Programming Elastic Services with AEON and PLASMA

Patrick Eugster
Università della Svizzera italiana (USI)
PL4DS&DDM 2019, Dagstuhl

Work in collaboration with Bo Sang, Srivatsan Ravi, Hui Lu, Gustavo Petri, Masoud Saeida Ardekani, Pierre-Louis Roman

Elastic Services

- Exploit *distributed* cloud (and edge) datacenter resources *at need*
- Benefits service provider
- Benefits cloud provider

Landscape

- Autoscalers
  - Focus mostly on middle tier
  - Straightforward based on containers, VMs etc.
- AWS Lambda et al.
  - Stateless functions
- Specifically designed elastic services
  - E.g. map/reduce, graph processing, key/value storage
- More generic?

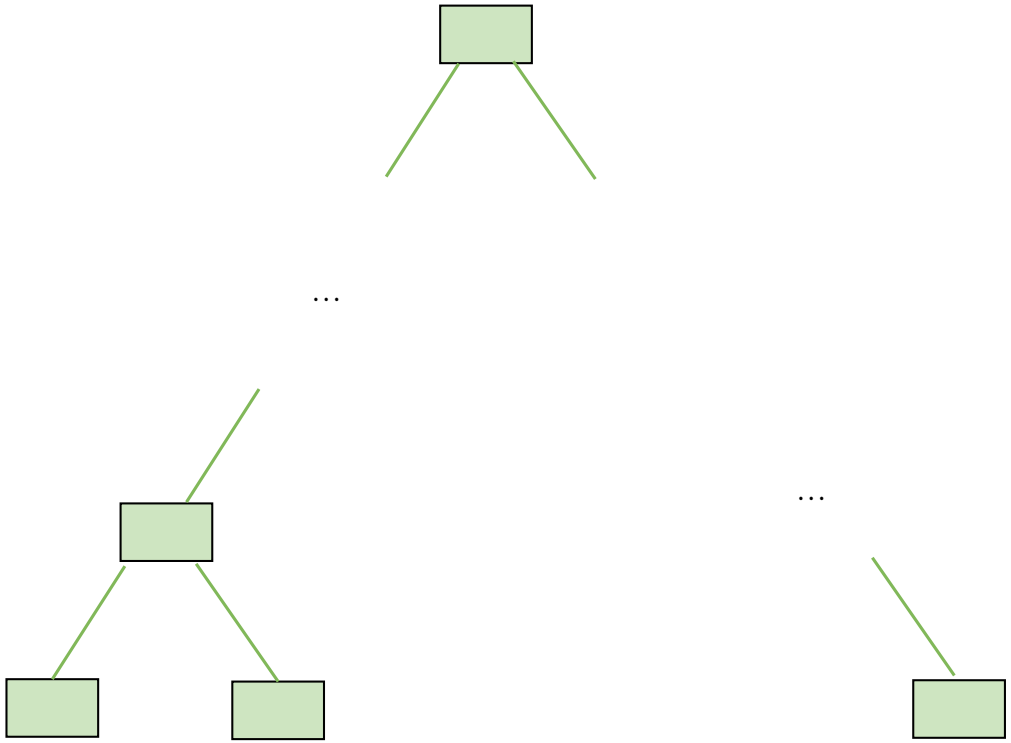
Roadmap

- Motivation
- AEON: Scalable serializable actor programming
- PLAS^{2}MA: Elasticity programming

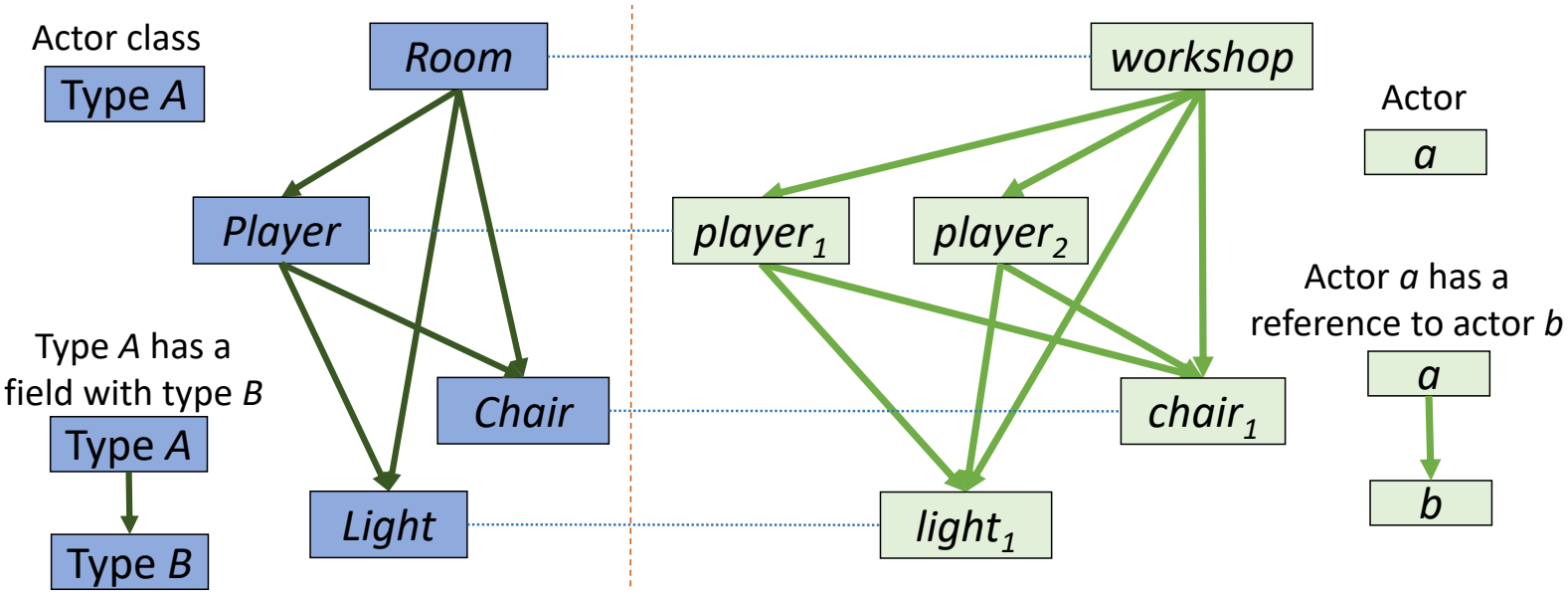
Actors and Events Ownership Network (AEON)

