Future Generation Computer Systems 123 (2021) 156–177

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

A survey of domains in workflow scheduling in computing infrastructures: Community and keyword analysis, emerging trends, and taxonomies

Laurens Versluis *, Alexandru Iosup

Vrije Universiteit Amsterdam, The Netherlands

ARTICLE INFO

Article history:
Received 2 November 2020
Received in revised form 9 March 2021
Accepted 21 April 2021
Available online 27 April 2021

Dataset link: https://atlarge-research.com/data/2020_fgcs_aip.pgsql

Keywords:
Cloud
Cluster
Grid
Workflow
Scheduling
Survey
Taxonomy
Formalism
Allocation
Provisioning
Applications
Services
Policies
Community
Meta-analysis

ABSTRACT

Workflows are prevalent in today's computing infrastructures as they support many domains. Different Quality of Service (QoS) requirements of both users and providers makes *workflow scheduling* challenging. Meeting the challenge requires an overview of state-of-art in workflow scheduling. *Sifting* through literature to find the state-of-art can be daunting, for both newcomers and experienced researchers. *Surveys* are an excellent way to address questions regarding the different techniques, policies, emerging areas, and opportunities present, yet they rarely take a systematic approach and publish their tools and data on which they are based. Moreover, the *communities* behind these articles are rarely studied. We attempt to address these shortcomings in this work.

We introduce and open-source an instrument used to combine and store article meta-data. Using this meta-data, we *characterize* and *taxonomize* the workflow scheduling community and four areas within workflow scheduling: (1) the workflow formalism, (2) workflow allocation, (3) resource provisioning, and (4) applications and services. In each characterization, we obtain important keywords overall and per year, identify keywords growing in importance, get insight into the structure and relations within each community, and perform a systematic literature survey per part to validate and complement our taxonomies

1. Introduction

Datacenters and cloud providers are increasingly becoming the go-to point for leasing additional computing power. Both industry and academia are embracing this new paradigm of computation on demand, ranging from financial institutes [1] to bioinformatics research communities [2,3]. Remaining up-to-date with the state-of-the-art and emerging trends through surveys is important for both scientists and engineers as improvements and new approaches are being introduced continuously. Interestingly, few surveys use a systematic approach to search for relevant articles and no surveys discuss the communities behind these articles. To address these aspects, in this work, we complement the use of traditional search methods with a systematic approach through an in-house developed instrument. Using this instrument, we obtain a dataset of meta-data on relevant articles, which in turn enables community analyses and systematic searches through the use of database queries. Additionally, this leads to large parts of our work becoming reproducible, something science as a whole is increasingly focusing on in general, and enables other researchers to re-use our dataset for future surveys, including different areas than covered in this work, as all software and the dataset itself are offered as open-access FAIR data.

Nowadays, many applications that run on cluster and cloud resources are *workflows*. These workflows can be modeled differently. The simplest model is the that of Coffman and Graham [4]. In this model, a workflow is represented as a Directed Acyclic Graph (DAG) where each vertex represents a task (i.e., a unit of work) and an edge a computation/data constraint. Other formalisms such as BPMN allow for cycles and human-in-the-loop elements. These workflows origin from various domains, ranging from bioinformatics to finance and from geology to astronomy. Datacenters, clusters, and clouds receive tens to millions

* Corresponding author.
E-mail address: l.f.d.versluis@vu.nl (L. Versluis).

https://doi.org/10.1016/j.future.2021.04.009

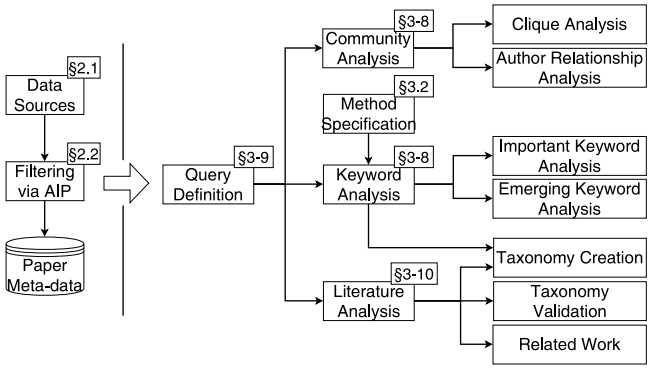


Fig. 1. The structure of and process applied in this survey. Left: obtaining a database of paper meta-data. Right: the usage of this meta-data for four type of analyses.

of such workflows per hour [5]. A key component in executing these workflows efficiently is the *scheduler* [6]. Scheduling these workflows to make efficient use of the available resources is a challenging task, demonstrated by the sheer amount of proposed scheduling systems and policies. Moreover, nowadays, the resource providers must adhere to different Quality of Service (QoS) requirements that can differ per workflow.

Performing well in workflow scheduling requires keeping up with the most recent advances in workflow scheduling. Especially with the recent developments in edge and serverless computing [7,8], it is important to remain up-to-date. The accelerating growth of the number of workflow scheduling articles published makes it a daunting task to get insights into the attempts made by research to solve these complex challenges. Semantic Scholar also underlines this challenge: "The rate of scientific publication is increasing every year, with more than 3 million papers published across 42,500 journals in 2018 alone. This unprecedented flow of information makes staying up-to-date with the scientific literature an increasingly pressing challenge for scholars".[1]

Questions arise such as: Which different techniques are being used nowadays to schedule workflow or resources? What structures do these schedulers have? What is currently important in the community? Which areas and topics are emerging? and which opportunities for research are there?

Surveys are an excellent way to get answers to such questions. They provide an overview of the current field by using taxonomies and other means such as tables to present and compare approaches, enumerate emerging topics and list challenges and possible directions for future work. Yet, survey articles rarely publish the tools and data on which they are based. This data is important to reproduce the survey's findings, verify the completeness, and use as a base for extensions.

In this work, aligned with our vision that scheduling is a first-class component when massivizing computer systems [9], we address these issues by following the process visualized in Fig. 1. Using an instrument we developed that parses and filters article meta-data, we gain insights into the workflow scheduling and four sub-communities. Additionally, we use our instrument to find relevant articles per topic next to using traditional search methods such as Google scholar. Using relevant articles and related work, we construct, extend existing, and validate taxonomies. We do not perform quantitative comparisons where we discuss and compare different algorithms. Such an endeavor requires an exploration of workload, metrics, operational environment, and careful details such as software versions which are out of scope due to the broad scope of this survey. Previous studies even on a small subset of these parameters have shown there are significant differences, and no conclusion about superiority can be drawn, i.e., no single best scheduling approach exists. An example of such a study is done by Kwok and Ahmad [10], where they compare in-depth several policies. The complexity and variety of today's policies would require even more work and detail.

Overall, we make the following five main contributions in this work:

1. We assemble a unique dataset of meta-data on relevant articles and create a specialized instrument to process it (Section 2). Our dataset combines data from three major curators into a comprehensive dataset for the computer systems community. Additionally, we develop a suite of tools for combining, filtering, and analyzing this dataset.
2. We perform a novel survey of the workflow scheduling community (Section 2). We propose a method for analyzing article meta-data on workflow scheduling, focusing on emerging keywords and community structure.
3. Using results from Section 2, we focuses on four areas in workflow scheduling (Sections 3–7). Our work proposes novel taxonomic aspects and significantly extends state-of-the-art taxonomies [11–13] on workflow scheduling in areas such as formalisms for workflow specification; workflow allocation policies, strategies, and structures; elasticity and serverless in resource provisioning; and various types of resources used in current applications and services. For each area, we also make observations about community, trending keywords, and emerging trends in the timespan 2011–2020.
4. We validate our taxonomies by mapping well-cited and recently introduced workflow allocation and resource provisioning policies using a systematic survey (Section 8). We map several elements of allocation and resource provisioning policies to our taxonomies to validate our taxonomies contain these elements.
5. The instrument, article meta-data database, and other software (e.g., scripts) used in this work are offered as open-source artifacts for the community to use. The database is suitable for similar studies on different topics. The instrument and tools are extensible and can be added to include more information sources and capture more properties. The database on which this article is based can be found at https://atlarge-research.com/data/2020_fgcs_aip.pgsql, AIP and other tools used to generate all floats in this article can be found at https://github.com/atlarge-research/AIP.

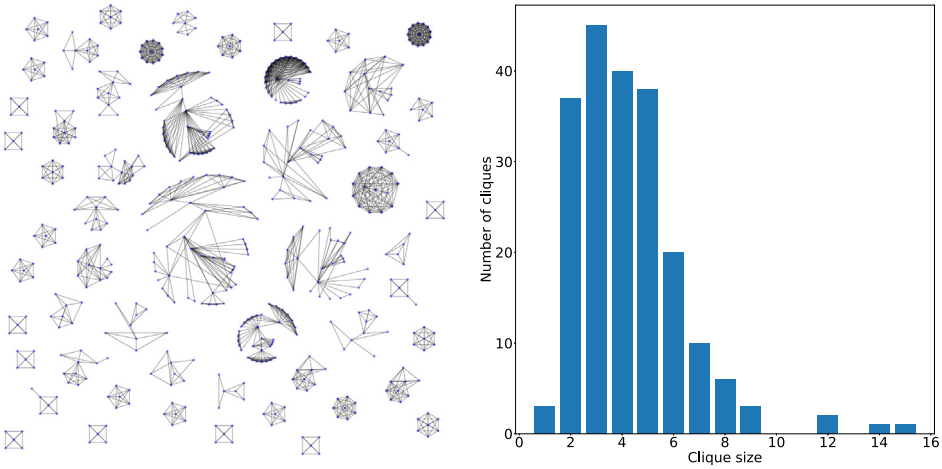
2. Analyzing and obtaining a dataset of article meta-data

As science is aiming for becoming increasingly reproducible and the amount of articles published per year is increasing, a more systematic approach is required next to traditional search methods. To facilitate such a systematic approach, we decided to develop an instrument, AIP, that gathers article meta-data from various sources and parses, complements, and filters this meta-data to store it in a database.

AIP filters and unifies data from DBLP [14], Semantic Scholar [15], and AMiner [16] to obtain meta-data on articles from the systems community. We outline the workings and details of AIP in our technical report [17]. As AIP uses a relational database, reproducibility and systematic searches become available through the use of queries. Additionally, these queries provide an complementary search method next to the traditional search methods (e.g., using Google scholar).

Using queries and AIP's database, we analyze:

[1] https://pages.semanticscholar.org/about-us.



(a) A visual representation of the workflow scheduling community. Vertices represent authors, an edge is drawn if authors have co-authored. Only components with cardinality 5 or higher are shown.

(b) A bar plot depicting the size and number of cliques within the workflow scheduling community.

Fig. 2. An overview of the workflow scheduling community.

Query 1: `SELECT * FROM publications WHERE year BETWEEN 2011 AND 2020 AND (lower(title) LIKE '%workflow%' OR lower(abstract) LIKE '%workflow%') AND (lower(title) LIKE '%schedul%' OR lower(abstract) LIKE '%schedul%')`

1. Community analysis, where we inspect the structure of the community and its characteristics. This can be useful for community managers and organizers.
2. Overall trend analysis, where we obtain the top-*n* most important keywords.
3. Analysis of trends over time, where we obtain the top-*n* most important keywords per year.
4. Emerging keyword analysis, where we attempt to identify *new* and *rising* keywords.

In this section, we introduce each analysis separately using articles on workflow scheduling, the main theme of this article, published in timespan 2011–2020, i.e., the last decade. Using insights obtained from each analyses, we select four sub-domains to survey in-depth.

2.1. Analysis of the workflow scheduling community

We take a look at the workflow scheduling community by visualizing the collaborations and determine how many mathematical cliques are within that community. To construct each component (i.e., community), we draw an edge from one vertex (an author) in the component to another if they have co-authored an article. To determine cliques, we use the definition of a clique introduced by Luce and Perry [18]. Additionally, we look at the clique sizes, the amount of citations per author on average and at maximum per clique, and how often authors co-author together. When ranking authors or communities, self-isolated cliques are often seen as less desirable [19]. A community with a lot of interaction between different (groups of) authors are signs of a *healthy* community. This kind of information is of interest to community leader and event organizers; how large is a community, how diverse are the collaborations, how is the organization of the community, etc.

To get insights into the workflow scheduling community, we analyzed the author and citation information of articles returned by **Query 1**. Fig. 2(a) present the structure of the community. From this figure we observe many large collaborating communities, which indicates a dynamic and collaborating community. Additionally, we observe plenty of authors forming "bridges" between two or more groups. We believe such bridges are positive, as they may facilitate individuals in these respective groups working together and gaining knowledge from the other groups.

Fig. 2(b) shows the number of cliques per clique size. As we observe, most cliques are of size 2–5, which is a normal set of authors on a single article, forming a clique per definition. Overall, there are only a few large cliques. Together with the visual of the community, it seems that the community is of a more collaborative nature than forming tightly connected, yet closed groups.

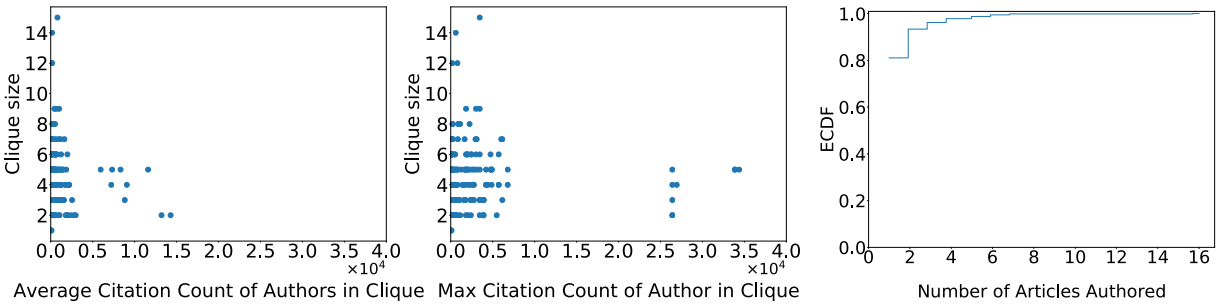
If we look at clique size versus average and maximum citation count per clique, visualized in Fig. 3(a), we observe that well-cited authors, both in maxima and on average, are not forming or participating in large cliques. This further adds to the intuition of a collaborative community.

Finally, Fig. 3(b) shows a CDF of the number of articles published per author in the workflow scheduling community. Roughly 80% of the authors publish a single article in the workflow scheduling domain in the span 2011–2020, with a long tail having authors publish up to twenty articles.

2.2. Method for keyword analysis

Identifying keywords is an effective way to obtain important topics within text [20]. What keywords are important given a set of articles? How often do we see the same keyword appear? Does the importance of keywords change over time? To obtain important keywords from articles matching a certain scope, defined by a database query, we apply the following process to sanitize and refine the data and then use Term Frequency–Inverse Data Frequency (TF-IDF). TF-IDF is a commonly applied technique in the information retrieval domain to obtain important keywords from text [21]. The process we apply is as follows.

First, two queries are defined. The first query is to construct a corpus that we will use to compare articles against. Such a corpus is required using TF-IDF to determine the commonality of words and thus rank their "uniqueness". The second query is to fetch articles of interest, e.g., articles having certain keywords in their title or abstract. The corpus we compare against is broader; it contains more articles using a broader scope, so that TF-IDF can identify the unique words within the community of interest targeted by the second query. In this work, we use as corpus all



(a) A scatterplot depicting the average (left) and maximum (right) citations of the authors for each clique in the workflow scheduling community.

