PL4DS

Patrick Eugster (USI)

Shonan Village

May 2019

I Had a Dream…

- … of a PL for all end-to-end DS programming
- But DSs are very “fragmented” — layers, tiers, …
- Nowadays just develop PL techniques for DSs

Ongoing Work

1. *Verification* of DSs, e.g., behavioral typing for DSs, SAT-based smart contracts

2. *Programming models* for DSs, e.g., elasticity programming, switch buffer management

3. *Infrastructure* for DSs, e.g., synchronous communication, network flow shape prediction

Ongoing Work

1. *Verification* of DSs, e.g., behavioral typing for DSs, SAT-based smart contracts

2. *Programming models* for DSs, e.g., elasticity programming, switch buffer management

3. *Infrastructure* for DSs, e.g., synchronous communication, network flow shape prediction

Ongoing Work

1. *Verification* of DSs, e.g., behavioral typing for DSs, SAT-based smart contracts
2. *Programming models* for DSs, e.g., elasticity programming, switch buffer management
3. *Infrastructure* for DSs, e.g., synchronous communication, network flow shape prediction

Ongoing Work

1. *Verification* of DSs, e.g., behavioral typing for DSs, SAT-based smart contracts
2. *Programming models* for DSs, e.g., elasticity programming, switch buffer management
3. *Infrastructure* for DSs, e.g., synchronous communication, network flow shape prediction

Co-design

1. Behavioral Typing

- Multi-party session types (MPSTs)
  - Targeting *middleware* for networked DSs, e.g., Spark
- Select features
  - E.g., no delegation, shared channels
  - Protocol types

Story in 3 Chapters

I. App-level failures (“exceptions”) [FORTE’17]

II. Partial (crash) failures, consensus service [ESOP’18]

III. Failures *integrated* with events, role *sets*, *sub*sessions

Story in 3 Chapters

I. App-level failures (“exceptions”) [FORTE’17]

II. Partial (crash) failures, consensus service [ESOP’18]

III. Failures *integrated* with events, role *sets*, *sub*sessions

$$
\begin{array}{lll}
G_{\text{top}} & ::= & \{C_i(\overline{r_R}, \underline{r_R}, R) = G \text{ with } r_R@r_R.G\}_I \\
G & ::= & \text{spawn } C(\overline{r_R}, \underline{R}, \overline{R}).G \mid r \to \mathbb{r}\{l_i : G_i\}_{i \in I} \mid \mu t.G \mid t \mid \text{end} \\
R & ::= & W \mid M \mid \ldots \qquad \mathbb{r} ::= r_R \mid R
\end{array}
$$

Story in 3 Chapters

I. App-level failures (“exceptions”) [FORTE’17]

II. Partial (crash) failures, consensus service [ESOP’18]

III. Failures *integrated* with events, role *sets*, *sub*sessions

Story in 3 Chapters

I. App-level failures (“exceptions”) [FORTE’17]

II. Partial (crash) failures, consensus service [ESOP’18]

III. Failures *integrated* with events, role *sets*, *sub*sessions

$$
\begin{array}{rcll}
P & ::= & (\overline{H}, c : R) & \text{Event loop} \\
 & | & \text{loop} & c\text{End} \\
 & | & x[y]!l.P & \text{Send} \\
 & | & x[y]?l.P & \text{Receive} \\
 & | & x.\text{spawn } C(\overline{y}) & \text{Spawn} \\
 & \ldots & &
\end{array}
$$

