An Efficient Formal Modeling Framework for Hybrid Cloud-Fog Systems

An Efficient Formal Modeling Framework for Hybrid Cloud-fog Systems

Xiao Chen, Jie Ding, Zhenyu Lu and Tianming Zhan

Abstract—Advanced communication technologies (e.g., 5G) probably elicit a complete change of network and its applications. For example, a growing number of services begin shifting from central clouds to vast mobile devices, as the hybrid use of cloud and fog computing technologies can provide enhanced quality of service and efficient utilization of resources. However, to design such complex hybrid cloud-fog (HCF) systems, it remains a challenge to implement time-consuming modeling and inefficient evaluation in its early design stage based on the conventional simulation or practical experimentation. Therefore, how to reduce design cost and improve development efficiency becomes a crucial issue in the process of designing large-scale HCF systems. To address the issue, this paper proposes a novel modeling framework for large-scale HCF systems based on a high-level formal language, i.e. performance evaluation process algebra (PEPA). Toward the key components of an HCF system, the proposed framework includes three crucial model prototypes: *compositional architecture model*, *abstract communication model* and *scheduling model*. Moreover, the scheduling model is designed with a novel smart scheduling scheme that integrates two atomic scheduling algorithms and a decision module to make an efficient algorithm selection. The smart scheduling algorithm can well adapt the HCF systems by yielding stable and fast response to end-users, particularly under dynamical system conditions. Finally, the framework is the first research achieving the full potential of formal methods to implement industry-level modeling and evaluation.

Index Terms—Formal Modeling, Hybrid Cloud-fog Systems, Scheduling, PEPA, Fluid Flow Approximaton.

1 Introduction

Since 2019, 5G communication networks have begun to be commercially deployed for mobile services. In the near future, not only mobile devices but also other appliances, such as vehicles or industrial equipment, will join the 5G network for more convenient and extensive services. Taking Internet of vehicles for example, with 5G or even future 6G connections [1], the vehicles can provide a greater variety of services through the networks [2]. As most smart vehicles integrate more powerful devices such as high-performance CPUs, massive storage and various sensors (e.g., video cameras), they can behave as not only service consumers but also producers. Therefore, many conventional service providers, particularly cloud-service merchants, might consider new solutions to combine their services with a mobile platform such as the vehicular cloud [20] and vehicular fog [4] platforms. In fact, fog computing technologies have been developed to offload data and computation from central clouds to a group of virtual servers formed by mobile devices [5], [6]. The fog computing offers supplemented physical resources (e.g., computation, network, and storage) closer to end users for better services [7]. The conventional cloud service providers have to face the challenge of new fog computing-based service platforms. As a result, a growing number of applications will be built on such a hybrid platform by merging cloud and fog systems, i.e., the hybrid cloud-fog (HCF) system. Our recent research [8] proposed a blockchain-based trust management system that conducts trust evaluation and preserves trust data based on all distributed participating nodes. Comparing to the conventional cloud-centralised system, the blockchain-based system needs to join all distributed end nodes together to perform trust evaluation and blockchain consensus operations by aggregating nodes' resources into a virtual fog server [9]. This design relies on an HCF system framework that provides efficient and scalable services without a cost increase of central cloud resources.

To shift services away from clouds, new systems need be designed in terms of specific application requirements. The large-scale system design usually suffers from a time-consuming process for the preliminary design, as well as design evaluation and several rounds of design refinement. To facilitate system design work, this paper presents a reasonable solution for the key issue, i.e., how to generate efficient modeling and reliable evaluation to facilitate the preliminary design. To address these issues, we propose a novel modeling framework to efficiently build models and generate performance analysis for the HCF-based systems.

The framework uses a novel formal modeling language, i.e., performance evaluation process algebra (PEPA) [10], to model all system designs. PEPA is a high-level formal modeling language for stochastic models, which describes a concurrent system as an interaction of the components engaging in activities. In contrast to the conventional process algebras, the activity duration of PEPA is assumed to be exponentially distributed. The PEPA language has a few

