A Systematic Design Space Exploration of Datacenter Schedulers

Fabian Mastenbroek
Delft University of Technology

Georgios Andreadis*
Delft University of Technology

Alexandru Iosup*
Delft University of Technology

ABSTRACT

Datacenter infrastructure has become vital for stakeholders across industry, academia and government. To operate efficiently, datacenter operators rely on a variety of complex scheduling techniques, to distribute user workloads across resources. In this work, we leverage a reference architecture for datacenter scheduling to design and implement an instrument for systematic design space exploration of datacenter schedulers. We construct a formal representation of the design space for datacenter schedulers, using scheduling policies collected from real-world schedulers. We then use a genetic algorithm in combination with trace-based simulation to explore the space, optimizing for workload metrics. Through several experiments, we assess the viability of the instrument. We find that our instrument is able to identify patterns in the workloads and adapt the scheduling policies appropriately. Overall, our work leads to numerous findings, which can become valuable for future comprehension and development of schedulers.

KEYWORDS

datacenter, scheduling, cloud computing, design space exploration, reference architecture, genetic algorithm, simulation

1 INTRODUCTION

Datacenter infrastructure is becoming ever more important in today's society. Numerous stakeholders across industry, academia, and government employ diverse cloud services, which, in turn, are hosted by datacenter infrastructure. Crucial for datacenter operation is the *scheduler*, which is concerned with planning and assigning user workloads to resources in the datacenter [26, 33]. Scheduling is extremely challenging, yet at the same time, the community regards schedulers as 'black-box' in the system, hindering understanding and comparison of schedulers. To address these issues, we recently proposed a reference architecture for datacenter scheduling [2]. Unifying the different approaches of schedulers, we envision in this work another use-case for the reference architecture: *systematic design space exploration of datacenter schedulers*. We represent the design space of schedulers using the reference architecture and design an instrument for systematic exploration of this design space.

Datacenter schedulers decide and enforce which resources to provision for a user or application and which parts of the application (typically, *tasks*) to map to provisioned resources, while at the same time optimizing for goals such as efficiency or sustainability, and honoring complex SLAs. Optimal scheduling is not feasible due to the NP-hardness of the problem; in practice, scheduling is solved using numerous heuristics and other online methods that merely approximate or satisfice the solution. Each part of enforcement raises additional technical issues and especially the combination of decisions and enforcement elements make the scheduler such a complex system. Consequently, the current approaches in the industry are problematic: although it is inconvenient for datacenter operators to keep much of their infrastructure idle, due to resulting high energy consumption and thus unnecessary costs [24], in practice, industry-wide, server utilization is only 6-12% [32, 19]. At the same time, co-locating workloads on machines might cause undesirable performance variability [21]. Furthermore, this decision making and enforcement needs to happen efficiently at unprecedented scale, yet schedulers require considerable time and resources for these activities [17].

Currently, the community develops highly complex schedulers ad hoc and yet treats schedulers as 'black-box' components in the system. This hinders both the likelihood of developing of good schedulers, and the analysis and comparison of schedulers, because the complexity of a scheduler and the diversity of actions that it needs to perform cannot be fully captured by an opaque component. Consequently, only few schedulers are well-understood and adopted by datacenter operators [20]. A conceptual model of datacenter scheduling could help understand how to design, build, and control such complex systems [16]. The reference architecture for datacenter scheduling [2] recently proposed by our group, identifies over 30 different components of datacenter schedulers, and the key data and control flows between these components. Fourteen real-world, well-known scheduler publications have been already been mapped to it.

Addressing the complexity of scheduler development, we propose in this work *an instrument for systematic design space exploration of datacenter schedulers*. Leveraging the high level of granularity provided by the reference architecture, we represent the construction of a datacenter scheduler as combinatorial optimization problem. Using a genetic algorithm, we explore the optimization landscape of scheduling policy combinations, potentially identifying novel combinations of scheduling techniques. With the instrument we analyze how the different policies, equipped by different components in the scheduler, can together affect the performance of the system in terms of scheduler decision-making.

Overall, this work has a three-fold contribution:

(1) We characterize the design space of datacenter schedulers (Section 2). We identify from real-world scheduler publications scheduling mechanisms and policies to include in the design space.
(2) We design a method for systematic design space exploration of datacenter schedulers (Section 3). We construct a design space from a selection of scheduling stage policies and examine appropriate search methods for exploring this design space.
(3) We develop a prototype of the instrument and evaluate through trace-based simulation the quality and viability of the instrument (Section 4).

*Supervision

2 DESIGN SPACE OF DATACENTER SCHEDULERS

We analyze in this section the domain of datacenter schedulers and define a representation for the design space of datacenter schedulers, appropriate for design space exploration.

2.1 Reference Architecture for Datacenter Schedulers

In traditional models of resource-management systems, datacenter schedulers usually play only a coarse-grained, 'black-box' role. Although we can evaluate and compare, on a high level, different schedulers using various metrics (such as job makespan), due their complex nature and diversity in approaches, it is often non-trivial to asses the impact of each individual design decision on system performance. In essence, this leaves little room for exploration of the design space. We could explore the space of an individual scheduler through a parameter sweep. However, this restricts the design space to the options and insights of that particular scheduler. Instead, we need to generalize the scheduling process across different scheduler implementations.

We consider in this work the *reference architecture for datacenter scheduling* proposed in 2018 by Andreadis et al. [2]. This is a conceptual model that captures the entire process of datacenter scheduling and uses a structured, workflow-based approach as depicted in Figures 1 and 2. The scheduler is modeled as a set of components (*stages*), each with specified inputs, outputs and side-effects, the combination of which define its function. Decisions taken by the scheduling stages are guided by *policies*, separated by design from the mechanism that executes the decisions. Numerous policies already exist for scheduling in datacenters and clouds [31, 28]. For instance, to assign tasks to the appropriate resources, schedulers could use the `Worst-Fit` policy, where tasks are placed on hosts with the most resources available. As another example, to rank the tasks eligible for scheduling, schedulers could use the *First-In-First-Out* (`FIFO`) policy to order tasks based on their arrival time. Alternatively, the scheduler might use the *Shortest-Remaining-Time-First* (`SRTF`), where tasks with shorter (estimated) runtime are given a higher priority over tasks with longer times.

The workload is modeled as a stream of jobs, where we assume that all jobs fit the morphology of *workflows* [1, 9, 22]: each job consists of a set of one or several tasks, with precedence constraints between tasks determining the order of execution. Execution starts at the central submission site, and progresses until job and task completion and cleanup stages. A scheduler iteration may be started periodically, on events such as job arrival or completion, or even manually.

Overall, we identify four main responsibilities in schedulers:

(1) *Job processing* (`J`): activities concerning the selection of jobs and the job life-cycle, such as job setup and cleanup,
(2) *Task processing* (`T`): stages of the task life-cycle, including more sophisticated stages for task migration, preemption, and replication,
(3) *Scheduler management* (`M`): stages facilitating scheduling hierarchies and (cloud) brokers, and
(4) *Resource management* (`R`): stages related to provisioning and allocating resources.

