# A Systematic Review of Scheduling Approaches on Multi-tenancy Cloud Platforms

Jia Ru[a,*], Yun Yang[a], John Grundy[b], Jacky Keung[c], Li Hao[d]

[a]*School of Software and Electrical Engineering, Swinburne University of Technology, Melbourne, Australia 3122*
[b]*Faculty of Information Technology, Monash University, Melbourne, Australia 3145*
[c]*Department of Computer Science, City University of Hong Kong, Hong Kong SAR*
[d]*Department of Business Intelligence, Allianz Australia Insurance Limited, Sydney, Australia 2000*

**Abstract**

*Context:* Scheduling in cloud is complicated as a result of multi-tenancy. Diverse tenants have different requirements, including service functions, response time, QoS and throughput. Diverse tenants require different scheduling capabilities, resource consumption and competition. Multi-tenancy scheduling approaches have been developed for different service models, such as Software as a Service (SaaS), Platform as a service (PaaS), Infrastructure as a Service (IaaS), and Database as a Service (DBaaS).
*Objective:* In this paper, we survey the current landscape of multi-tenancy scheduling, laying out the challenges and complexity of software engineering where multi-tenancy issues are involved. This study emphasises scheduling policies, cloud provisioning and deployment with regards to multi-tenancy issues. We conduct a systematic literature review of research studies related to multi-tenancy scheduling approaches on cloud platforms determine the primary scheduling approaches currently used and the challenges for addressing key multi-tenancy scheduling issues.
*Method:* We adopted a systematic literature review method to search and review many major journal and conference papers on four major online electronic databases, which address our four predefined research questions. Defining inclusion and exclusion criteria was the initial step before extracting data from the selected papers and deriving answers addressing our inquiries.
*Results:* Finally, 53 papers were selected, of which 62 approaches were identified. Most of these methods are developed without cloud layers' limitation (43.40%) and on SaaS, most of scheduling approaches are oriented to framework design (43.75%).
*Conclusion:* The results have demonstrated most of multi-tenancy scheduling solutions can work at any delivery layer. With the difference of tenants' requirements and functionalities, the choice of cloud service delivery models is changed. Based on our study, designing a multi-tenancy scheduling framework should consider the following 3 factors: computing, QoS and storage resource. One of the potential research foci of multi-tenancy scheduling approaches is on GPU scheduling.

*Keywords:* Systematic review, Survey, Cloud computing, Multi-tenancy, Scheduling

## 1. Introduction

Cloud computing enables resource sharing. However, different kinds of resources reflect different levels of dynamic behaviours and a diversity of user demands. This makes resource management very complex [1]. To make full use of the cloud's scalability and elasticity for cloud service providers, multi-tenancy is a key cloud characteristic that enables providers to share the same service instance, computational resources and storage among different tenants [2, 3]. In a multi-tenancy model, data and resources are deployed in the same cloud and are controlled and distinguished via labelling. This allows for the unique identification of resources owned by individual users [4]. Compared with a multi-user model, multi-tenancy customises *a single instance* based on multiple requirements of different tenants, as opposed to the multi-instance model, wherein each tenant has their own virtualised instance of the application [2, 5, 6]

More and more Application Service Providers (ASPs) and enterprises, especially Small to Medium sized Enterprises (SME), are using the SaaS delivery model. Databases hosted on the SaaS environment provide infrastructure and technologies that can support the instances running on applications with the same tables or schemes. Therefore shared hardware and software infrastructures can reduce the cost of deployment and operations. Multi-tenancy is a widely used technology in resource virtualisation and effectively help scale down costs [7, 8]. Multi-tenancy can scale up and down computing resources, as well as allocate resources according to the actual usage of the system/application, which is most widely used for SaaS applications. The scheduling strategies used in the multi-tenancy cloud take on a highlighted importance as they directly influence the runtime performance of software applications. Eventually, scheduling policies should be able to effectively improve the number of completed transactions (throughput), increase profit of service party, reduce completion time, reduce cost which is undertaken by service party when accepting applications and guarantee consistent Quality of Service (QoS) demands of clients [9].

The aim of conducting this systematic review is to discover the main methods of multi-tenancy scheduling and the previous challenges of multi-tenancy scheduling, as well as to investigate the current state-of-the-art of multi-tenancy scheduling approaches on cloud platforms and further research gaps and challenges. The main contributions of this work are summarised as ***Where***, ***How***, and ***What***:

- ***Where:*** to improve the system performance, which layer of the service model or which component is used to schedule tenants, data and instances to the resource pools?
- ***How:*** to improve the system performance or QoS, how do we modify scheduling approaches or resource allocation in the multi-tenant based cloud?
- ***What:*** are the limitations and weaknesses of the current scheduling methods in multi-tenant based cloud computing?

*Corresponding author
*Email addresses:* jiaruweiwei@gmail.com (Jia Ru), yyang@swin.edu.au (Yun Yang), john.grundy@monash.edu (John Grundy), Jacky.Keung@cityu.edu.hk (Jacky Keung), liucoolhao@gmail.com (Li Hao)

We systematically collected and reviewed domain-relevant papers published between 2009 to 2018. We identified several research questions for this systematic review that allow us to categorise existing research efforts to date. We restrict our attention to evidence-based guidelines that have appeared in scholarly publications [10].

The rest of this paper is organised as follows: Section 2 introduces the background and the overview of related work for this study. Section 3 briefly describes the methodology used to conduct this systematic review, including the defined research questions. Section 4 presents the results from the systematic review followed by a discussion in Section 5. Section 6 discusses the threats to validity in this study. Finally, Section 7 concludes our work.

## 2. Background

### *2.1. Multi-tenancy Implications*

Multi-tenancy is a specific characteristic of cloud applications that can change the underlying economics of applications through sharing infrastructure, platform and services. It allows each cloud application or "tenant", each with their own customers, processes and data, to obtain a single application instance [11]. A definition of multi-tenancy [12] is: *"Multi-tenancy refers to the architectural principle, where a single instance of the software runs on a SaaS vendor's servers, serving multiple client organisations (tenants). Multi-tenancy is contrasted to a multi-instance architecture where separate software instances (or hardware and software systems) are set up for different client organisations."*

### *2.2. Multi-tenancy Aware Cloud Applications*

Multi-tenancy aware applications can be defined as follows, they are suggested by Bezemer et al. [5, 13]: *"A* ***multi-tenant application*** *lets tenants (users) share the same hardware resources, by offering them one shared application and database instance, while allowing them to configure the application to fit their needs as if it runs on a dedicated environment."*

A multi-tenancy application looks and behaves similar to a single-tenant application, but the key difference is that multiple tenants can share the same cloud application services, platform and infrastructure. To achieve multi-tenancy awareness, sophisticated multi-tenancy support is required, which needs to be incorporated into the entire development and maintenance lifecycle of such applications [1, 14]. This support can be classified into two phases: development phase and deployment phase [14]. In the development phase, the multi-tenancy support is realised through high-degree configuration for different tenants. In the deployment phase, the service instances are introduced and deployed in the target cloud infrastructure according to different multi-tenancy requirements of the applications [14].

### *2.3. Overview of Related Reviews*

Cloud computing makes it possible to flexibly procure, scale and release computational resources on demand in response to workload changes. The work in [15] surveys the landscape of SLA-based cloud research to understand the state of the art and identify open problems. This work particularly aims at the resource allocation phase of the SLA life cycle while highlighting implications on other phases. This work contributes to

the fundamentals of engineering cloud SLA and their autonomic management, motivating further research and industry oriented solutions [15]. Results indicate that minimal number of SLA parameters are accounted for in most studies; heuristics, policies, and optimisation are the most commonly used techniques for resource allocation; and the Monitor-Analysis-Plan-Execute (MAPE) architecture style is predominant in autonomic cloud systems.

In recent years, the valuable knowledge that can be retrieved from petabyte scale datasets has led to the development of solutions to process information based on parallel and distributed computing [16]. Apache Hadoop [17] has attracted strong attention due to its applicability to processing data. The work in [16] provides a systematic literature review to assess research contributions related to Apache Hadoop, aiming to identify gaps, provide motivation for new research and to outline collaborations to Apache Hadoop. Their analysis led to some relevant conclusions: many interesting solutions developed in the studies were never incorporated into the framework; most publications lacked sufficient formal documentation of the experiments conducted by authors, hindering their reproducibility.

In cloud environments, resource allocation is an important issue due to the inherent uncertainty and dispersion of resources, which is caused by heterogeneity, dynamism, failures, etc. To provide efficient performance of workloads and applications, resource management techniques need to be addressed effectively. The work in [18] depicts a broad methodical literature analysis of autonomic resource management in the area of the cloud in general and QoS-aware autonomic resource management specifically. This work [18] helps researchers find the important characteristics of autonomic resource management and crucial properties of self-management (self-healing, self-configuring, self-optimising, and self-protecting), and helps to select the most suitable technique for autonomic resource management in a specific application along with significant future research directions. Efficient resource provision which can guarantee the satisfactory cloud services to the end users, lays the foundation for the success of commercial competition [19]. According to the deployment sequence, the work in [19] tackles the different deployment phases and the objectives of each phase. In [19], more than 150 articles from recent years are surveyed and the state-of-the-art algorithms to realise these objectives, e.g., cost, service quality and utility, are reviewed. Techniques employed in these algorithms are categorised and analysed systematically. Results show that almost all algorithms concentrate on the deployment phase [19].

Multiple scheduling algorithms in cloud environments have been proposed to ensure that short interactive jobs, large batch jobs and guaranteed-capacity production jobs running on these frameworks can deliver results quickly while maintaining a high throughput. However, only a few works have examined the effectiveness of these algorithms. The work in [20] conducts a systematic literature review of task scheduling algorithms that have been proposed for big data platforms. The work in [20] analyses the design decisions of different scheduling models proposed in the literature for Hadoop, Spark, Storm, and Mesos over the period between 2005 and 2016, and compares the algorithms in terms of performance, resources utilisation, and failure recovery mechanisms. The work in [20] identifies 586 studies and reports about different types of scheduling models (dynamic, constrained, and adaptive) and the main motivations behind them (including data locality, workload balancing, resources utilisation, and energy efficiency). A discussion of some open issues and future challenges pertaining to improving the current studies is

provided.

## 3. Method

Figure 1 illustrates the main procedure used for our systematic review. We followed the standard guidelines proposed by Kitchenham [10] and used a study protocol for our research. We chose to use a Systematic Literature Review approach, rather than a more general Systematic Mapping approach, to answer our focused research questions and to carry our a detailed comparative analysis of discovered approaches. Due to page limits, a detailed procedure describing the construction of our systematic review is provided online [1].

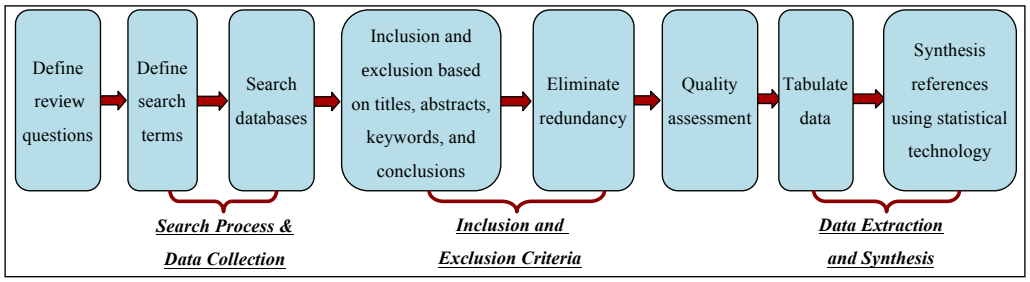


Figure 1: Systematic review procedure

### *3.1. Research Questions*

Constructing a systematic literature review requires a base set of research questions that drive the research methodology [10]. Referring to key prior studies [21, 22], we have identified four key research questions to investigate the approaches used for scheduling methods on a multi-tenancy cloud platform. We adopt a typical approach - PICOC criteria proposed by Petticrew et al. [23] to frame ourt systematic review questions. Using this we formulate our review questions based on 5 attributes: Population, Intervention, Comparison, Outcome, Context. Consequently, in our systematic review, we considered these 5 attributes, as shown in Table 1, to construct our research questions.