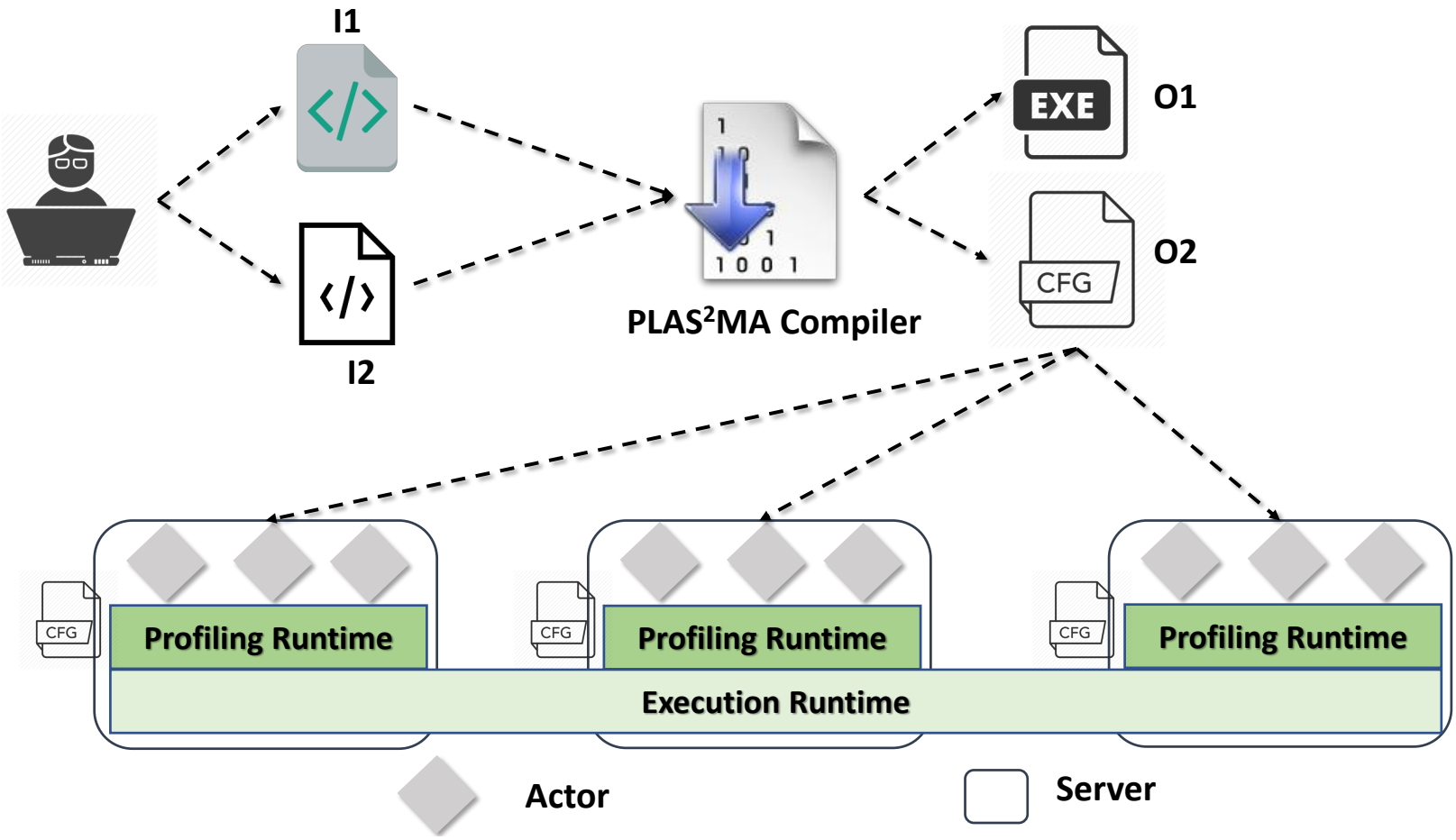
2. Elasticity Programming

- How to develop elastic software systems?
  - Middle tier
- Scalability prerequisite
  - AEON (Atomic Events via Ownership Network) [Middleware'16]
- Fine-grained elasticity, e.g., with actors?

