hecker, Georg Carle

1 Problem of Runtime Scheduling

Consider execution of services in distributed (e.g., virtualized) service environments:

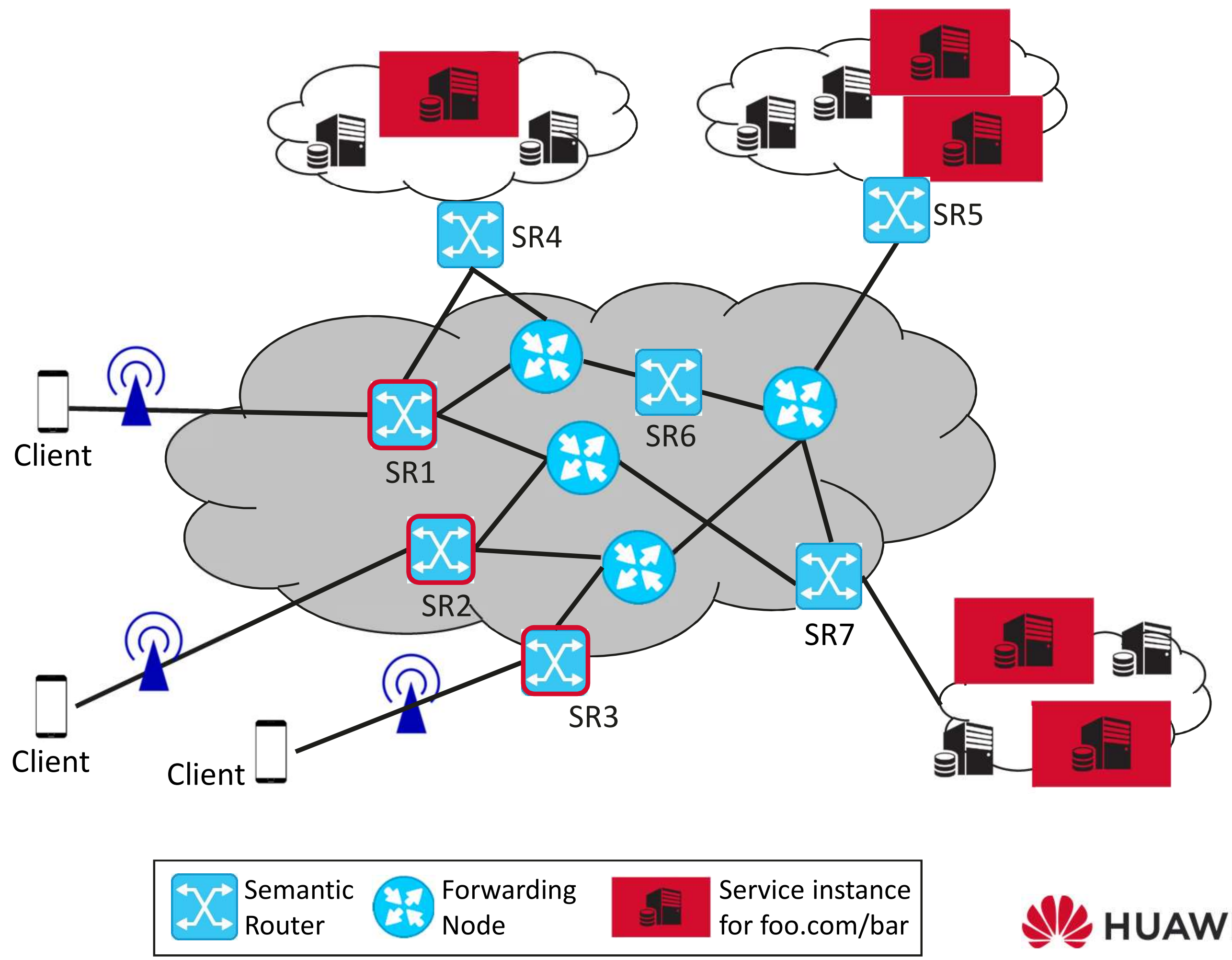
- A **service**, realized through a **service instance**, available in one or possibly more network locations
- **Service transaction** requires affinity to a service instance after the initial service request due to possible ephemeral state created