**RQ1: Scheduling on different service models**

Multi-tenancy has different meanings in different service models of the cloud computing. For instance, on the PaaS model (layer), applications are deployed on the same server to improve resource utilisation, however, on the SaaS layer, tasks are dispatched to multiple copies (instances) of the same software in a datacentre [24]. ***RQ1: How can we achieve multi-tenancy scheduling by using different service delivery models and layers?***

**RQ2: Scheduling with different objectives**

[1]The detailed procedure used in constructing this review can be found in Section 1 of Part A at https://github.com/Materials19/paper/blob/master/istslr2020online.pdf

Table 1: Review questions framed by PICOC criteria

| | |
|---|---|
| **Population** | Multi-tenancy scheduling approaches in the cloud |
| **Intervention** | Methodology/tool/technology for scheduling on multi-tenancy clouds |
| **Comparison** | The difference of scheduling approaches, especially showing evolution and improvement on traditional scheduling methods on cloud computing |
| **Outcome** | The effectiveness of scheduling approaches and implementation on multi-tenancy clouds |
| **Context** | Within the research domain of multi-tenancy clouds' scheduling, especially studies including experimental results |

Services with customisable non-functional qualities are used to address the requirements of tenants, such as response time, throughput, security, availability and the resulting price of a service. For example, GPUs have become prominent, both in high performance computing and in many cloud services, for many big data analysis tasks. Applications running on a GPU are different from those on a CPU. Thus, one can see how the issue of maximising GPU utilisation is one which is essential to enhancing performance and applicability. Due to the characteristics of sharing resources with multi-tenancy approaches, load balancing in multi-tenancy differs from traditional computing environments [25]. ***RQ2: Considering different needs, how do we schedule jobs for different tenants?***

**RQ3: Task management related to scheduling on different domains**

The entire scheduling process may include some other transactions, such as resource provision and tenant placement. For example, dynamic scaling performance of service-oriented applications hosted on the IaaS layer relies on the careful distribution of new VMs across physical hosts. In the IaaS layer, abstraction of physical hardware will result in resources being homogeneous and infinitely scaleable, providing linear increases in performance [26]. ***RQ3: How are related task management issues handled for multi-tenancy scheduling?***

**RQ4: Scheduling approaches on multi-tenancy clouds**

To support dynamically increasing demands from multi-tenants, cloud service providers have to deploy computing resources efficiently and cost-effectively to handle the fluctuation of requests from tenants using different methods, including algorithms, framework, etc. ***RQ4: What methods are used to schedule multi-tenancy cloud applications?***

### *3.2. Search Strategy and Process*

#### *3.2.1. Proposing Search Terms*

We first propose search terms to help us define topic relevancy and to classify articles' research topics. According to the lessons presented in [21, 22], we set up some important attributes and characteristics as search terms in this review. Our study background is scheduling on multi-tenancy clouds, and all the terms are expanded around this issue. The search terms are integrated by the Boolean "OR" or "AND" operation, which enables

each paper to at least have one of the relative terms. The "OR" operation groups different terms and the "AND" operation conjoins different terms which form search strings. An indicative search string used in our systematic review is shown in Figure 2. This is specialised for each digital library to be searched.

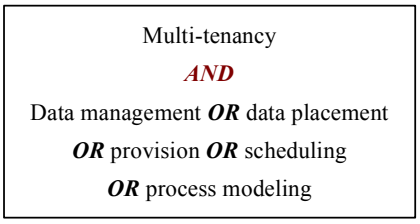


Figure 2: Search strings in our systematic review

*3.2.2. Search Process*

The search process was done in a manual manner to search for specific publications between 2009 to 2018. Primary data collection is by directly finding published papers (archival journals, and conference proceedings) from well-known and widely used online electronic digital libraries[2] We chose these four digital libraries as they provide a primary source for articles, contain all high repute venues for Computer Science and Software Engineering articles or relevance to our study, and have usable and accurate search engines for our search strings. We checked the references of all selected articles for any potentially relevant missed studies that should also be included in our search and analysis.

*3.3. Inclusion and Exclusion Criteria*

The main criterion of including journal and conference proceeding papers is based on their topic coverage relevant to our review questions. In our initial selection, we considered papers that clearly address our review questions, based on their titles, abstracts, keywords within the papers and their conclusions. Meanwhile, redundant publications were eliminated. However, the titles, abstracts and conclusions are not always enough to determine a paper's final inclusion. Therefore, in the final selection phase we retrieved the full context of those papers which were found to be relevant in the initial phase to decide on their final selection. Multi-tenancy and scheduling had to be the primary focus of the selected papers [3].

*3.4. Quality Assessment*

The quality of relevant papers can be accessed through the evidence presented in those studies. The conclusions drawn from a systematic review are only as strong as the evidence they are based on, so compiling an appropriate checklist to assess study

[2]Our selected digital libraries: ACM Digital Library (http://dl.acm.org), IEEE Xplore (http://ieeexplore.ieee.org/Xplore/home.jsp), ScienceDirect (http://www.sciencedirect.com), and SpringerLink (http://link.springer.com).

[3]The exclusion criteria is can be found in Section 2 of Part A at https://github.com/Materials19/paper/blob/master/istslr2020online.pdf

quality is important [10, 27]. A quality assessment checklist compiled by Kitchenham [10] was used to evaluate the quality of our located studies. Each question uses the same 3 level answer scale: "Yes" being a worth of 1 point value, "Partially" being worth of 0.5 and "No" being worth of 0. The total quality of each publication can be calculated by summing the quality scores of checklist questions. As the score of a paper increases, the paper will subsequently be better equipped to address the review questions in a more complex and profound manner [4]. The detailed quality assessment scores of all included articles are presented online [5].

### *3.5. Data Extraction and Synthesis*

The data extracted from the selected research papers provide a wide view of different scheduling approaches on multi-tenancy clouds. Aiming to address the review questions postulated in Section 3.1, we tabulate the data and analyse these data using a meta-analysis method [10]. To actualise the extraction of data from our included papers more explicitly, the data extraction schema used to collect the relevant data is provided online [6]. To conclude the data collection and review question analysis process, we summarise the quantitative data and then proceed to generalise and synthesise correlative answers addressing these review questions. Our results are presented in the following section.

### *3.6. Articles Classification Scheme*

Our classification scheme allows us to structure the literature in our work to map the literature in general and to answer our review questions in particular [28]. Classification of included papers considered several different schemes. Classification can simplify and reduce the systematic review complexity as well as improve the accuracy of the study. The classification scheme we developed is one of our novel contributions, providing a framework for categorising and describing scheduling approaches on multi-tenancy clouds. We classified each paper's approach using key words or phrases that describe the characteristics of the scheduling approaches:

**1. Classification on service model (layer)**

Scheduling polices are employed on different service models (layers).

- Software as a Service (SaaS)
- Platform as a Service (PaaS)
- Infrastructure as a Service (IaaS)
- Database as a Service (DBaaS)

**2. Classification on research aspect**

The main objective of this study is to find different multi-tenancy scheduling approaches. As discussed in Section 3.1, some relevant aspects are also considered relating to deployment scheduling polices that impact scheduling effort.

[4]The quality assessment checklist can be found in Section 3 of Part A at https://github.com/Materials19/paper/blob/master/istslr2020online.pdf

[5]Detailed quality assessment scores of included articles can be found in Part B at https://github.com/Materials19/paper/blob/master/istslr2020online.pdf

[6]Data extraction schema can be found in Section 4 of Part A at https://github.com/Materials19/paper/blob/master/istslr2020online.pdf

- Data management
- Data placement
- Process modelling
- Resource provision
- Scheduling

**3. Classification on methodological orientation**

- Algorithm
- Mechanism
- Framework or Architecture
- Model
- Others (System, Platform, Scheduler, etc.)

## 4. Results

Table 2: Distribution of reviewed papers in different digital libraries

| **Electronic database** | **Number of retrieved papers** | **Number of initial selected papers** | **Number of final included papers** | **Percentage in final inclusions (%)** |
|---|---|---|---|---|
| ACM Digital Library | 1106 | 205 | 13 | 24.53 |
| IEEE Xplore | 589 | 197 | 33 | 62.27 |
| ScienceDirect | 728 | 125 | 5 | 9.43 |
| SpringerLink | 612 | 48 | 2 | 3.77 |
| *Total* | 3035 | 575 | 53 | 100.00 |

We used different search terms to find relevant papers in the scientific digital libraries mentioned in Section 3.2.2. Using our search strings, we initially found 1106, 589, 728, and 612 results from ACM Digital Library, IEEE Xplore, ScienceDirect, and SpringerLink respectively. After initial selection based on the title, abstracts, keywords and conclusion, 575 relevant papers were assessed, 205 articles from ACM digital library, 197 articles from IEEE, 125 articles from ScienceDirect, and 48 articles from SpringerLink.

In our second selection phase, we reviewed the full context of the relevant papers selected in the initial phase according to the inclusion and exclusion criteria from Section 3.3. After eliminating redundant papers from different digital libraries, 53 papers were finally selected, with an acceptance ratio of 9.53%. These included papers and the main contributions of each paper are summarised online [7]. The distribution of reviewed papers from different databases is shown in Table 2. The 53 included papers were published in

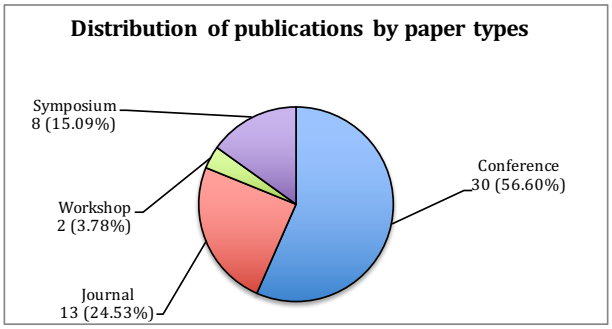


Figure 3: Distribution of papers by publication type

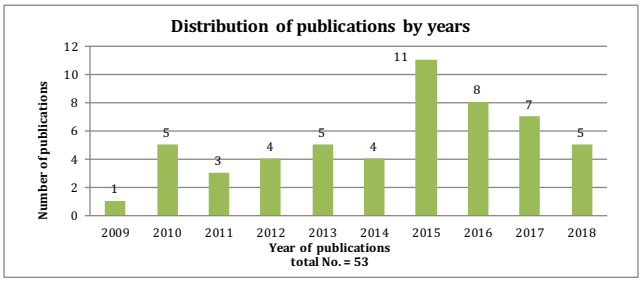


Figure 4: Distribution of papers by published year

40 different venues: 13 papers (24.53%) are from ACM Library, 33 papers (62.27%) are from IEEE, 5 papers (9.43%) are from ScienceDirect and 2 (3.77%) from SpringerLink.

The types of included papers can be classified into 4 categories: journal papers, conference papers, workshop papers and symposium papers. Figure 3 shows the specific distribution of different types. Journal papers occupy 24.53%, conference papers occupy 56.60%, workshop papers occupy 3.78% and symposium papers occupy 15.09%. More than half of the publications for scheduling approaches are from conferences.

Figure 4 shows the distribution of articles by year of publication. The trend in this figure indicates that research into multi-tenancy scheduling on cloud platforms has drawn more attention in recent years. Papers published in 2015 are the most included; the second most included papers were published in 2016.

We summarise the distribution of the overall results of our publication quality assessment, shown in Table 3. The score of most included papers is 8.0 (47.17%). According to our quality assessment scores, included papers should present some clear technical findings and provide experimental results as evidence to verify their proposed approaches.

