A Reference Architecture for Datacenter Scheduling

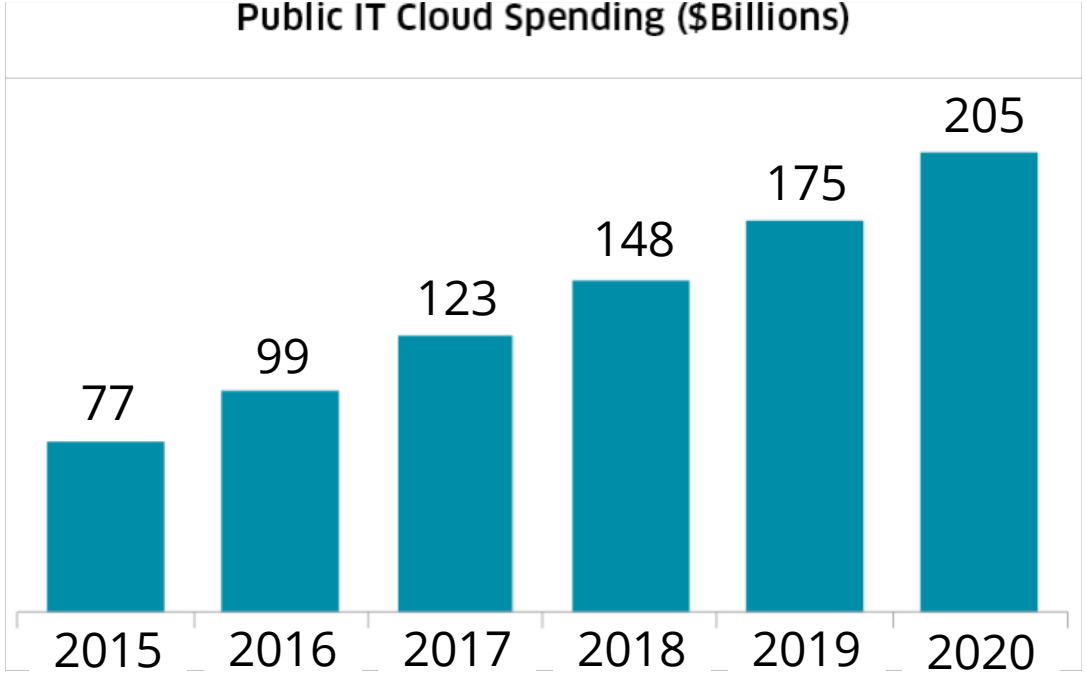
Design, Validation, and Experiments

Georgios Andreadis, Laurens Versluis, Fabian Mastenbroek, Alexandru Iosup

g.andreadis@atlarge-research.com
Delft University of Technology, Vrije Universiteit Amsterdam

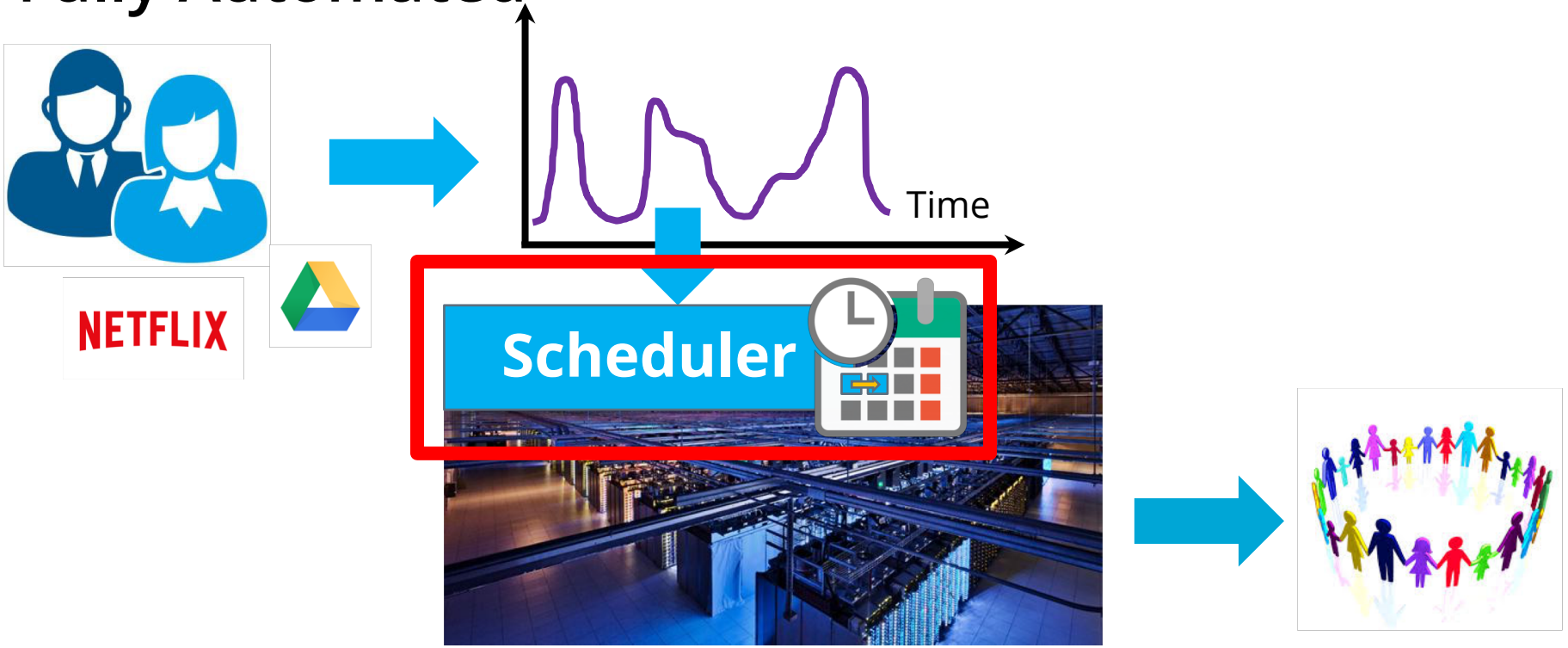
Delft > The Netherlands > Europe

Cloud Infrastructure is Vital to Digital Society



Source: https://business.nasdaq.com/marketinsite/2017/Cloud-Computing-Industry-Report-and-Investment-Case.html, 2018-11-02