Problem: Find the ‘best’ service instance to serve the client’s transaction at runtime, while preserving the affinity after the decision has been made

Our Contribution: Compute-Aware Distributed Scheduling (CArDS), where ‘best’ is utilizing knowledge of the compute capabilities of individual service instances

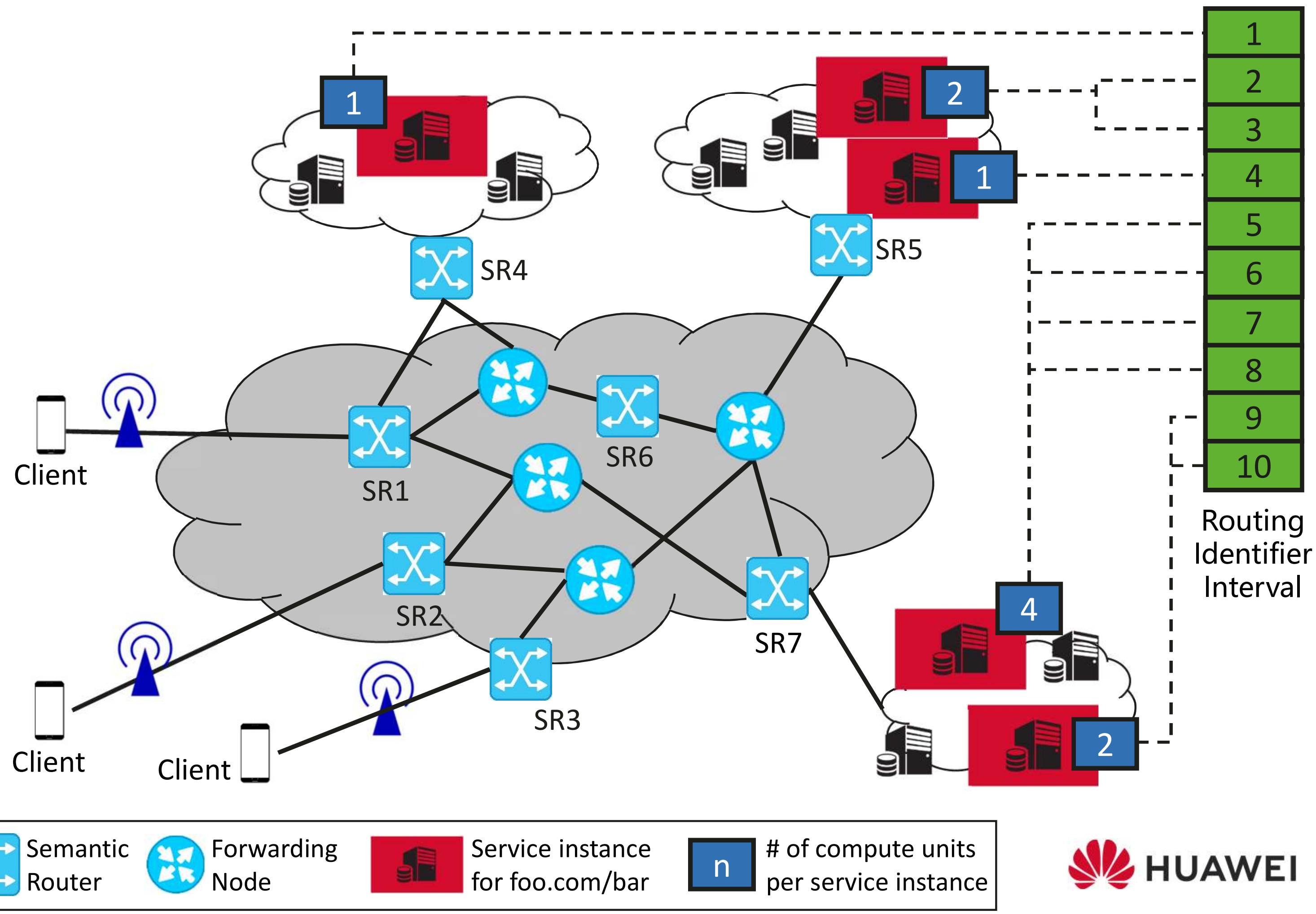
System Overview

- Geographically distributed sites at which service instances (SIs) for a given service are deployed
- Clients issue service requests destined to a **service identifier**
- The incoming semantic router forwards service requests towards a suitable destination, e.g., one of the possibly many service instances.
    - Performs an **on-path forwarding** decision compared to existing DNS+IP *off-path* systems
- **Affinity** is ensured by using IP locator for subsequent requests within same service transaction



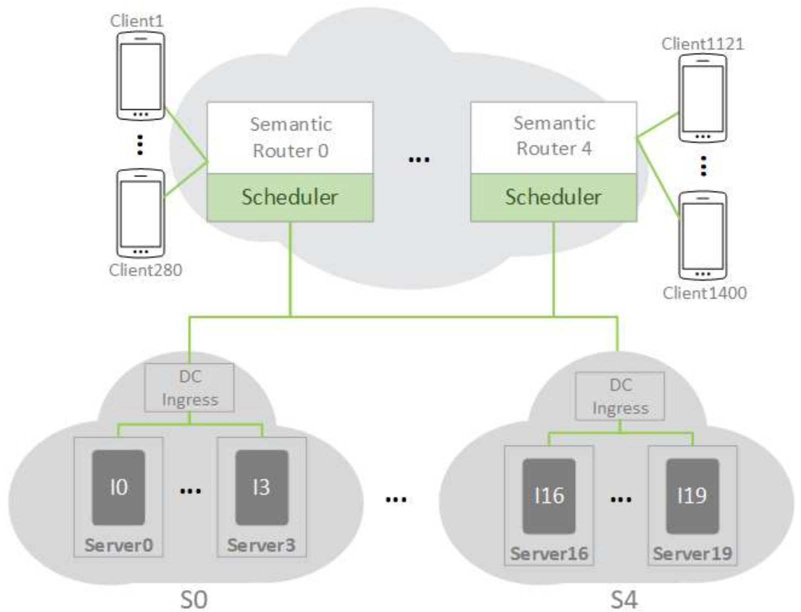
Compute-Aware Distributed Scheduling (CArDS)

- Each service instance assigned (during orchestration) a normalized **compute** (resource) **unit** (e.g., # cores, # threads)
- All compute units are flattened and joined in an identifier-specific **routing identifier interval**
  - Distributed to all routers only once after placement
- Scheduling is now distributed (i.e., within each ingress router) **round robin** over the interval for each incoming service request
- Implementable at **link speed**, e.g., in P4



Implementation & Evaluation Setup

- Event-based simulator using custom Python libraries
- 5 sites with 4 servers each and 1 service instance per server
- Compute units assignment of instances defined at start
- 5 ingress semantic routers
- All service requests are to one service function only, sent as single packet requests (all requests of one service identifier)