- Scalability is prerequisite for elasticity
- Actors to the rescue?
  - Scalable, familiar
  - But… little help for reasoning beyond single messages
  - Isolation
- Scalability vs consistency in distributed systems

Tree Intuition



Actor DAG



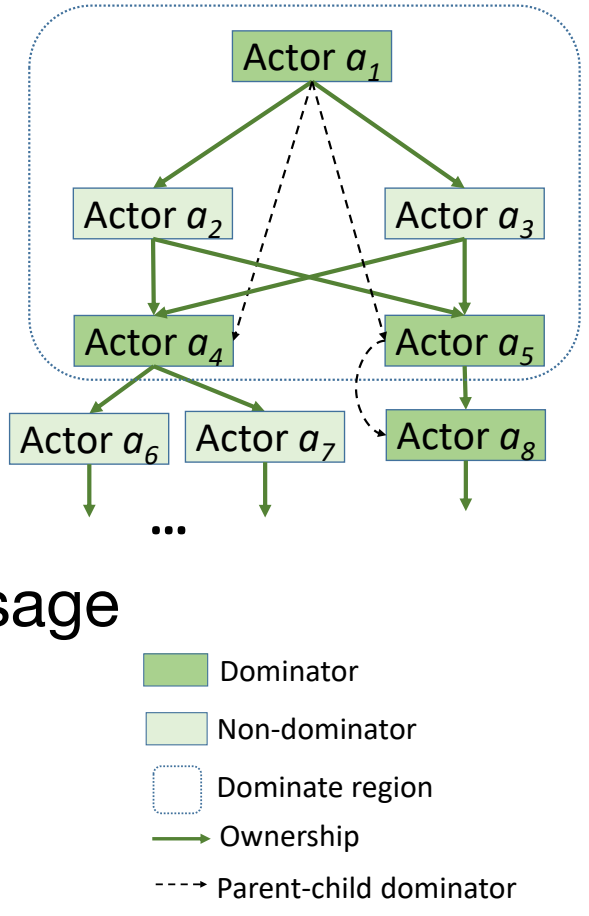
Restricted actor (interaction) graph

DAG Enforcement

- Ownership-style typing
  - Multiple owners
- Type-based program analysis
  - Program variables
- Leveraged for serializability and deadlock freedom

Leveraging the DAG

- Top-down locking
  - Cf. “hand-over-hand” locking
- Dominator
  - “LUB” for all actors involved in a message
  - Dominate region, queues
- Can use also for checkpointing



Programming Model

- Messages tagged at callsite
  - Events (no return) — external vs internal
  - Synchronous methods — default
  - Asynchronous methods (no return)
- Practical extensions
  - Calls outside DAG via yield fields — (sub)events
  - Readonly
  - Direct recursion

Mandatory Code Snippet

```
actorclass Room {
  // players in this room
  vector<Player> players;
  vector<Chair> chairs;
  vector<Light> lights;
  ...
  void turnOnLights() {
    // turn on lights in parallel
    for(int i=0 upto lights.size())
      async lights[i].turnOn();
  }
  void chairColorUpdate(...) {...}
}

actorclass Light {
  bool isOn;
  ...
  void turnOn() { isOn = true; }
  void turnOff() { isOn = false; }
}

actorclass Chair {
  ...
  void paint(...) {...}
  void putBag(...) {...}
}
```

```
class Role {...}

actorclass Player {
  Light light;
  Chair chair;
  Role role;
  yield Room room;
  vector<Chair> chairs;
  ...
  void cleanLight() {
    chair.putBag(...);
    light.turnOff();
  }
  void paintChair() {
    light.turnOn();
    chair.paint(...);
    event room.chairColorUpdate(...);
  }
  void paintChairs(...) {
    for(int i=0 upto chairs.size())
      // 3 options
      async chairs[i].paint(...);
      // chairs[i].paint(...);
      // event chairs[i].paint(...);
  }
}
```