Programmable Elasticity for Stateful Serverless Computing Apps (PLAS^{2}MA)

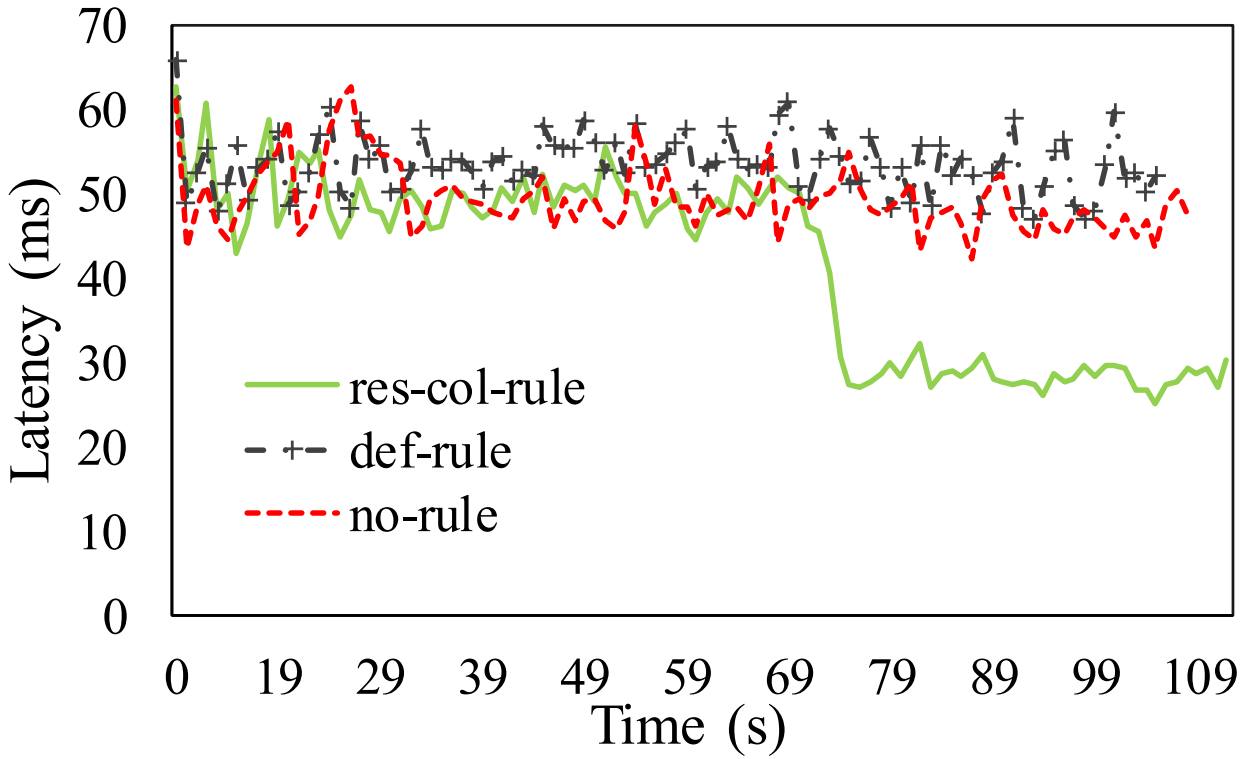
Policy	*pol*	::=	$\overline{rul}$	
Rule	*rul*	::=	$cond \Rightarrow \overline{beh};$	
Actor	*actor*	::=	$atype(var) \mid atype \mid var$	
Actor type	*atype*	::=	$aname \mid$ **any**	
Condition	*cond*	::=	$cond$ **or** $cond \mid cond$ **and** $cond$	
			$\mid$ **true** $\mid feat\ comp\ val$	
			$\mid actor$ **in** **ref**$(actor.pname)$	[F-IA]
Feature	*feat*	::=	$entity.res$	
			$\mid cllr.$**call**$(actor.fname).stat$	[F-IA]
Entity	*entity*	::=	$actor$	[F-RA]
			$\mid$ **server**	[F-RS]
Caller	*cllr*	::=	**client** $\mid actor$	
Statistic	*stat*	::=	**count** $\mid$ **size** $\mid$ **percent**	
Resource	*res*	::=	**cpu** $\mid$ **mem** $\mid$ **net**	
Comparison	*comp*	::=	$< \mid > \mid >= \mid <=$	
Behavior	*beh*	::=	**balance**$(\{\overline{atype}\}, res)$	[R-R]
			$\mid$ **reserve**$(actor,\ res)$	[R-R]
			$\mid$ **colocate**$(actor,\ actor)$	[R-I]
			$\mid$ **separate**$(actor,\ actor)$	[R-I]
			$\mid$ **pin**$(actor)$	[R-I]