(b) A CDF of number of articles authored per author in the workflow scheduling community.

Fig. 3. Clique size vs. author citation count (average and max) and a CDF of number of articles authored.

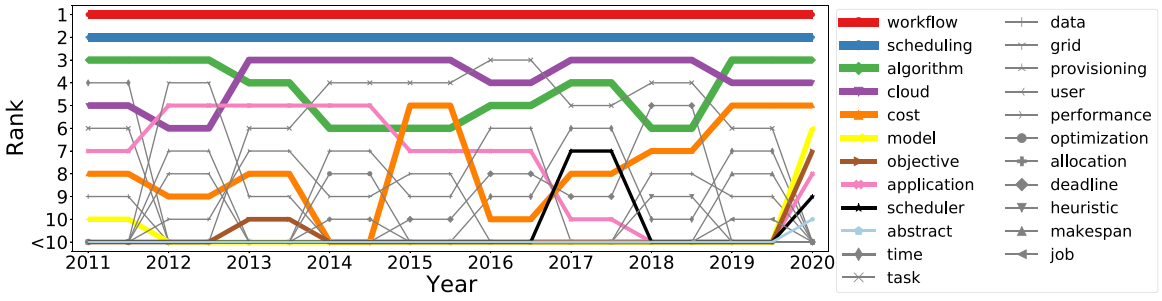


Fig. 4. Top-10 keywords in scheduling workflow articles in the past decade per year. Colored lines are part of 2020's top-10 and have a decreasing line thickness for better visual tracking. The legend shows the colored curves in order, and gray curves out of order.

Table 1
Top-10 keywords in articles on scheduling workflow published between 2011 and 2020.

Rank	1	2	3	4	5	6	7	8	9	10
Word	Workflow	Scheduling	Cloud	Algorithm	Task	Cost	Application	Time	Data	Deadline

articles from the systems community, i.e., the contents of the AIP database.

Text of articles that match these queries are preprocessed using a process similarly to [22].

Next, we compute keywords unique per article using TF-IDF. We extract the top-50 most important keywords per article based on their TF-IDF values. We count and store in a list the occurrences (term frequency) of the top-50 keywords found to obtain a ranking for all articles that match our second query. We limit the number of words we extract per article, otherwise we would end up counting all words in all articles matching the second query, defeating the purpose of TF-IDF.

We can then use this list to, e.g., create a top-n of keywords. If this top-n contains words that have no significant meaning, we manually filter them out and take the next meaningful word.

Scikit-learn 0.23.2 was used to compute the TF-IDF vector, TextBlob 0.15.3 was used for Lemmatization, Pandas 1.1.2 was used for computing and cleaning data, and NLTK 3.5 was used for stop word filtering along with a list of custom stop words.

To support reproducibility and FAIR data, all instruments and tools, scripts, and the database containing article meta-data used in this article are available as open-source artifacts.

2.3. Analysis of keywords in workflow scheduling articles

Observation-1 (O-1): The keywords "task", "time", "cost", and "deadline" are often mentioned in articles on workflow scheduling, highlighting the focus of the community on these topics.

To inspect what the focus is of articles working on scheduling and workflows, we first look at the top-10 most important keywords using the method described in Section 2.2. We inspect articles returned by **Query 1**.

The results are in Table 1. From this table we observe, besides the "workflow" and "scheduling", that the notion of clouds is the important keyword. This makes sense as most articles on workflow scheduling target either public or private cloud settings, which also explains why "cloud" is ranked third.

Consequently, "tasks", "application", "computing", and "cost" are popular keywords as these are closely aligned with workflow scheduling in clouds. In particular, plenty of scheduling policies focuses on the duo "cost" and "deadline". This can also be observed in our keyword analysis for workflow allocation (Section 5.1) and in the mapping in Section 8. As mentioned prior, "data" is a keyword that we did expect. Plenty of research on workflow scheduling produces or relies on data, from characterization to simulation studies. Scientific workflow applications are often used as a use-case for experimentation, but rarely is the data provided as auxiliary data for reproducibility purposes [5].

2.4. Analysis of keyword trends over time

O-2: The keyword "cloud" grew in importance over time, and remained in the top 5 since 2013, highlighting the importance of this topic.

O-3: The ranks of "task", "deadline", "user", "cost", "time", "heuristic", "makespan", and "algorithm" fluctuate, yet are often in the top-10 during the years 2011–2020.

To see how the focus of the community working on scheduling workflows shifted, we visualize in Fig. 4 the top-10 keywords per year. We take the output of **Query 1** per year, and apply the same method as described in Section 2.2.

From this figure, we observe that "cloud" already became a popular term in 2011. Since 2013, the term has been in the top-4 consistently. We also see a clear shift in focus: whereas "grid" was a popular term before "cloud" emerged, "grid" disappeared from the top-10 after 2013. We conjecture the general concept of "cloud" and related concepts such as Infrastructure as a Service (IaaS) increasing in popularity in industry and academia, with similar meaning and thus taking over. The keywords "cost", "deadline", and "heuristic" are keywords that have been rising in importance in recent years. Other keywords such as "algorithm", "task", and "data" appear to be consistently important, which makes sense provided they are general concepts and building blocks in workflow scheduling.

2.5. Discussion on emerging trends

O-4: Multi-objective and in particular makespan and deadline-aware scheduling are growing in popularity.

Emerging trends are often a good topic for research as they are deemed interesting by the community. We investigate which keywords increased in attention. Emerging keywords can indicate a further increase of attention in the future. We attempt to detect emerging trends in two ways:

New keywords: Keywords that were found by our method to be among the most common/important keywords in recent years, but did not come up in previous years. This type of analysis highlights keywords that previously were not common/important or are new and gaining traction fast. In this survey, we compare half of the selected time span, i.e. 2016–2020, with the remainder (2011–2015).

Rising keywords: Keywords that throughout the investigated years kept monotonically increasing in rank since their appearance. This type of analysis finds keywords of two categories:

1. Keywords that received more (or the same amount of) attention each year and thus indicates an interest.
2. Keywords that became emerging in the last year of the timespan checked.

For each year that we investigate, we apply the method outlined in Section 2.2, and take the top-10 most important/frequent keywords.

2.5.1. Emerging trends in workflow scheduling

We attempt to discover new and emerging keywords using articles that match **Query 1**.

The keywords found during *new keywords* analysis are as follows.

"deadline" "makespan" "model" "multi" "objective"

We observe the keywords "deadline" and "multi", "objective" research the top 10 in the last 5 years. This underlines the focus of workflow schedulers on several, often multi-objective metrics. "Makespan" is common metric which relates to deadlines. "Model" may relate to the workload or

If we look at *rising keywords*, we obtain the following.

"workflow" "model" "multi" "objective"

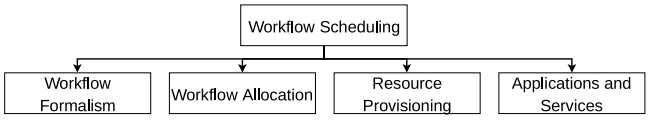


Fig. 5. The four areas within workflow scheduling focused on in this article.

Table 2
Possible relationships between all keywords in Fig. 4 and the areas related to workflow scheduling.

Area	Keywords found as important in Section 2.4
Workflow formalism	Task, data, graph, application, environment, job, user
Workflow allocation	Task, deadline, cost, algorithm, heuristic, graph, grid, application, time, environment, job, performance, user, optimization, objective, makespan
Resource provisioning	Task, cost, algorithm, heuristic, graph, grid, application, time, environment, job, performance, user, provisioning, optimization, objective, makespan
Applications and services	Data, cost, graph, grid, application, time, environment, job, performance, user, provisioning, optimization, makespan, scheduler

Again, we see the keywords "multi" and "objective", yet this time alongside "workflow". "workflow" is expected as it is the focus of our query. We observed from Fig. 4 that it is consistently at number one, thus being monotonically increasing. As for the keywords "multi" and "objective", they entered the top-10 in 2020, thus monotonically increasing. These keywords indicates that multi-objective schedulers are becoming more popular. From experience, single-metric schedulers no longer deliver the required performance with the diverse set of functional and non-functional requirements both the cloud provider and its users have; we expect the focus of multi-objective schedulers will remain and increase.

2.6. Future research directions inspired by meta-data analysis

Both the important keyword section and the emerging trend section suggest that non-functional requirements such as costs and deadlines are important. Moreover, multi-objective schedulers are increasing in importance, which makes sense given the complexity and demands around clouds and the applications being run. Our conjecture is that being deadline-aware whilst optimizing for other metrics such as costs, and energy consumption will continue to grow in importance and is an excellent topic for future work. Further refining these analyses and introducing new angles of investigation is another interesting item for future work.

3. Investigating and taxonomizing four areas within workflow scheduling

Taxonomies provide a structured and detailed decomposition of a certain topic and/or field. Decomposition allows for a good overview of possible and attempted avenues to tackle challenges. Using the overview, researchers attempt to find a feasible or optimal solution to challenges. These taxonomies can also provide new ideas for methods and/or combinations not attempted yet.

To limit the scope of this survey, using the keywords obtained in Section 2.4, we focus on four areas within workflow scheduling, depicted in Fig. 5: Workflow Formalism, Workflow

Query 2: SELECT * FROM publications WHERE year BETWEEN 2011 AND 2020 AND (lower(title) LIKE '%workflow%' OR lower(abstract) LIKE '%workflow%') AND ((lower(title) LIKE '%formalism%' OR lower(abstract) LIKE '%formalism%') OR (lower(title) LIKE '%language%' OR lower(abstract) LIKE '%language%'))

Allocation, Resource Provisioning, and Applications and Services. We select these four as they relate closely to the keywords found in Section 2.4. Table 2 shows our selection of keywords, per area.

Formalisms describe the way workflows are represented, and the possible features they can have, i.e., different formalisms support different notions of computation. Not many surveys focus on this aspect, yet we believe it is important as the formalism defines what properties can and cannot be captured.

Workflow allocation is the problem of assigning units of work to the available resources to adhere to the various QoS constraints set, while potentially attempting to improve other aspects such as resource utilization or power consumption.

Resource provisioning covers the research of when to allocate resources and how many given current and predicted demand. Adding the right amount of resources is crucial in lowering costs and improving the overall resource utilization, while avoiding slowdowns and other issues in the system.

Application and services cover the different type of resources, the execution model, and services that are considered in literature and available today.

These four main elements will be covered in the next four sections, each with their respective (sub-)taxonomies.

4. Taxonomy of workflow formalisms

A workflow formalism provides a language to construct workflows with. To create an overview and taxonomy of formalisms commonly used with workflow scheduling, we perform a systematic search to find articles on this topic and complement it with our experience. The query used to obtain articles on workflow formalisms for our systematic search is visible in **Query 2**.

Complementing the workflow structure taxonomy of Yu et al. [11], our taxonomy of workflow formalism is presented in Fig. 6 (left) and consists of two main branches: the enabled structure and what we call the core language, covered in Sections 4.2 and 4.3 , respectively.

4.1. Community and emerging keywords analysis

Based on results presented in our technical report [17], we make the following observations:

O-5: The formalism community is a small yet healthy community. A few large components exist and most author-relations are one-time.

O-6: Many of the emerging trend keywords indicate that users and convenience of use are growing in importance.

O-7: Larger cliques have a lower citation author citation count both on average and in maxima.

O-8: Over 90% author a single article.

O-9: Authors do not (co-)author more than two articles in this space.

4.2. Taxonomy of enabled structures

The enabled structure refers to the constructs possible within the workflow. We differentiate between Directed Acyclic Graph (DAG) and non-DAG. Due to the constraints between tasks and to prevent increasing complexity when having to deal with (complex) loops, most papers use the DAG formalism to represent workflows [23]. The DAG formalism is a simple and general concept often used in other fields. However, since this formalism is abstract, many implementations exist that allow developers to express their programs as a DAG.

Non-DAGs have the same entities as DAG, yet offer one additional instruction: iteration (or looping) [24]. Some workflow management systems support the non-DAG formalism, yet most well-known systems use DAGs. Many formalisms implementing a (non-)DAG formalism exist that are used by various other systems. Bastos et al. [25] look at the different structures of workflow formalisms for interchanging specification between workflow management systems.

Fig. 6 (right) presents a non-exhaustive overview of how workflow formalisms relate to the common abstracts of DAG and Non-DAG formalisms.

4.3. Taxonomy of core languages

Next to the structure enabled by the formalism, we use the term core language of the formalism to depict the language used to construct these workflows. We introduce this term to avoid ambiguity between the terms "formalism", and "language" which are used interchangeably in literature. Core languages can be generic purpose languages such as CSV, XML, YAML, and JSON. Example of formalisms based on generic core languages include AGWL, JS4Cloud, DAX, DIS3GNO, and CWL. We manually inspect the articles return by **Query 2** and cover the core languages mentioned, if any.

The AGWL is a formalism from the grid era based on XML [26]. The language explicitly models parallelism, loops, and forks such as if-else statements.

JS4Cloud is a JavaScript based workflow formalism for defining and executing data analysis workflows [27]. It has been implemented in the data mining cloud framework.

The Pegasus project uses an abstract workflow formalism called DAX. A DAX file describes a workflow as a DAG in XML format.

Cesario et al. [28] introduce a DAG-based workflow formalism for designing and executing distributed knowledge discovery workflows in their workflow executing framework named DIS3GNO.

The Common Workflow Language (CWL) is a formalism to describe command line tools and create a workflow out of them [29]. The formalism focuses on portability. An example of the usage of CWL in this context is done by Jansen et al. who use CWL to create a reducible file format called RED to improve the reproducibility of deep learning workloads and data-driven experiments [30].

Formalisms that use a specialized core language include BPMN, Petri net, YAWL, WED-Make, and UML.

Business Process Model and Notation (BPMN) is a formalism commonly used in businesses to outline workflows or processes within a company [31]. The formalism is comprehensive as it features over 100 symbols, including support for cycles and human interaction in workflows.

Petri net (PN) is a formalism commonly used in chemistry to model chemical processes and reactions. It is similar to the DAG formalism, yet uses tokens and weights on links to describe dependencies [32]. Many variations of the original Petri net

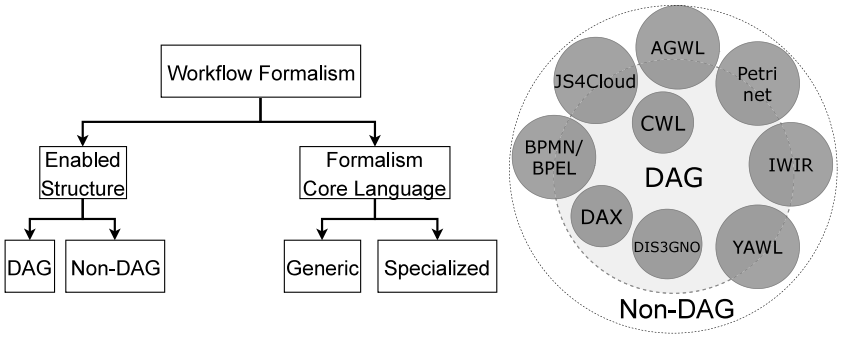


Fig. 6. Left: the taxonomy of workflow formalisms. Right: a Venn diagram showing how workflow formalisms relate to the abstract formalism of (non-)DAG.

formalism have been introduced to enhance its capabilities, for example the use of colored Petri nets [33]. Hoheisel et al. show how Petri nets can be used to model DAGs [34].