Capsule Semantics

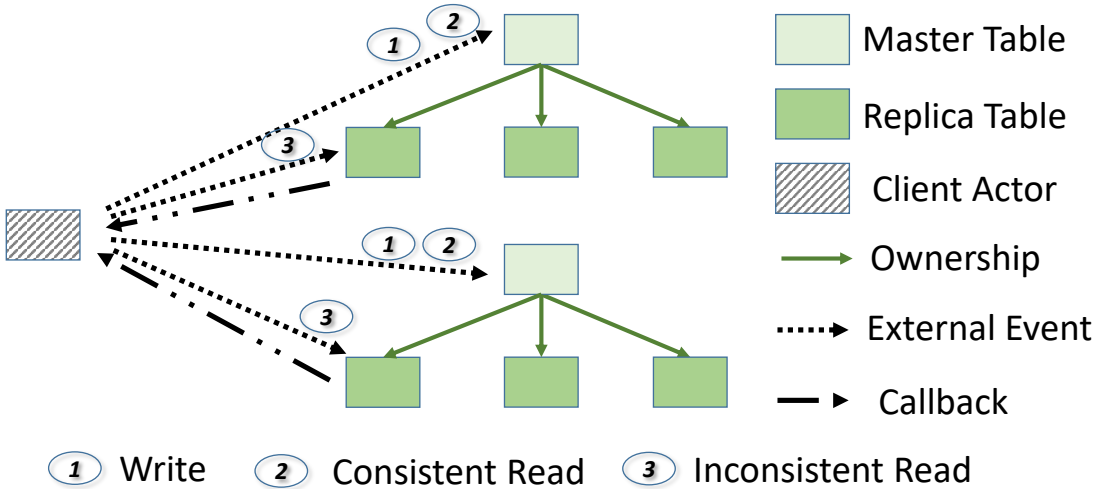
Caller	Call ($dc^?$)	Field (`yield`$^?$)	Method (`ro`$^?$)	Result	Blocking	Execution
External	`event`	`yield`	Any	Callback	Non-blocking	Serialized
Internal non-`ro`	`event`	Any	Any	Callback	Non-blocking	Serialized
Internal `ro`	`event`	Any	`ro`	Callback	Non-blocking	Serialized
Internal non-`ro`	Sync	Non-`yield`	Any	Return (T)	Blocking	Atomic
Internal non-`ro`	`async`	Non-`yield`	Any	Callback	Non-blocking	Atomic
Internal `ro`	Sync	Non-`yield`	`ro`	Return (T)	Blocking	Atomic
Internal `ro`	`async`	Non-`yield`	`ro`	Callback	Non-blocking	Atomic

Experiences

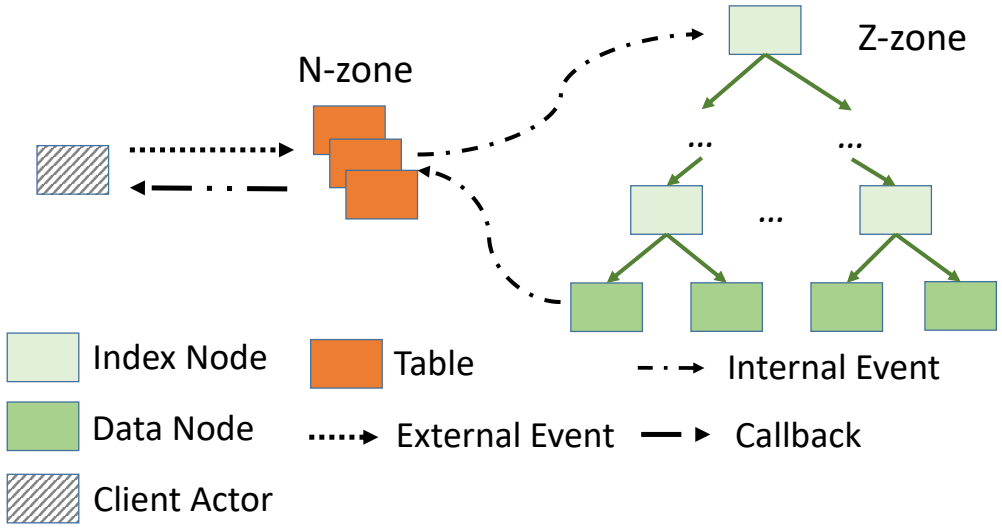
Application	Type	Ownership constraint	Multiple clusters	Uses `ro`	Uses `async`	Uses int. events	Actor classes	LoC
Halo Galactic	Game	None	Yes	No	No	Yes	3	313
Halo Presence	Game	None	Yes	No	No	Yes	3	287
Multi-player game	Game	DAG	No	No	No	No	4	564
Chirper	Social net	None	No	No	No	Yes	2	118
Graph engine	Computation	None	No	No	No	Yes	2	469
MapReduce	Computation	None	No	No	No	Yes	3	390
TPC-W-mini	Benchmark	None	No	No	No	Yes	2	236
TPC-C	Benchmark	DAG	No	Yes	Yes	No	7	1798
zExpander	Caching	Tree	Yes	Yes	Yes	Yes	3	506
Memcached	Caching	None	No	Yes	No	No	1	220
BigTable	Store	Tree	No	Yes	Yes	No	3	216
Cassandra	Store	Tree	No	Yes	Yes	No	2	221
Metadata store	Store	Tree	No	Yes	Yes	No	2	552
Silo	Store	DAG	No	Yes	Yes	No	7	1899
Bank account	Finance	Tree	No	No	No	No	4	176
Skip list	Data structure	DAG	No	Yes	Yes	No	2	322
LSM	Data structure	Tree	Yes	Yes	Yes	Yes	2	1587
B+ tree	Data structure	Tree	No	Yes	Yes	No	2	1457
B tree	Data structure	Tree	No	Yes	Yes	No	2	1437
Piazza	Course management	DAG	No	Yes	Yes	No	5	259