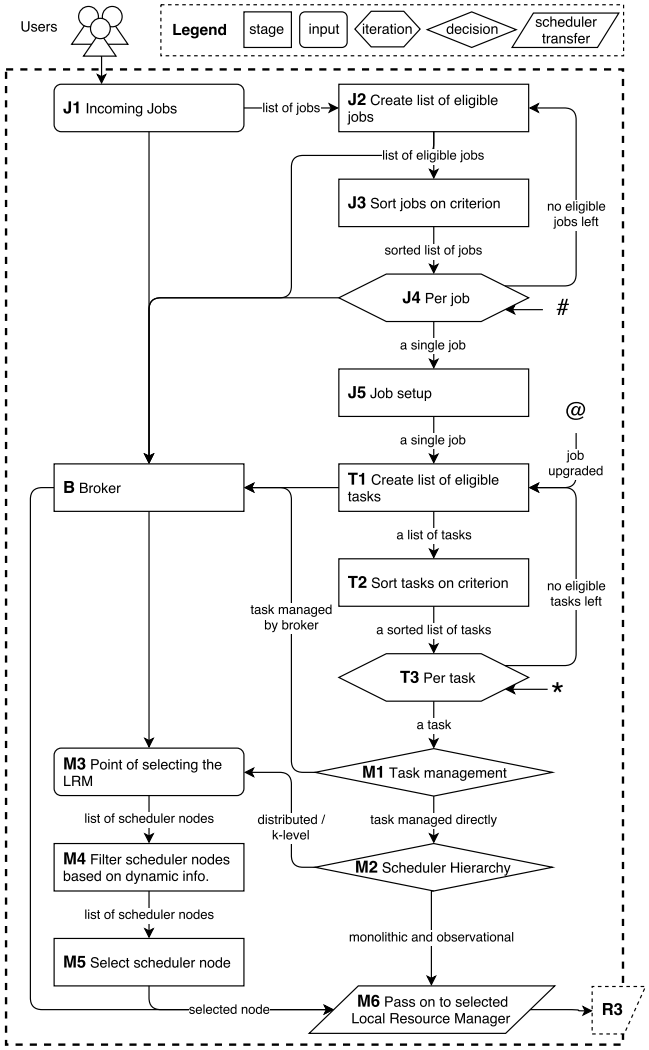


Figure 1: Reference architecture for datacenter scheduling [2]. Focus on the *global* scheduler. (Continued in Figure 2.)

To emphasize the scheduler hierarchy, we can further categorize stages by the level at which they run: Figure 1 visualizes the stages executed at datacenter-level by a global scheduler, whereas Figure 2 depicts stages predominantly executed at cluster-level by a subordinate, local scheduler. However, this categorization only serves as a suggestion as this division may not be applicable to all schedulers (e.g., a monolith scheduler may execute all stages on the same machine). The *Broker* component represents a stub; it may employ more complex sub-systems, however, in a design that remains external to the architecture.

2.2 Representation of the Design Space of Datacenter Schedulers

A suitable representation of the design space is essential for the effectiveness of the design-space exploration [18]. In this work,

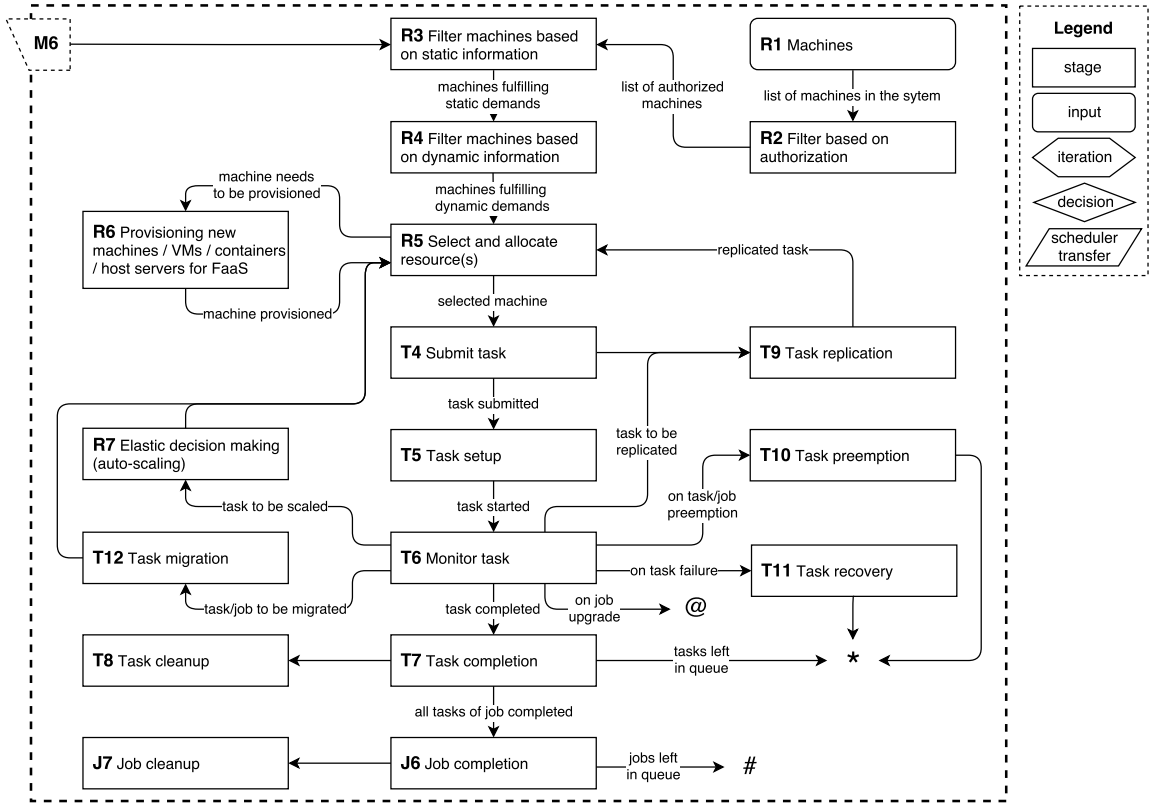


Figure 2: Reference architecture for datacenter scheduling [2]. Focus on the *local* scheduler. (Continued from Figure 1.)

we propose a design space representation based on the state-of-the-art reference architecture for datacenter schedulers proposed by Andreadis et al. [2]. We represent each datacenter scheduler (a point in the design space) as a particular combination of scheduling policies, with each policy corresponding to and implementing the intended functionality of a specific scheduling stage in the reference architecture. More formally, in our representation, a scheduler S is defined as the tuple $S = (p_{J1}, \ldots, p_{M1}, \ldots, p_{T1}, \ldots, p_{R7})$ where p_x represents a policy used by the scheduler for each of the scheduling stages x of the reference architecture.

Conceptually, we model policies of a scheduler using *atoms* and *operators* as building blocks. Atoms (Section 2.2.1) represent predefined and pre-programmed decision logic (algorithm) for a particular scheduling stage. To prevent 'magic' constants in policies and to expand our design space even further, atoms may be also parameterized, that is, they may accept arguments and change their behavior according to them. Operators (Section 2.2.2) are higher-order building blocks and are used to construct more complex policies from (a combination) of other scheduling policies.

Formally, we can now represent the policies of a scheduling stage x as the language P_x, defined inductively as follows:

(1) $Atom(y_1, \ldots, y_n)$ is a stage policy for every *Atom* and parameters y_1 to y_n.
(2) $Operator(p_{x_1}, \ldots, p_{x_n})$ is a stage policy for every *Operator* and stage policies p_{x_1} to p_{x_n}.

2.2.1 Construction of Scheduling Policies using Atoms. Atoms are the fundamental building blocks for stage policies in our representation of the design space. Although, in theory, the complexity and granularity of an atom may range from a single instruction to entire policies, we restrict ourselves in this work to basic heuristic scheduling policies. For example, to prevent queues from overflowing, schedulers might want to limit the number of total active jobs in the system during admission of new jobs into the queue (J2). To achieve this behavior, we implement the *Limit-Active*(n) atom using the pseudo-code listed in Algorithm 1.