Yet Another Workflow Language (YAWL) is a formalism inspired by PNs [35]. It features similar constructs to BPMN yet is more simple in its constructs. YAWL can be used to construct DAGs [36]. The formalism supports dynamicity and has extensive support for (unexpected) error handling.

WED-Make is a workflow formalism introduced by the Elba toolkit to define dependencies and commands for execution [37]. Using the formalism, hidden and implicit dependencies are found and declared, guaranteeing they are respected.

CARMA is a workflow language aimed at stochastic process algebra for the representation of systems developed in the Collective Adaptive Systems [38]. The authors describe it as an "attribute-based availability model" where both the workload and the physical machines can be modeled with.

Song and Tilevich introduce a dataflow-based DSL for constructing workflows for microservices to reliably and efficiently execute them [39].

Other formals such as UML and control and data flow approaches are also used [40].

4.4. Future directions

There are several future directions that we believe are worth pursuing in the context of workflow formalisms. We believe non-functional requirements (NFRs) can be better incorporated in the formalisms. Our preliminary investigation [23] found the "DAG"-based solutions are the most common and the most simple to extend, but further investigation is required. Another interesting direction is to allow the environment to give hints or suggestions to the workflow management system through the formalism. The CWL-project is investigating incorporating this aspect into their formalism through the form of splitters,[2] where an "executor" can provide tools to split and combine chunks.

Capturing provenance related elements has been a focus for a while by the community. In general, reproducibility has received attention as of late. We believe incorporating provenance details in the formalism, such as input parameters, file names, hardware details, and other (potentially) important elements deserves more attention.

5. Taxonomy of workflow allocation

Workflow allocation is the process of placing the workflows onto available resources in such a way that the scheduling targets are met (see Section 5.2) while not violating any constraint. To achieve this, a workflow scheduler needs to take into consideration both global and/or local constraints and focus on a single or multiple criteria (see Section 5.3).

In this section, we focus on the diverse sub-parts of workflow allocation, see the taxonomy in Fig. 7. Each of the sub-parts will be discussed with their respective sub-taxonomies, some of which extend state-of-the-art taxonomies, e.g., [11,12].

Query 3: `SELECT * FROM publications WHERE year BETWEEN 2011 AND 2020 AND (lower(title) LIKE '%workflow%' OR lower(abstract) LIKE '%workflow%') AND (lower(title) LIKE '%schedul%' OR lower(abstract) LIKE '%schedul%' OR lower(title) LIKE '%plan%' OR lower(abstract) LIKE '%plan%' OR lower(title) LIKE '%allocat%' OR lower(abstract) LIKE '%allocat%')`

Query 3 is used to find articles related to workflow allocating, which in turn are used to verify the completeness of our taxonomies regarding workflow allocation.

5.1. Community and emerging keywords analysis

Based on results presented in our technical report [17], we make the following observations:

O-10: The workflow allocation community is reasonably big. Many relationships are one-time. Plenty of authors exists that "bridge" two groups, i.e., being the single link between two connected components.

O-11: "Deadline", "cost", and "multi-objective" are emerging and important topics within the workflow allocation community. We observed the same for workflow scheduling articles, and as allocation is more common focus than resource provisioning (based on number of articles and community sizes), this result makes sense.

O-12: Similar to the workflow formalism community, larger cliques tend to have a lower average citation count among the authors. Different from the workflow formalism community, the maxima can be found in both larger and smaller sized cliques.

O-13: Over 80% of authors author a single article, yet a small number authored up to sixteen papers in the timespan 2011–2020.

5.2. Taxonomy of scheduling targets

In this section we cover the optimization metrics described in Taxonomy 8, which significantly extends the taxonomy of Yu et al. [11]. These metrics are obtained by combining personal

[2] https://github.com/common-workflow-language/common-workflow-language/issues/446.

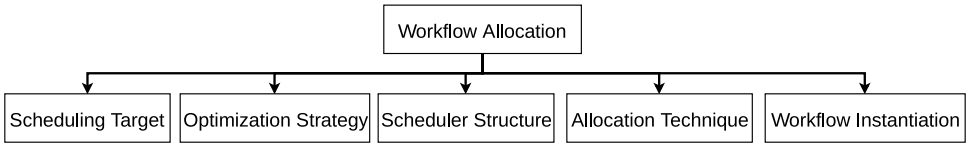


Fig. 7. Workflow allocation taxonomy.

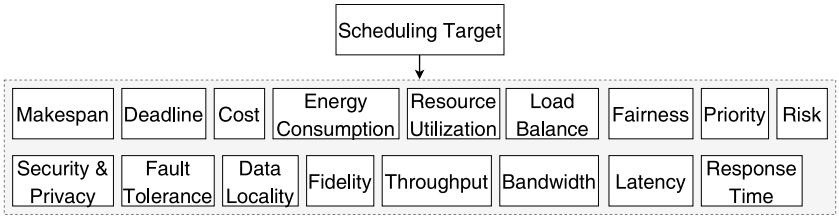


Fig. 8. The optimization target taxonomy. The gray box represent a collection and does not impose any difference between the elements.

experience complemented with analyzing the articles returned by the queries in this work, most noticeably **Query 3**. There are likely more optimization targets that policies use, yet the taxonomy discussed here covers a significant portion of them.

5.2.1. Makespan

Makespan (or *runtime*) is a common targeted metric when scheduling jobs. Makespan is the total time elapsed between the start and finish of the entire job. Several techniques have been used to minimize the makespan of jobs, including Particle Swarm optimization [41], simulated annealing [42], and min-cut/max-flow [43]. Dealing with latency sensitive applications also may require low makespans. An example of such a system is provided by Bonvin et al. [44].

5.2.2. Deadline

Related to makespan as scheduling target, deadlines are a more strict and may require different decisions of a scheduling system. Deadlines cover the total turnaround time, which is composed of wait time(s), makespan, and latency of submitting and obtaining a response [45].

5.2.3. Costs

Another common, yet important target for optimizing is costs. Cloud providers offer a pay-as-you-go model for leasing resources. Traditionally billing would be on an hourly basis, however, several cloud providers have moved towards a second-based billing granularity [46,47].

Cost is closely related to resource utilization (see Section 5.2.5). For example, autoscalers already are concerned with costs since resources pricing schemes differ per cloud provider.

Alkhanak et al. [48] provide an extensive overview and taxonomy of cost-aware approaches of workflow scheduling in cloud environments.

5.2.4. Energy consumption

With the growing importance of green computing, energy-aware scheduling is emerging, with new approaches and techniques being introduced. In 2014, datacenters already accounted for 2% of energy consumption in the US [49]. Datacenter operators are focusing on becoming energy-neutral, including Amazon [50], Google [51], and Microsoft [52]. Especially in high performance computing, the number of flops per watt has become increasingly important.[3]

[3] Keynote CCGrid 2018.

Articles in this domain focus on least-loaded machines [53], trade-offs between makespan and energy efficiency [54], Pareto-based scheduling [55], dynamic voltage and frequency scaling [56], power minimization in networks and protocols, and self-adaptive systems [57]. These techniques are sometimes combined as demonstrated in [58].

5.2.5. Resource utilization

Resource utilization denotes the efficient use of allocated resources. With the growth of cloud popularity, this metric is becoming increasingly important for cloud operators. Resource utilization levels of 70% are possible in domains such as super computing [59], yet the utilization of clouds is as low as 6%–12% are reported [60–62].

Cloud providers employ autoscalers (i.e. provisioning policies) to automatically scale resources based on the resource demand of the client. Autoscalers minimize under- and overprovisioning to improve resource utilization while not violating any QoS of the client. Especially when facing challenging, e.g., bursty or unpredictable workloads, autoscalers tend to perform differently [63,64]. The interplay between allocation and provisioning then becomes increasingly important to make sure resources are utilized properly.

5.2.6. Load balance

Some schedulers attempt to balance the load, i.e., distribute the work over workers in such a way that they are roughly equally loaded. This load can measured using various metrics; CPU utilization and RAM utilization are common metrics. Network can also be a target to balance [65]. Load balancing is related to Resource Utilization, yet differs on a few critical points that we believe warrant its own category:

1. You can have (non-uniform) load balancing, possibly without improving or even "worsening" resource utilization.
2. A lack of (uniform) load balancing can lead to higher failure rates for more highly utilized machines, so load balancing is not merely resource utilization maximization.
3. The two may even conflict in a multi-objective setting, latency and cost might lead to a conflicting state where a higher resource utilization is desired to reduce cost, but leads to a higher (tail) latency.

Load balancing can be used to impact other scheduling targets including latency, costs, throughput, and response time. Load balancing also plays a role in minimize resource contention, i.e., jobs delaying due to insufficient resources caused by all jobs requiring the same resources at the same time [66].

5.2.7. Fairness

The notion of fairness can have multiple definitions. Fairness can relate to an equal share of resources [67], sharing resources [68], equal slowdown [69], fair use in multi-resources with placement constraints [70], and slowdown [71].

Quang et al. [72] present a comparative analysis of two scheduling mechanisms for virtual screening workflows sharing the same infrastructure. They focus on fairness, overall system throughput, and response time.

5.2.8. (User-defined) priority

Some workflow execution systems support (user-defined) priorities. Deng et al. mention the use of priority-driven execution in their scheme [73]. An example of a policy taking priorities into account is PISA [74]. PISA differentiates between user priority levels e.g. free-tier and pay-tier users when scheduling. The level of priority determines the speed at which the user will be assigned the required resources when resource contention occurs. In the cluster traces released by Google, priority is also exposed as one of the field used to schedule [75].

5.2.9. Risk

Risk relates to allowing resource contention within acceptable bounds to reduce costs whilst still meeting the QoS requirements set by the customers. Van Beek et al. [76] describe risk based on CPU contention while running business-critical workloads. While focusing primarily on security, Li et al. [77] compute the risk rate proportional to the security levels and the distribution of risk to judge if it is within bounds.

5.2.10. Security & privacy

As public clouds are freely accessible by design, for some applications security is a desirable goal. Trust-based scheduling and result verification are necessary in these situations. Proposed solutions range include quiz systems [78], risk rate constraints [77], secure key sharing and fine-grained access control [79].

Shishido et al. [80] propose an extension to measure security overhead in CloudSim.

Following the recently enforced GDPR legislation in the European Union (EU), processing and storing data of EU citizens must happen on systems located in Europe. Data Protection Impact Assessments (DPIAs) methods are employed to identify and risks and rights of entity regarding data [81]. Countries such as Russia have similar legislation [82].

5.2.11. Fault tolerance

Fault tolerance is of vital importance when running business critical applications and required at several levels when running workflows. Replication, preemption, and checkpointing are common techniques to fault tolerance when executing tasks in datacenters.

Both *availability* and *reliability* are within the scope of fault-tolerance [83] where availability expresses the fraction of time a system is operational, and reliability the fraction of the system remaining operational during the processing of a task.

Fault-tolerance has been investigated when using spot and on-demand instances [84], using task replication and other resubmission techniques [85–87], challenges and tools [88].

A survey on the topic of fault-tolerance and taxonomies is provided by Poola et al. [89].

5.2.12. Data locality

With IO-intensive workflows, data locality can reduce the runtime and cost of workflows. It is especially important when sending data to and from cloud environments. Typically, sending data within the same cluster is free, yet communication to and from the cluster is not. Being data locality aware may also help in reducing costs by not having to send data.

Especially in the Map-Reduce domain, IO-intensive workflows are common. A well-know article on this topic is Xie et al. [90] who introduce a data placement scheme for MapReduce applications running on heterogeneous nodes. Articles such as [91] and [92] also focus on data-locality in MapReduce applications. The study by Wang et al. [92] has similarities to e.g. Duro et al. [93] who focus on the trade-off between data-locality and load balancing when executing generic workflow applications. Recent efforts also focus on data locality for entry (starting) tasks of workflows [94]. More articles on workflow scheduling with data locality exist, e.g. [95] and [96].

5.2.13. Fidelity

Fidelity relates to the quality of output of a workflow [11]. Cardoso et al. refer to fidelity as a function of effective design and in intrinsic property or characteristic of a good produced or service rendered [97]. Video streaming is a good example where fidelity versus computation power can be a trade-off. Another example is using dynamic voltage and frequency scaling to trade-off quality with power consumption [98].

5.2.14. Throughput

Throughput focuses on completing as many tasks in an as short as possible timespan. Different from focusing on makespan, throughput related approaches may not attempt to speed up the duration of tasks themselves by running them on, e.g., special hardware. Simply running more tasks in parallel could already be an feasible approach to improve throughput.

5.2.15. Bandwidth

Related to data locality, yet different, bandwidth can be a target as well. Some scheduling strategies involve messaging between components, where reducing bandwidth becomes important. An example of such work is that of Momenzadeh et al. [99]. Their work focuses on workflow segmentation to execute workflows on multiple Virtual Machines (VMs). Bandwidth, due to the message communication, becomes an important metric in such systems.

5.2.16. Latency

Latency defines the time it takes for the data to arrive at the computing infrastructure. Good examples of latency can be found in the IoT domain [100]. For example, Shell deploys IoT workflows for measuring pressure, temperatures, etc. in their oil refineries [101].

5.2.17. Response time

The total time it takes from submission to receiving the answer, i.e., output is the response time. Several policies exist that focus on this metric [102].

The latency, wait time (time spent in queue), processing time, and data transfer times to and from the computing infrastructure make up the response time, hence this metric can be improved across several dimensions.

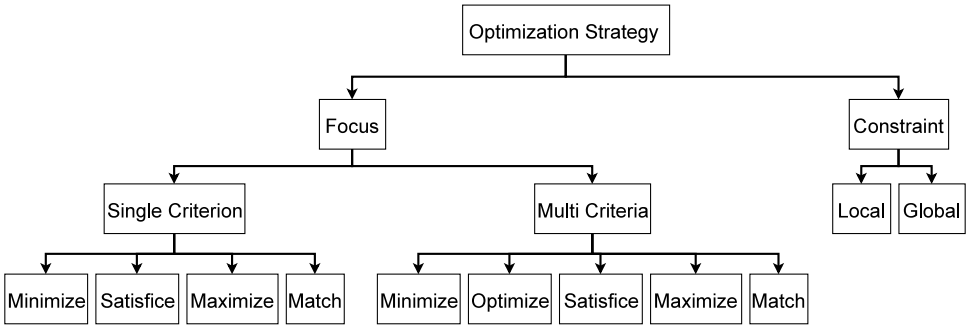


Fig. 9. Optimization strategy taxonomy.

5.3. Taxonomy of optimization strategies

The optimization strategy of policies varies in both *focus* and *constraint*. Fig. 9 presents the taxonomy for optimization strategies. The focus can be on a single criterion or multiple criteria. Popular single criterion for minimization are cost [103] and makespan [71]. Examples where policies must match a specific value can be found for example in Galaxy's workflow scheduler having to match tool versions [104], and Function as a Service (FaaS) instances relying on specific library versions [105]. Maximization policies focus on e.g. throughput [106] or fairness [71]. Satisficing is about generating "good enough" solutions. The term was introduced by Herbert A. Simon [107]. Jaeger et al. [108] use satisficing to modify business information system models using AI techniques. Zhang et al. [109] use iterative ordinal optimization to schedule scientific workflows in elastic cloud computing environments that satisfice the problem.