Examples

- Cassandra

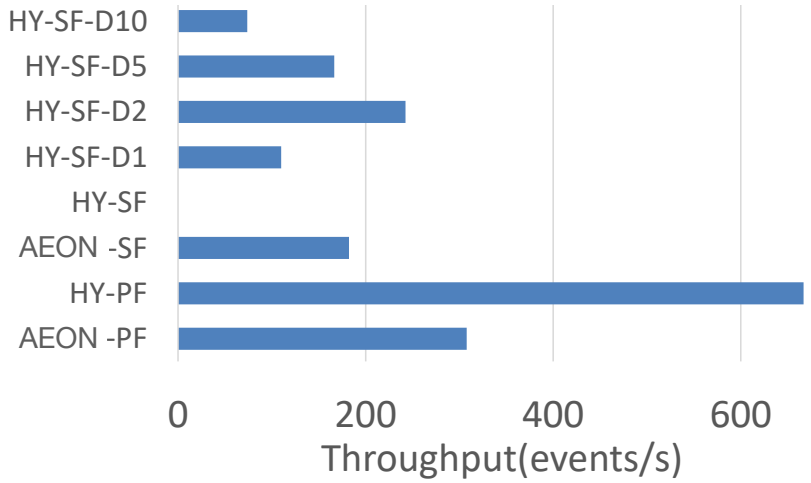


- zExpander

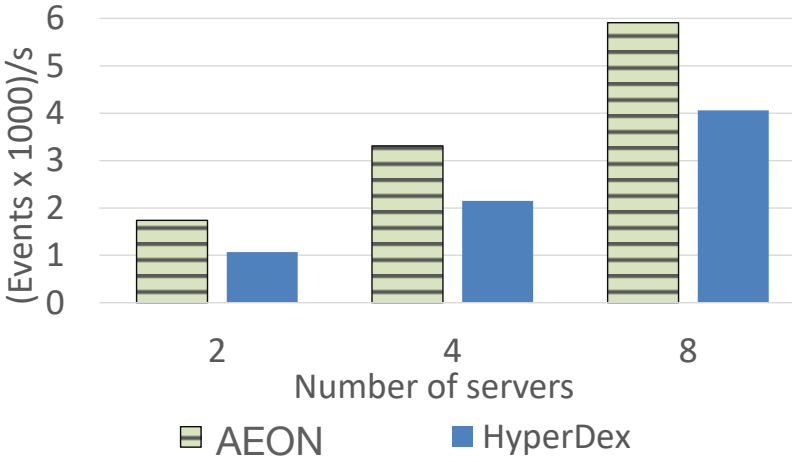


Performance

- AEON vs HyperDex Warp
  - Metadata server for Warp transactional file system



File creation

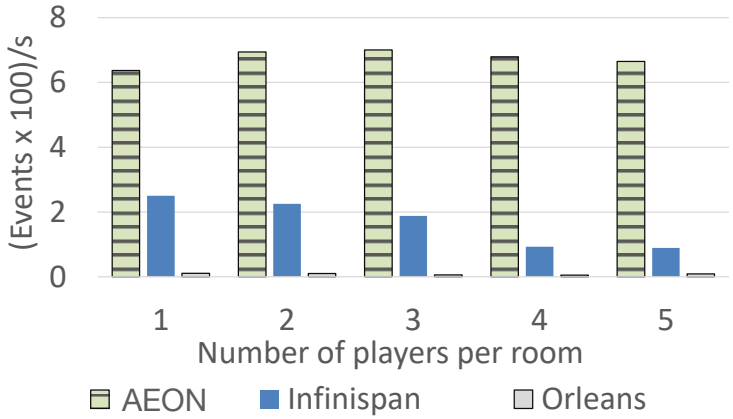


Open private inode

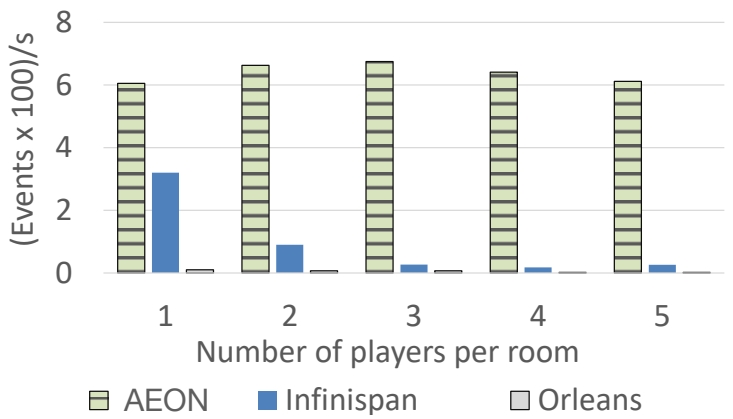