PLAS²MA Toolchain



Meta-Data Server

```
server.cpu > 80 and
client.call(Folder(fo).open).percent > 40 and
File(fi) in ref(fo.files) ⇒        reserve(fo, cpu);
                                   colocate(fo, fi);
```

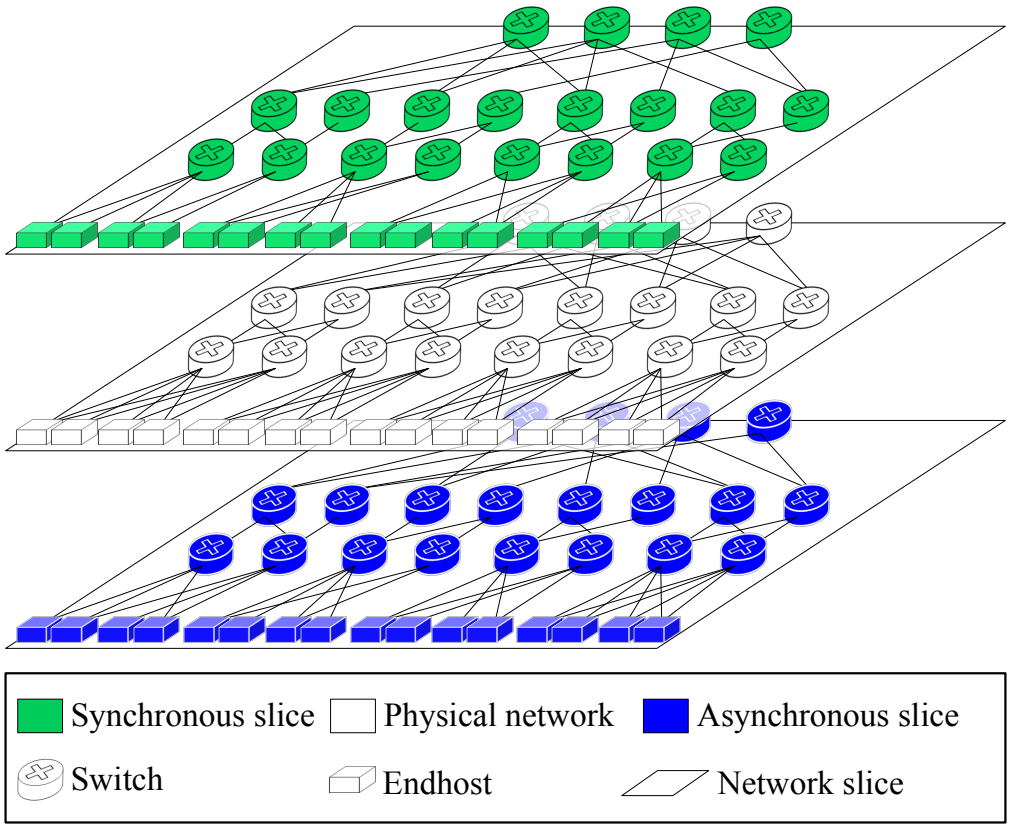


3. Synchronous Communication

- Distributed systems suffer from asynchrony
  - Failure or delay?
- Proposals for overall latency & jitter reduction in datacenter
  - Still “high” latency and/or jitter
  - Not needed for all traffic
- Several proposals for “INP” support for distributed protocols
  - Which one(s)? Ramifications?