Multi-criteria policies consider multiple metrics at once. Similar to single criterion policies, multi criteria policies can both maximize, minimize and match certain criteria. New to multi-criteria is optimize. With optimize, two or more metrics are trade-off to create an overall better outcome. A popular combination of criteria to optimize is cost while meeting deadlines such as [110]. Recently, energy-aware workflow scheduling is also becoming more prevalent, see e.g. [111,112].

Constraints can be both local and global. Some schedulers take into consideration all eligible workflows and the entire available resource environment and device a best plan across the entire workload, e.g., *globally*. Some schedulers only focus on a single workflow at a time without even though multiple may be eligible, or focus on only one specific (part of the) resource environment. This may lead to the best placement of workflow currently being allocated, yet may not lead to the overall best outcome. We consider the constraints these schedulers target to be *local*. Policies can use a mixture of both.

5.4. Taxonomy of scheduler structures

In the past decades, several scheduling *structures* have been proposed. Fig. 10 presents the taxonomy of different structures. Often, scheduler structures are divided into centralized, decentralized, and hierarchical structures, which is quite coarse grained.

The first differentiation is to be made between single-cluster and multi-cluster architectures. Next, for each of these levels, we focus on multiple different structures within these levels. This extends significantly the characterization done by of Moghaddam et al. [12].

5.4.1. Single-cluster architectures

For a single-cluster scheduling architecture, one can use either a centralized or a decentralized architecture in which the bootstrapping problem is solved centrally. Both of these architectures have their trade-offs.

A centralized scheduler, sometimes called a headnode, acts as a coordinator and keeps track of the global state. Using such architecture makes it easier to manage resource yet may become a bottleneck or single point of failure. An example of a centralized single cluster scheduling architecture is that of Kubernetes. Centralized systems are generally monoliths and can evolve into sophisticated systems that are hard to change, e.g., the situation at Google prior to Omega [113].

In a decentralized architecture, schedulers are responsible for a part of the resources. A decentralized scheduler is more resilient to failures, yet keeping a global state is more difficult and comes at the cost of having to communicate between the schedulers. HTCondor is an example of a distributed scheduler in which a central matchmaker delegates the work to nodes. Due to this centralized component, the bootstrapping problem is avoided.

5.4.2. Multi-cluster architectures

With a centralized meta-scheduler, a centralized headnode sends jobs to different clusters. Using such architecture makes it easier to manage resource yet may become a bottleneck or single point of failure. Firmament [43] is an example of a centralized workflow scheduler that is fast, even at large scale.

In a fully decentralized setting, costumers can check compare each cluster and independently submit to these clusters (observational scheduling). When load-sharing happens, this can turn into a fully decentralized, federated architecture, for example, OurGrid [114].

Hierarchical architectures attempt to combine some of the benefits centralized and decentralized architectures offer. In a hierarchical architecture, tasks often pass multiple schedulers in a layer fashion [115]. Examples of hierarchical schedulers are PUNCH, CCS, Moab/Torque, and Flux. PUNCH and CSS were one of the first tools to use a hierarchical architecture for scheduling in large-scale distributed environments with CSS being able to operate both in cluster and supercomputer environments [116]. Moab/Torque is a commercial scheduler that to date is still used in distributed environments. Flux is a scheduler framework that aims for scalable, easy-to-use, and portable execution of large workloads [117]. Flux enables (and encourages the use of) a hierarchical setup for scheduling using their APIs. For example, the Los Alamos National Laboratory uses Moab, as can be observed from recent workflow traces that originate from this lab [118].

With delegated matchmaking, instead of performing load-balancing by sending jobs to other clusters, clusters resources usage rights are delegated, rather than jobs [119]. Questions regarding who controls the delegation and notions of fairness arise in such architectures.

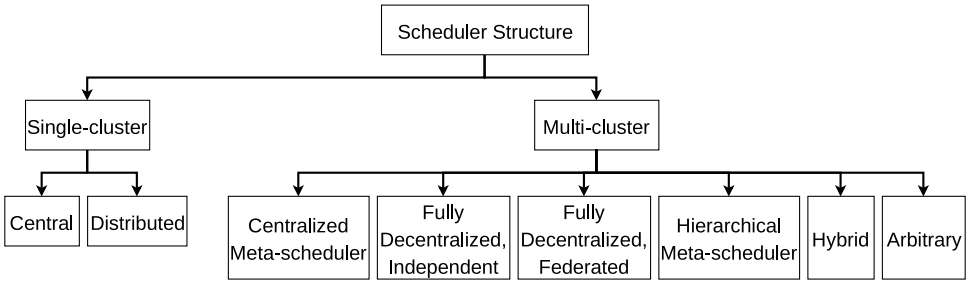


Fig. 10. Taxonomy of scheduler structures.

Shared-state scheduling is a special kind of fully decentralized, federated approach. It involves a situation in which multiple schedulers have an overview of and can manage and lay claim to *all* resources in a given environment. To schedule using this structure, you need to provide a unified notion of when resource allocations are permitted and a notion of precedence (who wins when competing). The Omega scheduler [113] is a well-known example of using shared-state scheduling.

Other arbitrary architectures includes architectures such as by a TAGS-based policy in which clusters serve different job depending on their runtime. This is implemented in KOALA-C in where jobs that run too long are preempted and moved to another queue for execution.

5.5. Taxonomy of allocation techniques

The *technique* used by a scheduler determines how the schedule is created. It has been shown that computing the optimal schedule is an NP-hard problem. Computing the optimal schedule is therefore infeasible in terms of time, especially with the ever-increasing dynamic workload.

To this end, many different techniques have been proposed to generate an optimal or near optimal schedule in feasible time. In this section we will discuss various techniques employed for task placement. Fig. 11 presents the taxonomy of task placement techniques.

5.5.1. Greedy

Selecting jobs and/or resources greedily sometimes helps in reducing time required to compute a schedule, or push a solution to a local optimum. Greedy algorithms are often used to generate "good enough" solutions within a timely manner.

Xiang et al. [120] introduce a greedy ant colony optimization algorithm that performs greedy machine allocation with low overhead. Yu et al. [121] mention several greedy algorithms for scheduling workflows in grid environments.

5.5.2. Game theory

Game Theory is another technique employed by schedulers to meet scheduling targets. The ICENI scheduler uses game theory for scheduling workflows, for example [122]. Yaghoobi et al. [123] use a game theory approach for scheduling workflows in grid environments to minimize turnaround time and cost. Duan et al. [124] introduce a workflow scheduling policy based on game theory that attempts to optimize for both makespan and cost while taking network bandwidth and storage into account.

5.5.3. Random

Scheduling eligible tasks randomly is often done to obtain a baseline when experimenting. First-come-first-serve (FIFO) is usually used as an allocation policy in this case. For example, Wu et al. [74] use the FIFO sequence for a random baseline.

Another random method is lottery ticket scheduling [125]. Resources get assigned a certain amount of tickets and for each task, a ticket is drawn at random. The task is then assigned to the corresponding resource if it fits, else a new ticket is drawn.

5.5.4. Heuristic

Heuristic approaches apply best-effort methods that work well for a given setting, e.g. workload of workflows, or use specific elements from a domain. Since scheduling tasks is a NP-hard problem, many policies rely on heuristic for task placement decisions. Examples of specific domain properties related to workflow scheduling can be task runtime, task size, etc.

Examples of policies that use heuristics are SJF [42] and HEFT [126].

5.5.5. Meta-heuristic

Meta-heuristics are the class of heuristic that are problem independent. Examples of such meta-heuristics applied for workflow scheduling are ant colony optimization [127], cat swarm optimization [128], Shuffled Frog Leaping [129], evolutionary algorithms such as genetic algorithms [130], and simulated annealing [131]. Wu et al. [132] present a revised particle swarm optimization approach.

5.5.6. Machine learning

Machine learning describes the notion where a system can make decisions based on prior seen (similar) situations. These systems require prior training before being able to classify new cases. Due to the agnostic principle of machine learning, it can be used for many different purposes, including scheduling. With the recent surge in interest for machine learning, the number of approaches in scheduling and autoscaling using this technique have increased too.

Vukmirovic et al. [133] use artificial neural networks for dynamically executing scheduling algorithms. Bauer et al. use machine learning for autoscaling multi-tier micro services [134].

5.5.7. Exhaustive search

Exhaustive search algorithms compute the optimal planning given a workload and resource settings. However, as scheduling workflows is NP-hard, in practice such solutions are infeasible. One of the most famous examples of scheduling is bin packing. Exhaustive search algorithms are infeasible in practice where real-time decisions need to be taken.

Examples of exhaustive search algorithms are ILAO and CO-LAO [135].

5.5.8. (Non-)linear programming

(Non-)linear programming can be used to construct mathematical models in which requirements can be specified by (non-)linear relationships. Such models can then be used to compute the optimal outcomes given the constraints. (Non-)linear programming has been used to construct workflow schedules, often with Service Level Objective (SLO) defined as constraints. Such schedules can be either used for verification or benchmarking purposes (comparison), or to use as scheduler (functionality).

The applicability of (non-)linear programming ranges from executing workflows in grid environments [136], multi cloud environments [137], to scheduling pipelined workflows [138].

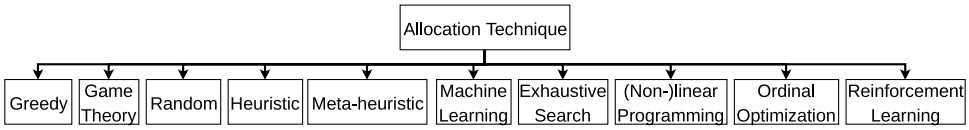


Fig. 11. Allocation policy technique taxonomy.

5.5.9. Ordinal optimization

Ordinal optimization was introduced by Ho et al. [109] to effectively generate local-optimal solutions (or "good enough") to NP-Hard problems. Zhang et al. [109] extended ordinal optimization and included an iterative approach to reduce the search space and overhead. Ordinal optimization has also been used in combination with other techniques. El-Zarif et al. [139] use ordinal optimization to improve parameter selection for their genetic algorithm approach. While this work is not in the context of (workflow) scheduling, this approach could be tested since genetic algorithm approaches have been introduced by related work (see Section 5.5.5).

5.5.10. Reinforcement learning

With reinforcement learning, the system contains a feedback loop that tunes parameters according to the feedback obtained. If implemented correctly, a system containing such a feedback loop will correct itself to changes in environment and workload. Examples of workflow scheduling using reinforcement learning are Ma et al. [101] who combine a Q-learning approach to portfolio scheduling, Wang et al. [140] apply a Deep-Q-network in a multi-agent reinforcement learning setting to improve both workflow makespan and cost,

5.6. Workflow instantiation

Yu et al. define the notion of abstract and concrete workflows [141]. An abstract workflow defines the tasks and their dependencies, yet lacks the detail of where each task will be ran and where the data should be read from and written to, which the concrete workflow entails. The concrete workflows is therefore an instantiation of the abstract workflow.

The process of instantiation can be done statically or dynamically. In the static case, concrete workflow plans are generated before execution in accordance to the latest state of the system. Any dynamic changes in this state are not taken into account. Dynamic schemes do make use of the dynamics in state as well as static information beforehand to make scheduling decisions at runtime.

User-directed and simulation-based scheduling are common when creating static schemes. In user-directed scheduling, the consumers themselves emulate the scheduling process and assign resources to tasks, or modify workflows themselves [11]. This process is often done by human experts who rely on their knowledge and can also incorporate preferences and/or other QoS criteria such as performance or availability. In simulation based scheduling, the "best" schedule is picked after simulating the workload on a set of resources according to defined metrics.

5.6.1. Partitioning technique

Other aligned methods arose to instantiate workflows. For example in graph processing where partitioning techniques are applied to the graphs first. The graph is partitioned based on the graph itself and the algorithm (often workflows) to be applied [142]. Similarly to workflow allocation policies managing data locality, the partitioning of the graph determines how the workflow is instantiated and which tasks of the workflows run where and on what data.

Query 4: `SELECT * FROM publications WHERE year BETWEEN 2011 AND 2020 AND (lower(title) LIKE '%workflow%' OR lower(abstract) LIKE '%workflow%') AND (lower(title) LIKE '%provision%' OR lower(abstract) LIKE '%provision%' OR lower(title) LIKE '%autoscal%' OR lower(abstract) LIKE '%autoscal%')`

5.6.2. Workflow and task optimization

Several workflow management systems perform optimization steps when having created an concrete workflow. An example of such a system is Pegasus [143]. First, the Pegasus Mapper holds the abstract workflow. The Mapper can e.g. reorder, group or prioritize tasks to improve performance. This workflow is then passed on to DAGMan, which turns the abstract workflow into a concrete workflow, by e.g. determining where each task will run and where the data will reside. DAGMan then monitors the execution of the workflow and tracks if task dependencies have been met. Finally, the HTCondor scheduler executed the workflow on the targeted resources.

5.7. Future research directions inspired by meta-data analysis

Cost and deadline-aware scheduling remains an important and growing topic within the workflow allocation community as seen in Section 5.1. Given the recent emerging topics of Edge/Fog, IoT, and serverless computing, we believe there are plenty of opportunities within this domain.

Metrics such as Risk and Fidelity are less studied in the context of workflow allocation. We believe especially for business critical workflows, these metrics are important. More work on these topics, especially in emerging areas as Edge/Fog etc.

Another topic we believe will grow in importance is green computing, i.e., a focus on reducing power consumption while adhering to all functional and non-functional requirements. Sustainable energy sources are already invested in heavily. With datacenters being major power consumers, and the power consumption is likely to rise, it is worthwhile investing into this topic [144].

Finally, we believe policies with multiple criteria or objectives will become the norm. Already, we see many policies focusing on more than one metric (see Section 8).

6. Taxonomy of resource provisioning

In this section, we discuss resource provisioning. The scope of this section is limited as this topic deserves a survey itself. As workflows do play an important role in several autoscalers, we will cover both provisioning in general and autoscalers. Novel compared to related work, we cover elasticity and the offloaded provisioning model.

Query 4 is used to find articles related to workflow resource provisioning, which in turn are used to verify the completeness of our taxonomies. For each article, we check if the proposed approach can be mapped onto our taxonomy, incrementally adding missing elements. We will show that recent policies map well to our taxonomy in Section 8.

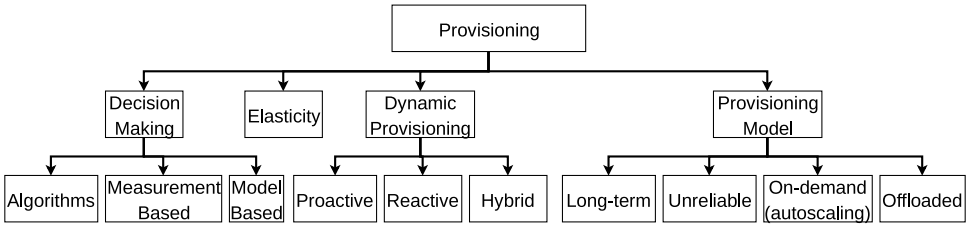


Fig. 12. Autoscaler taxonomy.

6.1. Community and emerging keywords analysis

Based on results presented in our technical report [17], we make the following observations:

O-14: The resource provisioning community is relatively small. Many authorship relationships are one-time. The connected components of size greater or equal to five are quite diverse and dynamic.