Challenge: Manage Resources Efficiently & Fully Automated



Scheduling is Hard

"30—70% scheduler decisions incorrect in datacenters"

Source: IEEE Computer '15

"current schedulers not efficient for many users, diverse services"

Source: Dutch industry, CCGRID '15

→ Need **Smarter** Schedulers

"new schedulers not used in datacenters, fear of failure"

Source: Euro-Par '16

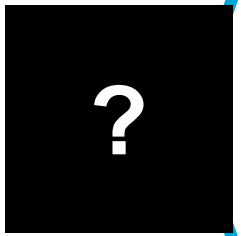
→ Need Scheduler **Reproducibility**

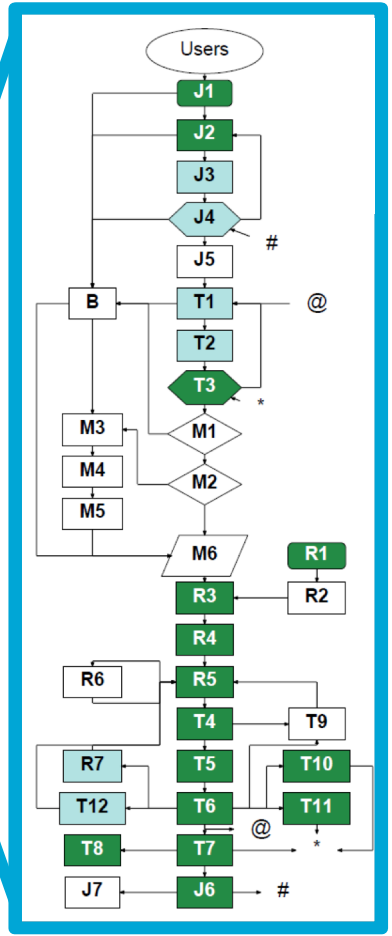
Previous Approaches vs. Our Solution

- Current models stay at black-box level
- Scheduler do many things
- Difficult to understand and compare
- Need for a common language

In this work:

1. Design a **reference architecture**
2. **Map** existing schedulers
3. Conduct **experiments**



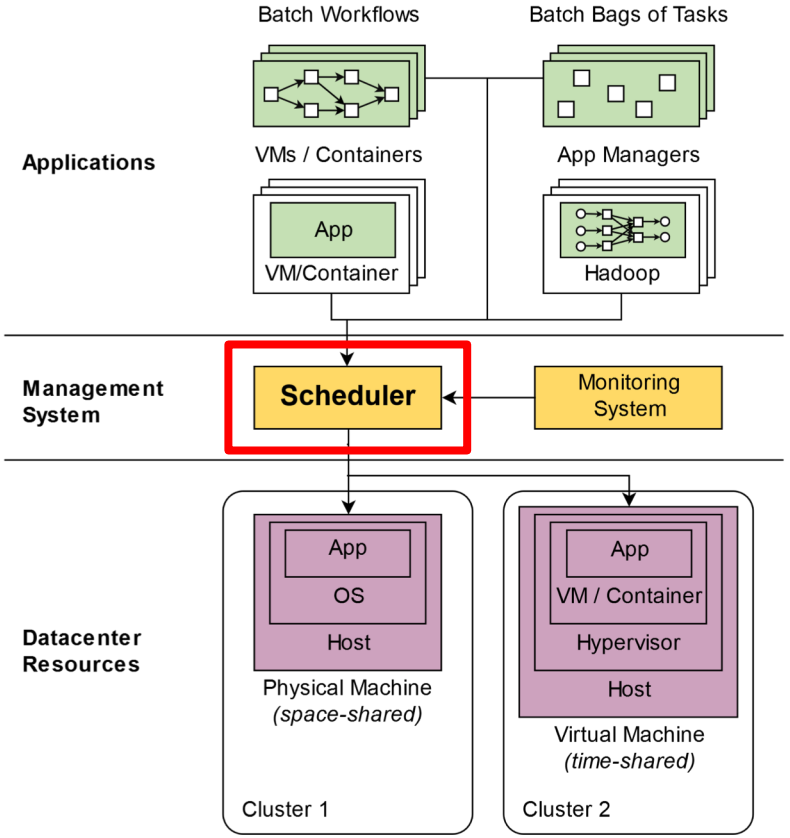


Closely Related Work

- Scheduling architectures
  - Schopf (10 steps when grid-scheduling): Main inspiration
  - Surveys (Rodriguez et al., Singh et al.): More coarse-grained and lacking features
- Architectures of cloud systems
  - Complements NIST Cloud and Big Data RAs
  - Fits in the ISO/IEC/IEEE 42010:2011 standard
- Large-scale software architectures
  - Rozanski et al., Bass et al.: Theory of Software Architectures

System Model

- Workflow-compliant workloads
- Users can specify requirements
- Physical and virtual resources
- The scheduler…
  - Allocates
  - Provisions
  - Replicates
  - Migrates
  - …



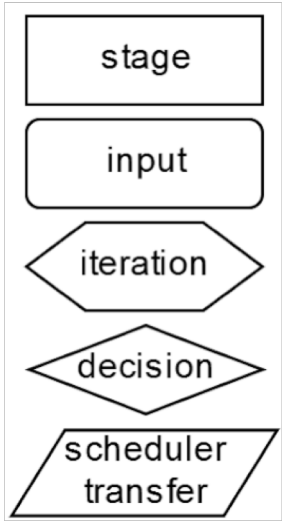
Design
Validation
Experiments

Requirements and Goals

- Validity
  - Accurately represents the field
  - Verify through mapping and peer-review
- Usefulness
  - Its utility to stakeholders
  - Empirical results from mapping and experiments