The distribution of papers based on our proposed classification scheme is shown in Table 4. To gain an intuitive view, Table 5 shows the numerical results. In this review, we can observe that more and more multi-tenancy scheduling approaches are proposed without the cloud layer's limitation (43.40% of included papers). Most solutions can be

[7]The main contributions of each paper can be found in Part B at https://github.com/Materials19/paper/blob/master/istslr2020online.pdf

Table 3: Distribution of overall quality assessment of relevant studies

| Score | Number of papers | Percentage (%) |
|---|---|---|
| 5.0 | 1 | 1.89 |
| 5.5 | 0 | 0 |
| 6.0 | 2 | 3.77 |
| 6.5 | 7 | 13.21 |
| 7.0 | 7 | 13.21 |
| 7.5 | 11 | 20.75 |
| 8.0 | 25 | 47.17 |
| *Total* | *53* | *100.00* |

used in any IaaS, PaaS and SaaS, with only a few solutions being employed on DBaaS (3.77% of included papers). However, most scheduling approaches that work on all layers are algorithm-based (41.66% of 23 included papers). One introduces a new delivery model - High Performance Computing as A Service (HPCaaS) and proposes a novel scheduling algorithm on HPCaaS. The results also indicate that to improve multi-tenancy scheduling performance, designing a better database or building a preferable table scheme are useful ways (3.77%).

Table 4: Distribution of papers based on the proposed classification schemes

| Service model | Research aspect | Methodological orientation | Used technology | Publications |
|---|---|---|---|---|
| SaaS | Scheduling | Algorithm | Load balancing | P1 [29] |
| | | | Model cluster-based duplication to allocate resource | P2 [30] |
| | | | SLA based weighing | P3 [31] |
| | | Framework | Database scheme design | P1 [29] |
| | | | Two duplication time strategies: lazy, proactive | P2 [30] |
| | | | Performance isolation and dynamic SLA control | P3 [31] |
| | | Mechanism | Weighted Round Robin (WRR) with a control feedback loop | P4 [32] |
| | Data management | Orchestrator | Follow the Observe-Decide-Act (ODA) loop structure; a software-level power capping for docker containers | P39 [33] |
| | | Framework | Optimise user distributions and provision strategies | P5 [34] |
| | | | Exploit the microservice architecture | P23 [35] |
| | Resource provision | Model | Resource allocation based on tenant isolation, VM instance allocation and load balancing | P6 [36] |
| | | Controller | Consolidation of multiple VMs; meet the Service Level Objective (SLO) | P45 [37] |
| | Data placement | Framework | Delete unnecessary replicas, ensure migration flexibility, create missing replicas, fix overloaded servers and reduce number of active servers | P7 [38] |
| | | | Resource estimation and business relations | P8 [7] |
| | | Algorithm | Robust Tenant Placement (RTP) and interleaving replicas across nodes | P7 [38] |
| | | | Tenant data formalisation | P8 [7] |

Table 4: continued

| Service model | Research aspect | Methodological orientation | Used technology | Publications |
|---|---|---|---|---|
| PaaS | Resource provision | Mechanism | Estimation method in Service Demand Law (SDL) combined with request admission control | P9 [39] |
| | Data placement | Framework | SLA negotiation by fixing a threshold for each query and cost-effective query optimisation problem | P43 [40] |
| | | Algorithm | Admission control; dynamic computing of tenants' priority | P9 [39] |
| | | | Load balancing based on M/G/s/s+r queueing model | P10 [41] |
| IaaS | Resource provision | Model | Elastic Application Container (EAC) based lightweight resource management | P11 [42] |
| | | Scheduler | Least-busy and load-aware VM placement | P12 [26] |
| | | System | Online resource demand prediction and prediction error handling | P26 [43] |
| | Data management | Framework | Attribute-based constraints specification and enforcement | P25 [44] |
| | | | Use stochastic hill climbing algorithm to find multiple resource reservations & prevent performance interference | P33 [45] |
| | | | Design of MetaDatabase schema and multi-tenant ResourceManager on Hadoop | P34 [46] |
| | | | Based on fuzzy cloud controller and online process miner | P40 [47] |
| | | Algorithm | Meta-heuristics solution on augmented shuffled frog leaping | P49 [48] |
| | | | Address VM sharing in the context of Workflow-as-a-Service (WaaS) by modeling the use of containers | P50 [49] |
| | | System | Load balancing scheme; auto scaling mechanism | P32 [50] |
| | Scheduling | Platform | Extension of a Distributed Shared Objects (DSO) middleware and redistribution of resources among different Java Virtual Machine (JVM) instances | P13 [51] |
| | | Algorithm | Workflow scheduling to map and manage the execution of inter-dependent tasks | P48 [52] |
| | | Framework | Design on Service-Oriented Architectures (SOAs) to leverage rate limiters and request schedulers | P31 [53] |
| DBaaS | Scheduling | Algorithm | Dynamic resource allocation, residual allocation and SLA enforcement | P14 [54] |
| | | Framework | 4 components: Monitor, Analyser, Predictor, Allocator; periodically re-allocates resource under SLA violation | P14 [54] |
| | | Model | Model resource allocation as unbounded knapsack problem using additional fairness constraint | P14 [54] |
| | Resource provision | Framework | Input: performance-based Service Level Objectives (SLOs), tenant workloads and hardware Stock Keeping Units (SKUs); Output: an recipe combined with SLO compliant tenant scheduling strategy and cost-minimising hardware provisioning strategy | P15 [55] |
| HPCaaS | Scheduling | Algorithm | based on capabilities of software-defined networking | P47 [56] |
| N/A | Resource provision | Framework | Tenancy Requirements Model mapping tenancy requirements with appropriate resources | P16 [57] |
| | | | Capable system topologies based on a Monitor-Analyse-Plan-Execute (MAPE) loop | P17 [58] |
| | | | Graph processing; Progress-Aware Disk Prefetching (PADP) policy | P44 [59] |
| | | Algorithm | Prediction based provision & tenancy requirement matching | P16 [57] |

Table 4: continued

| Service model | Research aspect | Methodological orientation | Used technology | Publications |
|---|---|---|---|---|
| | Data management | Mechanism | Customise compensation process in workflow system and extension of states transition model of Web Services Business Activity (WS-BA) | P18 [60] |
| | | Scheduler | Use *Strings*, decompose GPU scheduling into load balancing and per-device scheduling, treat GPUs as first class schedulable entities and manage GPU calls with a two-level scheduler | P19 [61] |
| | | Framework | Provisioning, managing and controlling the services (DBaaS, PaaS, SaaS and IaaS) | P24 [62] |
| | | | Based on the datacentre policies; efficient VM scheduling techniques to place the active VMs on minimum number of physical servers | P36 [63] |
| | | | Based on datacentre virtualisation & load balancing | P38 [64] |
| | | Algorithm | Based on dominant resource fairness; use *Cgroup* to control resource utilisation | P35 [65] |
| | | | Based on load balancing and specific multi-tenancy requirements (delay and priority) | P41 [66] |
| | | | Utilise a priori knowledge of the dataflow tasks to provide predictable scheduling behaviour | P51 [67] |
| | | | Obtain the exact optimal number of task resources; optimise resource provisioning | P52 [68] |
| | | System | Based on Business Process Management System (BPMS); control the execution of business processes | P37 [69] |
| | Process modelling | Mechanism | Tenant placement and subsequent, fine-grained batch planning and optimising the workload | P20 [70] |
| | Data placement | Algorithm | Optimise the sliding-scheduled application placement and routing problem | P42 [71] |
| | Scheduling | Scheduler | Monitor load on each application, build performance models, compute a priority for pending GPU-based requests | P21 [72] |
| | | Algorithm | Convert realistic task resource utilisation patterns into boxes | P27 [73] |
| | | | Characterise a resource isolation mechanism and share the resources available within a process equally or in proportion to some weights among tenants | P28 [74] |
| | | | Use system level, application-transparent checkpointing mechanism and save the progress of jobs | P30 [75] |
| | | | Use checkpoint optimisation to tolerate and eliminate the Byzantine faults; track server performance of virtual clusters | P53 [76] |
| | | System | System level abstraction for GPU, utilise GPUs without compromising fairness, deploy GPU requests across the multiple GPUs under Least-Attained-Service (LAS) of GPU request servicing | P22 [25] |
| | | Platform | Based on virtualisation technology for computing, storage and network resources | P29 [77] |
| | | Framework | Virtualisation; consider I/O performance | P46 [78] |

Table 5: Overall results on methodological/research orientation

| **Topic** | **No. of papers** | **Percent (%)** | **Research aspects** | **No. of methods** | **Percent (%)** | **Method type** | **No. of methods** | **Percent (%)** |
|---|---|---|---|---|---|---|---|---|
| SaaS | 11 | 20.75 | Scheduling | 7 | 43.75 | Algorithm | 5 | 31.25 |
| | | | Data management | 3 | 18.75 | Framework | 7 | 43.75 |
| | | | Resource provision | 2 | 12.50 | Mechanism | 1 | 6.25 |
| | | | Data placement | 4 | 25.00 | Model | 1 | 6.25 |
| | | | | | | Orchestrator | 1 | 6.25 |
| | | | | | | Controller | 1 | 6.25 |
| | | | *Total* | *16* | *100.00* | *Total* | *16* | *100.00* |
| PaaS | 3 | 5.66 | Resource provision | 3 | 75.00 | Mechanism | 1 | 25.00 |
| | | | Data placement | 1 | 25.00 | Framework | 1 | 25.00 |
| | | | | | | Algorithm | 2 | 50.00 |
| | | | *Total* | *4* | *100.00* | *Total* | *4* | *100.00* |
| IaaS | 13 | 24.53 | Resource provision | 3 | 23.08 | Model | 1 | 7.69 |
| | | | Scheduling | 3 | 23.08 | Platform | 1 | 7.69 |
| | | | Data management | 7 | 53.84 | Algorithm | 3 | 23.08 |
| | | | | | | Framework | 5 | 38.47 |
| | | | | | | Scheduler | 1 | 7.69 |
| | | | | | | System | 2 | 15.38 |
| | | | *Total* | *13* | *100.00* | *Total* | *13* | *100.00* |
| HPCaaS | 1 | 1.89 | Scheduling | 1 | 100.00 | Algorithm | 1 | 100.00 |
| | | | *Total* | *1* | *100.00* | *Total* | *1* | *100.00* |
| DBaaS | 2 | 3.77 | Resource provision | 1 | 25.00 | Algorithm | 1 | 25.00 |
| | | | Scheduling | 3 | 75.00 | Framework | 2 | 50.00 |
| | | | | | | Model | 1 | 25.00 |
| | | | *Total* | *4* | *100.00* | *Total* | *4* | *100.00* |
| N/A | 23 | 43.40 | Resource provision | 4 | 16.67 | Framework | 7 | 29.18 |
| | | | Data management | 10 | 41.66 | Algorithm | 10 | 41.66 |
| | | | Process modeling | 1 | 4.17 | Mechanism | 2 | 8.33 |
| | | | Data placement | 1 | 4.17 | Platform | 1 | 4.17 |
| | | | Scheduling | 8 | 33.33 | Scheduler | 2 | 8.33 |
| | | | | | | System | 2 | 8.33 |
| | | | *Total* | *24* | *100.00* | *Total* | *24* | *100.00* |
| ***Total*** | ***53*** | ***100.00*** | ***Total*** | ***62*** | ***100.00*** | ***Total*** | ***62*** | ***100.00*** |

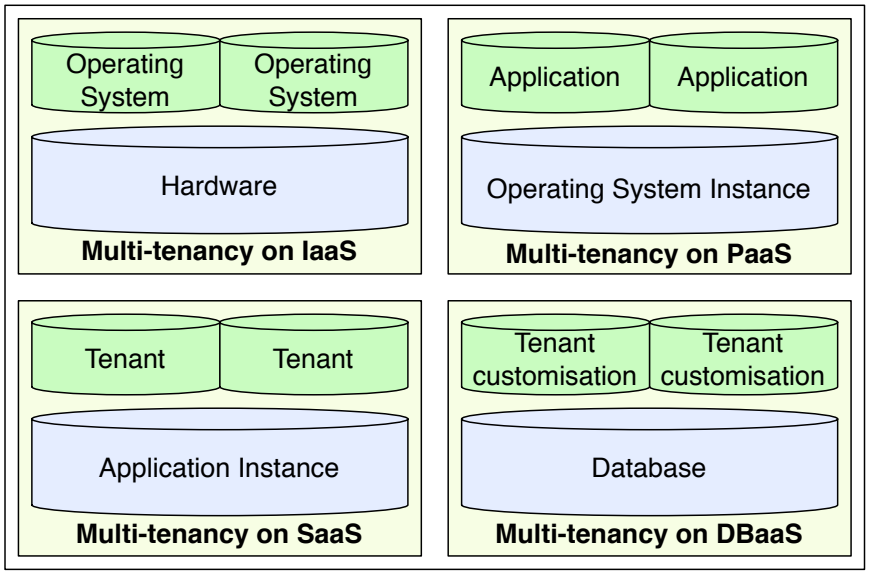


Figure 5: Multi-tenancy on different service layers

## 5. Detailed Results Analysis

In this section, we answer our four research questions (RQs) based on our collected results.