Algorithm 1 Psuedo-code for the *Limit-Active*(n) atom

```
Require: n ∈ ℕ
  function LIMIT-ACTIVE(scheduler, job)
      active ← Query the number of active jobs in scheduler
      if active < n then
          ADMIT-JOB(job)
      end if
  end function
```

We have implemented in total over 30 heuristic scheduling atoms to include in the design space of datacenter schedulers, listed in Tables 1 to 3 respectively. Note however that in this work, we consider only a subset of the scheduling stages defined by the reference architecture, consisting of non-trivial and important scheduling stages:

Interactive	Schedule immediately after submission
Batch(p)	Schedule every time quantum q
Random	Schedule after a random duration

(a) J1 – *Incoming jobs*

Always	Admit every incoming job
Limit-Load(t)	Admit if total system load does not exceed t
Limit-Active(n)	Admit if number of active jobs does not exceed n
Random(p)	Admit with probability p

(b) J2 – *Create list of eligible jobs*

Submission-Time(*asc*|*desc*)	Order by job submission time
Job-Size(*asc*|*desc*)	Order by number of tasks in job
Job-Duration(*asc*|*desc*)	Order by (estimated) critical path of job
Random	Order randomly

(c) J3 – *Sort jobs on criterion*

Table 1: Overview of the implemented atoms for job processing.

J1 – *Incoming jobs* (Table 1a): this input stage provides the list of jobs that users have submitted at the central submission site of the scheduling system. Policies of this stage decide whether to immediately pass the job to stage J2 or wait for some period.

J2 – *Create list of eligible jobs* (Table 1b): this stage makes a selection from the input list of jobs that only includes jobs that are eligible to be scheduled. A policy chosen for this stage may dictate that all jobs may be passed through, or may implement a restriction on eligibility, e.g. due to certain user restrictions or flow control measures. Any jobs rejected at this stage return to the list of incoming jobs and are reconsidered at the next scheduling iteration.

J3 – *Sort jobs on criterion* (Table 1c): this stage sorts a list of jobs based on a certain priority criterion and outputs the sorted list. Policies that determine this priority can take a variety of meta data into account, such as: the submitting user, a metric such as estimated time of completion, or even a composite score of different aspects.

T1 – *Create list of eligible tasks* (Table 2a): this stage filters the list of tasks provided as input, based on a filter-policy, e.g. a policy that allows tasks to pass through if and only if their dependencies have already finished.

T2 – *Sort tasks on criterion* (Table 2b): this stage takes a list of tasks and sorts it on a given criterion. This can be done to improve the throughput and latency of tasks.

R5 – *Select and allocate resource(s)* (Table 3a): in this stage, the selected task is matched with a (set of) resource(s). The match is passed on to the task submission stage (T4).

2.2.2 Combination of Same-Level Policies through Operators. Multiple scheduling policies can coexist in the same scheduling stage. For example, modern datacenter schedulers can use a portfolio of diverse policies, from heuristics [10, 25, 34] to optimization based on linear-integer programming [29], to make scheduling decisions. We use operators to construct such combinations of same-level scheduling policies in our design space. Similar to atoms, they may range in complexity and granularity. For example, we could implement a simple conditional operator, which selects a policy if a certain condition is met. On the other hand, we could also implement a complex portfolio scheduling operator, which dynamically selects an appropriate policy based on real-time information from the scheduler.

We consider in this work only the composition operator $p_1 \circ p_2$, which composes sequentially policies p_1 and p_2. Its exact behavior is dependent per stage and described in Table 4.

Always	Admit every incoming task
Limit-Load(t)	Admit if total system load does not exceed t
Limit-Active(n)	Admit if number of total active tasks does not exceed n
Limit-Job(n)	Admit if number of active tasks in the job does not exceed n
Job-Balance(t)	Admit if proportion of active tasks in a job to the average number of active tasks in a job does not exceed a threshold t
Random(p)	Admit with probability p

(a) T1 – *Create list of eligible jobs*

Submission-Time(*asc*|*desc*)	Order by task submission time
Task-Duration(*asc*|*desc*)	Order by (estimated) duration of task
Task-Duration-History(*asc*|*desc*)	Order by average duration of other finished tasks in job
Task-Dependencies(*asc*|*desc*)	Order by number of dependencies of a task
Task-Dependents(*asc*|*desc*)	Order by number of dependents of a task
Active-Per-Job(*asc*|*desc*)	Order by number of active tasks of the job of the task
Job-Completion(*asc*|*desc*)	Order by percentage of tasks finished in job
Random	Order randomly

(b) T2 – *Sort tasks on criterion*

Table 2: Overview of the implemented atoms for task processing.

First-Fit	Match task with first fitting resource
Best-Fit	Match task with resource with least amount of processing elements available to satisfy the request
Worst-Fit	Match task with resource with most amount of processing elements available to satisfy the request
Expensive	Match task with most expensive (fastest) resource
Cheap	Match task with cheapest (slowest) resource
Random	Match task with random resource

(a) R5 – *Select and allocate resource(s)*

Table 3: Overview of the implemented atoms for resource management.

J1	Not supported
J2	Admit if both p_1 and p_2 accept the job
J3	Order first by p_1, then by p_2
T1	Admit if both p_1 and p_2 accept the task
T2	Order first by p_1, then by p_2
R5	Match using p_1, then using p_2

Table 4: Behavior of the composition operator for the considered scheduling stages.

2.3 Evaluation of Datacenter Scheduler

To explore the design space, we need to be able to evaluate points in the design space. That is, we need to define an ordering of fitness between the points in the space. We use in this work the following four traditional metrics to evaluate the performance of a scheduler:

(1) Task response time (*TRT*): time elapsed from task submission to task completion,
(2) Job makespan (*JMS*): time elapsed from the first task-submission of a job, to the last completion of a task in the job,
(3) Normalized Job-Schedule Length [23] (*NJSL*): job makespan normalized by the length of the critical path (the shortest possible execution time of the job),
(4) Job waiting time (*JWT*): time elapsed from the first task-submission of a job, to the first start of a task of that job.

2.4 Problem Definition

Using this design space representation, we can formulate the design space exploration of datacenter schedulers as the following optimization problem:

> **Main Problem**: Given a datacenter environment E, a workload W and some objective function f, find the scheduler $S = (p_{J1}, \ldots, p_{R7})$ such that $f(S, E, W)$ *statisfices* [30, p.27].

2.5 What Is the Magnitude of the Design Space?

It is important to have an understanding of the *magnitude* of the design space, that is the order of its size. Concretely, we consider for our design space representation as magnitude the number of possible scheduling combinations in the design space. The magnitude of the design space directly influences the design of design of the instrument and may rule out certain approaches for exploration. For instance, for smaller design spaces brute-force search might be a feasible strategy. However, as the design space grows, brute-force search becomes less feasible and smarter methods are required to traverse the search space.

We investigate magnitude of the design using a three-point estimation. We assume that evaluation of a scheduler in the design space takes 1 second and we can evaluate up to 1024 schedulers at the same time.

Best-case Suppose none of the atoms accept any parameters and we have no operators. In this case, the magnitude of the design space must at least be the product of the number of atoms for each stage: $3 \times 4 \times 4 \times 5 \times 8 \times 6 = 11520$. Given our assumptions, a brute-force search of this magnitude would take only 12 seconds and therefore a feasible approach.

Average-case Suppose the parameters of each atoms have at most 100 choices and we can only use the composition operator twice per stage. We approximate the magnitude as follows, taking into account the number of combinations produced by composition, if applicable, by taking the sum of a stage to the power of three.
J1: $1 + 100 + 1 = 102$
J2: $(1 + 100 + 100 + 100)^3 = 301^3$
J3: $(2 + 2 + 2 + 1) = 7^3$
T1: $(1 + 100 + 100 + 100 + 100 + 100)^3 = 501^3$
T2: $(2 + 2 + 2 + 2 + 2 + 2 + 2 + 1)^3 = 15^3$
R5: $(1 + 1 + 1 + 1 + 1 + 1)^3 = 6^3$
Total: $J1 \times J2 \times J3 \times T1 \times T2 \times R5 \approx 9 \times 10^{29}$
In such a configuration, a brute-force search of the design space would take over 3×10^{18} years to complete given our assumptions, making it *intractable* to brute-force.