Specification: Our Workflow-based Model for Datacenter Scheduling

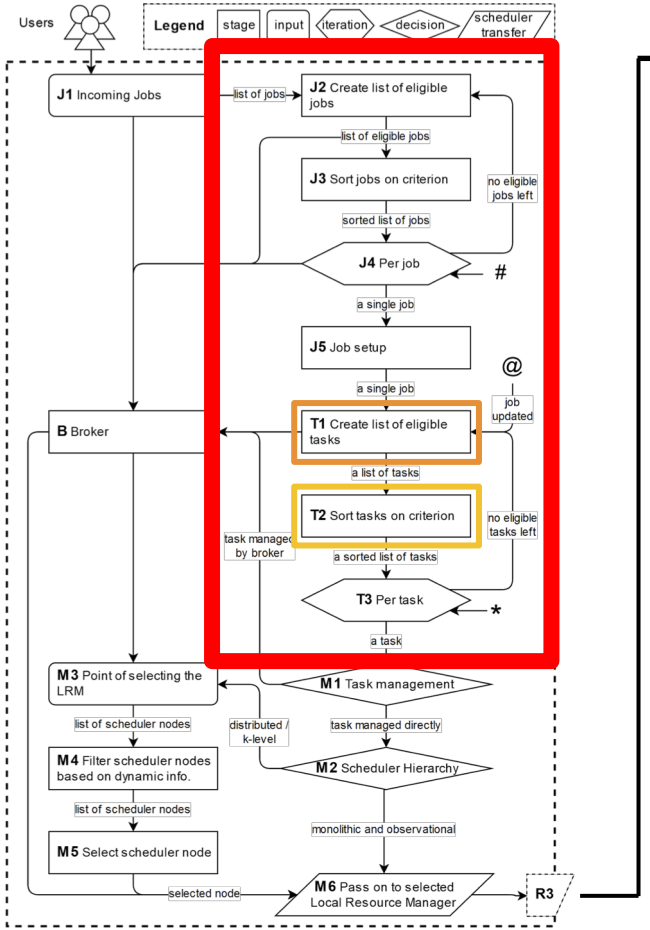
- 33 stages with control and data flows between them
- 5 modes of operation
- Support hierarchy of schedulers
- Stages can be divided into 4 groups:
  - Job processing (**J**)
  - Task processing (**T**)
  - Scheduler management (**M**)
  - Resource management (**R**)