- *X. Chen jointly worked with Jiangsu Key Laboratory of Meteorological Observation and Information Processing, Nanjing University of Information Science and Technology, Nanjing, China; and is with School of Informatics, University of Edinburgh, Edinburgh, EH8 9AB, UK. Email: xiao.chen@ed.ac.uk*
- *J. Ding is with China Institute of FTZ Supply Chain, Shanghai Maritime University, Shanghai 201306, China. Email: jding@shmtu.edu.cn*
- *Z. Lu is with School of Artificial Intelligence, Nanjing University of Information Science and Technology, Nanjing, China. Email: luzhenyu76@163.com*
- *T. Zhan is with School of Information and Engineering, Nanjing Audit University, Nanjing, China. Email: ztm@nau.edu.cn*

combinators, and its structured operational semantics can be represented as:

$$S ::= (\alpha, r).S \mid S + S \mid C_S,$$
$$P ::= P \underset{L}{\bowtie} P \mid P/L \mid C.$$

where S denotes a sequential component and P denotes a model component that executes in parallel. Here, C stands for a constant that denotes either a sequential component or a model component as introduced by a definition and C_S stands for constants that denote sequential components. The effect of this syntactic separation between these types of constants is to constrain legal PEPA components to be cooperations of sequential processes. More details of PEPA syntax can be obtained in [10].

PEPA is selected as our main tool due to its key features, which are not available in other well-known performance modeling paradigms (e.g., stochastic Petri-nets, queueing theory and discrete event simulation). The most important features of PEPA are: *compositionality*, the ability to efficiently model the system as the interaction of subsystems or components which clearly defines the architecture of a comprehensive system as well as the interactive system behaviors; *formality*, defining all terms in the PEPA language with precise meaning, which can yield accurate formal model specification of the system; *abstraction*, the ability to build up complex system models from detailed components and to separate the details into a set of submodels that can be combined with the abstract model for analysis. These features allow PEPA to efficient build models for large-scale systems and yield reliable numerical analysis based on stochastic simulation and fluid-flow approximation [11], compared to time-consuming discrete event simulation [18], [19], [24] and practical experiments [28]. The high-efficiency modeling and analysis process based on PEPA models can reduce the cost and time span of the preliminary design.

With the PEPA language, we propose the modeling framework for HCF systems by considering three detailed HCF-design issues. First, to generate a complete model of a specific HCF system, we need to define the system architecture that consists of all subsystems/components and their internal behaviors or external interactions. Based on the feature of compositionality, PEPA can effectively define the compositional structure of these components, including their behaviors and interactions. In this research, we adopt the Openstack cloud platform for the top cloud-layer of the HCF system. A compositional model is developed to demonstrate how a cloud-layer system model is efficiently built in PEPA language. This compositional model can be considered a universal prototype for various HCF-based systems which integrate an Openstack-like cloud platform. In addition, we also introduce stochastic analysis based on PEPA models, in which all parameters are observed directly from running an Openstack platform.

Second, with a multicomponent design, most HCF-based systems usually adopt different communication protocols between their subsystems (e.g., cloud and fog subsystems) or among components of the same subsystem [29]. Thus, it is inefficient to include all protocol-based communication behaviors in a single model, which makes the model overly complicated to define and maintain. Therefore, this research uses abstraction modeling to demonstrate a communication model prototype that defines the offloading process of the HCP system. The abstraction communication model moves protocol behaviors from a high-level offloading model to several submodels that can be analyzed together with the high-level model. The abstraction modeling can enable efficient model construction and maintenance without compromising its correctness.

Third, to obtain better offloading performance, HCF systems usually use a smart scheduling mechanism to adapt to changing system conditions. The smart scheduling scheme is proposed by using multidynamic scheduling algorithms and a decision model to achieve the optimal selection of scheduling algorithm. The scheduling algorithms and decision model are usually investigated with simulation tools due to their complex operations. To improve the modeling ability of formal language, this part aims to build a prototype for scheduling models and the corresponding decision model. Finally, we demonstrate this scheduling model prototype and prove its correctness by comparing its analysis results with those of equivalent simulation analysis. The main contributions can be highlighted as follows:

- The critical contribution of this work is to demonstrate an efficient modeling framework using a novel formal language (i.e. PEPA) to achieve comprehensive performance modeling and analysis towards complex systems (e.g., HCF systems). The framework includes three core model prototypes. The first one is the *compositional model* prototype representing the compositionality feature of PEPA language, which can be efficiently used to define system components and their interactions in a clear and layered structure.
- The second prototype is the *abstraction model* that is usually used to aggregate unobserved system components and behaviors for efficient model creation without losing analysis accuracy, which indicates the abstraction feature of PEPA models. To introduce this model prototype, we use PEPA to model system communication of offloading process based on an HCF system scenario and also introduce an efficient numerical analysis method (i.e. fluid flow analysis) through presenting a group of results.
- The last *scheduling model* presents a novel design of the smart scheduling scheme by integrating two atomic scheduling algorithms to adapt to changing system conditions. This design combines a novel time-preemptive scheduling algorithm with the classic queue-preemptive algorithm and provides a smart decision support module to aid the selection of two atomic scheduling algorithms by observing the changing system environments to improve system performance and quality of service. Besides, we present a PEPA prototype for modeling scheduling algorithm and the related numerical analysis, which can be considered an alternative method of performance modeling and evaluation besides conventional simulation approaches.