O-15: Elasticity and the environment seem to be emerging and trending keywords within the resource provisioning community.

O-16: The average and maximum citation count per clique situation is very similar to that of the workflow allocation community. The only difference is that there are less outliers.

O-17: Similar to the workflow formalism community, around 82% of authors author a single article. The highest number of co-authored papers by a single author was six in the year span 2011–2020.

6.2. Taxonomy of provisioning

We select from and extend the provisioning taxonomies of Smanchat et al. [13] and Shoaib et al. [145]. We only focus on the provisioning model, decision making, elasticity, and dynamic provisioning strategies as they relate closely to our scope of workflow scheduling (see Fig. 12).

Provisioning decisions are divided into three categories: algorithms that make decisions, (static) scaling decisions based on measurements, and scaling decisions based on models [145].

Algorithms typically consider multiple parameters, including deadlines, costs of resources, the workload, etc. These algorithms may incorporate models to make decisions.

Scaling decisions based on measurements are more simplistic. For example, when scheduling a bag-of-tasks, assuming that each task requires one CPU core, the amount of allocated machines might be straight forward.

Resource provisioning systems can also rely on performance models to make decisions. Shoaib et al. [145] refer to several in their survey.

Elasticity defines how well a provisioning approach scales with the need for resources. Ilyushkin et al. [63] define several novel metrics for system elasticity. We cover this topic in Section 6.4.

Dynamic provisioning refers how autoscalers respond to changes in resource requirements. Proactive approaches predict changes and act accordingly, to attempt to avoid over- and underprovisioning scenarios. The danger of such an approach is miscalculating the required resources. Reactive approaches respond to changes that have already taken place or are taking place. Reactive approaches might lead to short periods of over- and underprovisioning, yet follow changes in demand more closely albeit delayed. Finally, hybrid approaches combine both techniques. A typical combination is changing resources on a reactive basis approach when facing, e.g., bursts, yet be proactive with common patterns such as diurnal use of resources.

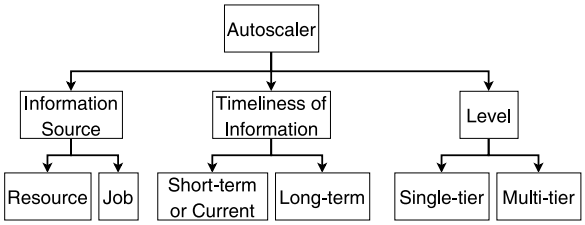


Fig. 13. Autoscaler taxonomy.

Finally, the provisioning model can vary. Long-term, unreliable, and on-demand provisioning are covered by Smanchat et al. [13]. Long-term resources are rented for extended periods of time, up to years. Unreliable provisioning relates to resources that may not always be available at a certain price, or simply available at all. Amazon's Spot Instances is an example of such unreliable resources. On-demand provisioning is the model of getting resources from a (usually) a fixed list from cloud providers. When scaling resources up and down to deal with sudden flash crowds, typically on-demand provisioning models using autoscalers are used. We cover this topic more in-depth in Section 6.3. Finally, we add to this category the offloaded model. In this case, the provisioning of resources is managed by the resource provider. An example of this is using the autoscalers of Amazon's autoscaling service.

6.3. Taxonomy of autoscalers

Autoscalers are employed to scale resources during changing resource requirements of workflows following a *provisioning policy*. The provisioning policy dictates how many and when resources are (de)allocated. Fig. 13 presents the taxonomy of autoscalers based on Ilyushkin et al. [63].

In workflow execution, autoscalers that solely monitor server-level information as their information source for making scaling decisions are agnostic to the workload. Examples of informative metrics used are current throughput, length of the task queue, and amount of available resources.

Workflow-specific autoscalers exploit the structure of workflows to improve provisioning decisions. Examples include estimating the level of parallelism in workflows [146] and construction partial execution plans for eligible tasks [63]. In our recent work [64], we demonstrated that autoscaler performance varies as the workload, environment, and other system components change. This indicates that careful benchmarking and proper identification of strengths and weaknesses of autoscalers is required. Such new insights can be exploited into new approaches and possibly new scheduler designs where allocation and provisioning are co-designed.

Timeliness of information refers to how recent the data is that is being used to estimate incoming workloads. There are two classes in this branch: short-term or current information and long-term. Autoscalers that only operate on the current incoming amount of workload or very recent information. There is no clear definition on how "recent" information should be, generally this differs per autoscaler. Long-term information spans days to even months or years where diurnal or even seasonal patterns can be observed.

The level at which autoscalers can operate is either single-tier or multi-tier. Single-tier autoscalers typically manage the resources for a single application. Multi-tier autoscalers such as Chamulteon [147] and FAHP [148] scale resources for multiple, different applications.

6.4. Elasticity

When additional resources are required, the autoscaler must obtain enough resources as fast as possible, potentially in accordance with other NFRs such as adhering to a budget. Similarly, when resources are no longer required, they should be deallocated to avoid unnecessary costs. *Elasticity* defines how well a system responds to changes in resource requirements without looking at the secondary requirements. The work of Ilyushkin et al. [63] use and introduce several metrics for elasticity. Among other elements, these metrics capture *overprovisioning*, i.e. the time and amount of resources that were idle and *underprovisioning*, i.e. the time and amount of resources that were required but not provisioned.

6.5. Allocation and provisioning policy interplay

The allocation policy can have a direct impact on the performance of an autoscaler (and the provisioning policy that goes with it). Versluis et al. [64] demonstrate that without task preemption resources may remain in use, yet underutilized due to the autoscaler being unable to deallocate these resources. Andreadis et al. [6] demonstrate that scheduler components, including the allocation and provisioning policies, are systematically underspecified. Underspecification of such policies and components can lead to significant differences in performance, hampering reproducibility.

Understanding both how resources are used through allocation and how they are provisioned are vital in creating a well-balanced system. Work such as that of Malawski et al. [149] investigate scheduling techniques that perform both resource allocation and provisioning. We conjecture that this interplay is important to investigate in order to improve resource efficiency and understanding systems better.

6.6. Future directions

Emerging areas present plenty of opportunities for resource provisioning research. With Edge datacenters becoming emergent in the Edge/Fog domain, resource provisioning policies should start taking these types of resources into account. Typically, for latency sensitive applications, the cost vs. benefit ratio can play an important role.

The rise in popularity of containerized applications through e.g. Docker is also gaining in popularity. Already products such as Docker Swarm and Kubernetes for container orchestrations are widely adopted by both academia and industry. Especially in the area of FaaS resource provisioning is an important aspect. Starting a VM or container incurs significant delay in function turnaround time as VMs or containers with specific libraries and/or versions have to be booted. Already work such as that of Aumala et al. [150] focus on package aware load balancing to speedup function deployment.

Moghaddam et al. [12] provide an extensive survey on resource provisioning and performance management. Several directions for future work are included in their work.

Another item for future work is reviewing and improving the interplay of the allocation and the provisioning policies. Improving this may lead to improved resource utilization and reduced resource consumption.

Query 5: `SELECT * FROM publications WHERE year BETWEEN 2011 AND 2020 AND (lower(title) LIKE '%cloud%' OR lower(abstract) LIKE '%cloud%') AND (lower(title) LIKE '%service%' OR lower(abstract) LIKE '%service%')`

7. Taxonomy of applications and services

Cloud providers offer several kinds of different services nowadays, most of which still eventually translate into running workflow applications. To this end, we divide this space using the taxonomy in Fig. 14. Each of these branches are covered in this section.

Query 5 is used to find articles related to cloud computing services, which in turn are used to verify the completeness of our taxonomies.

7.1. Community and emerging keywords analysis

Based on results presented in our technical [17], we make the following observations:

O-18: 86.8% of co-authorship relations are one time.

O-19: The cost and performance keywords became more important in the last five years in the span 2011–2020. The term "cloud" has been rising in importance as well.

O-20: Also for the applications and services communities holds that members of large cliques tend to have a lower total citation count on average and in maxima.

O-21: Around 70% of authors authored a single article in the timespan 2011–2020. This is surprising given the scope of this community (and our query).

7.2. Service types

7.2.1. IaaS

IaaS is the notion of renting resources from a cloud operator. These resources can either be virtualized or real. Typically, IaaS resources come with an clean OS on which dependencies, libraries, and packages, etc. have to be installed by the client. Until recently, resources were leased per hour, however most major vendors including Amazon and Microsoft now offer a per-second leasing granularity.

7.2.2. PaaS

Platform as a Service (PaaS) is the category of cloud services that allow users to install, configure, deploy, run, and manage their own applications without having to deal with any underlying infrastructure. It was derived from Software as a Service (SaaS) [151]. The deployment, maintenance, and upgrading of infrastructure are outsourced to the cloud provider. This service enables e.g. specific versioning or configuration of software when compared to SaaS.

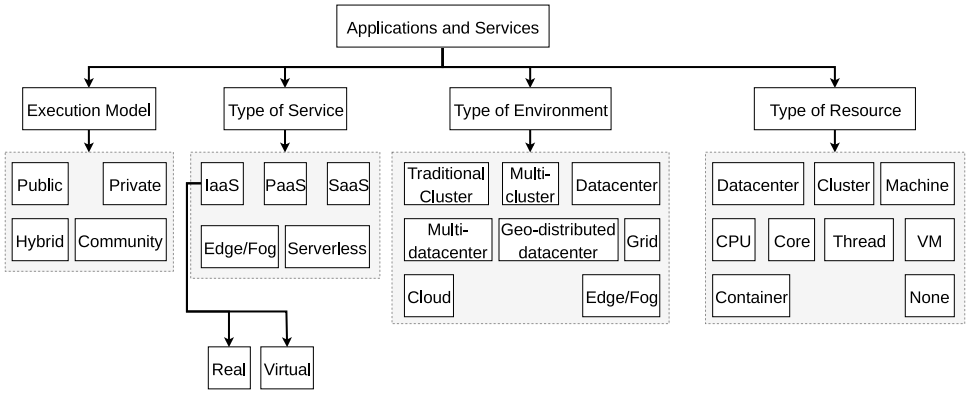


Fig. 14. Taxonomy of cloud computing services. Gray boxes depict a group of entities that fall under the same category.

7.2.3. SaaS

SaaS is a more restrictive form of both IaaS and PaaS that encapsulates a model where applications are offered as a service. Rather than having to install and set-up their own software, the hosting and installations are provided transparently by the cloud provider, eliminating any hosting intermediary. It is therefore that the cloud provider, hosting service, developer, and maintainer of the software are usually the same entity.

7.2.4. Edge/fog computing

Edge computing is an emerging paradigm where micro-datacenters and/or devices are put closer to the customer, often referred to as the edge of the network. This is also referred to as Fog computing [152]. By introducing such micro-datacenters, latency is reduced compared to sending data to the larger datacenters, further away. The general consensus is that such micro-datacenters are more expensive to use, as the datacenter operator must perform more management, often in various locations. In particular, IoT applications and mobile offloading strategies benefit from this new paradigm, enabling real-time processing and streaming of data.

For work done on Edge/Fog computing, the International Conference on Fog and Edge Computing (ICFEC) provides a good starting point. Surveys also provide starting points for open challenges and introduction to different concepts and applications, for example see [153] and [154]. Topics within Edge/Fog computing range from applications such as video streaming to resource consumption methods such energy-efficient scheduling, much alike traditional cloud topics.

7.2.5. Serverless

Serverless is an emerging paradigm where clients can choose not to (temporarily) own, or manage resources. In most cases, resource requirements still have to be specified, yet do not manually have to be provisioned and managed. This area recently gathered a lot of attention from the (cloud) community. The perceived benefits lie in the flexibility, cost-effectiveness, and availability properties. Several articles introduce both problems and opportunities for this new paradigm [155–157].

While emerging, the domain is growing fast with different areas of the domain being explored. Published articles range from historical [158] to frameworks. [159], and from exploration and characterization [160] to caching [105].

Examples of proposed applications using serverless are graph processing [161], chat bots [162], image processing [163,164], and data analytics [165].

7.3. Type of environments

Computing services can be offered on different type of environments. Traditionally, (local) *clusters* are used for additional computing. These are generally managed by a single department within a company/institution. *Multi-cluster* environments such as the Dutch DAS5 [166] offer resources in often geographically distributed clusters. These clusters can be managed by a single department or by the different institutions hosting them. *Datacenters* often comprise multiple clusters within a single location. These clusters can belong to a single or multiple entities, but generally, a datacenter consists of clusters belonging to multiple entities, either leased or bought. *Geo-distributed datacenters* are datacenters that are geographically distributed, often for fault-tolerance or legislation purposes. The different environments covered so far often have well-defined architectures and the hardware and infrastructure is known. *Grids*, *Clouds*, and *Edge/Fog* environments are more vague. Clouds are often composed of geographically distributed datacenters, where you can rent virtual machines in different physical locations using the now popular pay-as-you-go model. What makes the environments vague is that cloud providers rarely describe (in detail) the underlying hardware, schedulers, policies, and protocols in place. Grids are a mixture of hardware as they were often composed of a mixture of commodity and state-of-the-art hardware. This makes it difficult to assess the accessible hardware, nor were there any guarantees that a machine connected to the grid would not suddenly become unavailable. Finally, the Edge/Fog consists of many different devices “at the edge” of the network. These can be micro-datacenters, routers, mobile devices, smart devices, etc. Additionally, the communication established between these devices may be arbitrary.

7.4. Execution model

As Smanchat et al. [13] describe, when using computing services, the execution model can vary. *Public* resources are available to the public and can be leased from cloud providers. *Private* resources are only available to a single entity, e.g. company, it is not possible to execute work on these resources if you do not have (private) access. A *hybrid* model combines both public and private resources. Often when additional compute power is required, an entity can run (part of) the workload on a public cloud. *Community* resources such as the earlier mentioned DAS5 [166] enable a community to share maintain resources collectively.

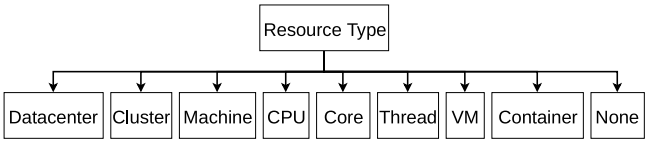


Fig. 15. Taxonomy of resource types.

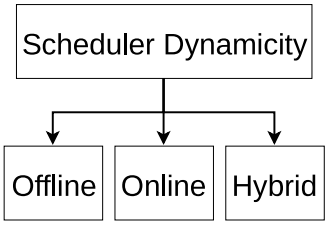


Fig. 16. Scheduling dynamicity taxonomy.

7.5. Taxonomy of resource types

When scheduling, different algorithms may consider different resources as the working unit. Fig. 15 depicts the taxonomy of resource types, which significantly extends the taxonomy presented in [13]. With different granularity possible and the heterogeneity of today's systems, plenty of work differentiates in resource considered. Typically, literature focuses on cores [167], VMs [140], and CPU [168]. Especially one task per CPU or VM is common [146,169]. Ma et al. [101] consider as resource types threads while scheduling tasks of an industrial IoT environment. Containers started receiving attention as possible alternatives to VMs. As a container does not include an operating system, the overhead can be reduced when using containers. With the popularity of Docker and Kubernetes, articles are investigating the use cases of containers [170,171]. Schedulers considering machines are used in e.g. parallel workflow computing [172]. Cluster schedulers were quite ubiquitous in Grid environments [42], and can be found in cloud environments as well [173,174]. The section "none" covers situations such as serverless (see Section 7.2.5) where cloud users need not consider the resources used. Naturally, resources are still consumed, yet these are entirely managed by the cloud operator and hidden from clients. Scheduling at the entire scale of a datacenter is common when dealing with super computers. Several applications that run on (a part) of a super computer can for example be found in the super computing community. For example, the algorithm for particle simulation that was deployed across a large part of the CORAL supercomputers [175].