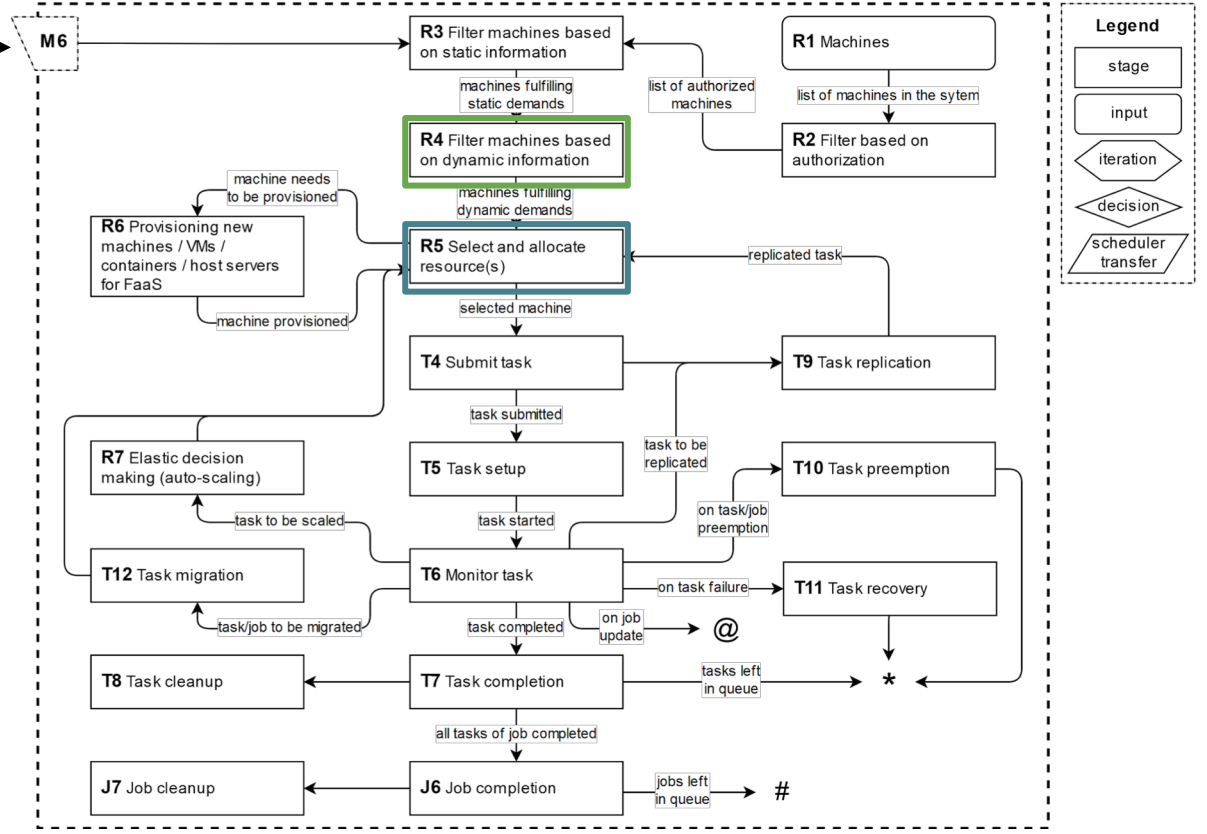


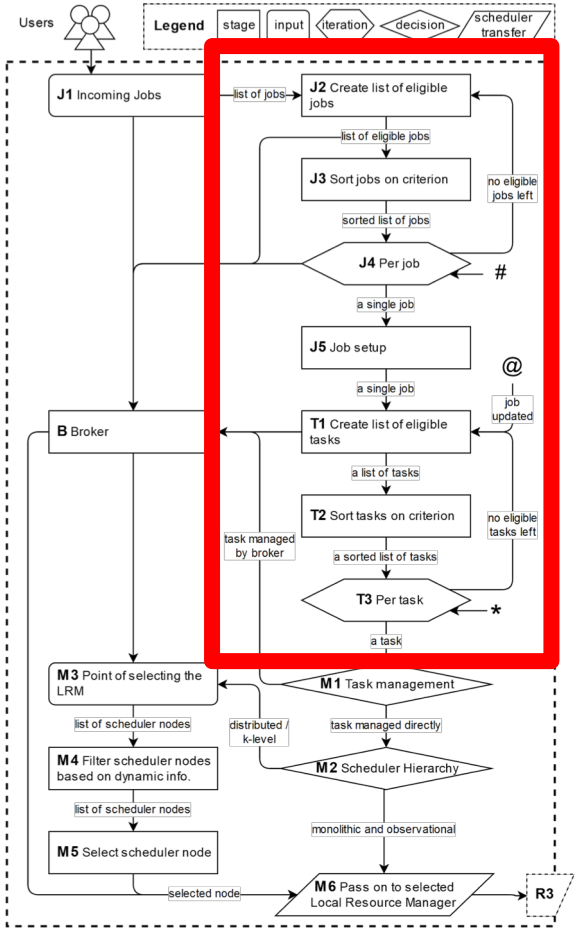
The 5 modes of operation for scheduling stages

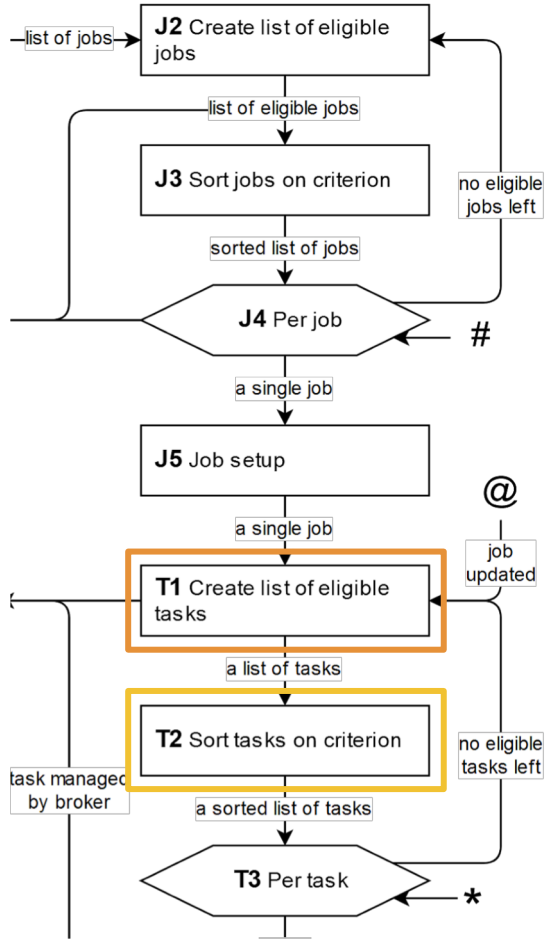
Process of Designing the Reference Architecture

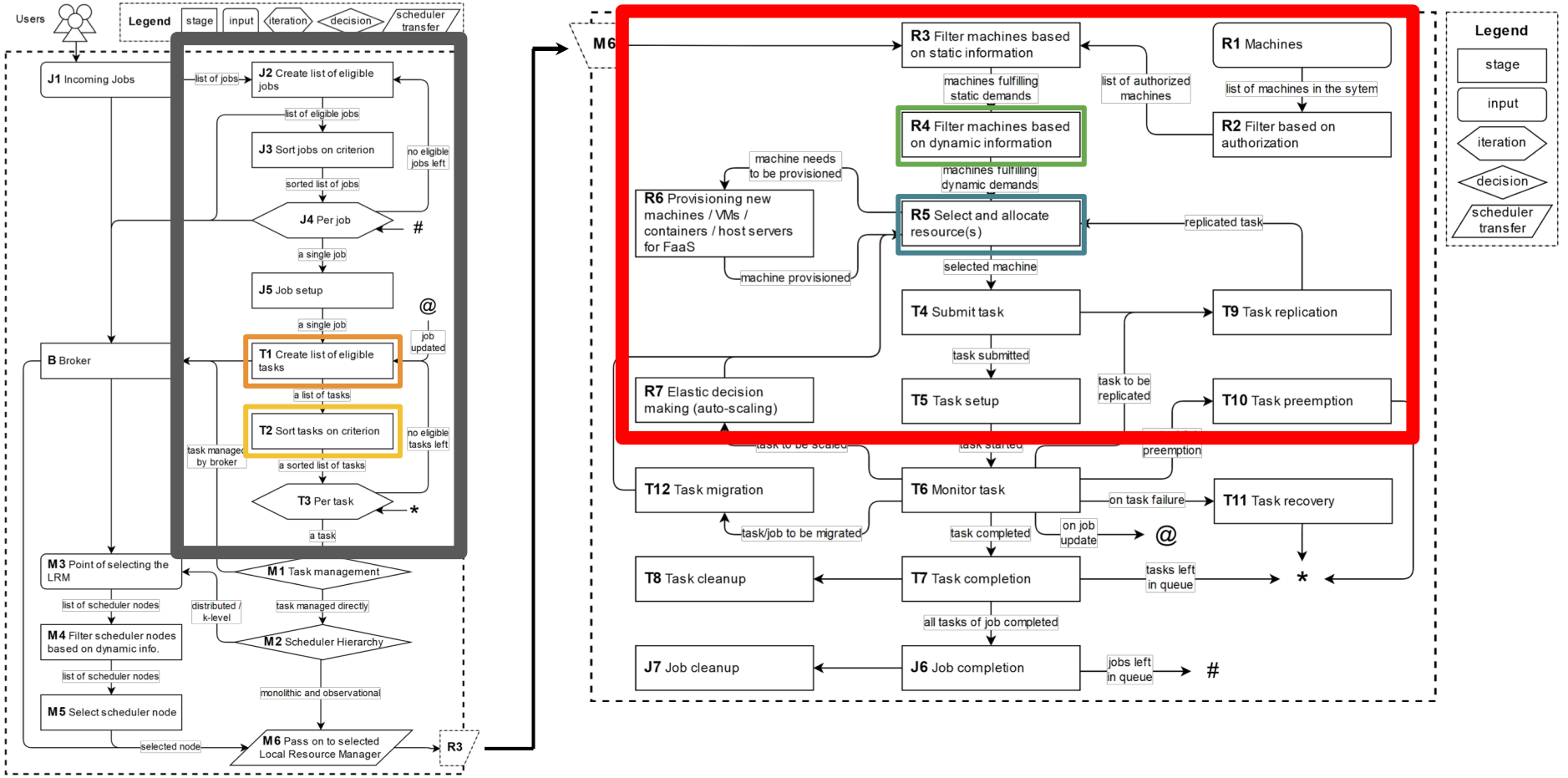
- Goal: Comprehensiveness
- Publications selected:
  - Scheduling core analyzed and presented
  - Highly cited or deployed in a large real-world environment
- Leading principles in this process:
  - Components with Clearly Distinct Responsibilities
  - Separation of Mechanism from Policy