### *5.1. RQ1: Scheduling on different service models*

Multi-tenancy has different implementations on different cloud service models (layers) as shown in Figure 5. In the PaaS model, multi-tenancy indicates that different applications share the same operating system instance. In the IaaS model, multi-tenancy means that multiple operating system instances, usually in the form of virtual machines, share the same physical hardware via a hypervisor. In the SaaS model, multi-tenancy with the highest level of isolation indicates that one application instance is shared across multiple tenants [41, 79]. However, Database-as-a-Service (DBaaS) is gaining important momentum [54]. Most multi-tenancy problems are metadata driven and use metadata as a way of configuring the platform and applications for each individual tenant [80]. Multi-tenancy on DBaaS means different tenants have tenancy specific customisations which present different requirements of tenants as metadata attributes on separate schemes in shared database.

A key problem in DBaaS is how to efficiently share resources among tenants while maintaining SLAs [8]. SLAs guarantee performance isolation to prevent one tenant from adversely affecting the performance of other tenants in an unpredictable manner [54]. Moreover, DBaaS providers need to maximise resource utilisation for additional revenue by dynamically allocating available resources to tenants who consume resources beyond their SLAs. The scheduling in DBaaS can be summarised as a resource allocation problem

with a constrained optimisation to realise objective functions, such as static workloads scheduling.

Multi-tenancy applications are deployed on the servers which provide computing resources. PaaS providers isolate code from different applications on the same OS instance to improve resource utilisation, which brings new challenges to load balancing as the server is shared by several applications, and response times may not be guaranteed due to the competition for shared resources [24]. The key scheduling problem in PaaS is how to realise efficiency provision resource through algorithms or mechanisms. These algorithms are used to dynamically estimate resources and calculate different tenants' priority. In IaaS, the scheduling problem focuses on VM placements and the redistribution of resources among different JVM instances. Additionally, multi-tenancy architecture is often used in SaaS. For example, a two-tier SaaS scaling and scheduling architecture working at service and application levels to save resources and avoid duplicating as proposed in [30].

According to Table 4 and Table 5, more and more scheduling approaches are designed without a delivery layer limitation (occupying 38.71% in all approaches). For instance, **P46** proposes a novel disk scheduling framework - PriDyn (DYNamic PRIority) on any delivery layers. This framework provides differentiated services to various I/O applications co-located on a single host based on their latency attributes and desired performance. This demonstrates that all I/O applications executing in VMs in multi-tenant environments will suffer degradation in disk throughput irrespective of their latency requirements, depending upon the number and the types of I/O patterns of concurrent applications running on other active VMs hosted on the same server.

According to Table 4 and Table 5, a new delivery layer – High Performance Computing as a Service (HPCaaS) is proposed. The cloud enables the HPC users to have access to supercomputing resources on demand and in a cost-efficient manner [56]. The providers of HPCaaS, who often own the service platform, administrate and maintain the virtual resources. They can either own the hardware or rent it from a cloud service provider. Desirable characteristics of the cloud such as on-demand access, resource pooling and cost effectiveness, have tempted industries and academia to embrace this technology into their businesses, including the HPC users. Nevertheless, standard clouds do not satisfy certain unique requirements of HPC such as batch processing, direct access to hardware, the ability to bypass the OS kernel, and high-performance execution [56]. For instance, **P47** uses this feature to design and implement an innovative SDN empowered task scheduling system for HPCaaS, called 'ASETS'. **P47** also proposes a novel algorithm for this system called SETSA (SDN Empowered Task Scheduling Algorithm) [81]. This algorithm takes advantage of the SDN capabilities and schedules the tasks to the available VMs such that the virtualised network bandwidth is maximised on an elastic HPCaaS architecture with a shared file system and on-demand number of nodes.

- **Outstanding Limitations and Challenges**

Virtualisation is one of the key enabling technologies for cloud computing. Although it facilitates improved utilisation of resources, virtualisation can lead to performance degradation due to the sharing of physical resources like CPU, memory, network interfaces, disk controllers, etc. Multi-tenancy can cause highly unpredictable performance for concurrent I/O applications running inside virtual machines that share local disk

storage in cloud [78]. Disk I/O requests in a typical cloud setup may have varied requirements in terms of latency and throughput as they arise from a range of heterogeneous applications having diverse performance goals. This necessitates providing differential performance services to different I/O applications [78]. In future, **multi-tenancy frameworks should provide on demand access to scalable resources**, offering better utilisation of physical resources and enabling energy savings [78]. For instance, the frameworks should include a proactive approach where it would be possible to ascertain in advance whether a particular I/O application can be handled on a given system based on its state before actual scheduling of the I/O requests. Such a framework can enable meta-scheduling of I/O applications and enable QoS based placement algorithms at a larger datacentre level [78].

Emerging Workflow as a Service (WaaS) platforms offer scientists a simple, easily accessible, and cost-effective way of deploying their applications in the cloud at any time and from anywhere [49]. Such workflows can be modeled as Directed Acyclic Graphs (DAGs). Once a DAG is pre-processed, the scheduling process of its tasks can begin on WaaS platforms, which are emerging with the vision of providing scientists with the ability to deploy their applications for execution in the cloud in a simple and cost-effective manner [49]. An important aspect, as is for any multi-tenant cloud-based framework, is **how to efficiently manage the execution of workflows belonging to different users and with different QoS requirements**. Design of multi-tenant frameworks on WaaS to manage the execution of a continuous workload of heterogeneous workflows is a key needed future direction. To achieve this, they leverage the compute, storage, and network resources offered by IaaS providers [49].

Further research focus should explore **the deployment of multiple simultaneous containers on a single VM**, in order to execute multiple tasks in parallel. Investigating the effects of sharing resources among multiple workflows and using containers on the consumption of energy is also very important. Finally, it **would be beneficial to collect and make use of workflow execution data** to better estimate the runtime of tasks, to address security and privacy issues that arise from their multi-tenant nature, and to develop failure recovery strategies at various levels of the framework. Different pricing models that WaaS providers could adopt to charge their users could also be investigated [49].

### *5.2. RQ2: Scheduling with Different Objectives*

Multi-tenancy scheduling enables multiple instances of an application to occupy and share resources from a large pool, allowing different users to have their own version of the same application running and coexisting on the same hardware but in isolated virtual spaces [1] with different objectives, such as performance isolation, resource isolation, QoS, load balancing, etc.

#### *5.2.1. Performance Isolation*

There are four isolation levels in a multi-tenancy pattern [82]. Native multi-tenancy is the most common one [7]. In native multi-tenancy patterns, all web applications share the same infrastructure, and tenants share the same application instances. Furthermore, tenant data is created to carry the subscription and configuration information. Multi-Tenancy Management System (MTMS) prepares resources for it and provisions it to

certain database server if the provisioning condition is satisfied. This is a normal tenant data provisioning process.

To guarantee isolated performance, it is essential to control the resources used by a tenant. The layers of the execution platform that are responsible for controlling resource usage, normally do not have knowledge about entities defined at the application level and thus they cannot distinguish between different tenants. Furthermore, it is hard to predict how tenant requests propagate through the multiple layers of the execution environment down to the physical resource layer. The intended abstraction of the application from the resource controlling layers does not allow one to solely solve this problem in the application [39]. A method proposed in [39] combines resource demand estimation techniques with a request-based admission control to address this problem. Some methods try to place tenants onto a defined set of nodes in a way such that their workload profiles do not interfere with others to avoid mutual performance influences. However, this is only possible if the workload profiles for the tenants are predictable [32]. Control feedback loops have been developed to ensure isolation with a priority-based scheduling mechanism to dynamically adjust the priorities of a tenant [32].

*5.2.2. Resource Isolation*

When tenants compete inside a process, traditional and well-studied resource management techniques in the operating system and hypervisor are unsuitable for protecting tenants from each other. In such cases, aggressive tenants can overload the process and gain an unfair share of resources. In the extreme, this lack of isolation can lead to a denial-of-service to well-behaved tenants and even system wide outages. It is crucial to provide resource isolation to ensure that a single tenant cannot get more than its fair share of resources, to prevent aggressive tenants or unpredictable workloads from causing starvation, high latencies, or reduced throughput for others. However, it is difficult to provide isolation in these systems because multiple tenants execute within the same process [74]. **P28** presents Two-Dimensional Fair Queueing (2DFQ), which spreads requests of different costs across different threads and minimises the impact of tenants with unpredictable requests. A request scheduling algorithm is proposed in **P28** to produce fair and smooth schedules in systems that can process multiple requests concurrently.

*5.2.3. Database Management*

Most cloud scheduling algorithms and database solutions address their problems independently, but most cloud components and functionalities are interconnected. Specifically, a task scheduling algorithm needs to consider database partitioning to provide an efficient solution for performance and scalability [29]. More specifically, a task assigned to a processor should host the appropriate data partitions, otherwise data updates and migration among caches and processors can be very expensive. The most scalable multi-tenancy architecture requires a SaaS scheduler that can dispatch tasks to multiple copies of the same software in a datacentre. As the same version of the software is used, user customisation must be stored in databases, and thus an integrated solution must address both scheduling and database partitioning [29]. Considering database partitioning and consistency, tenant data should focus on tenant placement, which should be partitioned well in the back-end database to support real-time high performance computing [29].

*5.2.4. QoS Performance*

QoS on the multi-tenant SaaS potentially includes response time, throughput and least resource cost. Response time especially has attracted more attention, such as work in [72] which focused on clusters delivering acceptable response times, and work in [41] which concentrated on the mean response times of applications. SLAs are used to configure service resources according to different service requirements [31]. Indeed, defining performance objectives and above all, guaranteeing them is very challenging. Database queries have different levels of complexity, so defining a unique performance objective for all queries is not realistic [40]. Different tenants may have very different SLA requirements that need to be met by the same shared cloud application. For example, an SLA-based scheduling algorithm [31] guarantees the service quality of tenants and improves the system performance using request load as the measure of resource utilisation. **P43** defines a negotiation framework such that the tenant and the provider could define this threshold together in a rather fair way. The aim is to find a performance objective that is satisfactory or at least acceptable by the tenant and reachable (i.e. technically achievable and financially profitable) for the provider. In the SLA, **P43** also fixes the pricing policy, which is used to adjust the price according to the real performance and the parameter values.

SLA defines the parameter objective, calculation method, parameter threshold and violated action. An SLA template should contain parties of the agreement, the correlative information and service level objectives (SLO). SLO is the main body of SLA specifications which includes objective validity period, SLA parameters, metrics related to SLA parameters and so on [31]. Services that are elastically provisioned in the cloud are able to use platform resources on demand. Instances can be spawned to meet the SLO during periods of increasing workload and removed when workload drops. Enabling elastic provisioning saves the cost of hosting services in the cloud, since cloud users only pay for the resources that are used to serve their workload [37]. **P45** identifies and controls the different sources of unpredictability and builds Hubbub-Scale - an elasticity controller that is reliable in the presence of performance interference and achieves high resource utilisation without violating the SLO.