Performance (2)

- Game application

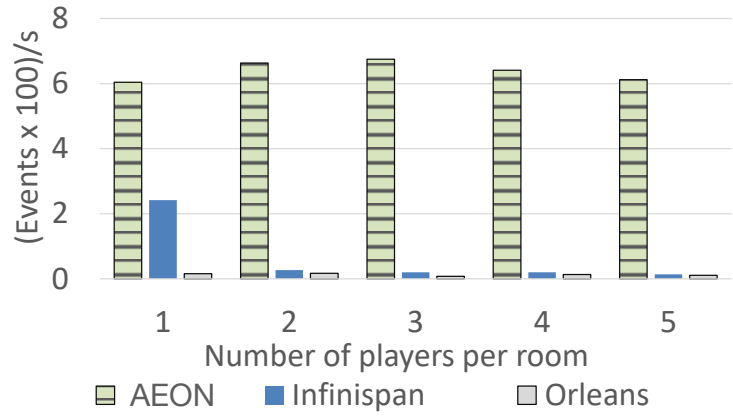
100% UseItem



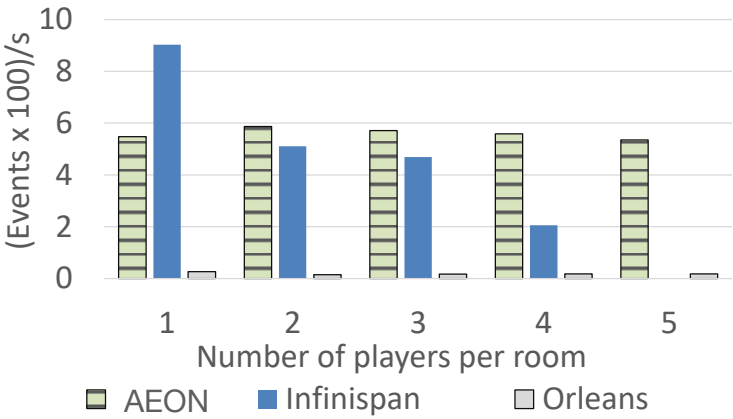
80% UseItem
20% NewItem



50% UseItem
50% NewItem



100% NewItem



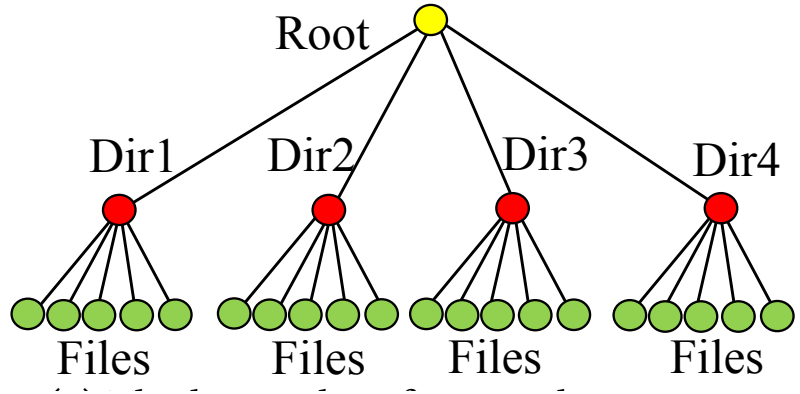
Programmable Elasticity for Stateful Serverless Computing Applications (PLAS^{2}MA)