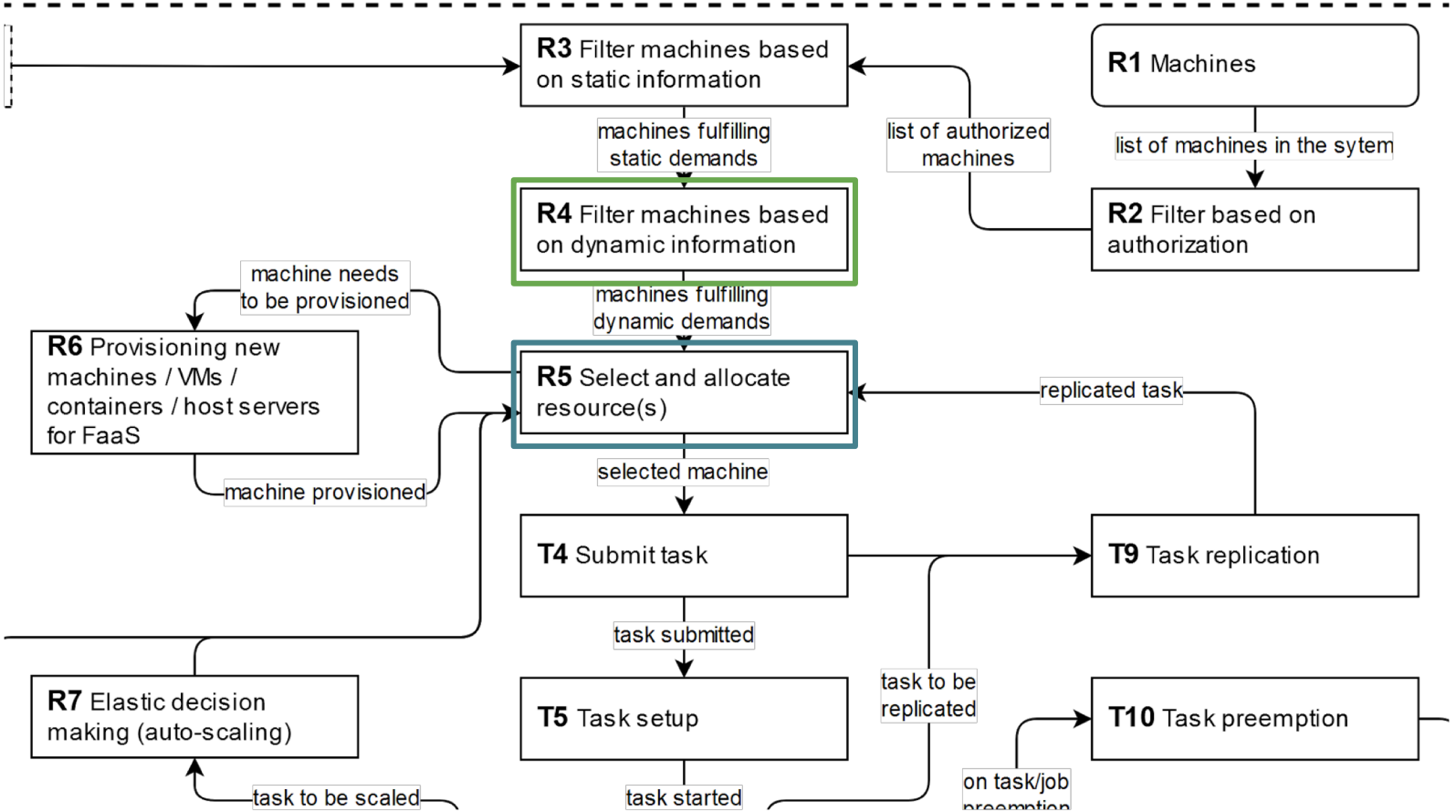

Users
Legend
stage
input
iteration
decision
scheduler transfer
J1 Incoming Jobs
list of jobs
J2 Create list of eligible jobs
list of eligible jobs
J3 Sort jobs on criterion
no eligible jobs left
sorted list of jobs
J4 Per job
#
a single job
J5 Job setup
@
a single job
job updated
B Broker
T1 Create list of eligible tasks
a list of tasks
T2 Sort tasks on criterion
no eligible tasks left
task manager by broker
a sorted list of tasks
T3 Per task
*
a task
M3 Point of selecting the LRM
M1 Task management
list of scheduler nodes
distributed / k-level
task managed directly
M4 Filter scheduler nodes based on dynamic info.
M2 Scheduler Hierarchy
list of scheduler nodes
monolithic and observational
M5 Select scheduler node
selected node
M6 Pass on to selected Local Resource Manager
R3