DBaaS providers want to support high performance to each tenant, but without introducing a heavy trade-off between cost and performance. A DBaaS provision and scheduling framework proposed in [55] is one of the solutions, which optimises operating costs while adhering to desired performance based SLOs.

*5.2.5. Multi-tenancy GPU Scheduling*

A GPU is treated as a device chosen by the applications. There are some key challenges of GPU scheduling:

1. Static collisions: Applications running on a multi-GPU node may compete for the same GPU, thus not able to leverage the availability of multiple on-node GPU accelerators. This will lead to the serialisation of GPU requests that otherwise could have been served in parallel [61].
2. Character collision: Applications are unaware of each other's GPU usage and their relative GPU intensities, and thus they cannot assess the performance implications of sharing a single GPU [61].

3. Static and character collisions become even more important when nodes have heterogeneous GPUs, each with different capabilities in terms of their compute and memory levels of GPU utilisation. This is particularly for web application-driven capacities, and memory bandwidths [25].
4. Single applications have difficulties achieving high utilisation by end user requests. [25].

In current multi-tenancy GPU scheduling challenge is static GPU provisioning, where applications explicitly and programmatically select their desirable GPU devices to run. GPU scheduling problem can be decomposed into load balancing and per-device scheduling [61]. Device-level scheduling efficiently uses all of a GPU's hardware resources, including its computational and data movement engines. Load balancing goes beyond obtaining high throughput, to ensure fairness through prioritising GPU requests that have attained least service.

*5.2.6. Load Balancing*

Most of the current QoS-aware load balancing policies are not suitable in a multi-tenancy environment since they are not aware about the mutual intervention among applications within the server. Thus, some applications' response time would exceed their threshold due to the lower throughput on the server [41]. Tenant-based load balancing considers isolation parallelism in web applications, requests from users are totally independent of each other [36].

Load balancing is a key service in the cloud and refers to the routing of packets from a source to a chosen destination [50]. Auto scaling helps in scaling a system horizontally in presence of a load spike by adding more instances of the application that can serve incoming requests. When load reduces or goes back to its normal state, some of the running instances of the application can then be stopped to ensure that the cloud tenant does not incur extra costs for running idle instances (since most public clouds charge by the time an instance runs). Auto scaling is best implemented as part of the load balancer service itself [50]. For example, **P32** presents HAvEN - a system for holistic load balancing and auto scaling in a multi-tenant cloud environment that is naturally distributed and hence scalable.

- **Outstanding Limitations and Challenges**

Each of the different isolation strategies require further work to improve them for multi-tenancy cloud applications. Some scheduling mechanisms have been introduced with concerns to their isolation capabilities, and all these can provide a certain degree of isolation, but these have low efficiency, especially when system becomes overcommitted due to constant priorities for each tenant. **More efficient performance isolation** is needed in many domains. There is also a **strong need for finer-grained resource isolation.**

Different tenants may have have vastly differing SLA requirements in terms of Quality of Service needs. Further work is needed **to address the balancing of different tenant SLAs** in a multi-tenancy cloud platform. A further important problem is **how DaaS providers can better balance multi-tenancy with performance-based SLOs** and how they deploy resources to tenants [55].

Future work should **address memory issues with multi-tenancy GPU clouds**, and in addition, **explore the use of runtime binary translation (of GPU kernels)** to further broaden the potential targets a load balancer can choose for running GPU requests (e.g., by running requests on otherwise idle CPUs) [35].

From the datacentre layer, **to achieve better load balancing among data partitions** to optimise the overall database performance, a more effective algorithm is highly desirable. This should migrate, distribute and duplicate tenants among partitions through monitoring the load [29].

Further work is needed to support better auto-scaling in multi-tenancy cloud environments. Such auto-scaling algorithms need **to fully support multi-tenancy and consider the utilisation levels of different resources** in the cloud as part of their load balancing and auto scaling algorithms.

### *5.3. RQ3: Task Management Related to Scheduling on Different Domains*

Table 6: Distribution of research domains

| **Research aspects** | **Publications** |
|---|---|
| Data placement (Total: 6) | P7, P8, P9, P10, P42, P43 |
| Data management (Total: 20) | P5, P18, P19, P23, P24, P25, P32, P33, P34, P35, P36, P37, P38, P39, P40, P41, P49, P50, P51, P52 |
| Resource provision (Total: 10) | P6, P9, P11, P12, P15, P16, P17, P26, P44, P45 |
| Scheduling (Total: 17) | P1, P2, P3, P4, P13, P14, P21, P22, P27, P28, P29, P30, P31, P46, P47, P48, P53 |
| Process modelling (Total: 1) | P20 |

Table 6 describes the distribution of different research domains of included papers. In Table 6, we clearly see that related data management methods and resource provision approaches can also help optimise multi-tenancy scheduling (occupying 55.56% in all the research objectives). Paper **P9** presents a resource allocation strategy and a data placement algorithm simultaneously. **P9** uses an estimation method in Service Demand Law (SDL) to predict the resources demands with the aim of realising dynamic resource allocation and also uses a admission control mechanism to dynamically compute tenants' priority with the aim of flexibly placing tenants on cloud nodes. For example, **P33** proposes a workload-aware resource reservation framework, named Argus, which targets multiple resource reservations and aims to prevent performance interference, in terms of fair throughput violation, in NoSQL stores [45]. Specifically, Argus focuses on cache and disk reservations. It enforces the cache reservation by splitting the cache space among tenants. It approximates the disk usage by throughput of the underlying file system and uses a request scheduler to enforce throughput reservation.

#### *5.3.1. Data Management*

From the data management perspective, framework design is an effective way to achieve efficient scheduling. When designing, service providers need to design and implement a specific class, and create an object of the class, which serves the requirements of

multiple users efficiently. A workflow system builds, executes, manages, and evolves its own process-oriented business applications, which is a reflection of data management [60]. Combining a workflow system with a SaaS mechanism provides a flexible and affordable environment for modem enterprise composite application development. The compensation process is a series of operation nodes to roll back the faulting transaction, which is suitable to handle special demand of compensation. Customisation of compensation process is an important concern when building a multi-tenancy scheduling framework.

Cloud hosted NoSQL data stores is for economic reasons often shared amongst multiple tenants simultaneously. The NoSQL provider consolidates multiple tenants' access into a shared NoSQL instance and provides a dedicated view for each tenant. This multi-tenancy has tenants' data and workloads coexisting in the same node, which under certain conditions can lead to performance degradation of one tenant caused by another. To better schedule applications among tenants, ones may investigate the multi-tenant interference in a common NoSQL store and enforce cache reservation by splitting the cache space and disk reservation by scheduling requests to a distributed file system (DFS) [45].

*5.3.2. Scheduling*

The scheduling strategies used in multi-tenancy cloud then become important, as they directly influence the runtime performance of software applications. Servers with multiple deployed applications need a proper request scheduling policy to guarantee their QoS. It is essential to design algorithms on SaaS, aiming to dynamically estimate the needed resources of each tenant while weighing SLAs and determining how the scheduling algorithms on IaaS focus more on resource usage and distribution on different VM instances. In addition, QoS-aware scheduling algorithms should be concerned with the mutual interactions among applications deployed on the same server and load balancing [41]. Scheduling approaches focusing on multi-tenant, instance-intensive workflows should consider three other aspects: (1) the quality of service experience (QoSE) of tenants in different SLAs, (2) the average execution time of multiple workflow instances, and (3) the execution cost saving for service providers [83].

*5.3.3. Resource Provisioning*

Resource provisioning means the selection, deployment, and run-time management of software (e.g., database management servers, and load balancers) and hardware resources (e.g., CPU, storage, and network) for ensuring guaranteed performance for applications [84]. Before resource provisioning can be completed, one needs to calculate the resource requirements (such as CPU, memory and storage) for the multi-tenancy in a shared application instance. This must satisfy some constraints (such as response time, availability, business transaction rate, and database request rate while minimising the cost) without violating SLA requirements. This is a very complex task to both perform and to engineer software applications to support [82]. Moreover, cost-effective scalability is not achieved if idle processors and other resources are unused but are charged to application providers. Over and under provisioning of cloud resources are still major problems. Current cloud virtualisation mechanisms do not provide cost-effective pay-per-use model for SaaS applications and just-in-time scalability is not achieved by simply deploying SaaS applications to cloud platforms. Due to elasticity, it is necessary to calculate the number of VMs needed when the SaaS platform is running and its tenants are consuming virtualised resources [36].

*5.3.4. Data Placement and Process Modelling*

In SaaS applications on in-memory databases, there is the challenge of the trade-off between low operational cost for the provider and performance as perceived by tenants: only so much consolidation can occur without significant impact on responsiveness. To manage this trade-off, the service provider must address two issues: resource modeling and data placement. Estimation of shared resource consumption in the presence of multi-tenancy on a single server is important, which can be solved by characterising the dominating resources (CPU, RAM, disk I/O) and quantifying how much each tenant utilises them [38]. Finally, it is important to model users' requirements with a set of services. Services should be modeled in a customisable way so that each tenant can customise the services according to their specific requirements [85].

- **Outstanding Limitations and Challenges**

From the data management perspective, **quantifying the impact of writes on reads and modelling the I/O behaviour** through offline sampling needs further research. Resource reservation also should consider the memory usage for writes. Increasing the size of the write buffer will boost the write performance but harms read performance as the size of cache block decreases. It is beneficial to set the sizes of cache and write buffer according to different workload characteristics [45].

From resource management perspective, the lack of end-to-end visibility and complex request execution structures make it challenging to regulate two key metrics in an SOA across multiple tenants: the end-to-end throughput (and thereby, the load at every process) and the end-to-end latency. To regulate system load, **the resource management system must correctly attribute overload** to a specific subset of tenants, and limit of the entire chain of API invocations for only those tenants, with minimal impact on others. Approaches to **end-to-end latency goals** requires further work, including making better local request scheduling decisions despite limited visibility into the full request execution graph [53].

*5.4. RQ4: Scheduling Approaches on Multi-tenancy Clouds*

Table 7: Distribution of method types

| **Research aspects** | **Publications** |
|---|---|
| Algorithm (Total: 22) | P1, P2, P3, P7, P8, P9, P10, P14, P16, P27, P28, P30, P35, P41, P42, P47, P48, P49, P50, P51, P52, P53 |
| Framework (Total: 22) | P1, P2, P3, P5, P7, P8, P14, P15, P16, P17, P23, P24, P25, P31, P33, P34, P36, P38, P40, P43, P44, P46 |
| Mechanism (Total: 4) | P4, P9, P18, P20 |
| Model (Total: 3) | P6, P11, P14 |
| Others (Total: 11) | **System:** P22, P26, P32, P37; **Scheduler:** P12, P19, P21; **Platform:** P13, P29; **Orchestrator:** P39; **Controller:** P45 |

As shown in Table 7, to achieve multi-tenancy scheduling, optimising the design of algorithms and frameworks (70.97% of reported approaches) is one of the most commonly used methods for the development of efficient multi-tenancy scheduling. Moreover, to support the proposed multi-tenancy frameworks, researchers have also proposed some algorithms to meet a framework's functional requirements (Such as **P1**, **P2** and **P3**). For instance, a framework proposed in **P14** periodically re-allocates resource to tenants, aiming to maximise the resource utilisation while tolerating a low risk of SLA violations, especially for highly dynamic workload, and in this framework, a dynamic resource allocation algorithm for DBaaS is proposed if there is a higher variance intensity. Additionally, an extensible dynamic provisioning framework presented in **P16** began with a Tenancy Requirements Model (TRM). The model in **P16** is based on the mapping of functional and non-functional tenancy requirements with appropriate resources, their parameters, and health monitoring policy allows dynamic re-provisioning for existing tenants based on either changing tenancy requirements or health grading predictions.