Worst-case Suppose we consider all atoms with every possible combination of parameters and all operators: extremely high amount of combinations, so also intractable to brute-force.

3 DESIGN OF AN INSTRUMENT FOR DESIGN SPACE EXPLORATION OF DATACENTER SCHEDULERS

In this section, we present the design for an instrument for design-space exploration of datacenter schedulers, and motivate the decisions behind our proposed design. We first define the problem formally. Subsequently, we describe our set of requirements, present a high-level architectural overview of the instrument and discuss the design details.

3.1 Requirements

We identify five key requirements for an instrument for design-space exploration of datacenter schedulers:

R1 *Usability:* The instrument should require only minimal involvement from the user during the exploration process.
R2 *Feasibility:* The instrument should be able to find a solution that satisfices within an acceptable time frame.
R3 *Explainability:* The instrument must be able to explain scheduler configuration selections to the user, such that they can understand *why* a specific configuration was chosen. Such a property is crucial for the adoption of new scheduling approaches [20].
R4 *Reproducibility:* The outputs of the instrument must be reproducible.
R5 *Configurability:* The instrument must support optimizing for different user-specified metrics, workloads and environments.

3.2 Design Overview

The instrument follows the high-level process depicted in Figure 3 and consists of three main components, which interact during runtime: (1) the *exploration algorithm*, (2) the *simulator*, (3) the *result processing and analysis* component. It is an iterative process: the instrument uses feedback from previous iterations to improve its next results.

The exploration algorithm is responsible for constructing a fixed-size population of scheduler candidates (*individuals*) based on the configuration specified by the user. This configuration consists of the workload, the datacenter environment and the objective (the fitness function) to optimize for (**R5**). Moreover, the user may also specify meta-parameters such as the population size or the seed of the pseudorandom number generator (**R4**), or supply the exploration with additional atoms and operators for construction of scheduling policies. We construct the scheduler candidates using the genetic algorithm described in Section 3.3 (**R2**) and then sent them to the simulator for evaluation.

The simulator receives the population of scheduler candidates from the exploration algorithm and will simulate (**R4**) the configured workload and datacenter environment using each of the individuals in parallel. During execution, the simulator monitors, through the monitoring component, the progress of the workload and tracks various metrics of the system, such as system load, task response time and job makespan. This information is then stored in the *result database* to be processed and analyzed.

The result processing and analysis component is responsible for processing the results of each round of simulation. For every

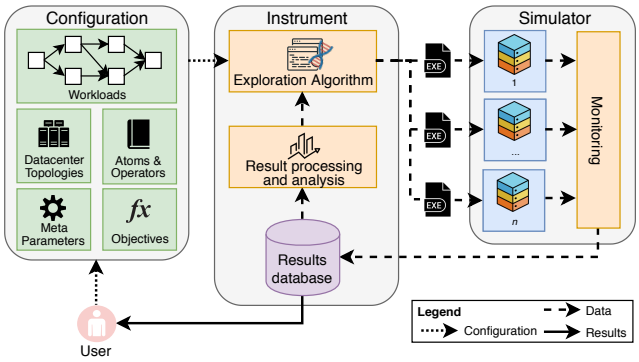


Figure 3: Architectural overview of the instrument.

individual in the population, it determines the fitness based on the objective function specified by the user. This information is communicated back to the exploration algorithm, where this feedback will be used to construct a new population of scheduler candidates. In addition, this component also analyzes after each round of simulation, the difference between the elite (the fittest individual) of last round and the elite of the current round, and tries to identify the policy that caused a drop in fitness.

3.3 Exploration of Design Space using Genetic Search

Numerous methods for exploring a design space exist. The most straightforward approach is to perform a brute-force search of the design space. Although effective for smaller design spaces, brute-force search quickly becomes infeasible as the magnitude of a design space grows. As we have shown in our estimation of magnitude of the design space in Section 2.5, brute-force exploration of our design is intractable. We instead use an evolutionary approach proposed by Holland [12]. This approach is inspired by the process of natural selection and encodes each point in the design space as a set of chromosomes which can be mutated and altered.

We encode each scheduler combination as a phenotype with its scheduling stages represented as chromosomes. The genes of each chromosome represent the scheduling policies of a particular scheduling stage. If a chromosome contains more than one gene, we use the composition operator to combine the genes into a single scheduling policy that accepted by the simulator.

A high-level overview of the evolution process is depicted in Figure 4, which proceeds as follows:

(1) A population of random individuals is generated based on the population size and seed specified by the user.
(2) The fitness of every individual is evaluated. We send the individuals simulator component and wait for feedback from the result processing and analysis component.
(3) The stop criteria are evaluated. The instrument limits the maximum number of generations to 500 by default. Furthermore, we terminate the evolution process when the average fitness of the last 10 generations differs at most 0.01% from the average fitness of the last 30 generations, in which case we deem the fitness as converged.

(4) A selection of the more fit individuals is made from the current population. We use tournament selection, where the best individual from a random sample of three individuals (drawn with replacement) is chosen from the population. Tournament selection is a good choice due to its lack of stochastic noise and independence to scaling of the fitness function.
(5) The genetic operators, described below, are applied to the selection. We proceed to step 2 to evaluate the new population.

To explore the design space using a genetic algorithm, genetic operators are used to either converge or diverge the solution. We consider for our instrument the following genetic operators:

(1) Uniform crossover – The genes at index i of two chromosomes are swapped with probability 0.2. Empirical studies show that this approach leads to better exploration of the design space while maintaining the exchange of good information [8].
(2) Unguided mutation – A gene is changed to a random policy with probability $\sqrt[3]{0.1}$.
(3) Guided mutation – With probability probability $\sqrt[3]{0.05}$, the parameter of an atom is changed. The new value is picked based on a Gaussian distribution around the current value of the parameter. This allows for exploration for better solutions near the current individual.
(4) Length mutation – The length of a chromosome is mutated with probability $\sqrt[3]{0.02}$. In turn, with probability $\frac{1}{3}$ we remove the first gene from the chromosome and with probability $\frac{2}{3}$ we add a random gene to the chromosome.
(5) Redundancy pruning – We prune redundant combinations from the chromosome. For example, the composition of an allocation policy with itself is redundant and may be reduced to a single policy.

Note that in this work we are not concerned with the efficiency of search methods for design space exploration of datacenter schedulers. Instead, the goal of this work is to display the feasibility and potential of such a design space exploration. We leave efficiency as an interesting concern to explore in the future.

3.4 Simulation of Datacenter Schedulers

To evaluate scheduling candidates, we use simulation, which is a powerful and cost-effective tool for computer-based modeling and analysis of complex systems. Simulation is often useful when other techniques such as mathematical modeling or direct measurement of a system become infeasible due to complexity or costs. Several software packages already exist for simulating schedulers [5, 4, 6, 15, 13], which can be used to optimize scheduling policies and test scenarios which are prohibitively expensive in physical datacenters.

We implement a prototype of the instrument on top of the community-driven, open-source OpenDC simulation platform [13], making configurable the stages `J1`, `J2`, `J3`, `T1` `T2` and `R5` and implementing the atoms and operators described in Section 2.2.1 and Section 2.2.2 respectively.

In theory, the instrument could employ real-world experiments for short scenarios using real-world experimental infrastructure. To support this functionality, our implementation of the scheduler with the various stages of the reference architecture should be adapted to delegate task to real machines, instead of simulated machines. However, the results of the instruments will most-likely suffer due to the non-determinism of real-world systems. This causes the fitness of individuals to include noise, which hampers the performance of the genetic search algorithm.