M6
R3 Filter machines based on static information
R1 Machines
machines fulfilling static demands
list of authorized machines
list of machines in the sytem
R4 Filter machines based on dynamic information
R2 Filter based on authorization
machine needs to be provisioned
machines fulfilling dynamic demands
R6 Provisioning new machines / VMs / containers / host servers for FaaS
R5 Select and allocate resource(s)
replicated task
machine provisioned
selected machine
T4 Submit task
T9 Task replication
task submitted
R7 Elastic decision making (auto-scaling)
T5 Task setup
task to be replicated
T10 Task preemption
task to be scaled
task started
on task/job preemption
T12 Task migration
T6 Monitor task
on task failure
T11 Task recovery
task/job to be migrated
task completed
on job update
@
T8 Task cleanup
T7 Task completion
tasks left in queue
*
all tasks of job completed
J7 Job cleanup
J6 Job completion
jobs left in queue
#
Legend
stage
input
iteration
decision
scheduler transfer


Users
Legend
stage
input
iteration
decision
scheduler transfer
J1 Incoming Jobs
list of jobs
J2 Create list of eligible jobs
list of eligible jobs
J3 Sort jobs on criterion
no eligible jobs left
sorted list of jobs
J4 Per job
#
a single job
J5 Job setup
@
job updated
a single job
B Broker
T1 Create list of eligible tasks
a list of tasks
T2 Sort tasks on criterion
no eligible tasks left
task managed by broker
a sorted list of tasks
T3 Per task
*
a task
M3 Point of selecting the LRM
M1 Task management
list of scheduler nodes
distributed / k-level
task managed directly
M4 Filter scheduler nodes based on dynamic info.
M2 Scheduler Hierarchy
list of scheduler nodes
monolithic and observational
M5 Select scheduler node
selected node
M6 Pass on to selected Local Resource Manager
R3


list of jobs
J2 Create list of eligible jobs
list of eligible jobs
J3 Sort jobs on criterion
no eligible jobs left
sorted list of jobs
J4 Per job
#
a single job
J5 Job setup
@
a single job
job updated
T1 Create list of eligible tasks
a list of tasks
T2 Sort tasks on criterion
no eligible tasks left
task managed by broker
a sorted list of tasks
T3 Per task
*


Users
Legend
stage
input
iteration
decision
scheduler transfer
J1 Incoming Jobs
list of jobs
J2 Create list of eligible jobs
list of eligible jobs
J3 Sort jobs on criterion
no eligible jobs left
sorted list of jobs
J4 Per job
#
a single job
J5 Job setup
@
a single job
job updated
B Broker
T1 Create list of eligible tasks
a list of tasks
T2 Sort tasks on criterion
no eligible tasks left
task manager by broker
a sorted list of tasks
T3 Per task
*
a task
M3 Point of selecting the LRM
M1 Task management
list of scheduler nodes
distributed / k-level
task managed directly
M4 Filter scheduler nodes based on dynamic info.
M2 Scheduler Hierarchy
list of scheduler nodes
monolithic and observational
M5 Select scheduler node
selected node
M6 Pass on to selected Local Resource Manager
R3
M6
R3 Filter machines based on static information
R1 Machines
machines fulfilling static demands
list of authorized machines
list of machines in the sytem
R4 Filter machines based on dynamic information
R2 Filter based on authorization
machine needs to be provisioned
machines fulfilling dynamic demands
R6 Provisioning new machines / VMs / containers / host servers for FaaS
R5 Select and allocate resource(s)
replicated task
machine provisioned
selected machine
T4 Submit task
T9 Task replication
task submitted
R7 Elastic decision making (auto-scaling)
T5 Task setup
task to be replicated
T10 Task preemption
task started
preemption
T12 Task migration
T6 Monitor task
on task failure
T11 Task recovery
task/job to be migrated
task completed
on job update
@
T8 Task cleanup
T7 Task completion
tasks left in queue
*
all tasks of job completed
J7 Job cleanup
J6 Job completion
jobs left in queue
#
Legend
stage
input
iteration
decision
scheduler transfer


R3 Filter machines based on static information
R1 Machines
machines fulfilling static demands
list of authorized machines
list of machines in the sytem
R4 Filter machines based on dynamic information
R2 Filter based on authorization
machine needs to be provisioned
machines fulfilling dynamic demands
R6 Provisioning new machines / VMs / containers / host servers for FaaS
R5 Select and allocate resource(s)
replicated task
machine provisioned
selected machine
T4 Submit task
T9 Task replication
task submitted
task to be replicated
R7 Elastic decision making (auto-scaling)
T5 Task setup
T10 Task preemption
task to be scaled
task started
on task/job

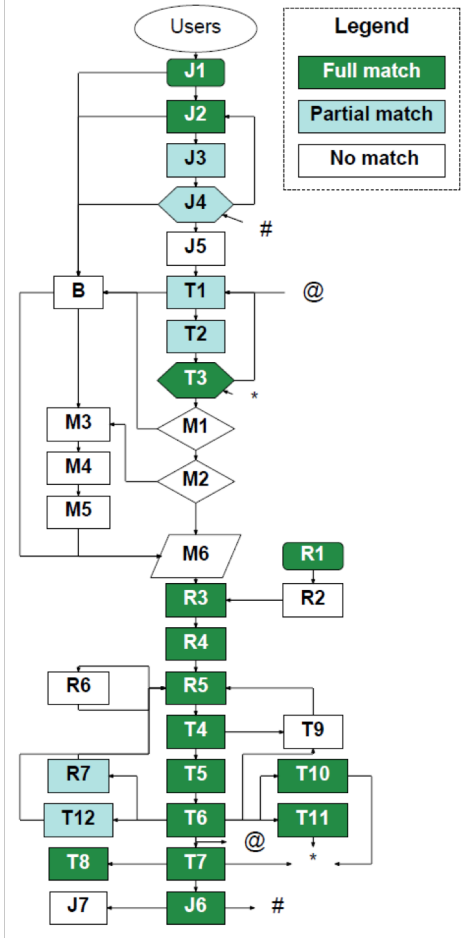
Design

Validation

Experiments

Mapping Schedulers to the Model

- 14 scheduler publications analyzed
- For each stage: 3 possible types of a ‘match’
  - **Full match:** Stage is described in detail
  - **Partial match:** Detail on the ‘how’ is lacking
  - **No match:** Remaining cases **Underspecification**
- Here, a Borg-like scheduler, given no open-source