7.6. Taxonomy of scheduling dynamicity

Scheduling dynamicity describes how allocation policies plan their allocations. Fig. 16 presents the taxonomy of scheduling dynamicity. Offline policies construct a total plan for all tasks yet to be scheduled in the system. The system must then adhere strictly to this plan. Any issues or runtime variations (e.g. stragglers) are not taking into account at runtime, while some plans do include some *slack* in their schedules. Online policies construct plans responsively. Incoming tasks are appended to the plan, or the plan is (partially) reconstructed to optimize for the optimization goal. Hybrid solutions use a combination of offline and online dynamicity. Hybrid policies often create an initial execution plan that is then changed proactively.

Query 6:
```
SELECT * FROM publications WHERE (lower(title) LIKE '%workflow%' OR lower(abstract) LIKE '%workflow%') AND ((lower(title) LIKE '%allocat%' OR lower(abstract) LIKE '%allocat%') OR (lower(title) LIKE '%schedul%' OR lower(abstract) LIKE '%schedul%') OR (lower(title) LIKE '%plan%' OR lower(abstract) LIKE '%plan%')) ORDER BY n_citations DESC
```

8. Mapping allocation and provisioning policies

The process of scheduling workflows in computing environments consists of many distinct steps [6]. In general, it boils down to two main elements; the workflows and theirs tasks need to be placed on resources (allocation), where these resources should be acquired accordingly (provisioning). Managing these two parts can be done separately using agnostic policies, or can be done in synergy where the two policies work together.

To demonstrate the different policies for both allocation and provisioning can be mapped to our taxonomies, in this section, we map a number of recent state-of-the-art and well-known policies to our taxonomies. As we cannot possibly map all existing policies to our taxonomies, we believe mapping well-known and state-of-the-art policies provide adequate empirical examples of the applicability and coverage of our taxonomies. We first describe our method used to obtain these lists of policies followed by an enumeration of their main properties.

8.1. Method

To obtain our list of policies, we use a systematic approach. We first create a query to filter for allocation and provisioning policies, respectively. Using this query, we extract sixteen policies per category: ten described in the most recent articles according to our database (using citation count as tie-breaker) and the six most popular by citation count. Articles that are a false positive, i.e., do not describe a policy are skipped.

As we have run these queries on an older version of our database for an earlier version of this article, we have amended more recent published articles leading to mapping twenty-six policies in Section 8.2 and eighteen policies in Section 8.3.

For allocation policies, we note the optimization goal, strategy, target, and technique used as well as if the policy computes the scheduling plan offline ahead of time, or dynamically at runtime. For provisioning policies, we note the type of decision-making, dynamic provisioning method, and provisioning model used. Additionally, for each article we also list the number of citations to provide a rough indication of popularity.

8.2. Allocation policies

In this section we provide an overview of well-cited and state-of-the-art workflow allocation policies. Table 3 presents a list of the six most cited and twenty recent allocation policies, sorted by citations. We focus on the scheduling technique used, the optimization goal and scheduling strategy. To obtain articles based on *citation count* we use **Query 6**. To obtain the *most recent* policies, we use **Query 7**.

8.3. Provisioning policies

In this section we provide an overview of well-cited and state-of-the-art resource provisioning policies. Table 4 presents a list of the six most cited and twelve recent provisioning policies, sorted by citations. Ten originate from the first version of this article, since then two articles matching **Query 9** were added to the

Table 3
Overview of allocation policies.

Name	#Citations	Year	ST	OG	OS
Titan [176]	522	2003	H	Makespan, Deadline, Resource Utilization	Multi/Min/Global
SCS [177]	433	2011	G	Cost	Multi/Min/Global
Min–min task [2]	383	2005	H	Runtime	Single/Min/Local
Blythe et al. [2]	383	2005	G+R	Runtime	Single/Min/Local
Weighted min–min [2]	383	2005	H	Runtime	Single/Min/Local
IC-PCP [178]	360	2012	H	Deadline, Cost	Multi/Min/Global
NN-DNSGA-II [179]	1	2020	MH+ML	Makespan, Cost, Energy consumption, Load balance, Reliability, Resource Utilization	Multi/Min+Max/Global
Ebadifard et al. [180]	0	2020	MH	Makespan, Cost, Resource Utilization	Multi/Min+Max/Global
COMSE [181]	0	2020	H	Deadline, Cost	Multi/Min/Global
GRP-HEFT [182]	0	2020	G	Makespan, Cost	Single/Min+Satisfice/Local
MOWO [183]	0	2020	MH	Response time, Fault tolerance, Cost	Multi/Min+Max/Global
Genez et al. [184]	0	2020	(I)LP	Deadline, Cost	Single/Satisfice+Min/Local
NearDeadline [185]	0	2020	H	Energy consumption, Fairness	Multi/Optimize/Global
Barika et al. [186]	NA	2020	MH	Cost	Single/Min/Local
SERAS [187]	NA	2020	G	Energy consumption, Fault Tolerance, Deadline	Multi/Min+Satisfice/Local
CLOSURE [188]	NA	2020	GT+H	Security	Single/Min/Global
Qureshi [189]	5	2019	H	Cost, Power, Resource Utilization	Multi/Min+Max/Local
FDHEFT [190]	5	2019	H	Cost, Makespan	Multi/Min/Global
WSADF [99]	3	2019	H	Bandwidth, Resource Utilization, Throughput	Multi/Min+Max/Global
Stavrinides et al. [98]	3	2019	H	Power Consumption, Cost	Multi/Min/Local
BDAS [191]	2	2019	H	Cost, Deadline	Multi/Min/Local
MOGA [192]	2	2019	MH	Makespan, Cost, Deadline, Power Consumption	Multi/Min/Global
PSFS [193]	1	2019	H	Deadline, Cost	Multi/Opt/Global
CSFS-H [193]	1	2019	H	Cost	Single/Min/Global
DNCSPO [194]	1	2019	MH	Makespan, Cost	Multi/Opt/Global
Wu et al. [195]	1	2019	MH	Makespan, Cost	Multi/Opt/Global

Header: ST = scheduling technique, OG = optimization goal, OS = optimization strategy. ST column: G = Greedy, H = Heuristic, MH = Meta-heuristic, R=Random, ML = Machine Learning, (I)LP = (Integer) Linear Programming, GT = Game Theory.

Query 7:
```
SELECT * FROM publications WHERE (lower(title) LIKE '%workflow%' OR lower(abstract) LIKE '%workflow%') AND ((lower(title) LIKE '%allocat%' OR lower(abstract) LIKE '%allocat%') OR (lower(title) LIKE '%schedul%' OR lower(abstract) LIKE '%schedul%') OR (lower(title) LIKE '%plan%' OR lower(abstract) LIKE '%plan%')) ORDER BY year DESC, n_citations DESC, id ASC
```

Query 8:
```
SELECT * FROM publications WHERE (lower(title) LIKE '%workflow%' OR lower(abstract) LIKE '%workflow%') AND (lower(title) LIKE '%provision%' OR lower(abstract) LIKE '%provision%' OR lower(title) LIKE '%autoscal%' OR lower(abstract) LIKE '%autoscal%') ORDER BY n_citations DESC
```

Query 9:
```
SELECT * FROM publications WHERE (lower(title) LIKE '%workflow%' OR lower(abstract) LIKE '%workflow%') AND (lower(title) LIKE '%provision%' OR lower(abstract) LIKE '%provision%' OR lower(title) LIKE '%autoscal%' OR lower(abstract) LIKE '%autoscal%') ORDER BY year DESC, n_citations DESC, id ASC
```

database. We focus on the information source used, the timeliness of information on which decisions are based, and the level at which the autoscaler operates.

To obtain articles based on *citation count* we use **Query 8**. To obtain the *most recent* policies, we use **Query 9**.

Table 4
Overview of provisioning policies.

Name	NC	Year	IS	ToI	L
DPDS [149]	236	2015	R	Short-term	Single-tier
WA-DPDS [149]	236	2015	R+J	Short-term	Single-tier
SPSS [149]	236	2015	J	Current	Single-tier
Dörnemann et al. [196]	173	2009	R	Current	Single-tier
PBTS [197]	153	2011	J	Current	Single-tier
Singh et al. [198]	107	2007	J	Current	Single-tier
GRP-HEFT [182]	0	2020	J	Current	Single-tier
Nguyen et al. [199]	0	2020	J	Current	Single-tier
CLUES [200]	0	2019	R	Current	Single-tier
Prob [201]	0	2019	J	Current	Single-tier
EPSM [202]	15	2018	J	Current	Single-tier
React [63]	10	2018	R	Short-term	Single-tier
Adapt [63]	10	2018	R	Short-term	Single-tier
Hist [63]	10	2018	R	Long-term	Multi-tier
Reg [63]	10	2018	R	Long-term	Single-tier
ConPaaS [63]	10	2018	R	Long-term	Single-tier
PPDPS [203]	8	2018	J	Current	Single-tier
BioCloud [204]	6	2018	J	Current	Single-tier

Header: NC = Number of Citations, IS = Information Source, ToI = Timeliness of Information, L = Level. IS column: R = Resource, J = Job.

9. Related work

There are several surveys that either overlap this work or that present formalisms which we extend. As we mention and cite articles of which we use elements or extend directly in the sections themselves, we discuss our contributions in contrast to several categories of related work. For a differentiation per article, we refer to our technical report [17].

Surveys on workflow allocation. Several surveys cover parts of the taxonomies we present [13,48,89,205–209].

Surveys on workflow provisioning. Resource provisioning has been surveyed before [12,13,63,206,208]. Our work primarily adds the recent additions on elasticity, provisioning models, and the level at which autoscalers operate. The focus of related work that we do not cover includes anomaly detection, multi-tenancy, and static vs. dynamic provisioning.

Workflow formalisms Several articles discuss workflow formalisms and their model [25,40]. Our survey focuses on the notion of DAG vs. non-DAG and the core language, which is not or less extensively addressed by related work.

Additionally, our survey contains some novel elements none of the mentioned related work performs. The first element is the survey of the community as we present in this article. Collaborative relationships have been investigated before, but not at the granularity and on the topics that we cover in this survey. We feel there is room for additional work in this direction, but leave it as future work due to limit in scope. A second element is investigation of important keywords per topic. Related work does mention emerging trends and recommend future directions, yet rarely is this based on techniques from the information retrieval domain. Extracting keywords from sparse text such as paper meta-data is a challenging problem due to limited length of the text. Using different techniques such as LDA or LSI to obtain more insights is left for future work.

Finally, most of our taxonomies are either new or significantly extend those of related work. We have validated taxonomies by mapping well-cited and state-of-the-art approaches on them, which lacks in most related work as well.

10. Threats to validity

We perceive four main threats to the validity of this work, which we address in this section.

The first main threat is the lack of depth by, e.g., comparisons between policies, workloads, parameters, a combination of former, etc. As outlined in Section 1, a comparison even for a subset of policies using an identical workload and computing environment is already difficult for several reasons. First, even if all parameters are known, not all sources are available. From experience, these source code used to experiment with, or other related configuration files are important as even the smallest guess work can lead to significant differences in performance [6].

Second, by design our survey is meant to be broad instead of in-depth. Due to the areas we are covering, going in-depth is out of scope. Third, we already compare both well-cited and state-of-the-art policies in Section 8 on a high-level where we outline several properties per policy.

Third, in several instances, the outcome of TF-IDF produced noise, i.e., false positives. This is mainly due to titles and abstracts being sparse text. Extracting meaningful keywords automatically form sparse text is difficult and ongoing work in both academia and industry. We believe we got meaningful results, yet approaches using Latent Dirichlet Allocation or text clustering approaches might yield interesting insights too. We leave this for future work. Perhaps obtaining the full article text may lead to improved results. However, extracting text from PDFs is a tough problem.[4]

Finally, the fourth threat is the combination of traditional search methods for relevant articles and the use of queries. On the one hand, using solely traditional search methods makes reproducibility of results very hard if not impossible due to search bubbles when, e.g., using Google scholar. On the other hand, solely relying on queries on article meta-data databases such as introduced in this article may lead to missing important results if the query is not covering the field adequately or will yield a large set of false positives if defined too broad. The coverage of queries also requires knowledge of which keywords to search on, something that requires the use of traditional search methods to become familiar with all synonymous terms used in the community. We therefore decided to combine these two techniques when searching for related work to complement the results. Next to outlining all queries used, situations that did allow for full reproducibility of results by only using queries were included where possible, such as the mapping of articles describing allocation policies in Section 8.2 and provision policies in Section 8.3. Overall, we believe the use of our article meta-data database already exceeds the effort of related surveys that solely rely on traditional search methods.

[4] See for example https://www.filingdb.com/pdf-text-extraction.

11. Conclusion

Clouds and other infrastructures have been widely adopted to run workloads on. In particular, workflows are a popular workload model nowadays as they support the workload of many domains. The mixture of QoS requirements, different scheduling targets from both the user and resource provider makes workflow scheduling a complex problem. Getting insight into research done in this topic can be a daunting task, with the high-volume of publications in this research area, which is likely to intensify in the future.

Surveys are an excellent way to learn about the current status of a field, emerging trends, and open challenges. Unfortunately, surveys rarely publish the data on which their survey is based. Moreover, surveys rarely focus on the community itself; the structure, authors relationships, and citation information can provide insight into their health and operations, which can be interesting to community leaders and organizers.

In this work, we address these issues by performing various types of analyses.

We start by introducing and making open-source our instrument used to gather, filter, and unify article meta-data. Using this meta-data, we obtain insights into the workflow scheduling community and four related areas. For each, we analyze in-depth the community, look at important keywords both overall and per year, and identify emerging keywords. Additionally, using this meta-data we were able to perform a systematic literature survey to construct and validate our taxonomies.

We observe that for all areas, 70+% of the authors only author a single article in the timespan 2011–2020. Furthermore, our meta-data suggests that larger cliques tend to have members with a lower average and maximum citation count when compared to small and moderately sized cliques. This may indicate that the likelihood junior researchers are part of such cliques is higher, and that well-cited authors do not engage with all members of large communities. However, more research in this direction is required to draw more definitive conclusions. Our provided open-source instrument provides an excellent start for such future work.

Finally, using our instrument we map the most recent and top-cited allocation and provisioning policies to our taxonomies to demonstrate their completeness.

All software and artifacts that we introduce to obtain paper meta-data and visualizations are made open-source. We believe these tools are valuable to the community for finding related work (we already experienced this multiple times first-hand), reproduce or perform a survey similar to this study, or redo this study in the future to observe new trends.