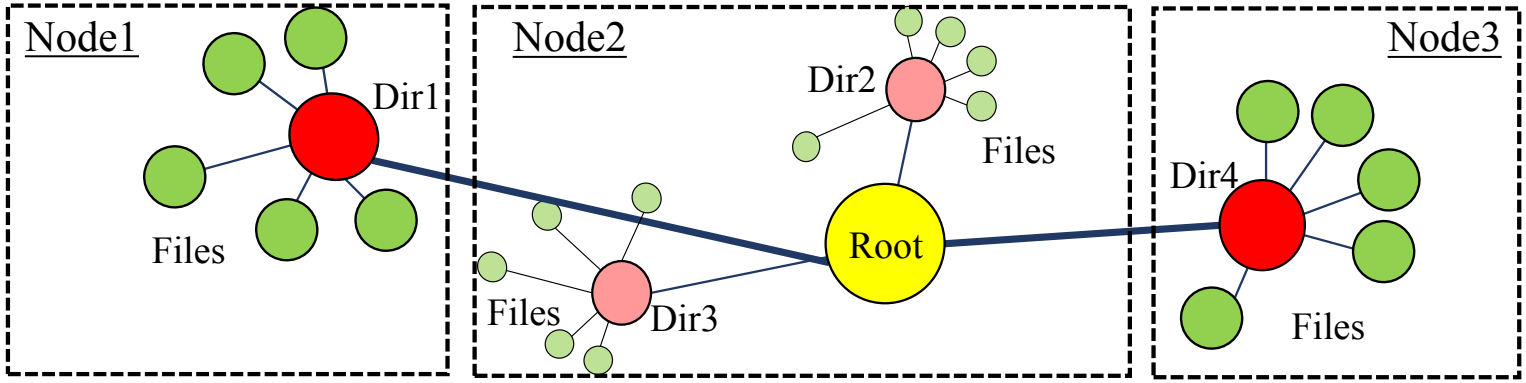
☑ Scalable distributed programming model

☑ Migration

- How to know when/what to migrate?
  - Automated approaches for VMs etc.