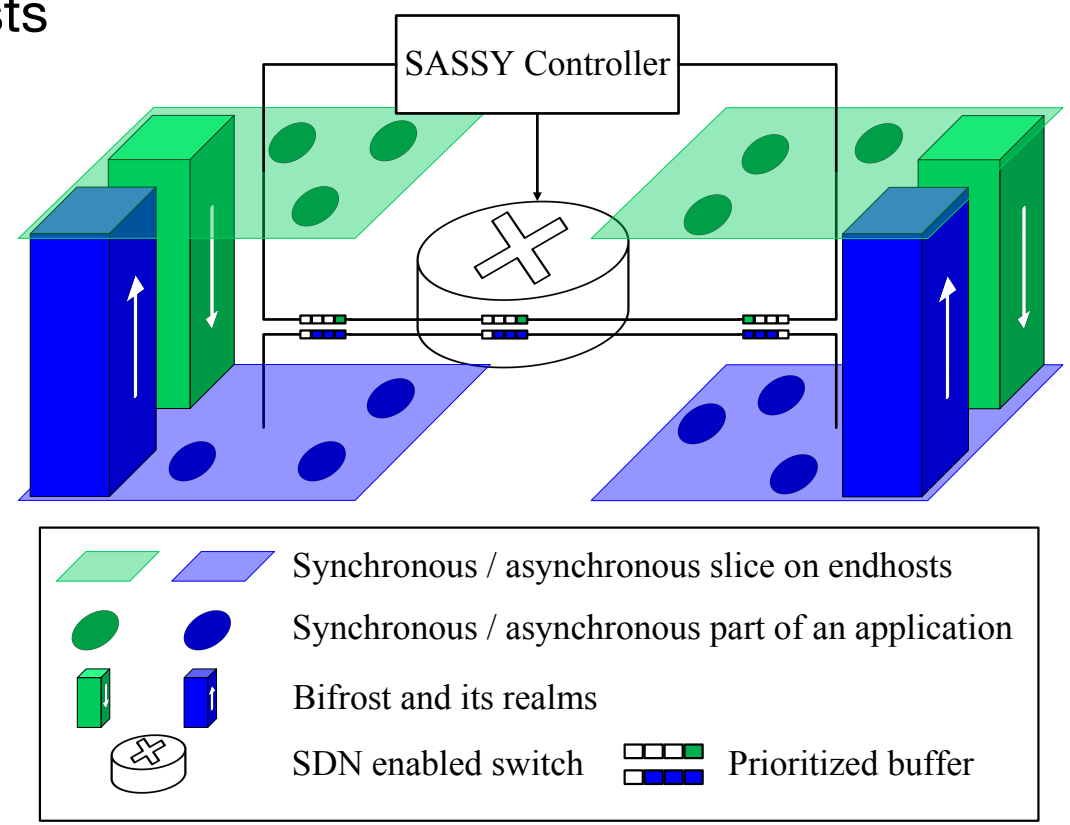
Sliced Network for Asynchronous/ Synchronous Communication (SASSY)

- 2 slices
  1. Latency & deadline-centric “control” traffic
     - No penalty for not using
  2. “Regular” bandwidth-centric traffic
- Commodity HW and SW (Linux)
  - Extensions for SmartNIC



Co-Existing Slices

- API for 1. slice traffic requests
- Switches
  - 2 priorities
    - Cf. buffering
- Links
  - Traffic engineering
- Endhosts
  - Reserve core, mask interrupts, etc.



Synchronous Slice Performance Overview

- SAP datacenter, 17 Intel XEON 52 cores, hyperthreading, 128 GB

- Arista 7280CR-48 switches

Approach	**Latency** (µs)	**Jitter** (ns)	**FFT**
SASSY	5,137	144,052	8,1
SASSY_{sNIC}	4,089	31,995	4,652
SASSY_{0}	5,253	162,478	10,581
QJump	7726,405	7721311,38	4275821,503
DPDK	3831,707	395873,272	33128,168

(Via 1 switch)

Open Questions

- What abstractions are appropriate for what level/layer?
  - How to deal with “layering”?
- Do we need more “OSI layers”?
  - Breaking up the “application layer”?