Mapping of the Borg Scheduler to the RA

💡 Overall: Schedulers tend to be **underspecified**

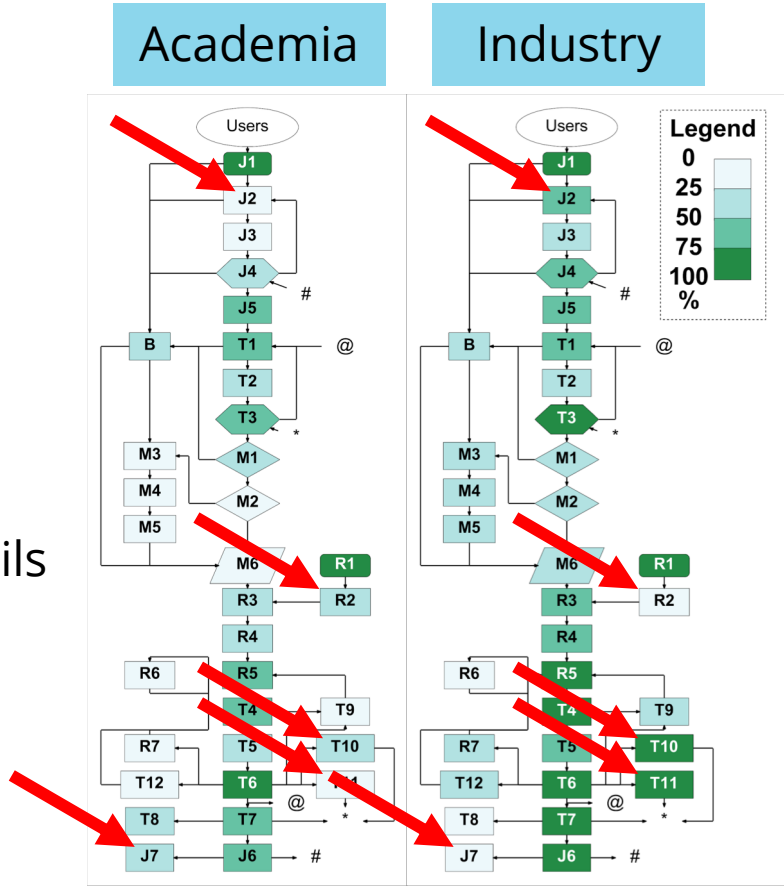
More on the Mapping Process

- Subset of the publications used for design
- Mapping done in parallel by two reviewers
- Small set of publications used as calibration
  - No significant difference in interpretation

Discussion on mapping schedulers with different levels of expertise:
bit.ly/sc18-scheduling

Head-to-Head: Academia vs. Industry

- Comparison through **heat maps**
- Specified more by industry
  - T11: Task recovery
  - T10: Task preemption
  - J2: Job filtering
  - → More attention to the technical details
- Specified more by academia
  - R2: Filtering resources based on authorization
  - J7: Job cleanup



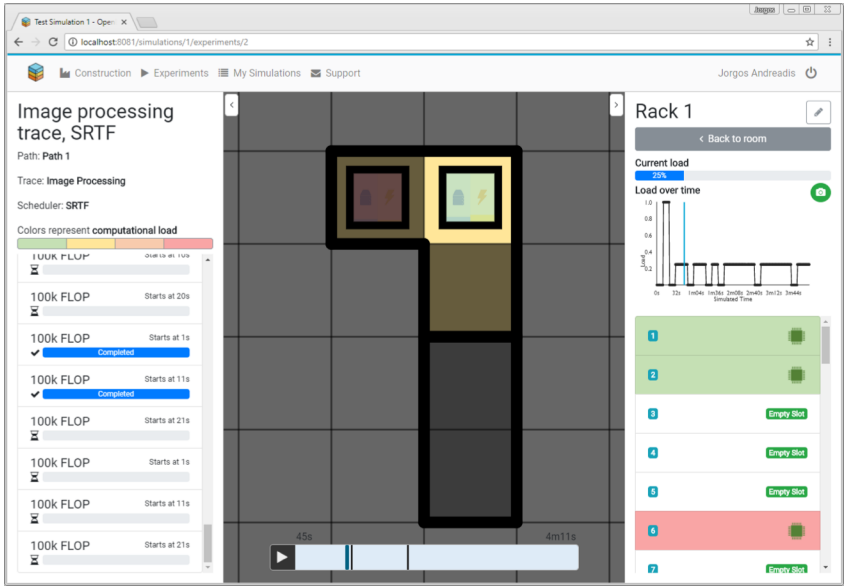
Design
Validation
Experiments

Q1: Impact of underspecification on performance?
Q2: Impact of underspecification on real-world performance?

Experimental Setup

- Prototype implemented in **OpenDC**
- Selected subset of stages made configurable
- Real-world engineering and industrial traces
- Standard DC topology