To the best of our knowledge, this is the first research that investigates a universal framework based on the formal

TABLE 1
Summary of Related Work

Ref. No.	Issue Domain	Model Techniques	Formality/Composition	Diverse Conditions
[14]	Architecture	SPN	Yes / No	No
[15]	Architecture	SRN	Yes / No	No
[16], [17]	Architecture	Queueing	No / Yes	No
[5], [6], [18], [19]	Offloading	Simulation	No / No	No
[20], [21], [22], [23]	Offloading	Queueing	No / Yes	No
[24], [25], [26], [27]	Scheduling	Simulation	No / Yes	No
This Work	**All Issues**	**PEPA**	**Yes / Yes**	**Yes**

modeling method which can effectively promote the use of formal modeling in industry-level design, since the proposed framework is able to meet more complex modeling demands with those basic model prototypes. The rest of this paper will provide the details of those three model prototypes and related performance analysis approaches.

2 Related Work

Cloud architecture modeling has been implemented through various techniques, such as stochastic Petri nets (SPNs) and stochastic reward nets (SRNs). Silva [14] and Ghosh [15] apply SPNs and SRNs to model mobile cloud and cloud computing systems, respectively. SPNs/SRNs support a formal representation of system architecture and behaviors; nevertheless, their syntax confines the representation for compositional system architectures, which entails an inferior compositionality in model representation. In addition, another modeling technique, i.e., queueing theory, can be used for similar performance modeling and analysis. Chang [16] developed a novel approximate analytical model to conduct performance evaluation in IaaS clouds with a queueing model, and Li [17] also uses the M/M/c/r queueing system to propose an approximate analytical model for cloud computing centers by representing the rendering service platform as a multistation, multiserver system. These queueing models can efficiently represent a compositional architecture; however, they present the drawback of providing such compositional models using a formalized definition. In contrast to these modeling techniques, PEPA can provide both formality and compositionality in terms of its features stated in Section 1.

Regarding the communication models, Su's research [18] presents a device-to-device-based communication framework that is used for content delivery by employing resources of parked vehicles. Moreover, Chen [19] proposes a novel hybrid task offloading framework in fog computing so that end users have the flexibility of selecting among multiple options for task execution, including local mobile execution, device-to-device (D2D) offloading, and cloud offloading. Both Su and Chen use simulation for performance evaluation. Most related research (e.g., [5], [6]) also utilizes simulation as the main tool for performance modeling and numerical analysis, though the simulation is informal and quite time-consuming, particularly for large-scale models. To improve modeling efficiency, some other researchers [20] have explored an offloading solution by enabling it for real-time traffic management in fog-based Internet of Vehicles (IoV). In this research, parked and moving vehicle-based fog nodes are modeled as an M/M/1 queue. Similarly, Wu [23] builds partial and full offloading models based on a modulated M/M/1 queue in a two-phase (fast and slow) Markov random environment. Moreover, Mehmeti [21] proposed a queueing analytic model for delayed offloading to analyze mean delay, offloading efficiency, and other metrics of interest, and Cheng [22] investigates WiFi offloading performance by establishing an explicit relation between offloading effectiveness and average service delay with an M/G/1/K queueing model in order to minimize the trade-off between those two aspects. As we noted before, a queueing model can represent the compositional architectures of systems but cannot formalize the model definition. Thus, this work defines the proposed D2D offloading scheme in a formal high-level PEPA model to achieve both formality and compositionality and adopts fluid-flow approximation to generate numerical analysis.