- The network load is varied by configuring the total number of clients, distributed equally across 5 ingress semantic routers
  - 100% workload is simulated using 1550 clients
- Main metric is mean **request completion time** (RCT) of service requests



Scenario 1a - Centralized vs Scaled Distributed Scheduling

- **Aim**: Observe the effects of distribution as well as scaling the number of distributed ingress semantic routers
- Idealized, centralized scheduler vs increasing # of distributed schedulers
- Observations:
  - Negligible effect of distribution on mean RCTs, which stays within reasonable bounds.
  - Only when load approaches maximum capacity, an increase in mean RCT observed
  - This deterioration grows with the scheduling distribution scale, i.e. a 11% increase for **5** schedulers and 29% increase in RCT for **50** schedulers is observed

Scenario 1b - Scheduling to Instances vs Via Site-Local Load Balancer

- **Aim**: Observe effect of scheduling to a DC-ingress load balancer, compared to scheduling directly to instances
- Certain deployment scenarios may not want to expose the instances directly to network-level routing but use DC-internal mechanisms instead
  - In our evaluation, represented by simple, ‘random’ load balancer at each site ingress
- Observations:
  - Lack of compute awareness at the load balancers has **significant** impact on the mean RCT
  - The impact increases with an increased network load
  - With a network load as low as 30%, the mean RCT of scheduling to sites is almost double than that of directly scheduling to instances, while when the load is 80% of the compute resources, this grows to more than 100 times higher.