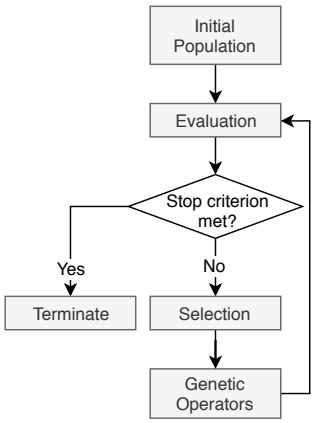


Figure 4: Exploration of the design space through genetic search.

3.5 Explainability

Many new scheduling approaches are developed every year, but they are not well-understood and face lack of adoption by datacenter operators [20]. Although the instrument might be able to identify novel scheduling combinations, in light of adoption, it is crucial that we are able to understand and explain the performance of a scheduling combination. Moreover, this information is valuable for the development of new scheduling policies.

To address this problem, we integrate a notion of explainability into the instrument. That is, the instrument will by itself investigate the cause of a certain scheduling combination performing better. After each generation, the instrument analyses and compares the fittest individual of the current generation against the one of the last generation. The instrument will take the stages that changed between the generation and apply one-by-one a changed stage to the individual of the last generation. Next, it will run for each of these changes a simulation to asses the impact of this single change on the fitness of the individual. If the fitness is within 10% of the current individual's fitness, we consider that particular change to explain the fitness increase and report this to the user.

However, note that it might be possible that there is no single change whose fitness is within 10% of the current individual's fitness. In this case multiple policies together have caused the fitness to increase.

4 EXPERIMENTAL EVALUATION WITH THE INSTRUMENT

We present in this section an experimental evaluation with the instrument. Our main findings are:

MF1 The instrument supports exploration long-term and large-scale scenarios.
MF2 The instrument is able to identify patterns in the workloads and adapt appropriately the stages of the scheduler using the portfolio of policies available.
MF3 The instrument is sensitive to changes in workload and environment.
MF4 The instrument blindly optimizes for a single objective and does not take into account other metrics.

4.1 Experimental Setup

In this section, we describe the experiment setup used throughout our experiments.

4.1.1 Hardware and Software Environment. The prototype of the instrument is implemented on top of the community-driven, open-source OpenDC simulation platform [13], which itself is written in the Kotlin language (version 1.3).

Experiments are carried out on the Google Cloud Platform, using a standard machine[1] (`n1-standard-64`) running Linux (Ubuntu 19.04) and Java 12 (OpenJDK 12.0.1).

4.1.2 Workloads. We use three traces collected from real-world datacenter-like environments by the community of various sizes: Google [27], Askalon [14] and Chronos [25]. Google is a heterogeneous engineering workload from Google that captures a month of activity of over 12k machines running the Borg [33] resource management system. Askalon is an engineering workload that uses workflows to simulate chemical processes. Chronos is an industrial workload that uses workflows to process data collected from an IoT production-environment monitoring industrial equipment. Table 5 gives an overview of the characteristics of the workloads: Google is a long-running, highly dynamic workload, varying over time and features, but is driven by many short-running jobs. Askalon has more complex workflows, whereas Chronos includes a large job-burst at start.

4.1.3 Datacenter topology. We consider in this work a datacenter environment with a varying number of common-off-the-shelf resources (machines), where half of these machines contain each an Intel i7 processor, with 4 cores, at a clock rate of 4.1GHz and the other half contain each an Intel i5 processor, with 2 cores, at a clock rate of 3.5GHz.

[1] https://cloud.google.com/compute/docs/machine-types

Table 5: Workload characteristics.

Workload	Application domain	Workflows	Tasks
Google	Engineering	494,179	17,810,002
Askalon	Engineering, chemistry	3,551	122,105
Chronos	Industrial, IoT	1,024	3,072

4.2 Experiment 1: Exploration for a Long-term and Large-scale Scenario

With the ever more growing demand for cloud services, datacenter operators such as Google and Amazon must maintain efficient operation at unprecedented scale [3]. It it therefore essential that datacenter schedulers they employ are sufficiently optimized for the heterogeneous workloads they face.

Our instrument could prove valuable for datacenter operator, but this requires the instrument to support for the tremendous scale and time-frame in which typical modern datacenters operate. Demonstrating the viability of exploring long-term and large-scale scenarios of the instrument, we optimize a scheduler running the Google [27] workload for average job makespan. This workload consists of over 400k workflows and captures a month of activity. Moreover, we consider for this experiment a datacenter environment with 24k machines using the setup described in Section 4.1.

Figure 5 shows the progression of the elite fitness over the generations of the exploration. On top of that, we depict the scheduling combinations that were used in the particular generation. We highlight their effect on the fitness using the dashed red line, indicating a switch in scheduling policy.

We observe that there is only few variation in the policies that have been selected. In particular, the stages `J2`, `J3` and `R5` do not change at all during evolution. Moreover, we also observe the use of random policies for stages `J1`, `J2` and `T2`. This suggests that for this particular workload, our current portfolio of policies for these scheduling stages is unsuitable.

Overall, the results seem to indicate that the instrument tries to limit large jobs from taking over the scheduling queue. Instead, preference is given to jobs of smaller size with the least amount of tasks running. This observation seems to agree with the conclusion [27] that the Google trace is driven by many short jobs that demand quick scheduling decisions.

This suggests that the instrument is able identify global patterns in the workload and is able to adapt appropriately.

4.3 Experiment 2: Sensitivity Analysis for Workload, Topology, and Other Parameters

In this experiment, we investigate the sensitivity of the input workload, topology and meta-parameters of the instrument. This is important as chosen parameters should not have too large of an impact on the instrument output.

We first analyze the effect of the seeding of the pseudo-random number generator on the results of the instrument. We optimize the Askalon workload for makespan twice using a different seed each run. We consider a datacenter environment of 256 machines. In Figure 6, we show the effect of different seeding on the progression of the fitness of the elite solution. Additionally, we show the scheduling combinations that were used in the particular generation.

The results indicate that the progression of the fitness is definitely affected by the seeding of the psuedo-random number generator. While Figure 6a progresses overall slower than Figure 6b, Figure 6b slows down immediately after the second generation. However, we observe that both figures converge to the same point. Moreover, they also overlap in the scheduling policies they select.

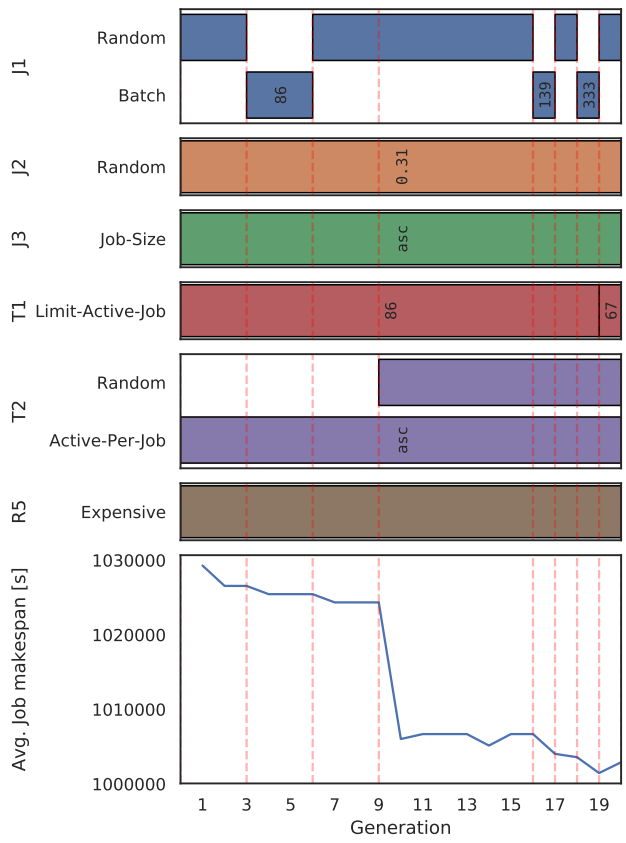


Figure 5: Policy selection per scheduling stage and fitness per generation. Each bar represents the period of generations that a policy is active. The text inside a represents the parameter of that policy.