There are many ways to achieve multi-tenancy scheduling, such as virtualisation, prediction, interference, theoretical scheduling and boxing technologies. For instance, virtualisation is a technique that facilitates the sharing of resources in datacentres, which transforms a huge collection of bare-metal hardware into cloud infrastructure with high flexibility, predictable performance, reliability, controllability and security. Among the recent advances of network virtualisation, **P38** uses the software-defined networking (SDN) and its underlying Distributed Virtual Switch (DVS). SDN is a revolutionary innovation in computer networks in the sense that network control is decoupled from the data forwarding function and is directly programmable. The OpenFlow protocol is a fundamental element for building SDN solutions. It is an industry-standard SDN communications protocol, allowing operators to address complex network behaviour, while optimising performance and allowing operators to leverage a richer set of capabilities. The proposed framework in **P38** takes advantage of the application of DVS, and leverages the adoption of OpenFlow protocols to accommodate heterogeneous network communication patterns by supporting arbitrary traffic matrices among VMs in virtual private clouds (VPCs). The framework generates multi-tenancy oriented private clouds, offering great flexibility to cloud users. The framework achieves global load balancing on the underlying physical network, thus delivering predictable performance. Moreover, it is adaptive to a dynamic network environment. For example, **P27** converts realistic task resource utilisation patterns into resource boxing to directly explore theoretical and applied scheduling in cloud computing. **P27** designs resource boxing to convert real task patterns to theoretical scheduling inputs. Theoretical scheduling studies optimal allocation of boxes to machines in adherence to an objective function. In terms of computing, boxes and machines are represented as tasks that execute within machines.

Most frameworks are employed at the SaaS level. Since duplication if the application may result in significant resource waste, some proposed frameworks focus on the application level. Designing a framework should consider three factors: ***computing***, ***QoS*** and ***storage resource***. Under the constraint of a certain number of database servers, how to place an incoming tenant in a distributed computing environment in order to balance the servers [7]. A potential solution is consumption of computing resources in database servers regarding several aspects: maintaining database middleware, tenant access data via applications, cross-tenant (selected or inserted request access multiple tenant data isolation) access [7]. To better enable providers to model services with QoS and non-

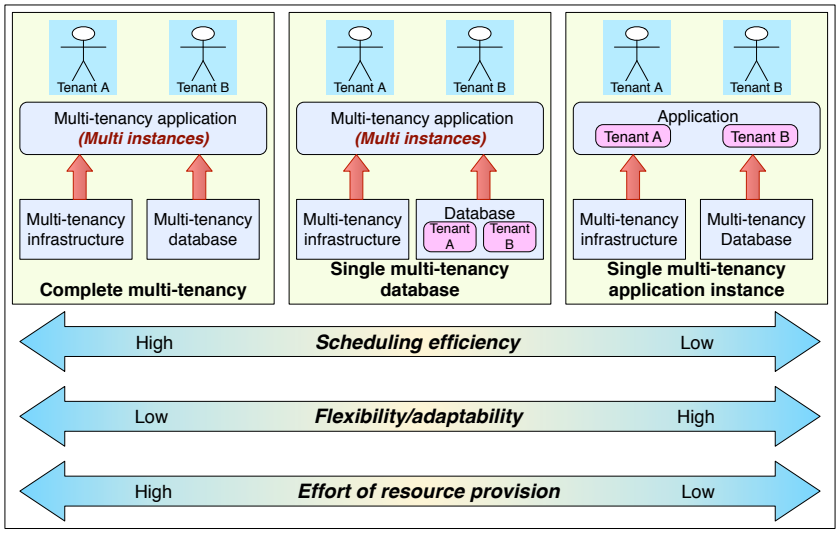


Figure 6: Achievement of multi-tenancy

functional properties [34]. A potential solution is to use algorithms to compute efficient tenant distributions that take requirements of tenants into account [34].

- **Outstanding issues and challenges**

Multi-tenancy for cloud applications can help service providers lower costs through economical scalability, improve resource utilisation and reduce service customisation time, through sharing hardware resources by multiple users [86]. To achieve multi-tenancy, we can focus on many different components, such as application code, operating system, data storage software and computing resources (referring to Table 4).

Figure 6 describes what components can be shared. Even in data storage (referring to single multi-tenancy database mode), there are many different shared options, such as separate database, shared database and separate schemas, shared database and shared schema, etc. Complete multi-tenancy can efficiently realise specific tenant behaviour and each tenant or each group of tenants adopt their own service to achieve their own specific behaviour. Each tenant is served by its own instance and it may result in load unbalance on an application service [1, 41]. However, this kind of multi-tenancy can provide better resource provision through the construction of multi-tenancy frameworks which focus on dynamic resource allocation (best-fit resources for different tenancy requirements) [57] and by optimising tenants' distributions and provisioning strategies [34]. Due to the previously discussed aspects of complete multi-tenancy, the flexibility of this type of multi-tenancy is subsequently less than other tenancy models. In comparison, in single multi-tenancy application instance modes, a single instance service can be adopted to deploy once and have the same behaviour for all the tenants. It does not need special consideration regarding multi-tenancy. In addition, application deployment is much easier for service providers, as only one service instance of the application needs to be deployed [2]. When updating the application, the service will only be updated once

for all the tenants. Since all the tenants share all the data and tenants cannot specify their requirements, the scheduling effort and system performance are not good enough, therefore making scaling have fewer infrastructure implications.

According to our analysis results, multi-tenancy scheduling frameworks usually use virtualisation technology, but often do not support the features of the full virtualisation stack [63]. Thus, in the future, researchers should **focus more heavily on full virtualisation technology**. From the perspective of resource boxing technology, boxes are unable to capture the dynamicity of resource utilisation patterns inherited within cloud computing. Failure to capture this behaviour results in reduced applicability due to the disconnect between the evaluation of theoretical scheduling derived from real world operations. In other words, there is a requirement **to find an accurate representation of execution patterns that are composed of boxes**. Such a technique would allow for a direct application of the theoretical scheduling algorithms to real-life execution patterns [73]. Further work about resource boxing should **consider more extreme workflow patterns** to evaluate the accuracy of resource boxing. Currently, only few works focus on the platform design so far. Future scholars should seek to investigate the **scalability of a resource provisioning platform**. These novel platforms will be applied on a large-scale datacentre infrastructure to serve large number of tenants and applications with diverse resource requirements.

### *5.5. Key future directions*

Other authors use very different approaches to achieve multi-tenancy scheduling, such as resource allocation (**P28**) and tenant placement (**P1**). Although concerned with performance, some works present solutions covering important related areas, for example, data management (**P25**), database scheme design (**P34**) and resource reservation (**P33**), which reflects indirectly on scheduling performance in multi-tenancy cloud environments. Thus, scheduling in multi-tenancy clouds can also be addressed by some of these studies. Ultimately, some approaches develop mechanisms for design of models in web service business activity (**P18 and P20**), and some develop systems based on business process management systems (**P37**), or abstraction of GPU computation (**P22**).

For improved support for scheduling in multi-tenancy clouds, the following major research challenges still need to be solved:

- Due to the variety of tenants' resource demands and the difficulty in predicting the resource demand of tenant's workload/application, **there is a need for a resource management technique** that can easily make the right decisions regarding dynamic scaling of resources and workloads/applications.

- Currently it is difficult to achieve performance isolation for multi-tenancy on different service layers. To resolve this problem, **there is a need for isolation of resources and better abstraction of applications**. It is crucial to provide resource isolation to ensure tenants to fair share resources. To isolate resources, we need to better determine which part of the resources are to be isolated depending on tenants' requirements, such as database, performance and resource isolation.

- **There is a need to develop scheduling approaches that optimise both QoS targets**: tenant-centric (deadline and throughput) and resource-centric (reliability, availability, and utilisation).

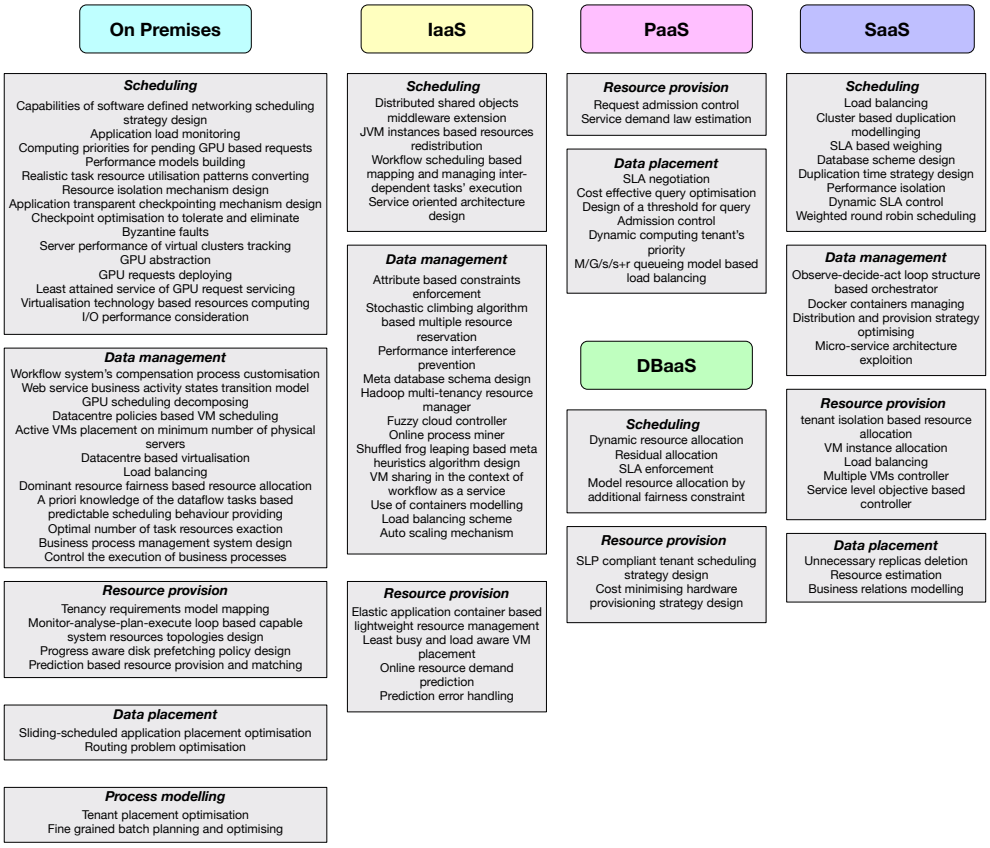


Figure 7: Research road map for multi-tenancy scheduling

- **There is a need to develop GPU scheduling for multi-tenants**. The difficulty of GPU scheduling involves static collisions and character collision. GPU scheduling problem can be decomposed into load balancing and per-device scheduling.

- In centralised distributed systems, it is very difficult to manage the large number of tenant requests in multiple service queues, which further leads to performance degradation. **New multi-tenancy scheduling approaches need to be more decentralised.**

Figure 7 shows a proposed future research map for work in multi-tenancy scheduling. In this figure, we can see that if we would like to improve multi-tenancy scheduling performance, which service model can be improved, what aspects can be focused on, and what technologies can be used. For instance, On SaaS, we can enforce data management to improve scheduling performance. Inside, we can manage docker containers and exploit micro-service architecture.

## 6. Threats to Validity

The results of this SLR might have been affected by some limitations such as bias in the selection of primary studies, inaccuracy in performing data extraction, and assessing quality of the studies.