The scheduling efficiency is an important issue in cloud/fog-based systems, which has been explored in many recent research endeavors. Tan [24] has studied the approach of minimizing the total weighted response time for all tasks by constructing a general model, where the tasks are generated in arbitrary order and times at the mobile devices and are offloaded to servers with delays. In the research, the scheduling issue is investigated in a steady condition without considering its diversity. Other researchers [25] have had a similar goal of minimizing task completion time by designing an efficient task scheduling and resource management strategy through extensive simulation-based studies. However, these researchers neglect the diverse system conditions while using nonformal simulation tools for performance evaluation. Moreover, some other researchers, such as [26] and [27], also overlook the influence on the performance of proposed schemes caused by the changing system conditions. To address this issue, we will focus on developing a novel smart dynamic scheduling scheme to adapt this changeable system environment, which can be integrated with HCF systems to obtain higher performance. Moreover, we have had some initial research (seeing [35]) to model the Join-the-shortest-queue scheduling algorithm, which help us get some experience of modeling scheduling algorithm under fog computing environments. This research will explore a new PEPA-based modeling framework and a novel smart scheduling algorithm for HCF systems.

In summary, the most recent research commonly utilizes either noncompositional (e.g., SPNs and SRNs) or nonformal techniques (e.g., queueing models and simulations) to

construct system models; meanwhile, none of them has considered the realistic system environment (i.e., the system with changing conditions) that can cause unexpected influence on the performance of offloading or scheduling scheme. All related literature is summarized in Table 1 in order to compare key features. Hence, this research aims to facilitate modeling work in system design and proposes a novel formal modeling framework to address several key issues of HCF-based systems by considering a more realistic system environment.

3 Compositional Model of Cloud Systems

This section introduces a PEPA-based compositional model prototype that is suitable for modeling systems with multiple components and layered structure due to PEPA's key feature, i.e., *compositionality*. The benefit of compositional modeling will be demonstrated by modeling a central cloud architecture on the basis of Openstack.

Generally, a central cloud system is joined with a hybrid cloud-fog system, providing powerful computing and huge storage services for mobile users. In an HCF system, the cloud system can be considered a subsystem connecting to a fog-based subsystem and end users. Thus, it is important to accurately model the cloud system to investigate the performance of the entire HCF system. Hence, a PEPA-based compositional architecture model is developed to build cloud system models and generate performance analysis.

3.1 Cloud Model Scenario

To illustrate PEPA-based compositional modeling under a real-world cloud scenario, we use the Openstack platform as our basic scenario to generate the system architecture model and observe running figures as model parameters.

Openstack is a free and open-source cloud computing platform that controls large pools of computation, storage, and networking resources throughout a data center and is managed through a dashboard or via the Openstack API. The architecture of Openstack, shown in Fig. 1, is logically organized as four key nodes (i.e., *Controller*, *Compute*, *Network*, and *Storage*). *Controller*, i.e., *Horizon*, is the Openstack Dashboard which provides administrators and users with access to the cloud resources. *Compute*, i.e., *Nova*, provides a way to provision compute instances and supports the creation of virtual machines (VMs). *Network*, i.e., *Neutron*, manages all networking facets for the Virtual Networking Infrastructure (VNI) and the access layer aspects of the physical networking infrastructure (PNI) in the Openstack environment. *Storage* includes *Cinder* for Openstack Block Storage service, providing volumes to Nova virtual machines, Ironic bare metal hosts, containers and more and *Swift* for Openstack Object Storage, offering scalable storage for users to store and retrieve large amounts of data with a simple API.

The scenario shown in Fig.1 presents the conceptual architecture of Openstack to demonstrate the compositional architecture modeling technique. Some detailed components and their connections are simplified in the scenario. If they are required in the specific research, all of these components and their behaviors can be modeled by following our proposed model prototype.

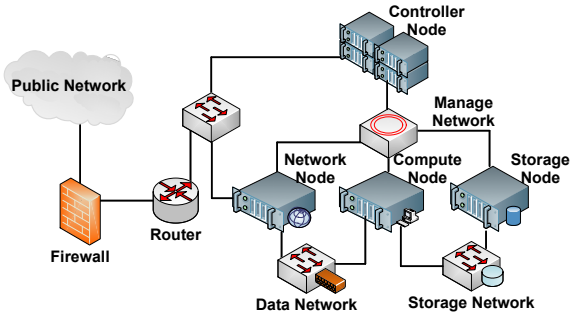


Fig. 1. Physical Architecture of Cloud Deployment

3.2 Cloud-side Architecture Modelling

This subsection will introduce the compositional architecture modeling through defining the VM creation process based on the Openstack scenario.