In both cases, the scheduling policies *Interactive*, *Limit-Active-Job*, *Job-Balance* and *Active-Per-Job*.

In summary, while the seeding of the pseudo-random number generator affects the fitness progression, the fitness tends to converge to the same point eventually.

Next, we investigate the effect of different workloads and datacenter environments on the fitness progression. We depict in Figure 7 the fitness progression as a fraction of its initial fitness. We observe that the Chronos workload does not optimize well for the environment of 512 machines and the environment of 64 machines. Strangely enough, Chronos is able to achieve a rather large speedup when using an environment of 256 machines. Moreover, we observe that for the Askalon workload, the different environments do show different fitness progression. However, in all cases, they seem to converge to the same fitness.

Overall, the results indicate that the instrument is indeed sensitive to the input workload and environment.

4.4 Experiment 3: Exploration with Diverse Objectives

Datacenters and other large-scale computing infrastructure often have different needs and requirements. In traditional batch scheduling systems, jobs are usually queued and processed at a later moment in the background. In contrast, interactive scheduling systems require immediate feedback to incoming requests. In essence, these systems optimize for different requirements.

As such, we analyze the effect of optimizing a scheduler for different objectives (metrics). We run in this experiment the Askalon workload and optimize for the job makespan (*JMS*), job waiting time (*JWT*) and normalized job-schedule length [23] (*NJSL*). We consider a datacenter of 256 machines.

Figure 8 depicts the different metrics per generation as different metric is being optimized. The top graph represents the evolution of the job makespan, the middle graph the evolution of the normalized job-schedule length and the bottom graph the evolution of the job waiting time.

We observe that the job waiting time and normalized job-schedule length show some correlation. In contrast, the job makespan does not seem to have any correlation with other two metrics. Interestingly, the rather chaotic relation between on one side the job waiting time and normalized job-schedule and on the other hand the job makespan show that even when the fitness is converging, other metrics may still fluctuate a lot.

This suggests that there exists solutions where we both metrics are optimize, yet the instrument blindly focuses on its given objective.

5 THREATS TO VALIDITY

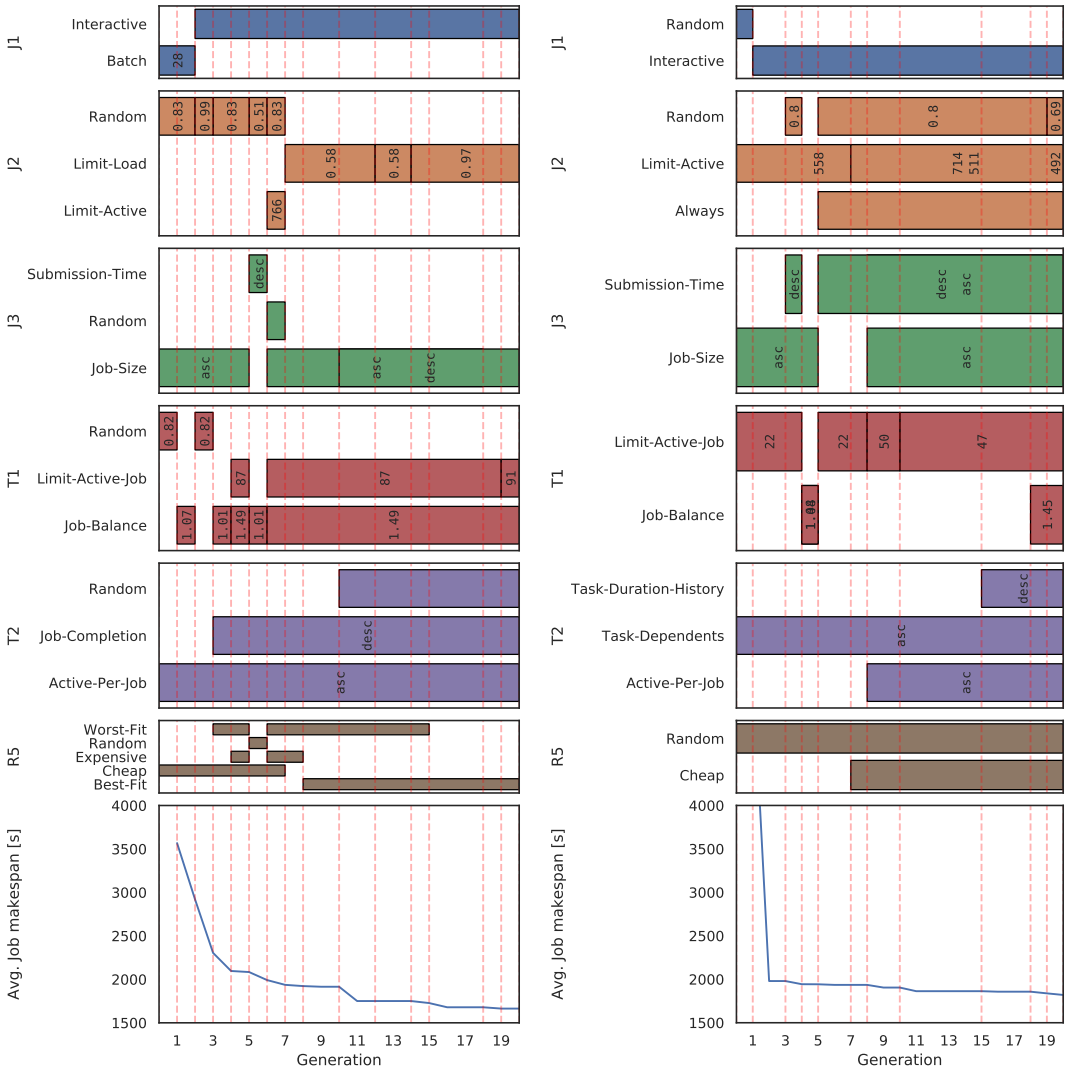
We discuss in this section several points which might threaten the validity of the work, and how these are mitigated.

5.1 Validity of Simulator

A possible threat to the validity of this work is the use of a simulator in the instrument, instead of real-world experimentation, for evaluating schedulers in the design space. To mitigate this risk, we have validated the simulator by running real-world workloads and verifying the outcomes manually. However, currently existing alternatives are not suitable and could suffer from the same or even deeper problems: mathematical analysis, where the scheduler represented as a mathematical model (e.g., hierarchical and queuing models), is limited because its accuracy relies on preexisting data from which a model derived. Considering further the complexity and responsibilities of modern datacenter schedulers, this approach becomes infeasible. Another approach is real-world experimentation. However, such experimentation is expensive, hard to reproduce and cannot capture the scale at which datacenter infrastructure is running at.

5.2 Limits of the Reference Architecture

Another threat to the validity of this work is the use of the reference architecture by Andreadis et al. [2]. In particular, the use of the reference architecture might restrict our design space by possibly not being able to represent some of the scheduling approaches. However, more than ten real-world and state-of-the-art scheduling



(a) Seed: 456 **(b) Seed: 789**

Figure 6: The effect of different seeds on the fitness. Each bar represents the period of generations that a policy is active. The text inside a represents the parameter of that policy

systems have already been mapped to the reference architecture. These systems cover the multiple dimensions of the domain, such as background, age and deployment.

Moreover, we have found that alternative models are even more restricting to the design space. They lack the fine granularity of the reference architecture and miss support for advanced concepts typical in modern datacenter scheduling.

5.3 Optimality of Solutions

Another concern for this work is the optimality of solutions as the instrument might suffer from the local optima problem. That is, the exploration algorithm might get stuck at a local optimum in the design space. However, we try to overcome this problem by using (1) unguided mutation as divergence operator to explore possible other regions of the design space and (2) uniform crossover to combine two schedulers, which also leads to better exploration of the design space [8].