The primary threat of this review is potential publication selection bias. We addressed the issue of bias in study selection through comprehensive searching from search databases commonly used in existing SLRs (e.g. [87]). The procedure that we adopted in this study follows the guidelines of Kitchenham et al. [10], which helps to minimise the possibility of missing evidence. The searching phase of this work also faced some limitations due to the limited availability of suitable search options in the search engine and online databases. For example, the IEEE Xplore does not support a lengthy search string. In the case of IEEE Xplore, it is clearly mentioned that the maximum number of search terms is 15. Hence, to overcome this issue, we created several sub-strings, which were executed separately and the results from each execution was recorded and then accumulated [87]. Our search was organised as a multi-step process including manual and automatic searches. To ensure the selection was unbiased, we defined a review protocol. During the search strategy, we accessed relevant papers based on the appropriateness of the search strings. If the title, abstract and keywords were not very clear and straightforward, we chose to underestimate the importance of these research papers and subsequently excluded them from the review. Moreover, the keywords used to retrieve literature may be extended to the fields of distributed or large-scale database, distributed data, machine learning and grid computing that have light correlation with big data and cloud computing. Due to time restrictions, we only considered some commonly used databases: IEEE Xplore, ACM Library, ScienceDirect and SpringerLink. Although this may result in bias and present a threat to validity, the primary conferences and journals of this domain have been searched to reduce such limitations. We also did not take into account of surveys, technical reports, theses and non-published papers.

The secondary potential threat is inaccuracy in our data extraction. We used a data extraction schema mentioned in Section 3.5 and a quality assessment described in Section 3.4. The evaluation of quality level of primary studies was considerably subjective. For instance, some studies did not explicitly define the variables or measures used in their research design. To reduce the likelihood of erroneous results, different researchers conducted separate extractions and evaluate the quality assessment independently, as well as clearly report the observations and evidences of each included paper related to the review questions.

## 7. Conclusion

Multi-tenancy is a new software architectural pattern with a single instance of applications (or customising the data and configuration) running on service provider's infrastructure. In this study, we have systematically reviewed research studies related to multi-tenancy scheduling on cloud platforms, which attempts to investigate applied methods and key research trends to solve multi-tenancy scheduling issues in an emerging area of software engineering in cloud environment.

The results have demonstrated that most of multi-tenancy scheduling solutions (43.40%) without delivery layers limitation, which can work at any layer. It introduces a new de-

livery model - HPCaaS. With the difference of tenants' requirements and functionalities, the choice of cloud platforms are changed from SaaS to DBaaS. IaaS more focuses on virtual machines placement on servers to realise resource provision. Based on our study, designing a multi-tenancy scheduling framework should consider the following three factors: computing, QoS and storage resource. Yet multi-tenancy scheduling algorithms are most used to dynamically estimate needed resource, and compute tenants' priority or other QoS. To combine with scheduling algorithms and database solution, scheduler on SaaS is a better choice. Some related task management issues can achieve multi-tenancy scheduling, such as data management, data placement, and some other approaches focus on data operation, such as building database or data partition.

Additionally, multi-tenancy scheduling approaches mostly consider load balancing problem and QoS parameters, such as response time through negotiating SLA violations and system performance. Nowadays, GPU are treated as explicitly selected devices in high performance applications. GPU scheduling can be decomposed into load balancing and per-device scheduling and further direction of GPU scheduling is static GPU provisioning. We have provided a comprehensive overview of different scheduling approaches dedicated multi-tenant based cloud platform, which is important for future development of multi-tenant based applications for cloud computing.

In our own future work, we will evaluate the performance of several multi-tenancy scheduling algorithms and make comparisons of these algorithms using empirical methods to evaluate these scheduling methods.

## Acknowledgment

This research is supported by a scholarship from Swinburne University of Technology. Grundy is supported by ARC Laureate Fellowship FL190100035.

## References

[1] R. Jia, J. Grundy, J. Keung, Software engineering for multi-tenancy computing challenges and implications, in: Proceedings of ACM International Workshop on Innovative Software Development Methodologies and Practices, ACM, 2014, pp. 1–10.

[2] C.-P. Bezemer, A. Zaidman, Multi-tenant saas applications: maintenance dream or nightmare?, in: Proceedings of the Joint ERCIM Workshop on Software Evolution (EVOL) and International Workshop on Principles of Software Evolution (IWPSE), ACM, 2010, pp. 88–92.

[3] M. Almorsy, J. Grundy, I. Müller, An analysis of the cloud computing security problem, in: Proceedings of 30th IEEE Conference on Asia-Pacific Software Engineering (APSEC), Cloud Workshop, IEEE, 2010.

[4] K. Wood, M. Anderson, Understanding the complexity surrounding multitenancy in cloud computing, in: Proceedings of 8th IEEE International Conference on e-Business Engineering (ICEBE), 2011, pp. 119–124.

[5] C.-P. Bezemer, A. Zaidman, B. Platzbeecker, T. Hurkmans, A. Hart, Enabling multi-tenancy: An industrial experience report, in: Proceedings of IEEE International Conference on Software Maintenance (ICSM), IEEE, 2010, pp. 1–8.

[6] G. C. Frederick Chong, R. Wolter, Multi-tenant data architecture, in: http://msdn.microsoft.com/en-us/library/aa479086.aspx.

[7] K. Tang, Z. B. Jiang, W. Sun, X. Zhang, W. S. Dong, Research on tenant placement based on business relations, in: Proceedings of 7th IEEE International Conference on e-Business Engineering (ICEBE), IEEE, 2010, pp. 479–483.

[8] S. Aulbach, T. Grust, D. Jacobs, A. Kemper, J. Rittinger, Multi-tenant databases for software as a service: schema-mapping techniques, in: Proceedings of ACM SIGMOD International Conference on Management of Data, ACM, 2008, pp. 1195–1206.
[9] R. Jia, J. Keung, An empirical investigation on the simulation of priority and shortest-job-first scheduling for cloud-based software systems, in: Proceedings of 22nd IEEE International Conference on Software Engineering Conference (ASWEC), IEEE, 2013, pp. 78–87.
[10] B. Kitchenham, Procedures for performing systematic reviews, Keele, UK, Keele University 33 (2004) 2004.
[11] L.-J. Zhang, J. Fiaidhi, I. Bojanova, J. Zhang, Enforcing multitenancy for cloud computing environments, IT Professional 14 (1) (2012) 0016–18.
[12] D. Banks, J. Erickson, M. Rhodes, Multi-tenancy in cloud-based collaboration services, Information Systems Journal.
[13] C.-P. Bezemer, A. Zaidman, Challenges of reengineering into multi-tenant SaaS applications, Delft University of Technology, Tech. Rep. TUD-SERG-2010-012.
[14] R. Mietzner, T. Unger, R. Titze, F. Leymann, Combining different multi-tenancy patterns in service-oriented applications, in: Proceedings of IEEE International Conference on Enterprise Distributed Object Computing (EDOC'09), IEEE, 2009, pp. 131–140.
[15] F. Faniyi, R. Bahsoon, A systematic review of service level management in the cloud, ACM Computing Surveys (CSUR) 48 (3) (2016) 43.
[16] I. Polato, R. Ré, A. Goldman, F. Kon, A comprehensive view of Hadoop research-a systematic literature review, Journal of Network and Computer Applications 46 (2014) 1–25.
[17] T. White, Hadoop: The definitive guide, O'Reilly Media, 2012.
[18] S. Singh, I. Chana, QoS-aware autonomic resource management in cloud computing: a systematic review, ACM Computing Surveys (CSUR) 48 (3) (2016) 42.
[19] J. Zhang, H. Huang, X. Wang, Resource provision algorithms in cloud computing: A survey, Journal of Network and Computer Applications 64 (2016) 23–42.
[20] M. Soualhia, F. Khomh, S. Tahar, Task scheduling in big data platforms: A systematic literature review, Journal of Systems and Software 134 (2017) 170–189.
[21] P. Brereton, B. Kitchenham, D. Budgen, M. Turner, M. Khalil, Lessons from applying the systematic literature review process within the software engineering domain, Journal of systems and software 80 (4) (2007) 571–583.
[22] B. Kitchenham, P. Brereton, D. Budgen, M. Turner, J. Bailey, S. Linkman, Systematic literature reviews in software engineering–a systematic literature review, Information and software technology 51 (1) (2009) 7–15.
[23] M. Petticrew, H. Roberts, Systematic reviews in the social sciences: A practical guide, Wiley. com, 2008.
[24] T. Zhao, H. Sun, Y. Tang, X. Liu, A load balancing algorithm in multi-tenancy environment, in: Proceedings of ACM/IFIP/USENIX International Middleware Conference, ACM, 2013, p. 16.
[25] D. Sengupta, R. Belapure, K. Schwan, Multi-tenancy on gpgpu-based servers, in: Proceedings of 7th ACM International Workshop on Virtualization Technologies in Distributed Computing, ACM, 2013, pp. 3–10.
[26] W. Lloyd, S. Pallickara, O. David, M. Arabi, K. Rojas, Dynamic scaling for service oriented applications: Implications of virtual machine placement on IaaS clouds, in: Proceedings of IEEE International Conference on Cloud Engineering (IC2E), IEEE, 2014, pp. 271–276.
[27] D. Azhar, E. Mendes, P. Riddle, A systematic review of web resource estimation, in: Proceedings of 8th International Conference on Predictive Models in Software Engineering, ACM, 2012, pp. 49–58.
[28] D. Maplesden, E. Tempero, J. Hosking, J. Grundy, Performance analysis for object-oriented software: A systematic mapping, IEEE Transactions on Software Engineering 41 (7) (2015) 691–710.
[29] W.-T. Tsai, Q. Shao, Y. Huang, X. Bai, Towards a scalable and robust multi-tenancy SaaS, in: Proceedings of 2nd ACM Asia-Pacific Symposium on Internetware, ACM, 2010, pp. 1–15.
[30] W.-T. Tsai, X. Sun, Q. Shao, G. Qi, Two-tier multi-tenancy scaling and load balancing, in: Proceedings of 7th IEEE International Conference on e-Business Engineering (ICEBE), 2010, pp. 484–489.
[31] X. Cheng, Y. Shi, Q. Li, A multi-tenant oriented performance monitoring, detecting and scheduling architecture based on SLA, in: Proceedings of IEEE Joint Conference on Pervasive Computing (JCPC), IEEE, 2009, pp. 599–604.
[32] R. Krebs, A. Mehta, A feedback controlled scheduler for performance isolation in multi-tenant applications, in: Proceedings of 3rd IEEE International Conference on Cloud and Green Computing (CGC), IEEE, 2013, pp. 195–196.
[33] A. Asnaghi, M. Ferroni, M. Santambrogio, DockerCap: A software-level power capping orchestra-