Scenario 2 - Comparisons with Existing Network-level Solutions

Aim: Compare CArDS performance against other distributed scheduling mechanisms with respect to:

- factoring compute capabilities into scheduling decision
- performing scheduling at ingress nodes vs at sites
- the impact of the compute unit distribution across sites and instances within sites

Compared against:

- Random scheduler –
  - positioned at ingress nodes
  - compute-unaware
  - performs random load balancing across sites: selects an instance uniformly at random from all the instances of the network
- STEAM [1] –
  - positioned at site-ingress; ingress nodes forward requests to sites uniformly at random
  - compute-unaware
  - uses load estimation and local instance state information in scheduling decision

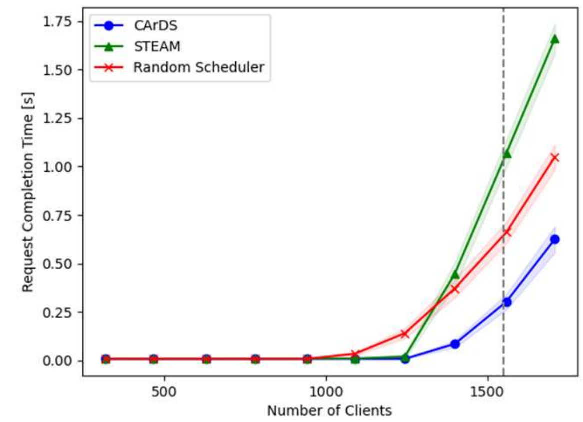
[1] M. Blocher, R. Khalili, L. Wang, and P. Eugster, "Letting off STEAM: ¨Distributed runtime traffic scheduling for service function chaining," in IEEE INFOCOM 2020 - IEEE Conference on Computer Communications, pp. 824–833, 2020.

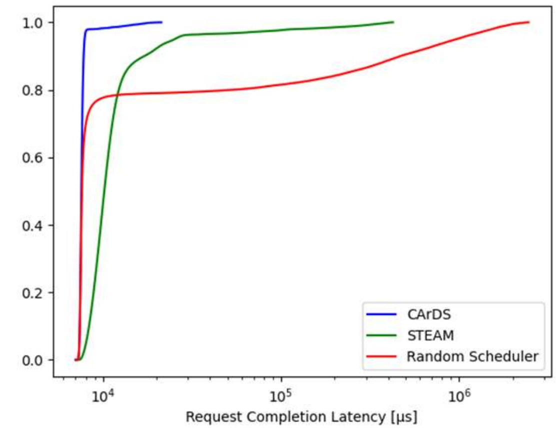
Scenario 2a – Uniform CU Distribution Across and Within Sites

- CArDS significantly **reduces RCT** in high load settings, i.e. > 80%

- CDF at 80% network load (1245 clients) shows:
  - Random Scheduler - tail is very heavy, STEAM - comparatively lighter, but CArDS – very small tail

CArDS able to improve on the average performance in terms of latency as well as significantly reduce the variance