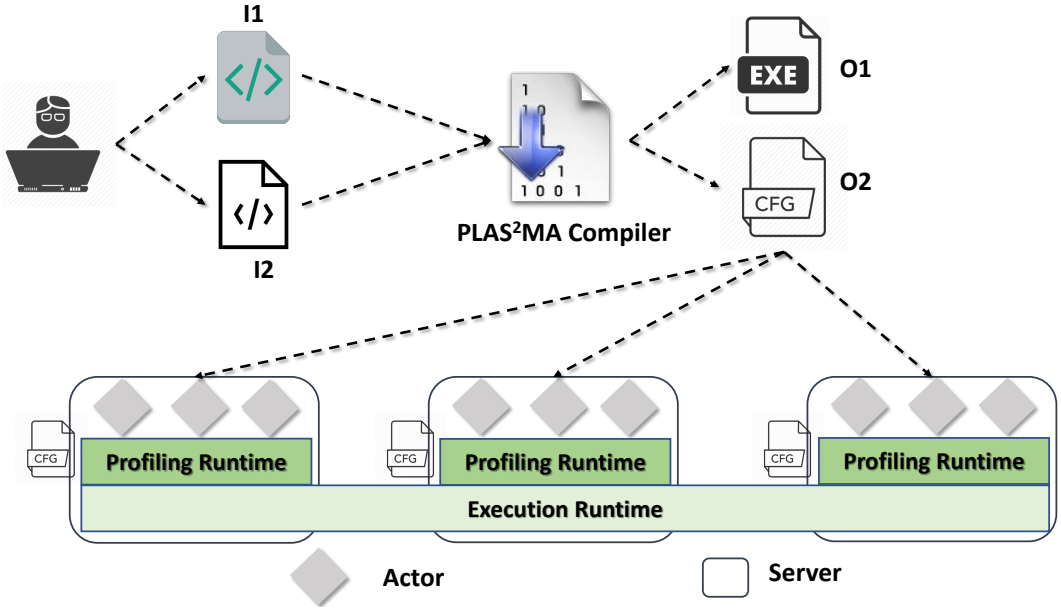
Metadata Server Revisited





2-Level Programming

1. Application programming (e.g., AEON)
2. Elasticity programming
   - Assumes actor types



- Elasticity rules

  [R-R] Resource elasticity rules

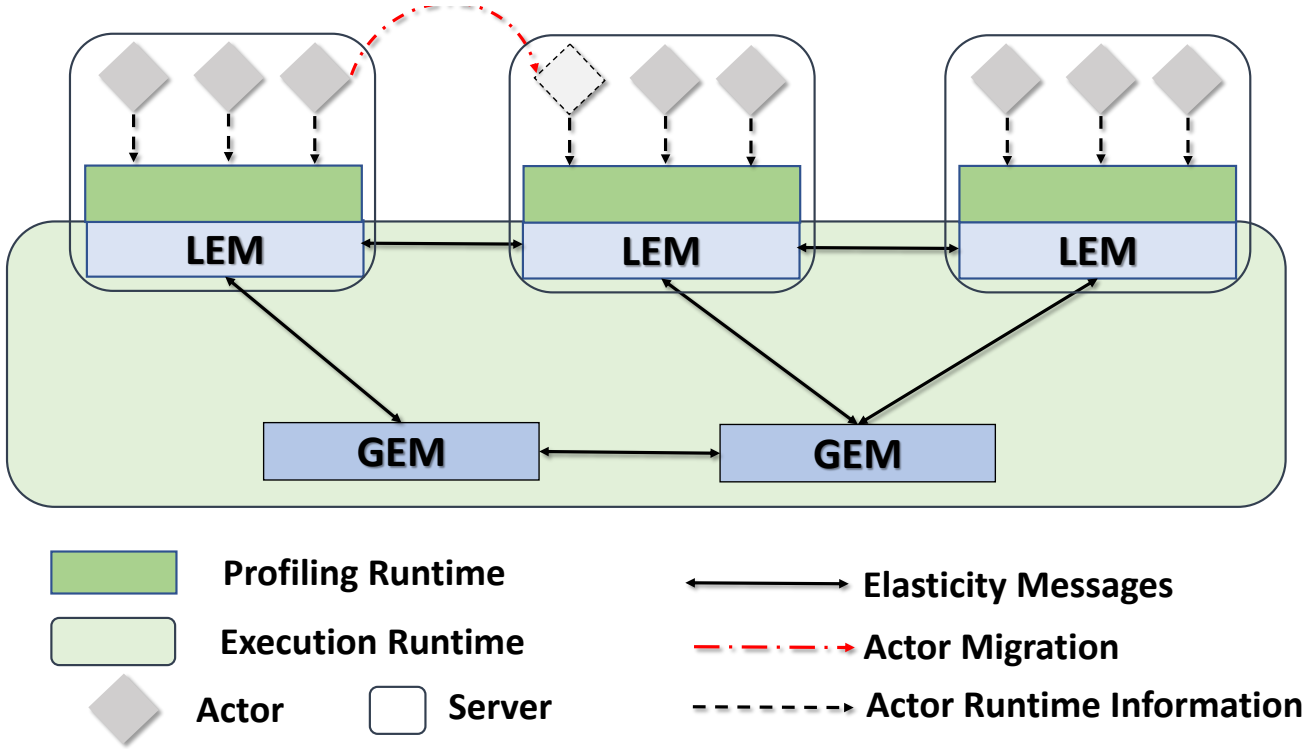
  - Actor (e.g., CPU usage) and server resources (e.g., network usage)

  [R-I] (Actor) interaction elasticity rules (e.g., message rate)