Furthermore, we focus in this work on finding a solution that satisfices [30, p.27], rather than full minimization of the fitness of a scheduler, given that this is most likely unachievable or intractable due to extreme magnitude of the design space (see Section 2.5).

5.4 Inter-Dependence of Stages

The inter-dependence of stages in the scheduler might pose another threat to the validity of this work. This is the effect that policies of the different scheduling stages may not be fully independent

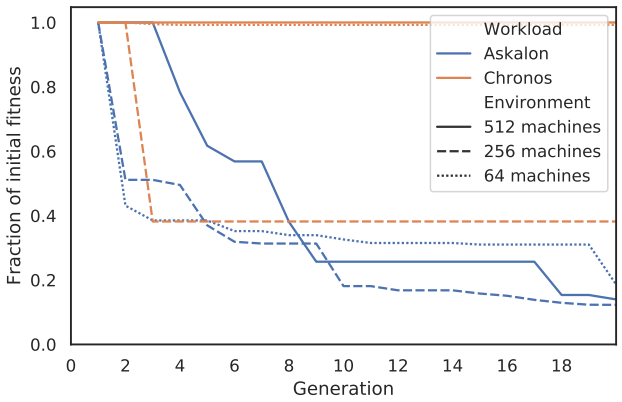


Figure 7: The effect of optimizing different workloads and environments.

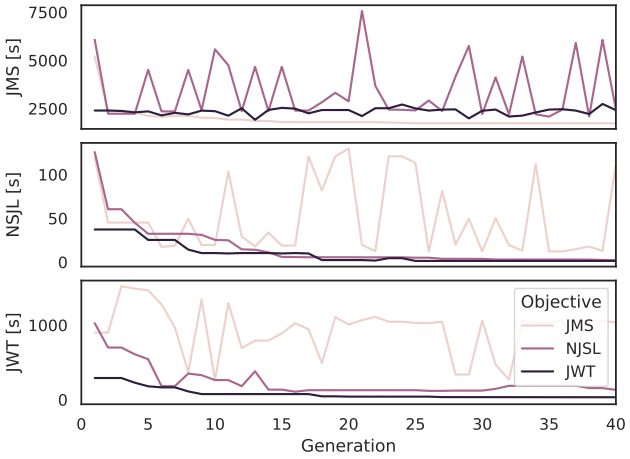


Figure 8: *JMS*, *NJSL* and *JWT* per generation while optimizing for different objectives.

with respect to other scheduling stages and the performance of the system, such that using either one of two policies will not give the desired speedup, but both will.

While it very valuable for the instrument to be able to recognize and analyze inter-dependence between the different stages of a scheduler, trying to accomplish this will lead to combinatorial explosion as trying to analyze the difference between two schedulers will require in the worst-case an exponential amount of combinations.

As such, we do not take this property into account in this work. Instead, we assume that each scheduling policy is independent during analysis of scheduling combinations.

6 RELATED WORK

The field of scheduling simulation is broad. Various taxonomies of scheduling have been proposed in literature [7, 28]. Rodriguez and Buyya additionally surveyed numerous scheduling algorithms and classified them according to their proposed taxonomy. These taxonomies focus on the characteristics of the broader architecture, but do not elaborate into the interplay between these characteristics.

A survey by Singh and Chana [31] on resource scheduling identifies several responsibilities of schedulers, including provisioning assignment and monitoring, but lacks the more advanced concepts present in modern datacenters, such as checkpointing, auto-scaling, replication and migration of workloads.

Closest to our work, and surveying the most relevant alternatives for simulation is our own previous publication on the reference architecture for datacenter scheduling [2] in 2018. In this work, four stages of the reference architecture were implemented using the community-driven, open-source OpenDC platform for datacenter simulation [13]. The authors conduct experiments with the four implemented stages of the general model. The work proposed here naturally complements this prior work. In this work, we extend OpenDC platform with more stages of reference architecture and implement over 30 scheduling policies. We further design and implement an instrument for exploring the optimization landscape of datacenter schedulers. We systematically construct scheduling combinations of various policies and using simulations on traces from real-world datacenter-like environments, we analyze the effect of different scheduler combinations on system performance (e.g., in terms of job makespan).

Estrada et al. propose a similar idea for design space exploration of datacenter schedulers [11]. They use genetic programming approach to construct parse trees of conditional statements which represent resource allocation policies. While we also use a genetic algorithm for constructing scheduling policies, we work at the granularity of a single policy (e.g., Select host with most available resources). In contrast, Estrada et al. work at finer granularity, constructing allocation policies using inequalities and logical expressions. Moreover, in their work they consider only the resource allocation stage (R5) of the reference architecture. In this work, we also explore important stages related to job and task processing, such as admission and sorting.

7 CONCLUSION AND FUTURE WORK

Datacenter infrastructure power today's digital society. Crucial for their operation is the *scheduler*, responsible for allocating datacenter resources for user workloads. Although the community develops highly complex and advanced schedulers, they struggle to perform efficiently both in terms of output performance and runtime. Moreover, due to their complex nature and diverse approaches, they are difficult to comprehend and compare, hindering adoption of new schedulers and policies [20]. The reference architecture for datacenter scheduling proposed by Andreadis et al. [2] tries to address this problem using a conceptual workflow-based model, in which over 30 components including their key data and control flows represent the processes of scheduling in datacenters.

We design and implement in this work an instrument for systematic design space exploration of datacenter schedulers. Using the reference architecture, we construct a formal representation of a design space for datacenter scheduler. We use a genetic algorithm to explore the space and analyze design candidates using simulation. Using real-world workloads, we analyze the performance of the instrument.

Our main findings consist of:

MF1 The instrument supports exploration long-term and large-scale scenarios.

MF2 The instrument is able to identify patterns in the workloads and adapt appropriately the stages of the scheduler using the portfolio of policies available.

MF3 The instrument is sensitive to changes in workload and environment.

MF4 The instrument blindly optimizes for a single objective and does not take into account other metrics.

For the future, we plan to extend the design space with more policies and implement more stages of the reference architecture. Furthermore, we aim to investigate the decomposition of policies into a finer granularity, similar to the genetic programming approach by Estrada et al. [11]. Lastly, analysis of the efficiency of the exploration algorithm is left out in this work, but this might be an interesting concern to explore in the future, allowing for possibly larger-scale experiments.

REFERENCES

[1] Ilkay Altintas, Chad Berkley, Efrat Jaeger, Matthew Jones, Bertram Ludäscher, and Steve Mock. 2004. Kepler: an extensible system for design and execution of scientific workflows. In *Proceedings of the 16th International Conference on Scientific and Statistical Database Management (SSDBM 2004), 21-23 June 2004, Santorini Island, Greece*, 423–424.

[2] Georgios Andreadis, Laurens Versluis, Fabian Mastenbroek, and Alexandru Iosup. 2018. A reference architecture for datacenter scheduling: design, validation, and experiments. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis, SC 2018, Dallas, TX, USA, November 11-16, 2018*, 37:1–37:15.

[3] Beyer et al. 2016. *Site Reliability Engineering: How Google Runs Production Systems*. O'Reilly Media.

[4] Rajkumar Buyya and M. Manzur Murshed. 2002. Gridsim: a toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. *Concurrency and Computation: Practice and Experience*, 14, 13-15, 1175–1220.

[5] Rodrigo N. Calheiros, Rajiv Ranjan, César A. F. De Rose, and Rajkumar Buyya. 2009. Cloudsim: A novel framework for modeling and simulation of cloud computing infrastructures and services. *CoRR*, abs/0903.2525.

[6] Henri Casanova. 2001. Simgrid: A toolkit for the simulation of application scheduling. In *First IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2001), May 15-18, 2001, Brisbane, Australia*, 430–441.