According to Fig. 2, the *User* component only has interactions with the *Controller*; however, the *Controller* cooperatively interacts with all other nodes (i.e., *Network*, *Compute*, and *Storage*). Moreover, the inner components of each node (e.g., *Bandwidth*, *VM*, *Volume*) usually have limited interactions with other components. For example, *Bandwidth* and *Volume* only interact with *VM* and *Storage*, respectively; and *VM* cooperatively interacts with both *Bandwidth* and *Compute*. It is worthy of mention that volumes in Openstack are block storage devices which are attached to instances (VMs) to enable persistent storage; volumes can be attached to or detached from a running instance or be attached to another instance at any time. Based on these interaction features, the whole system can be modeled in a layered architecture, which is shown in Fig. 3. In the figure, the arrows indicate interactions between components in different layers.

As the *Controller* manages all other key components, its modeling process is used as an example to illustrate the compositional architecture modeling with PEPA language. According to the activity flow of the *Controller* in Fig. 2, the VM generation process can be formally defined with PEPA:

$$
\begin{aligned}
\mathbf{CTN} &\overset{def}{=} (cre\_vm, r_{cre\_vm}).CTN_{req1};\\
CTN_{req1} &\overset{def}{=} (req1_1, r_{req1_1}).CTN_{schd1}\\
&\quad + (req1_2, r_{req1_2}).CTN;\\
CTN_{schd1} &\overset{def}{=} (schd1, r_{schd1}).CTN_{rpc\_cVM};\\
CTN_{rpc\_cVM} &\overset{def}{=} (rpc\_cVM, r_{rpc\_cVM}).CTN_{cre\_net};\\
CTN_{cre\_net} &\overset{def}{=} (cre\_net, r_{cre\_net}).CTN_{rpc\_ip};\\
CTN_{rpc\_ip} &\overset{def}{=} (rpc\_ip, r_{rpc\_ip}).CTN_{image};\\
CTN_{image} &\overset{def}{=} (image_1, r_{image_1}).CTN_{img\_file}\\
&\quad + (image_2, r_{image_2}).CTN_{vif\_plug};\\
CTN_{img\_file} &\overset{def}{=} (img\_file, r_{img\_file}).CTN_{vif\_plug};\\
CTN_{vif\_plug} &\overset{def}{=} (vif\_plug, r_{vif\_plug}).CTN_{up\_state1};\\
CTN_{up\_state1} &\overset{def}{=} (up\_state1, r_{up\_state1}).CTN_{return1};\\
&\cdots\\
CTN_{return3} &\overset{def}{=} (return3, r_{return3}).CTN.
\end{aligned}
$$

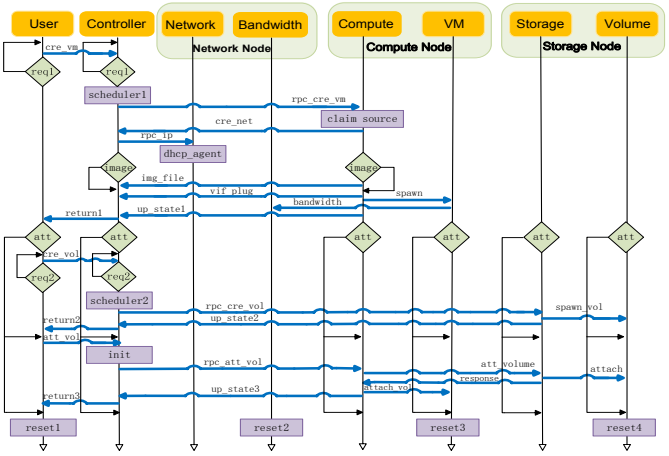


Fig. 2. Virtual Machine Generation Process on the Cloud Layer

The controller component (CTN) performs its first action $create\_vm$ once it accepts a VM generation request from a user. Then, a choice $req1$ is made to validate the request and examine the available resources. Action $schd1$ represents the behavior of selecting a suitable *Compute* node to create a VM. The follow-up action is rpc_cVM, which cooperates with the *Compute* node to spawn a new VM in the hypervisor. Thereafter, the CTN awaits the response of the logic-network creating action (cre_net), and thereby notifies the *Network* node about this change through another action rpc_ip. Thereafter, the CTN decides the execution of img_file action by an $image_1$ action, and then transmits image files (represented as $image\_file$) by cooperating with the *Compute* node; otherwise, it skips this transmitting action ($image\_file$) and proceeds with the following vif_plug action until the completion of network deployment based on the preceding logic-network. Finally, the CTN performs the up_state1 action to obtain the response from the $Compute$ node and notifies the user by the $return1$ action. The initial VM creation is thus finished, and the following process aims to create an extension of VM storage, which is omitted in the model definition.