opendc.org



T1 T2 R4 R5

Impact of Underspecification on Performance

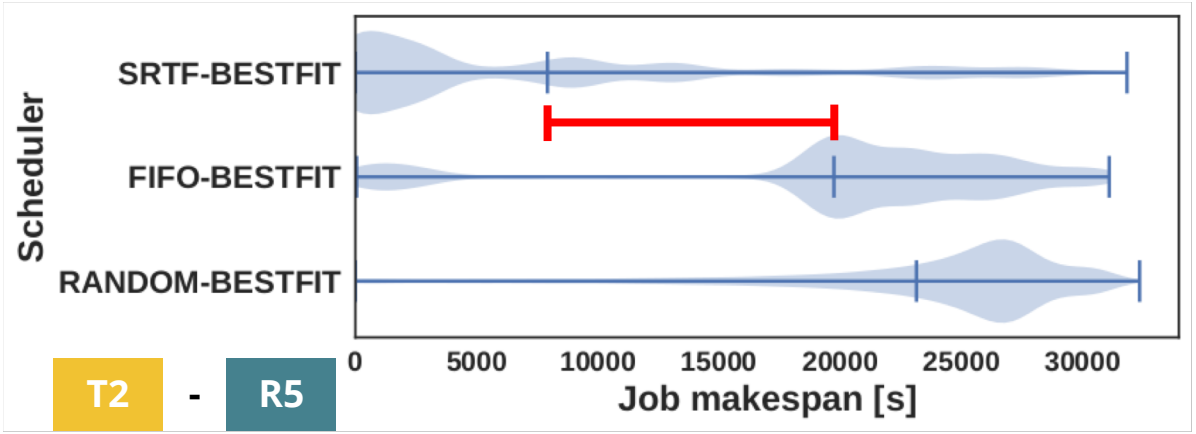
- Selected two underspecified stages in a Borg-like system
- Compare various metrics of different configurations

Significant difference in performance!

💡 Different policy combinations lead to different results

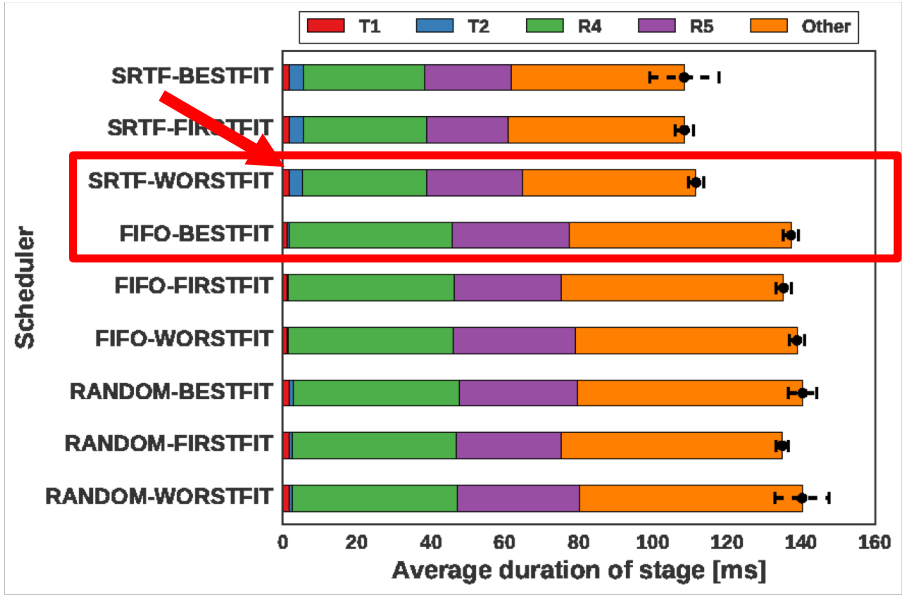
T2: Sort tasks

R5: Select resources



Stage Complexity at Runtime

- Instrumented simulation run
- Measure time spent in each stage
- Observation: SRTF leads to longer sorting times, but shortest total duration

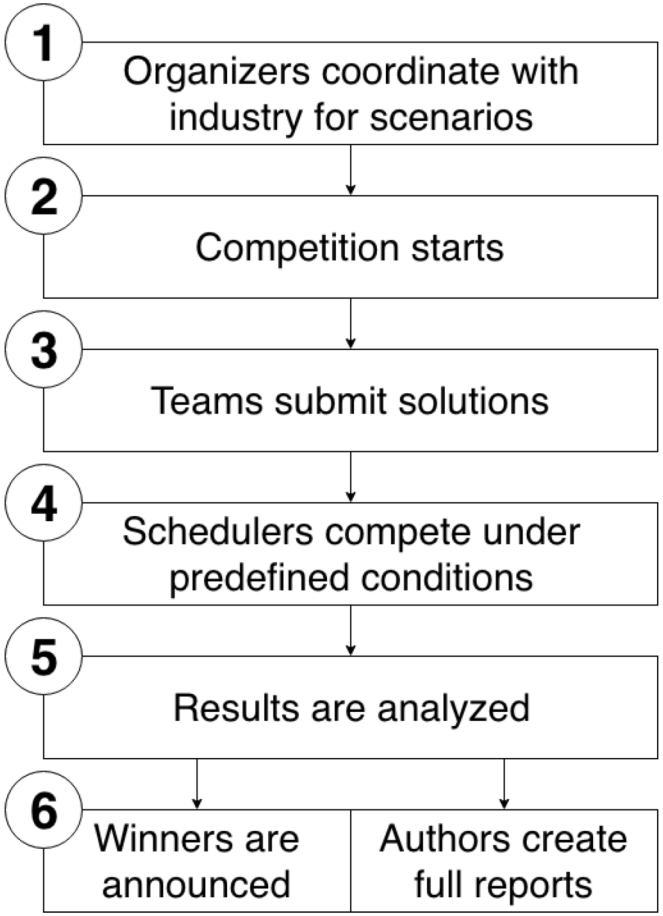


💡 Stage-policy can have a non-trivial impact on stage durations

Future Directions

- Extended validation
- API/Language for scheduler design
- Dialogue with scheduler designers
- A global competition of schedulers
  - Future edition of JSSPP

We're collecting use cases for the community: bit.ly/jsspp-cfp



Speculation: Scheduler Design Space Exploration

- Systematic exploration of stage implementation combinations
- Using real-world stage policies
- “Bruteforcing” scheduler design

Will the results differ from human designs?

Take-Home Message

- Schedulers are complex and difficult to compare
- They remain underspecified
- Important implications for reproducibility, DevOps
- A guideline for new scheduler publications
- Conceptual model can help design, analyze, and improve schedulers

Learn more about OpenDC:
opendc.org

Read the Technical Report:
bit.ly/sc18-scheduling

Fill in the short survey:
bit.ly/sc18-scheduling-survey

Georgios Andreadis
g.andreadis@atlarge-research.com
atlarge-research.com/gandreadis