Elasticity Programming

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

Runtime System

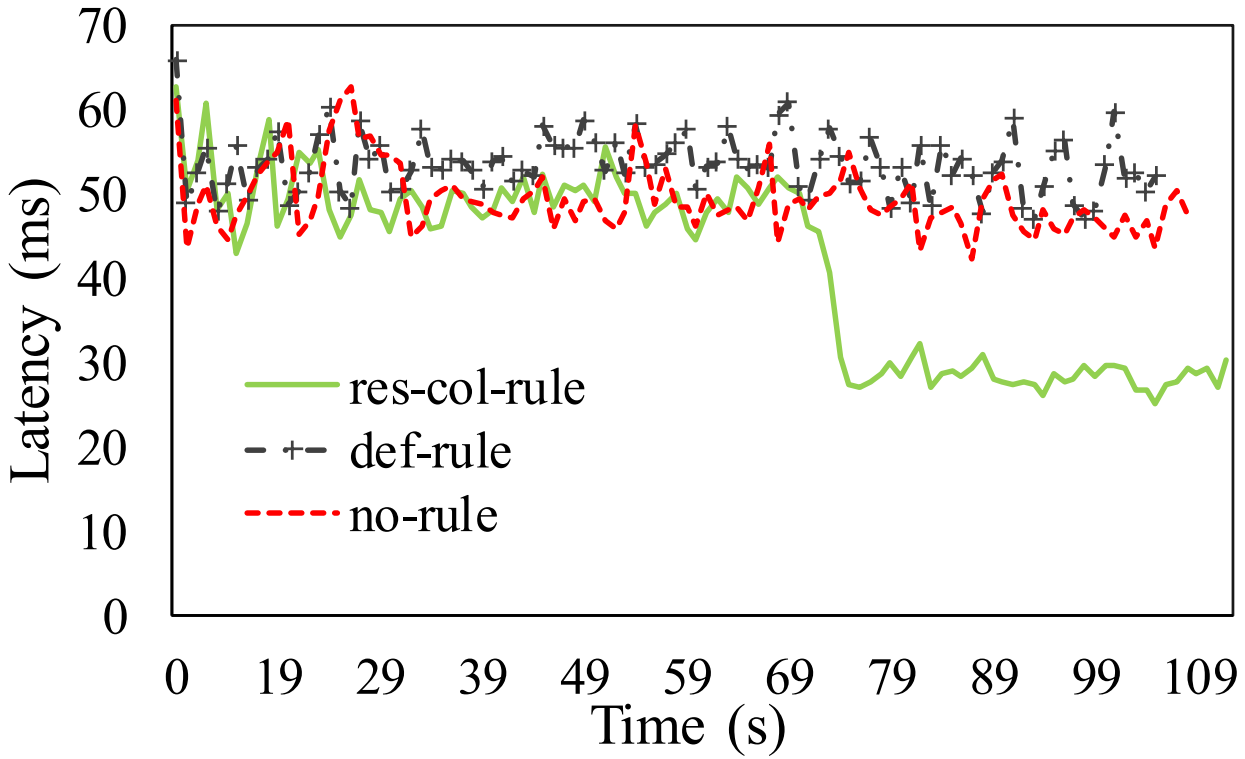


Experiences

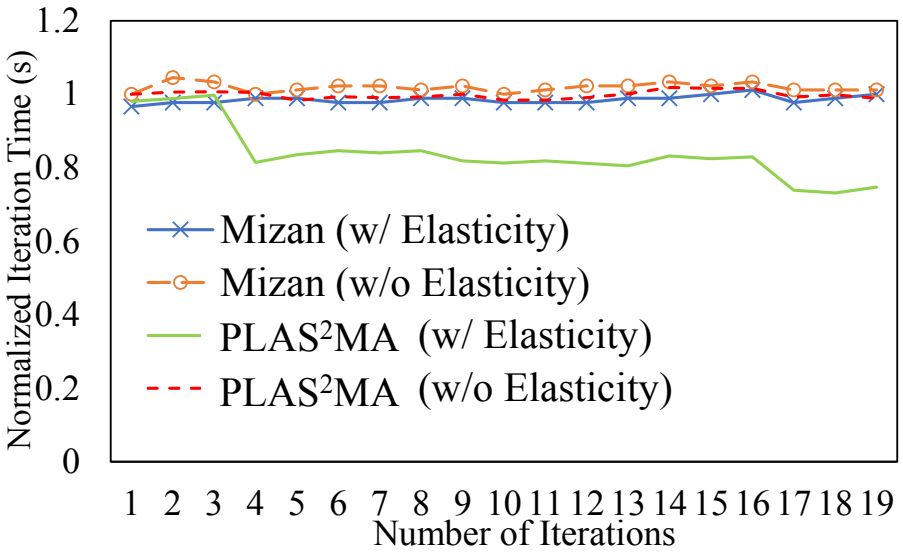
Application	LoC	Elasticity rules (and number of rules)
Metadata Server	253	**1.** Colocate Folder with Files in it (since they are accessed together)
PageRank	465	**1.** Balance CPU workload
Halo Presence Service	287	1. Balance CPU workload of Routers **2.** Colocate Session with Players in it
Media Service	756	1. Balance network workload for FrontEnds 2. Provide VideoStream with enough CPU 3. Colocate linked VideoStream and UserInfo 4. Avoid migrating MovieReview **5.** Balance CPU workload of ReviewChecker
B+ tree	1457	1. Colocate parent-child inner nodes **2.** Put leaf nodes on separate servers
Piccolo	–	1. Balance CPU workload for Workers **2.** Colocate Worker and Table that Worker reads data from
E-Store	645	1. Put hot Keys on idle servers **2.** Colocate parent-child Keys
zExpander	–	**1.** Put leaf nodes on idle servers
Cassandra	–	**1.** Put table replicas on different servers

Metadata Server

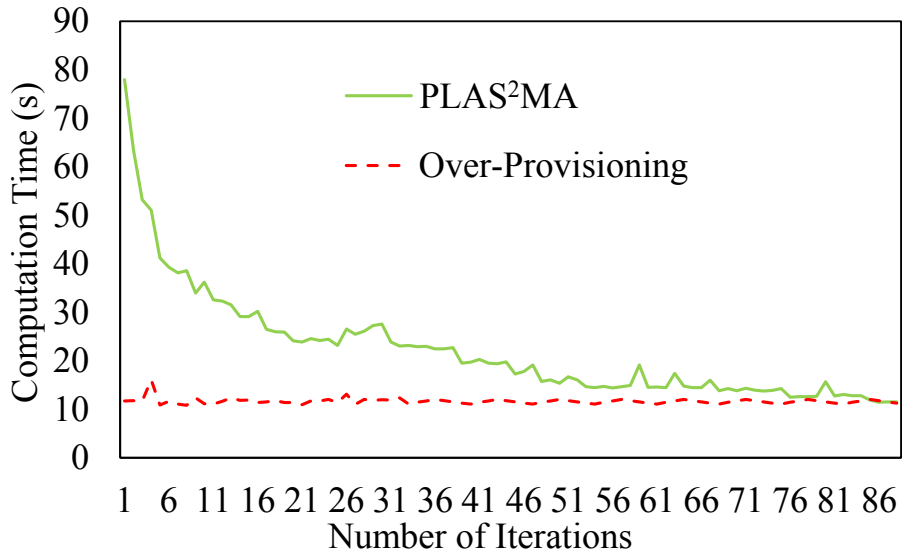
```
server.cpu.perc > 80 and
client.call(Folder(fo).open).perc > 40 and
File(fi) in ref(fo.files) ⇒
    reserve(fo, cpu); colocate(fo, fi);
```



PageRank



vs Mizan



vs over-provisioning

Conclusions

- AEON: combines scalability and serializability in a well-known programming model with pragmatic tradeoffs
- PLAS^{2}MA: fine-grained elasticity with few programmer instructions
- Ongoing
  - Extensions, e.g., futures
  - Livelocks with ownership updates
  - High availability