As the definitions of other components, such as network node (NWN), compute node (CPN) and storage node (STN), have similar formulations, their model details are omitted. The whole system can be defined as the cooperation of these components in a layered structure:

$$
\begin{aligned}
Sys = \; & \underbrace{User[m] \underset{S_1}{\bowtie} \{CTN[n]}_{\textbf{Layer 1}} \\
& \underset{S_2}{\bowtie} \underbrace{[(Network[k_1] \,||\, Compute[k_2] \underset{S_3}{\bowtie} Storage[k_3])}_{\textbf{Layer 2}} \\
& \underset{S_4}{\bowtie} \underbrace{(BandWidth[p_1] \underset{S_5}{\bowtie} VM[p_2] \,||\, Volume[p_3])]\}}_{\textbf{Layer 3}}.
\end{aligned}
$$

The Sys component represents the formal definition of a complete cloud system by joining all components and specifying their cooperative actions in the sets ($S_{1,\cdots,5}$).

According to Fig. 3, the three-layer architecture of the cloud system is clearly depicted by the Sys component of PEPA models. Moreover, a set of components and their

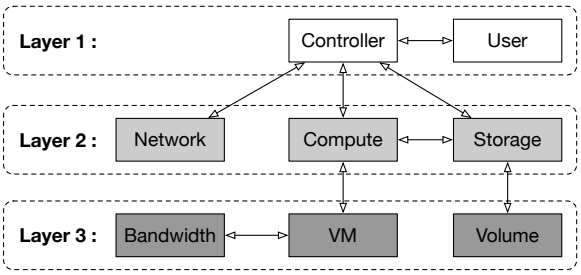


Fig. 3. Layered Architecture of Cloud Deployment

interactions, shown in Fig. 2, are specified in other model components (e.g., *CTN*). In comparison with graph-based modeling languages (e.g., SPN or Queueing Model), the PEPA-based model exhibits its superior compositionality in representing system architecture without generating over-complicated and indistinguishable model representation. Hence, it is particularly suitable to model large-scale systems with multiple components and complex interactions (e.g., HCF systems).

3.3 Performance Evaluation with Stochastic Analysis

PEPA models include information about the duration of activities and their relative probabilities through a racing policy. It is possible to generate a corresponding continuous time Markov chain (CTMC) by elaborating the model against the structured operational semantics of the PEPA language, and it also supports the related analysis technique based on CTMCs. In this subsection, PEPA-based stochastic simulation will be demonstrated with performance analysis on PEPA-based compositional architecture models. Experimental parameters are observed from running a small-scale Openstack cloud platform on cluster servers. All figures are observed from timestamping each operation defined in the model. Performance analysis is conducted through evaluating the response time and resource utilization, which are obtained from stochastic simulation on CTMCs derived from PEPA models.

The stochastic simulation is an optional analysis technique supported by PEPA in which realizations of the random variables are generated and inserted into a model of the system. Outputs of the model are recorded, and then the process is repeated with a new set of random values. These steps are repeated until a sufficient amount of data is gathered. In the end, the distribution of the outputs shows the most likely estimates, as well as a frame of expectations regarding what ranges of values the variables are more or less likely to fall within. This section presents the analysis results based on a given stochastic simulation algorithm, i.e., the Gillespie algorithm.

Figs. 4 and 5 indicate the completion probability of creating VMs based on the two types of conditions (varying number of users or CTNs). In Fig. 4, with the growth of users from 80 to 120, the probability of completing users' requests is reduced from 100% to approximately 50%. This means that the current capacity of the platform can provide a rapid response to nearly half of users if the user number is 120. This analysis can predict the probability of completing different numbers of users based on the current server