Besides the directions for future work that were provided in each section, we believe more directions can be investigated

and surveyed. In particular, our taxonomies can be extended and integrated in other taxonomies; several surveys that we marked as related work expand in different directions with respect to our work. Deeper analysis into the communities using more data, different angles, and statistical methods may provide additional insights. As for promising research directions, Edge, IoT, and serverless are emerging fields with many potential directions. Another directions that is worth pursuing across all areas is energy efficiency. With the increasing power consumption due to the growth of datacenters, even a small reduction percentage-wise will have a major impact on the absolute power consumption. Research on efficiently reducing energy consumption while adhering to all QoSs is gaining traction and will grow in importance.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Availability of data and software artifacts

All data and instruments used in this work are available as open-access, FAIR data. The database on which this article is based can be found at https://atlarge-research.com/data/2020_fgcs_aip.pgsql, AIP and other tools used to generate all floats in this article can be found at https://github.com/atlarge-research/AIP.

Acknowledgments

We thank Erwin van Eyk, Sacheendra Talluri, Oana Inel, Benno Kruit, and Alexandru Uta for their helpful comments. This work is sponsored by COMMIT, The Netherlands and Vidi MagnaData, The Netherlands.

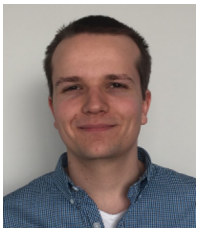
References

[1] Ron, Why cloud computing matters to finance, Strateg. Financ. 92 (2011) (2011).
[2] Blythe, et al., Task scheduling strategies for workflow-based applications in grids, in: CCGrid, 2005.
[3] Stein, The case for cloud computing in genome informatics, Genome biology 11 (2010) (2010).
[4] Edward G. Coffman Jr., Ronald L. Graham, Optimal scheduling for two-processor systems, Acta Inf. 1 (1972) (1972).
[5] Versluis, et al., The Workflow Trace Archive: Open-Access Data from Public and Private Computing Infrastructures – Technical Report, 2019, arXiv 2019.
[6] Andreadis, et al., A reference architecture for datacenter scheduling: design, validation, and experiments, in: SC, 2018.
[7] Shi, Edge computing: Vision and challenges, IOTJ 3 (2016) (2016).
[8] Eyk, et al., Serverless is more: From PaaS to present cloud computing, IC 22 (2018) (2018a).
[9] Iosup, Versluis, et al., Massivizing computer systems: A vision to understand, design, and engineer computer ecosystems through and beyond modern distributed systems, in: ICDCS, 2018, pp. 1224–1237.
[10] Kwok, Ahmad, Static scheduling algorithms for allocating directed task graphs to multiprocessors, ACM Comput. Surv. 31 (1999) (1999).
[11] Yu, Buyya, A taxonomy of scientific workflow systems for grid computing, SIGMOD 34 (2005) (2005).
[12] Kardani-Moghaddam, et al., Performance-aware management of cloud resources: A taxonomy and future directions, CSUR 52 (2019) (2019).
[13] Smanchat, Viriyapant, Taxonomies of workflow scheduling problem and techniques in the cloud, FGCS 52 (2015) (2015).
[14] The dblp team: dblp computer science bibliography, Monthly snapshot release of 2019, 2019, https://dblp.org/xml/release/dblp-2019-11-01.xml.gz.
[15] Ammar, et al., Construction of the literature graph in semantic scholar, in: NAACL, 2018.
[16] Tang, et al., Arnetminer: extraction and mining of academic social networks, in: SIGKDD, 2008.
[17] Versluis, Iosup, A Survey and Annotated Bibliography of Workflow Scheduling in Computing Infrastructures: Community, Keyword, and Article Reviews – Extended Technical Report, 2020, arXiv e-prints 2020.
[18] R. Duncan Luce, Albert D. Perry, A method of matrix analysis of group structure, in: Psychometrika, (1949) 1949.
[19] Liu, et al., Co-authorship networks in the digital library research community, IPM 41 (2005) (2005).
[20] Onan, et al., Ensemble of keyword extraction methods and classifiers in text classification, J. ESWA 57 (2016) (2016).
[21] Zhang, et al., TFIDF, LSI and multi-word in information retrieval and text categorization, in: ICSMC, IEEE, 2008.
[22] Gupta, et al., A survey of text mining techniques and applications, JETWI 1 (2009) (2009).
[23] Versluis, et al., An analysis of workflow formalisms for workflows with complex non-functional requirements, in: HotCloudPerf, 2018a.
[24] Li, et al., Trust-based and qos demand clustering analysis customizable cloud workflow scheduling strategies, In: Cluster workshops, 2012.
[25] Bastos, et al., Scientific workflow interchanging through patterns: Reversals and lessons learned, in: E-Science, 2015.
[26] Fahringer, et al., Specification of grid workflow applications with AGWL: an abstract grid workflow language, in: CCGrid, Vol. 2, 2005.
[27] Marozzo, et al., JS4Cloud: script-based workflow programming for scalable data analysis on cloud platforms, CCPE 27 (2015) (2015).
[28] Cesario, et al., Programming knowledge discovery workflows in service-oriented distributed systems, CCPE 25 (2013) (2013).
[29] Amstutz, et al., Common workflow language, Draft 3 (2016) (2016).
[30] Christoph Jansen, et al., Reproducibility and performance of deep learning applications for cancer detection in pathological images, in: CCGRID, 2019.
[31] Börger, Approaches to modeling business processes: a critical analysis of BPMN, workflow patterns and YAWL, SOSYM 11 (2012) (2012).
[32] Aalst, The application of Petri nets to workflow management, J. Circuits Syst. Comput. 8 (1998).
[33] Aalst, et al., Modelling and analysing workflow using a Petri-net based approach, in: CSCW Workshop, Petri nets and related formalisms, 1994..
[34] Hoheisel, Der, Dynamic workflows for grid applications, in: Cracow Grid Workshop, Vol. 3, 2003.
[35] Van, Ter, YAWL: yet another workflow language, Information systems 30 (2005) (2005).
[36] Wüst, et al., Generation of interactive questionnaires using YAWL-based workflow models, Manage. Stud. 3 (2015) (2015).
[37] Rodrigo Alves Lima, et al., Systematic construction, execution, and reproduction of complex performance benchmarks, in: Cloud, Vol. 11513, 2019.
[38] Hongwu Lv, et al., An attribute-based availability model for large scale IaaS clouds with CARMA, TPDS 31 (2020) (2020).
[39] Zheng. Song, Eli. Tilevich, Equivalence-enhanced microservice workflow orchestration to efficiently increase reliability, in: ICWS, 2019.
[40] Deelman, et al., Workflows and e-science: An overview of workflow system features and capabilities, FGCS 25 (2009) (2009).
[41] Zhan, Huo, Improved PSO-based task scheduling algorithm in cloud computing, JICS 9 (2012) (2012).
[42] Yu, et al., Workflow scheduling algorithms for grid computing, in: MSDCE, 2008.
[43] Gog, et al., Firmament: Fast, centralized cluster scheduling at scale, in: OSDI, 2016.
[44] Bonvin, et al., Autonomic SLA-driven provisioning for cloud applications, in: CCGrid, 2011.
[45] Abrishami, et al., Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds, FGCS 29 (2013) (2013a).
[46] Jeff Barr, New–Per-Second Billing for EC2 Instances and EBS Volumes | Amazon Web Services, 2018, https://aws.amazon.com/blogs/aws/new-per-second-billing-for-ec2-instances-and-ebs-volumes/.
[47] Becky. Peterson, Google takes a page from amazon web services and adds per-second billing for its cloud, 2017, http://uk.businessinsider.com/google-cloud-matches-amazon-web-services-with-per-second-billing-2017-9.
[48] Alkhanak, et al., Cost-aware challenges for workflow scheduling approaches in cloud computing environments: Taxonomy and opportunities, FGCS 50 (2015) (2015).
[49] [N. d.] here's how much energy all US data centers consume, 2019, https://www.datacenterknowledge.com/archives/2016/06/27/heres-how-much-energy-all-us-data-centers-consume, (Accessed: 2019-07-22).
[50] [N. d.] amazon plans wind farm to power its datacenters, 2019, https://phys.org/news/2015-01-amazon-farm-power-datacenters.html, (Accessed: 2019-07-22).
[51] [N. d.] renewable energy - data centers, 2019, https://www.google.com/about/datacenters/renewable/, (Accessed: 2019-07-22).

[52] [N. d.] with our latest energy deal, microsoft's cheyenne datacenter will now be powered entirely by wind energy, keeping us on course to build a greener, more responsible cloud, 2019, https://blogs.microsoft.com/on-the-issues/2016/11/14/latest-energy-deal-microsofts-cheyenne-datacenter-will-now-powered-entirely-wind-energy-keeping-us-course-build-greener-responsible-cloud, (Accessed: 2019-07-22).
[53] Li, et al., An energy-efficient scheduling approach based on private clouds, JICS 8 (2011) (2011).
[54] Durillo, et al., Multi-objective workflow scheduling: An analysis of the energy efficiency and makespan tradeoff, in: CCGrid, 2013.
[55] Durillo, et al., Multi-objective energy-efficient workflow scheduling using list-based heuristics, FGCS 36 (2014) (2014).
[56] Pietri, Sakellariou, Energy-aware workflow scheduling using frequency scaling, in: ICPPW, 2014.
[57] Berl, et al., Energy-efficient cloud computing, CJ 53 (2010) (2010).
[58] Benoit, et al., Performance and energy optimization of concurrent pipelined applications, in: IPDPS, 2010.
[59] Jones, Nitzberg, Scheduling for parallel supercomputing: A historical perspective of achievable utilization, in: JSSPP, 1999.
[60] Vasan, et al., Worth their watts? - an empirical study of datacenter servers, in: HPCA, 2010.
[61] Heinze, et al., Auto-scaling techniques for elastic data stream processing, in: DEBS, 2014.
[62] Dougherty, et al., Model-driven auto-scaling of green cloud computing infrastructure, FGCS 28 (2012) (2012).
[63] Ilyushkin, et al., An experimental performance evaluation of autoscalers for complex workflows, TOMPECS 3 (2018) (2018).
[64] Versluis, et al., A trace-based performance study of autoscaling workloads of workflows in datacenters, in: CCGrid, 2018b.
[65] Thomas Bauer, et al., Intra-subnet load balancing in distributed workflow management systems, IJCIS 12 (2003) (2003).
[66] Alexandru Uta, et al., Is big data performance reproducible in modern cloud networks?, in: NSDI, 2020.
[67] Bittencourt, Madeira, Towards the scheduling of multiple workflows on computational grids, JGC 8 (2010) (2010).
[68] Ghodsi, et al., Dominant resource fairness: Fair allocation of multiple resource types, in: Nsdi, Vol.11, 2011.
[69] Zhao, Sakellariou, Scheduling multiple DAGs onto heterogeneous systems, in: IPDPS, 2006.
[70] Wang, et al., Multi-resource fair sharing for datacenter jobs with placement constraints, in: SC, 2016b.
[71] Genez, et al., A flexible scheduler for workflow ensembles, in: UCC, 2016.
[72] Quang, et al., A comparative analysis of scheduling mechanisms for virtual screening workflow in a shared resource environment, in: CCGrid, 2015.
[73] Deng, et al., A scheme for scheduling hard real-time applications in open system environment, in: ECRTS, 1997.
[74] Wu, et al., A priority constrained scheduling strategy of multiple workflows for cloud computing, in: ICACT, 2012.
[75] John. Wilkes, More Google cluster data. Google research blog, 2011, Posted at http://googleresearch.blogspot.com/2011/11/more-google-cluster-data.html.
[76] Beek, et al., [n. d.] A CPU Contention Predictor for Business-Critical Workloads in Cloud Datacenters, ([n. d.]).
[77] Li, et al., A security and cost aware scheduling algorithm for heterogeneous tasks of scientific workflow in clouds, FGCS 65 (2016) (2016).
[78] Zhao, Result Verification and Trust-Based Scheduling in Open Peer-To-Peer Cycle Sharing Systems, 2004.
[79] Zhu, Jiang, A secure anti-collusion data sharing scheme for dynamic groups in the cloud, TPDS 27 (2016) (2016).
[80] Shishido, et al., A cloudsim extension for evaluating security overhead in workflow execution in clouds, in: CANDAR, 2018.
[81] Alnemr, et al., A data protection impact assessment methodology for cloud, in: Annual Privacy Forum, 2015.
[82] RF Federal Law, [N. d.] on amendments to certain legislative acts of the Russian Federation in connection with the improvement of the legal status of the State (municipal) institutions of may 08, 2010, 83-FZ.
[83] Arun K. Somani, Nitin H. Vaidya, [n. d.]Understanding Fault Tolerance and Reliability - Guest Editors' Indroduction, in: Computer 30, ([n. d.]).
[84] Poola, et al., Fault-tolerant workflow scheduling using spot instances on clouds, Procedia Comput. Sci. 29 (2014) (2014).
[85] Ramakrishnan, et al., Vgrads: enabling e-science workflows on grids and clouds with fault tolerance, in: SC, 2009.
[86] Jayadivya, et al., Fault tolerant workflow scheduling based on replication and resubmission of tasks in cloud computing, JCSE 4 (2012) (2012).
[87] Zhu, et al., Fault-tolerant scheduling for real-time scientific workflows with elastic resource provisioning in virtualized clouds, TPDS 27 (2016) (2016).
[88] Bala, Chana, Fault tolerance-challenges techniques and implementation in cloud computing, IJCSI 9 (2012) (2012).
[89] Poola, et al., A taxonomy and survey of fault-tolerant workflow management systems in cloud and distributed computing environments, in: SABDC, 2017.
[90] Xie, et al., Improving mapreduce performance through data placement in heterogeneous hadoop clusters, in: IPDPSW, 2010.
[91] Guo, et al., Investigation of data locality in mapreduce, in: CCGrid, 2012.
[92] Wang, et al., Maptask scheduling in mapreduce with data locality: Throughput and heavy-traffic optimality, TON 24 (2016) (2016a).
[93] Duro, et al., Exploiting Data Locality in Swift/T Workflows using Hercules, (2014) 2014.
[94] Georgios L. Stavrinides, Helen D. Karatza, Orchestration of real-time workflows with varying input data locality in a heterogeneous fog environment, in: FMEC, 2020.
[95] Choi, et al., Data-locality aware scientific workflow scheduling methods in HPC cloud environments, JPP 45 (2017) (2017).
[96] Tanaka, Tatebe, Workflow scheduling to minimize data movement using multi-constraint graph partitioning, in: CCGrid, 2012.
[97] Cardoso, et al., Workflow quality of service, in: ICEIMT, 2002.
[98] Stavrinides, Karatza, An energy-efficient, QoS-aware and cost-effective scheduling approach for real-time workflow applications in cloud computing systems utilizing DVFS and approximate computations, FGCS 96 (2019) (2019).
[99] Momenzadeh, Safi-Esfahani, Workflow scheduling applying adaptable and dynamic fragmentation (WSADF) based on runtime conditions in cloud computing, FGCS 90 (2019) (2019).
[100] Raafat Omar Aburukba, et al., Scheduling Internet of Things requests to minimize latency in hybrid Fog-Cloud computing, FGCS 111 (2020) (2020).
[101] Ma, et al., Ananke: A Q-learning-based portfolio scheduler for complex industrial workflows, in: ICAC, 2017.
[102] Simranjit. Kaur, Pallavi. Bagga, Rahul. Hans, Harjot. Kaur, Quality of Service (QoS) aware workflow scheduling (WFS) in cloud computing: A systematic review, AJSE 44 (2019) (2019).
[103] Bittencourt, Madeira, HCOC: a cost optimization algorithm for workflow scheduling in hybrid clouds, JISA 2 (2011) (2011).
[104] Goecks, et al., Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences, Genome Biol. 11 (2010) (2010).
[105] Abad, et al., Package-aware scheduling of faas functions, in: CHotCloud-Perf, 2018.
[106] Liu, et al., A throughput maximization strategy for scheduling transaction-intensive workflows on SwinDeW-G, CCPE 20 (2008) (2008).
[107] Simon, Rational choice and the structure of the environment, Psychol. Rev. 63 (1956) (1956).
[108] Jaeger, et al., A framework for automatic improvement of workflows to meet performance goals, in: ICTAI, 1994.
[109] Zhang, et al., Ordinal optimized scheduling of scientific workflows in elastic compute clouds, in: CloudCom, 2011.
[110] Yu, Buyya, Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms, SP 14 (2006) (2006).
[111] Goiri, et al., Greenslot: scheduling energy consumption in green datacenters, in: SC, 2011.
[112] Yassa, et al., Multi-objective approach for energy-aware workflow scheduling in cloud computing environments, SWJ 2013 (2013) (2013).
[113] Schwarzkopf, et al., Omega: Flexible, Scalable Schedulers for Large Compute Clusters, 2013, (2013).
[114] Andrade, et al., Ourgrid: An approach to easily assemble grids with equitable resource sharing, in: JSSPP, 2003.
[115] Deng, Liu, Scheduling real-time applications in an open environment, in: RTSS, 1997.
[116] Iosup, A framework for the study of grid inter-operation mechanisms, (2009) 2009.
[117] Dong H. Ahn, et al., Flux: Overcoming scheduling challenges for exascale workflows, FGCS 110 (2020) (2020).
[118] Amvrosiadis, et al., On the diversity of cluster workloads and its impact on research results, in: ATC, 2018.
[119] Iosup, et al., Inter-operating grids through delegated matchmaking, SP 16 (2008) (2008).
[120] Xiang, et al., Greedy-ant: ant colony system-inspired workflow scheduling for heterogeneous computing, IEEE Access 5 (2017) (2017).
[121] Yu, et al., Deadline/budget-based scheduling of workflows on utility grids, MOGUC 200 (2009) (2009).
[122] McGough, et al., Workflow enactment in ICENI, in: UK E-Science All Hands Meeting, Vol. 9, 2004.
[123] Yaghoobi, et al., A non-cooperative game theory approach to optimize workflow scheduling in grid computing, in: PACRIM, 2013.
[124] Duan, et al., Multi-objective game theoretic schedulingof bag-of-tasks workflows on hybrid clouds, TCC 2 (2014) (2014).
[125] Waldspurger, Weihl, Lottery scheduling: Flexible proportional-share resource management, in: OSDI, 1994.