tor for docker containers, in: Proceedings of 15th IEEE International Symposium on Distributed Computing and Applications for Business Engineering (DCABES), IEEE, 2016, pp. 90–97.
[34] C. Fehling, F. Leymann, R. Mietzner, A framework for optimized distribution of tenants in cloud applications, in: Proceedings of 3rd IEEE International Conference on Cloud Computing (CLOUD), IEEE, 2010, pp. 252–259.
[35] S. Kalra, Prabhakar, Towards dynamic tenant management for microservice based multi-tenant SaaS applications, in: Proceedings of the 11th ACM International Conference on Innovations in Software Engineering, ACM, 2018, pp. 1–5.
[36] J. Espadas, A. Molina, G. Jiménez, M. Molina, R. Ramírez, D. Concha, A tenant-based resource allocation model for scaling Software-as-a-Service applications over cloud computing infrastructures, Future Generation Computer Systems 29 (1) (2013) 273–286.
[37] N. Rameshan, Y. Liu, L. Navarro, V. Vlassov, Hubbub-scale: Towards reliable elastic scaling under multi-tenancy, in: Proceedings of 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), IEEE, 2016, pp. 233–244.
[38] J. Schaffner, T. Januschowski, M. Kercher, T. Kraska, H. Plattner, M. Franklin, D. Jacobs, RTP: robust tenant placement for elastic in-memory database clusters, in: Proceedings of ACM International Conference on Management of Data, ACM, 2013, pp. 773–784.
[39] R. Krebs, S. Spinner, N. Ahmed, S. Kounev, Resource usage control in multi-tenant applications, in: Proceedings of 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), IEEE, 2014, pp. 122–131.
[40] S. Yin, A. Hameurlain, F. Morvan, Sla definition for multi-tenant dbms and its impact on query optimization, IEEE Transactions on Knowledge and Data Engineering 30 (11) (2018) 2213–2226.
[41] H. Sun, T. Zhao, Y. Tang, X. Liu, A QoS-aware load balancing policy in multi-tenancy environment, in: Proceedings of 8th IEEE International Symposium on Service Oriented System Engineering (SOSE), IEEE, 2014, pp. 140–147.
[42] S. He, L. Guo, Y. Guo, C. Wu, M. Ghanem, R. Han, Elastic application container: A lightweight approach for cloud resource provisioning, in: Proceedings of 26th IEEE International Conference on Advanced information networking and applications (AINA), IEEE, 2012, pp. 15–22.
[43] Z. Shen, S. Subbiah, X. Gu, J. Wilkes, Cloudscale: elastic resource scaling for multi-tenant cloud systems, in: Proceedings of 2nd ACM Symposium on Cloud Computing, ACM, 2011, pp. 1–14.
[44] K. Bijon, R. Krishnan, R. Sandhu, Mitigating multi-tenancy risks in IaaS cloud through constraints-driven virtual resource scheduling, in: Proceedings of 20th ACM Symposium on Access Control Models and Technologies, ACM, 2015, pp. 63–74.
[45] J. Zeng, B. Plale, Workload-aware resource reservation for multi-tenant NoSQL, in: Proceedings of IEEE International Conference on Cluster Computing (CLUSTER), IEEE, 2015, pp. 32–41.
[46] H. Won, M. C. Nguyen, M.-S. Gil, Y.-S. Moon, Advanced resource management with access control for multitenant Hadoop, Journal of Communications and Networks 17 (6) (2015) 592–601.
[47] G. Acampora, M. L. Bernardi, M. Cimitile, G. Tortora, A. Vitiello, A fuzzy-based autoscaling approach for process centered cloud systems, in: Proceedings of IEEE International Conference on Fuzzy Systems (FUZZ-IEEE), IEEE, 2017, pp. 1–8.
[48] P. Kaur, S. Mehta, Resource provisioning and work flow scheduling in clouds using augmented shuffled frog leaping algorithm, Journal of Parallel and Distributed Computing 101 (2017) 41–50.
[49] M. Rodriguez, R. Buyya, Scheduling dynamic workloads in multi-tenant scientific workflow as a service platforms, Future Generation Computer Systems 79 (2018) 739 – 750.
[50] R. Poddar, A. Vishnoi, V. Mann, HAVEN: Holistic load balancing and auto scaling in the cloud, in: Proceedings of 7th IEEE International Conference on Communication Systems and Networks (COMSNETS), IEEE, 2015, pp. 1–8.
[51] J. Simao, N. Rameshan, L. Veiga, Resource-aware scaling of multi-threaded java applications in multi-tenancy scenarios, in: Proceedings of 5th IEEE International Conference on Cloud Computing Technology and Science (CloudCom), Vol. 1, 2013, pp. 445–451.
[52] B. P. Rimal, M. Maier, Workflow scheduling in multi-tenant cloud computing environments, IEEE Transactions on Parallel and Distributed Systems 28 (1) (2017) 290–304.
[53] L. Suresh, P. Bodik, I. Menache, M. Canini, F. Ciucu, Distributed resource management across process boundaries, in: Proceedings of ACM International Symposium on Cloud Computing, ACM, 2017, pp. 611–623.
[54] J. Zhu, B. Gao, Z. Wang, B. Reinwald, C. Guo, X. Li, W. Sun, A dynamic resource allocation algorithm for Database-as-a-Service, in: Proceedings of IEEE International Conference on Web Services (ICWS), IEEE, 2011, pp. 564–571.
[55] W. Lang, S. Shankar, J. M. Patel, A. Kalhan, Towards multi-tenant performance SLOs, in: Pro-

ceedings of 28th IEEE International Conference on Data engineering, IEEE, 2012, pp. 702–713.
[56] S. Jamalian, H. Rajaei, Asets: A sdn empowered task scheduling system for hpcaas on the cloud, in: Proceedings of IEEE International Conference on Cloud Engineering (IC2E), IEEE, 2015, pp. 329–334.
[57] A. Gohad, K. Ponnalagu, N. C. Narendra, Model driven provisioning in multi-tenant clouds, in: Proceedings of IEEE SRII Global Conference (SRII), IEEE, 2012, pp. 11–20.
[58] T. Ritter, B. Mitschang, C. Mega, Dynamic provisioning of system topologies in the cloud, in: Enterprise Interoperability V, Springer, 2012, pp. 391–401.
[59] M. R. Rahman, I. Gupta, A. Kapoor, H. Ding, OPTiC: Opportunistic graph processing in multi-tenant clusters, in: Proceedings of IEEE International Conference on Cloud Engineering (IC2E), IEEE, 2018, pp. 113–123.
[60] X. Ding, R. Luo, M. Hui, A multi-tenant oriented customizable compensation mechanism for workflows, in: Proceedings of 3rd IEEE International Conference on Broadband Network and Multimedia Technology (IC-BNMT), IEEE, 2010, pp. 951–955.
[61] D. Sengupta, A. Goswami, K. Schwan, K. Pallavi, Scheduling multi-tenant cloud workloads on accelerator-based systems, in: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, IEEE Press, 2014, pp. 513–524.
[62] Rangavittala, Sanjay, S. Salvi, Enhanced multi-tenant architecture for DaaS, PaaS, IaaS and SaaS in Edu-Cloud: Simplifying the service provisioning in Edu-Cloud by multi-tenant architecture, in: Proceedings of 6th ACM International Conference on Computer and Communication Technology, ACM, 2015, pp. 51–56.
[63] N. Jain, Lakshmi, PCOS: Prescient cloud I/O scheduler for workload consolidation and performance, in: Proceedings of International Conference on Cloud Computing and Big Data (CCBD), IEEE, 2015, pp. 145–152.
[64] J. Duan, Y. Yang, A load balancing and multi-tenancy oriented data center virtualization framework, IEEE Transactions on Parallel and Distributed Systems 28 (8) (2017) 2131–2144.
[65] R. Jia, J. Grundy, Y. Yang, J. Keung, H. Li, Providing fairer resource allocation for multi-tenant cloud-based systems, in: Proceedings of 7th IEEE International Conference on Cloud Computing Technology and Science (CloudCom), IEEE, 2015, pp. 306–313.
[66] E. P. Neto, G. Callou, F. Aires, An algorithm to optimise the load distribution of fog environments, in: Proceedings of IEEE International Conference on Systems, Man, and Cybernetics (SMC), IEEE, 2017, pp. 1292–1297.
[67] A. Shukla, Y. Simmhan, Model-driven scheduling for distributed stream processing systems, Journal of Parallel and Distributed Computing 117 (2018) 98 – 114.
[68] P. Nghiem, S. Figueira, Towards efficient resource provisioning in MapReduce, Journal of Parallel and Distributed Computing 95 (2016) 29 – 41.
[69] G. Rosinosky, S. Youcef, F. Charoy, An efficient approach for multi-tenant elastic business processes management in cloud computing environment, in: Proceedings of 9th IEEE International Conference on Cloud Computing (CLOUD), IEEE, 2016, pp. 311–318.
[70] C. Momm, W. Theilmann, A combined workload planning approach for multi-tenant business applications, in: Proceedings of 35th IEEE International Conference Workshops on Computer Software and Applications Conference Workshops (COMPSACW), IEEE, 2011, pp. 255–260.
[71] A. Dalvandi, M. Gurusamy, K. C. Chua, Application scheduling, placement, and routing for power efficiency in cloud data centers, IEEE Transactions on Parallel and Distributed Systems 28 (4) (2017) 947–960.
[72] M. Rafique, S. Cadambi, K. Rao, A. R. Butt, S. Chakradhar, Symphony: A scheduler for client-server applications on coprocessor-based heterogeneous clusters, in: Proceedings of IEEE International Conference on Cluster Computing (CLUSTER), IEEE, 2011, pp. 353–362.
[73] B. Primas, P. Garraghan, K. Djemame, N. Shakhlevich, Resource boxing: converting realistic cloud task utilization patterns for theoretical scheduling, in: Proceedings of 9th IEEE/ACM 9th International Conference on Utility and Cloud Computing (UCC), IEEE, 2016, pp. 138–147.
[74] J. Mace, P. Bodik, M. Musuvathi, R. Fonseca, K. Varadarajan, 2DFQ: Two-dimensional fair queuing for multi-tenant cloud services, in: Proceedings of ACM International Conference on Special Interest Group on Data Communication (SIGCOMM), ACM, 2016, pp. 144–159.
[75] J. Li, C. Pu, Y. Chen, V. Talwar, D. Milojicic, Improving preemptive scheduling with application-transparent checkpointing in shared clusters, in: Proceedings of 16th ACM Middleware Conference, ACM, 2015, pp. 222–234.
[76] S. Chinnathambi, A. Santhanam, J. Rajarathinam, Senthilkumar, Scheduling and checkpointing optimization algorithm for byzantine fault tolerance in cloud clusters, Cluster Computing 22 (6)

(2018) 14637–14650.
[77] Z. Li, L. Wang, Y. Zhang, T. Truong-Huu, E. S. Lim, P. M. Mohan, S. Chen, S. Ren, M. Gurusamy, Z. Qin, Integrated QoS-aware resource provisioning for parallel and distributed applications, in: Proceedings of 19th IEEE International Symposium on Distributed Simulation and Real Time Applications, IEEE Press, 2015, pp. 171–178.
[78] N. Jain, Lakshmi, Pridyn: enabling differentiated i/o services in cloud using dynamic priorities, IEEE Transactions on Services Computing 8 (2) (2015) 212–224.
[79] A. Azeez, S. Perera, D. Gamage, R. Linton, P. Siriwardana, D. Leelaratne, S. Weerawarana, P. Fremantle, Multi-tenant SOA middleware for cloud computing, in: Proceedings of 3rd IEEE International Conference on Cloud computing (cloud), IEEE, 2010, pp. 458–465.
[80] J. den Haan, Multi-tenancy and Model Driven Engineering, necessary assets of a Platform-as-a-Service, http://www.theenterprisearchitect.eu/blog/2010/04/27/multi-tenancy-and-model-driven-engineering-necessary-assets-of-a-platform-as-a-service/.
[81] S. Jamalian, H. Rajaei, Data-intensive hpc tasks scheduling with sdn to enable hpc-as-a-service, in: Proceedings of 8th IEEE International Conference on Cloud Computing, IEEE, 2015, pp. 596–603.
[82] T. Kwok, A. Mohindra, Resource calculations with constraints, and placement of tenants and instances for multi-tenant SaaS applications, in: Proceedings of International Conference on Service-Oriented Computing (ICSOC), Springer, 2008, pp. 633–648.
[83] L. Cui, T. Zhang, G. Xu, D. Yuan, A scheduling algorithm for multi-tenants instance-intensive workflows, Applied Mathematics & Information Sciences 7 (1) (2013) 99–105.
[84] B. Nagesh, Guruprasad, Resource provisioning techniques in cloud computing environment-a survey, IJRCCT 3 (3) (2014) 395–401.
[85] W.-T. Tsai, Q. Shao, X. Sun, Real-time service-oriented cloud computing, in: Proceedings of 6th World Congress on Services, 2010, pp. 473–478.
[86] M. Almorsy, J. Grundy, A. Ibrahim, Adaptable, model-driven security engineering for SaaS cloud-based applications, Automated Software Engineering (2013) 1–38.
[87] A. B. Soomro, N. Salleh, E. Mendes, J. Grundy, G. Burch, A. Nordin, The effect of software engineers' personality traits on team climate and performance: A systematic literature review, Information and Software Technology 73 (2016) 52–65.