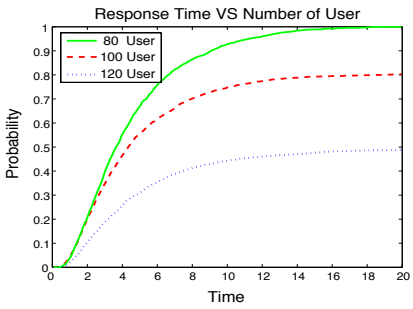


Fig. 4. Response Time Varying against the Number of Users

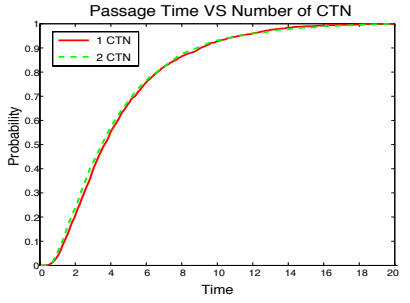


Fig. 5. Response Time Varying against the Number of CTNs

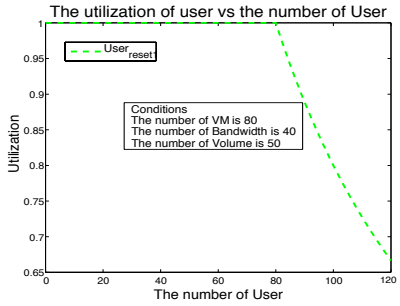


Fig. 6. Utilization of $User_{reset}$ Varying against the Number of Users (1)

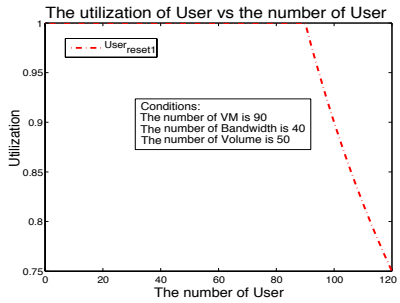


Fig. 7. Utilization of $User_{reset}$ Varying against the Number of Users (2)

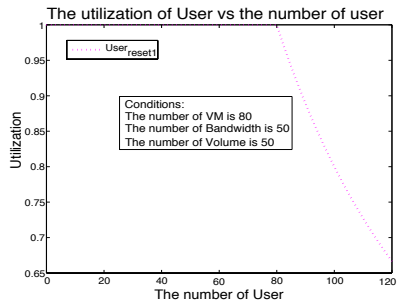


Fig. 8. Utilization of $User_{reset}$ Varying against the Number of Users (3)

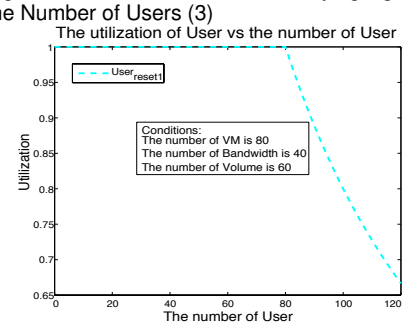


Fig. 9. Utilization of $User_{reset}$ Varying against the Number of Users (4)

resources (e.g., a fixed number of CTN resources). Similarly, we can also consider changing the number of CTN resources for a fixed number of users. Taking Fig. 5 for example, all users can be fully served (i.e., the probability is 100%) with 2 CTNs, and 100% probability remains when we change two CTNs to only one. This means that adequate CTN resources are provided to users. In other words, in comparison with the changing of CTNs, varying the number of users can generate more serious performance change. Hence, the analysis of completion probability can help achieve both prediction of user-request completion and elucidation of the dominant factor for performance variation.

Figs. 6, 7, 8, and 9 indicate the utilization of a specific type of physical resource under different numbers and amounts of VMs, bandwidth, volume and users, which aims to represent the trade-off among these four factors. In the figures, the decrease of utilization indicates the loss of performance, and so the inflection point of utilization indicates the maximum number of users than can be served with high quality of service (QoS) under given system resources (e.g., the numbers of VMs, volumes and amount of bandwidth). Furthermore, by comparing these figures, we can observe which factor can exert more influence upon the resource utilization. For example, Fig. 6 indicates that the maximum number of users is approximately 80 before a decrease in utilization under the given resources (i.e., 80 VMs, bandwidth of 40 and 50 volumes) in the component $User_{reset1}$. While changing one of the given resources, respectively, such as increasing VMs from Fig. 6 to 7, increasing bandwidth from Fig. 6 to 8 and increasing volumes from Fig. 6 to 9, we can find that the maximum number of users increases from 80 to 90 by comparing Fig. 6 and 7. Hence, the number of VMs yields more influence on the resource utilization with the growth of users.

In conclusion, PEPA-based models can provide effective compositional modeling and support stochastic simulation-based analysis. This section demonstrates the method of building compositional system models in PEPA and yielding performance analysis with stochastic simulation.