[126] Zhao, Sakellariou, An experimental investigation into the rank function of the heterogeneous earliest finish time scheduling algorithm, in: Euro-Par, 2003.
[127] Chen, Zhang, An ant colony optimization approach to a grid workflow scheduling problem with various QoS requirements, TSMC 39 (2009) (2009).
[128] Bilgaiyan, et al., Workflow scheduling in cloud computing environment using cat swarm optimization, in: IACC, 2014.
[129] Kaur, Mehta, Resource provisioning and work flow scheduling in clouds using augmented Shuffled Frog Leaping Algorithm, JPDC 101 (2017) (2017).
[130] Song, et al., Security-driven Heuristics and a fast genetic algorithm for trusted grid job scheduling, in: IPDPS, 2005.
[131] Young, et al., Scheduling architecture and algorithms within the ICENI grid middleware, in: UK E-Science All Hands Meeting, 2003.
[132] Wu, A revised discrete particle swarm optimization for cloud workflow scheduling, in: CIS, 2010.
[133] Vukmirović, et al., Optimal workflow scheduling in critical infrastructure systems with neural networks, JART 10 (2012) (2012).
[134] Bauer, et al., Chameleon: A hybrid, proactive auto-scaling mechanism on a level-playing field, in: TPDS, (2018) 2018.
[135] Malik, et al., Co-locating and concurrent fine-tuning MapReduce applications on microservers for energy efficiency, in: IISWC, 2017.
[136] Afzal, et al., Qos-constrained stochastic workflow scheduling in enterprise and scientific grids, in: GRID, 2006.
[137] Genez, et al., Using time discretization to schedule scientific workflows in multiple cloud providers, in: CLOUD, 2013.
[138] Benoit, et al., A survey of pipelined workflow scheduling: Models and algorithms, CSUR 45 (2013) (2013).
[139] El-Zarif, Awad, An ordinal optimization like GA for improved OFDMA system carrier allocations, in: ICIS, 2012.
[140] Wang, et al., Multi-objective workflow scheduling with Deep-Q-network-based multi-agent reinforcement learning, IEEE Access 7 (2019) (2019).
[141] Yu, Shi, A Planner-Guided Scheduling Strategy for Multiple Workflow Applications, In: ICPP Workshops, 2008.
[142] Guo, et al., Modeling, analysis, and experimental comparison of streaming graph-partitioning policies, JPDC 108 (2017) (2017).
[143] Deelman, et al., Pegasus and DAGMan from concept to execution: Mapping scientific workflows onto today's cyberinfrastructure, in: HPC, 2006.
[144] Whitney, Delforge, Data center efficiency assessment, in: NRDC, (2014) 2014.
[145] Shoaib, Das, Performance-oriented Cloud Provisioning: Taxonomy and survey, 2014, CoRR abs/1411.5077 2014.
[146] Ilyushkin, et al., Scheduling workloads of workflows with unknown task runtimes, in: CCGrid, 2015.
[147] Bauer, et al., Chamulteon: Coordinated auto-scaling of micro-services, in: ICDCS, 2019.
[148] Khorsand, et al., FAHP Approach for autonomic resource provisioning of multitier applications in cloud computing environments, Softw. - Pract. Exp. 48 (2018) (2018).
[149] Malawski, et al., Algorithms for cost- and deadline-constrained provisioning for scientific workflow ensembles in IaaS clouds, FGCS 48 (2015) (2015).
[150] Aumala, et al., Beyond load balancing: Package-aware scheduling for serverless platforms, in: CCGrid, 2019.
[151] [N. d.] google angles for business users with 'platform as a service', 2018, https://www.theguardian.com/technology/2008/apr/17/google.software, (Accessed: 2018-06-14).
[152] Mahmud, et al., Quality of Experience (QoE)-aware placement of applications in Fog computing environments, JPDC 132 (2019) (2019).
[153] Yi, et al., A survey of fog computing: concepts, applications and issues, in: Mobidata, 2015.
[154] Mahmud, et al., Fog computing: A taxonomy, survey and future directions, in: Internet of Everything, 2018.
[155] Baldini, et al., Serverless computing: Current trends and open problems, in: Research Advances in Cloud Computing, 2017.
[156] Fox, et al., Status of serverless computing and function-as-a-service (faas) in industry and research, 2017, arXiv 2017.
[157] Eyk, et al., A SPEC RG cloud group's vision on the performance challenges of FaaS Cloud Architectures, in: HotCloudPerf, 2018c.
[158] Eyk, et al., Serverless is more: From paas to present cloud computing, IC 22 (2018) (2018b).
[159] Pérez, et al., [n. d.] On-premises Serverless Computing for Event-Driven Data Processing Applications, ([n. d.]).
[160] Wang, et al., Peeking behind the curtains of serverless platforms, in: ATC, 2018.
[161] Toader, et al., [n. d.]Graphless: Toward Serverless Graph Processing. ([n. d.]).
[162] Yan, et al., Building a chatbot with serverless computing, in: MOTA, 2016.
[163] Akkus, et al., SAND: Towards high-performance serverless computing, in: ATC, 2018.
[164] Oakes, et al., SOCK: Rapid task provisioning with serverless-optimized containers, in: ATC, 2018.
[165] Nastic, et al., A serverless real-time data analytics platform for edge computing, IC 21 (2017) (2017).
[166] Bal, et al., A medium-scale distributed system for computer science research: Infrastructure for the long term, IEEE Comput. 49 (2016) (2016).
[167] Lee, et al., Resource-efficient workflow scheduling in clouds, Knowl.-Based Syst. 80 (2015) (2015).
[168] Topcuoglu, et al., Performance-effective and low-complexity task scheduling for Heterogeneous computing, TPDS 13 (2002) (2002).
[169] Seo, Energy efficient scheduling of real-time tasks on multicore processors, TPDS 19 (2008) (2008).
[170] Gerlach, et al., Skyport: container-based execution environment management for multi-cloud scientific workflows, in: DataCloud, 2014.
[171] Liu, et al., Flexible container-based computing platform on cloud for scientific workflows, in: ICCCRI, 2016.
[172] Rajakumar, et al., Workflow balancing strategies in parallel machine scheduling, JAMT 23 (2004) (2004).
[173] Verma, et al., Large-scale cluster management at Google with Borg, in: EuroSys, 2015.
[174] Pollard, et al., Evaluation of an interference-free node allocation policy on fat-tree clusters, in: SC, 2018.
[175] Berkowitz, et al., Simulating the weak death of the neutron in a femtoscale universe with near-exascale computing, in: SC, 2018.
[176] Cao, et al., Gridflow: Workflow management for grid computing, in: CCGrid, 2003.
[177] Mao, Humphrey, Auto-scaling to minimize cost and meet application deadlines in cloud workflows, in: SC, 2011.
[178] Abrishami, et al., Deadline-constrained workflow scheduling algorithms for infrastructure as a Service Clouds, FGCS 29 (2013) (2013b).
[179] Goshgar Ismayilov, Haluk Rahmi Topcuoglu, Neural network based multi-objective evolutionary algorithm for dynamic workflow scheduling in cloud computing, FGCS 102 (2020) (2020).
[180] Fatemeh Ebadifard, Seyed Morteza Babamir, Scheduling scientific workflows on virtual machines using a Pareto and hypervolume based black hole optimization algorithm, TJS 76 (2020) (2020).
[181] Hao Wu, et al., Scheduling large-scale scientific workflow on virtual machines with different numbers of vCPUs, in: TJS, (2020) 2020.
[182] Hamid Reza Faragardi, et al., GRP-HEFT: a budget-constrained resource provisioning scheme for workflow scheduling in iaas clouds, TPDS 31 (2020) (2020).
[183] Vincenzo De Maio, Dragi Kimovski, Multi-objective scheduling of extreme data scientific workflows in Fog, FGCS 106 (2020) (2020).
[184] A.L. Thiago, Genez, et al., [n. d.]Time-discretization for speeding-up scheduling of deadline-constrained workflows in clouds, FGCS 107, ([n. d.]).
[185] Emile Cadorel, et al., Online multi-user workflow scheduling algorithm for fairness and energy optimization, in: CCGRID, 2020.
[186] Mutaz Barika, et al., Cost effective stream workflow scheduling to handle application structural changes, FGCS 112 (2020) (2020).
[187] Hadeer A. Hassan, A smart energy and reliability aware scheduling algorithm for workflow execution in DVFS-enabled cloud environment, FGCS 112 (2020) (2020).
[188] Yawen Wang, et al., CLOSURE: A cloud scientific workflow scheduling algorithm based on attack-defense game model, FGCS 111 (2020) (2020).
[189] Qureshi, Profile-based power-aware workflow scheduling framework for energy-efficient data centers, FGCS 94 (2019) (2019).
[190] Zhou, et al., Minimizing cost and makespan for workflow scheduling in cloud using fuzzy dominance sort based HEFT, FGCS 93 (2019) (2019b).
[191] Arabnejad, et al., Budget and deadline aware e-science workflow scheduling in clouds, TPDS 30 (2019) (2019).
[192] Rehman, et al., Multi-objective approach of energy efficient workflow scheduling in cloud environments, CCPE 31 (2019) (2019).
[193] Pietri, Sakellariou, A Pareto-based approach for CPU provisioning of scientific workflows on clouds, FGCS 94 (2019) (2019).
[194] Xie, et al., A novel directional and non-local-convergent particle swarm optimization based workflow scheduling in cloud-edge environment, FGCS 97 (2019) (2019).
[195] Wu, et al., An integrated algorithm for multi-agent fault-tolerant scheduling based on MOEA, FGCS 94 (2019) (2019).
[196] Dörnemann, et al., On-demand resource provisioning for BPEL workflows using Amazon's elastic compute cloud, in: CCGrid, 2009.
[197] Byun, et al., Cost optimized provisioning of elastic resources for application workflows, FGCS 27 (2011) (2011).
[198] Singh, et al., A provisioning model and its comparison with best-effort for performance-cost optimization in grids, in: HPDC, 2007.
[199] Minh Nguyen, et al., A black-box Fork-join latency prediction model for data-intensive applications, TPDS 31 (2020) (2020).

[200] Pérez, et al., On-premises serverless computing for event-driven data processing applications, in: Cloud, 2019.
[201] Zhou, et al., Incorporating probabilistic optimizations for resource provisioning of data processing workflows, in: ICPP, 2019a.
[202] Rodriguez, Buyya, Scheduling dynamic workloads in multi-tenant scientific workflow as a service platforms, FGCS 79 (2018) (2018).
[203] Singh, et al., A novel cost-efficient approach for deadline-constrained workflow scheduling by dynamic provisioning of resources, FGCS 79 (2018) (2018).
[204] Senturk, et al., A resource provisioning framework for bioinformatics applications in multi-cloud environments, FGCS 78 (2018) (2018).
[205] Hilman, et al., Multiple workflows scheduling in multi-tenant distributed systems: A taxonomy and future directions, 2018, CoRR abs/1809.05574 2018.
[206] Rodriguez, Buyya, A taxonomy and survey on scheduling algorithms for scientific workflows in IaaS cloud computing environments, in: CCPE, (2016) 2016.
[207] Wieczorek, et al., Towards a general model of the multi-criteria workflow scheduling on the grid, FGCS 25 (2009) (2009).
[208] Wu, et al., Workflow scheduling in cloud: a survey, TJS 71 (2015) (2015).
[209] Mainak Adhikari, et al., A survey on scheduling strategies for workflows in cloud environment and emerging trends, ACM Comput. Surv. 52 (2019) (2019).



Laurens Versluis received his B.Sc. and M.Sc degrees in computer science from Delft University of technology, The Netherlands.

Currently, he is a Ph.D. student with the Massiving Computer Systems Group of the Department of Computer Science, Faculty of Sciences, VU Amsterdam, The Netherlands.

His research interests include cloud computing, distributed systems, scheduling, complex workflows, and data science. Contact him at l.f.d.versluis@vu.nl

Alexandru Iosup is tenured full Professor and University Research Chair with the Vrije Universiteit Amsterdam, the Netherlands. He is also Chair of the SPEC Research Cloud Group. He received a Ph.D. in computer science from TU Delft, the Netherlands (2009). He was awarded the yearly Netherlands Prize for Research in Computer Science (2016), the yearly Netherlands Teacher of the Year (2015), and several SPECtacular awards (2012–2017). His research interests are in massivizing computer systems, that is, making computer systems combine desirable properties such as elasticity, performance, and availability, yet maintain their ability to operate efficiently in controlled ecosystems. Topics include cloud computing and big data, with applications in big science, big business, online gaming, and (upcoming) massivized education.