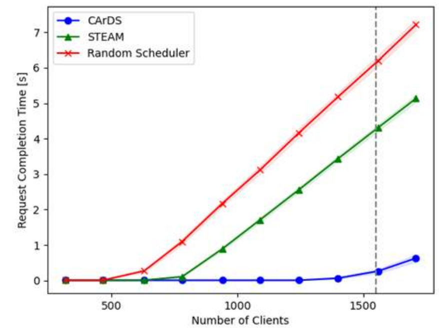


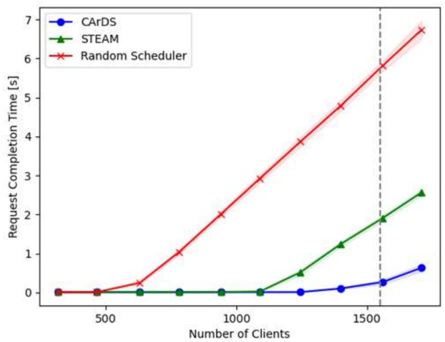


Scenarios 2b & 2c – Imbalance of CU Distribution Across/Within Sites

2b - Imbalance across sites, uniform within site

- STEAM and Random Scheduler both performing very poorly compared to CArDS
  - Lack of compute awareness + site selected uniformly at random by both





2c - Uniform across sites, imbalance within site

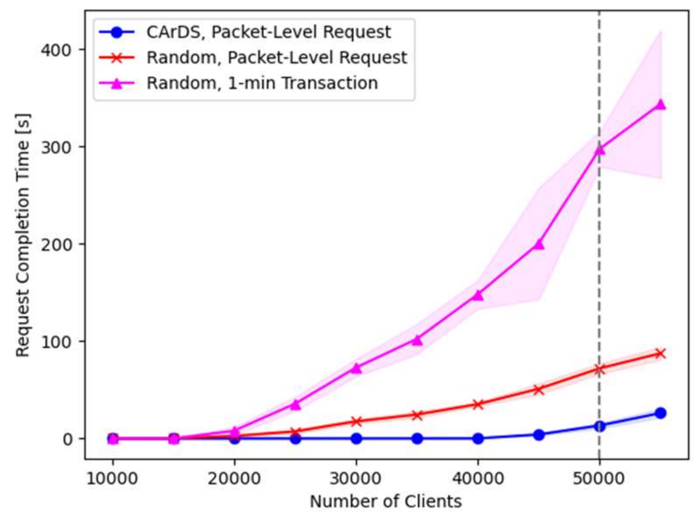
- STEAM able to handle resulting contention within a site performing better than 2b
- Random Scheduler performs as badly as in 2b, unaware of compute capabilities (selecting instance uniformly at random)
- CArDS able to improve on both and maintains performance at a much lower request completion time, even at high loads

Scenario 3 – Use-Case Driven Analysis

Aim: Evaluate performance of CArDS in applications that would benefit from improved RCTs of individual service requests, e.g., content retrieval, compared against existing long-lived approaches

Observations:

- Cases where transactions maintain longer affinities (as in application level solutions), result in high contention and very high RCTs
- Bringing scheduling decision down to packet-level allows for a significant improvement in RCT
- Improvement in overall system utilization, based on assumption of 1.5s as upper bound latency
    - Random Scheduler already able to improve on maximum number of clients that can be served within bound by compared to 1-minute affinity - by 12.5% (almost **2000 more clients**)
    - CArDS able to further improve by serving almost **24000 more clients (~133%)** with the same service completion time compared to the random packet level scheduling
    - CArDS can serve **162%** more clients within bounded latency compared to the long-lived affinity scheduling



Conclusion

- CArDS is a solution to integrate **compute awareness** with the steering of service requests at the data plane level

- This compute-awareness in the scheduling decision leads to **significant performance improvements** in RCT over both network-level and application solutions

- CArDS improves on system utilization by supporting **more than 160% more clients** in a use case with bounded request times, significantly **lowering costs** for service delivery

Follow-up Work

- IFIP Networking paper compared CArDS against two other mechanisms (at L3)
  -> horizontal comparison
- What about using CArDS at different layers of the system, e.g., L3 vs L7?
  -> vertical comparison

Approach:

- Identify defining difference between an L7 and L3 system
  - On-path traffic steering vs off-path (indirection) resolution
  - This compares system in slide 3 (and similar efforts) against, e.g., DNS, GSLB, QUIC_LB, …

Our findings on this vertical comparison will be presented at the upcoming ACM SIGCOMM FIRA workshop

Thank you.