[7] Thomas L. Casavant and Jon G. Kuhl. 1988. A taxonomy of scheduling in general-purpose distributed computing systems. *IEEE Trans. Software Eng.*, 14, 2, 141–154.

[8] Pravir K. Chawdhry, Rajkumar Roy, and Raj K. Pant. 1997. *Soft Computing in Engineering Design and Manufacturing*. Springer-Verlag.

[9] Ewa Deelman, Karan Vahi, Mats Rynge, Gideon Juve, Rajiv Mayani, and Rafael Ferreira Da Silva. 2016. Pegasus in the cloud: science automation through workflow technologies. *IEEE Internet Computing*, 20, 1, 70–76.

[10] Kefeng Deng, Junqiang Song, Kaijun Ren, and Alexandru Iosup. 2013. Exploring portfolio scheduling for long-term execution of scientific workloads in iaas clouds. In *International Conference for High Performance Computing, Networking, Storage and Analysis, SC'13, Denver, CO, USA - November 17 - 21, 2013*, 55:1–55:12. DOI: 10.1145/2503210.2503244. https://doi.org/10.1145/2503210.2503244.

[11] Trilce Estrada, Michael R. Wyatt, and Michela Taufer. 2015. A genetic programming approach to design resource allocation policies for heterogeneous workflows in the cloud. In *21st IEEE International Conference on Parallel and Distributed Systems, ICPADS 2015, Melbourne, Australia, December 14-17, 2015*, 372–379.

[12] 1984. *Genetic algorithms and adaptation. Adaptive Control of Ill-Defined Systems*, 317–333.

[13] Alexandru Iosup, Georgios Andreadis, Vincent van Beek, Matthijs Bijman, Erwin Van Eyk, Mihai Neacsu, Leon Overweel, Sacheendra Talluri, Laurens Versluis, and Maaike Visser. 2017. The opendc vision: towards collaborative datacenter simulation and exploration for everybody. In *16th International Symposium on Parallel and Distributed Computing, ISPDC 2017, Innsbruck, Austria, July 3-6, 2017*, 85–94.

[14] Alexandru Iosup, Hui Li, Mathieu Jan, Shanny Anoep, Catalin Dumitrescu, Lex Wolters, and Dick H. J. Epema. 2008. The Grid Workloads Archive. *Future Generation Comp. Syst.*, 24, 7, 672–686.

[15] Alexandru Iosup, Ozan Sonmez, and Dick Epema. 2008. Dgsim: comparing grid resource management architectures through trace-based simulation. In *Proceedings of the 14th International Euro-Par Conference on Parallel Processing*, 13–25.

[16] Alexandru Iosup, Alexandru Uta, Laurens Versluis, Georgios Andreadis, Erwin Van Eyk, Tim Hegeman, Sacheendra Talluri, Vincent van Beek, and Lucian Toader. 2018. Massivizing computer systems: a vision to understand, design, and engineer computer ecosystems through and beyond modern distributed systems. *CoRR*, abs/1802.05465.

[17] Svilen Kanev, Juan Pablo Darago, Kim Hazelwood, Parthasarathy Ranganathan, Tipp Moseley, Gu-Yeon Wei, and David Brooks. 2015. Profiling a warehouse-scale computer. In *Proceedings of the 42Nd Annual International Symposium on Computer Architecture*, 158–169.

[18] Eunsuk Kang, Ethan Jackson, and Wolfram Schulte. 2011. An approach for effective design space exploration. In *Proceedings of the 16th Monterey Conference on Foundations of Computer Software: Modeling, Development, and Verification of Adaptive Systems*, 33–54.

[19] James M Kaplan, William Forrest, and Noah Kindler. 2008. Revolutionizing data center energy efficiency. In *Uptime Institute Symposium*.

[20] Dalibor Klusáček and Šimon Tóth. 2014. On interactions among scheduling policies: finding efficient queue setup using high-resolution simulations. In *Euro-Par 2014 Parallel Processing*, 138–149.

[21] Younggyun Koh, Rob C. Knauerhase, Paul Brett, Mic Bowman, Zhihua Wen, and Calton Pu. 2007. An analysis of performance interference effects in virtual environments. In *2007 IEEE International Symposium on Performance Analysis of Systems and Software, April 25-27, 2007, San Jose, California, USA, Proceedings*, 200–209.

[22] Mehmet Can Kurt, Sriram Krishnamoorthy, Kunal Agrawal, and Gagan Agrawal. 2014. Fault-tolerant dynamic task graph scheduling. In *SC*, 719–730.

[23] Yu-Kwong Kwok and Ishfaq Ahmad. 1998. Benchmarking the task graph scheduling algorithms. In *IPPS/SPDP*, 531–537.

[24] Jacob Leverich and Christos Kozyrakis. 2010. On the energy (in)efficiency of hadoop clusters. *SIGOPS Oper. Syst. Rev.*, 44, 1, 61–65.

[25] Shenjun Ma, Alexey Ilyushkin, Alexander Stegehuis, and Alexandru Iosup. 2017. Ananke: A q-learning-based portfolio scheduler for complex industrial workflows. In *ICAC*.

[26] Kay Ousterhout, Patrick Wendell, Matei Zaharia, and Ion Stoica. 2013. Sparrow: distributed, low latency scheduling. *ACM Symposium on Operating Systems Principles (SOSP)*, 69–84.

[27] Charles Reiss, Alexey Tumanov, Gregory R. Ganger, Randy H. Katz, and Michael A. Kozuch. 2012. Heterogeneity and dynamicity of clouds at scale: google trace analysis. In *Proceedings of the Third ACM Symposium on Cloud Computing* (SoCC '12), 7:1–7:13.

[28] Maria Alejandra Rodriguez and Rajkumar Buyya. 2017. A taxonomy and survey on scheduling algorithms for scientific workflows in iaas cloud computing environments. *Concurrency and Computation: Practice and Experience*, 29, 8, e4041.

[29] Siqi Shen, Kefeng Deng, Alexandru Iosup, and Dick H. J. Epema. 2013. Scheduling jobs in the cloud using on-demand and reserved instances. In *Euro-Par 2013 Parallel Processing - 19th International Conference, Aachen, Germany, August 26-30, 2013. Proceedings*, 242–254. DOI: 10.1007/978-3-642-40047-6_27. https://doi.org/10.1007/978-3-642-40047-6%5C_27.

[30] Herbert A. Simon. 1996. *The Sciences of the Artificial (3rd Ed.)* MIT Press, Cambridge, MA, USA. ISBN: 0-262-69191-4.

[31] Sukhpal Singh and Inderveer Chana. 2016. A survey on resource scheduling in cloud computing: issues and challenges. *Journal of Grid Computing*, 14, 2, 217–264.

[32] Arunchandar Vasan, Anand Sivasubramaniam, Vikrant Shimpi, T. Sivabalan, and Rajesh Subbiah. 2010. Worth their watts? - an empirical study of datacenter servers. In *16th International Conference on High-Performance Computer Architecture (HPCA-16 2010), 9-14 January 2010, Bangalore, India*, 1–10.

[33] Abhishek Verma, Luis Pedrosa, Madhukar Korupolu, David Oppenheimer, Eric Tune, and John Wilkes. 2015. Large-scale cluster management at google with borg. In *Proceedings of the Tenth European Conference on Computer Systems*, 18:1–18:17.

[34] Maria A. Voinea, Alexandru Uta, and Alexandru Iosup. 2018. POSUM: A portfolio scheduler for mapreduce workloads. In *IEEE International Conference on Big Data, Big Data 2018, Seattle, WA, USA, December 10-13, 2018*, 351–357. DOI: 10.1109/BigData.2018.8622215. https://doi.org/10.1109/BigData.2018.8622215.