4 Abstraction Model of Communication Network

As the communication network is a crucial part of HCF, this section aims to develop a prototype HCF communication network with the PEPA-based abstraction modeling. The abstraction communication model is built to represent message passing during offloading operations in a high-level PEPA model by aggregating some protocol-level details. The abstraction modeling allows us to focus on the core behaviors that need to be observed, rather than those irrelevant model details, which can improve modeling efficiency without sacrificing accuracy. The aggregated details can be decomposed when they are required for observation. Since PEPA provides effective support of abstraction and decomposition modeling, the following sections will introduce how the abstraction modeling is used to represent the offloading-based communication network.

4.1 Communication Model Scenario

The network architecture of an HCF system is represented in a layered architecture as shown in Fig. 10. The top cloud-layer users often suffer from extra and high communication cost via the interlayer due to the increasing amount of service on the mobile clients. Therefore, fog-side computing service is applied to address this problem by offloading some message communications from base stations to mobile nodes with direct device-to-device (D2D) connections.

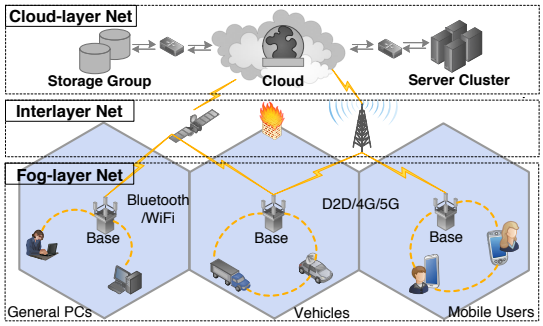


Fig. 10. Communication Deployment of Hybrid Cloud-Fog System

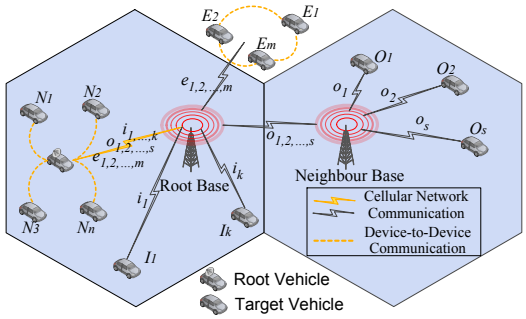


Fig. 11. Wireless Communication Offloading Deployment on Fog Layer

In this section, a vehicular fog computing scenario is considered due to its mobility feature. D2D communication is applied between two nearby vehicles, which can support the offloading of message exchange from passing base stations to direct D2D exchange. Fig. 11 depicts this process based on a vehicular fog scenario. Fig. 11 indicates a high-level communication model between a root vehicle and other vehicles in different areas. $N_{1,2,3,4}$ represents a group of vehicles near the root vehicle, which can create D2D data transmission instead of cellular communication; $I_{1,2}$ and $O_{1,2,3}$ are vehicles located in the root base-station coverage and its neighbor, respectively, but they all move out of the D2D coverage. Thus, the communication to the root vehicle requires the support of base stations. $E_{1,2,3}$ vehicles operate jointly to generate beamforming to enhance communication signals from the edge of cellular coverage. In this situation, the vehicles in Group E are required to share their data to be sent with other vehicles to form a data sequence through D2D; thereafter, they send the data sequence cooperatively in the manner of beamforming.

4.2 Fog-side Communication Modeling

In this section, a formal communication model is defined with PEPA; moreover, a novel fluid-flow approximation is applied for model analysis to address the state-space explosion problem in solving its underlying CTMCs. Based on Fig. 11, the D2D-offloading model can be defined as:

Network Model 1: Offloading Model

$$\boldsymbol{V_N} \stackrel{\text{def}}{=} (req\_N, r_{reqN}).(v\_encrypt, r_{venc}).V_{N'};$$
$$V_{N'} \stackrel{\text{def}}{=} (txD2D, r_{D2D}).(s\_decrypt, \tau).(process, \tau).V_N;$$
$$\boldsymbol{V_E} \stackrel{\text{def}}{=} (req\_E, r_{reqE}).(v\_encrypt, r_{venc}).V_{E'};$$
$$V_{E'} \stackrel{\text{def}}{=} (txD2D, r_{D2D}).(txD2D, r_{D2D}).V_{E''};$$
$$V_{E''} \stackrel{\text{def}}{=} (tx\_cell\_root, \tau).(s\_decrypt, \tau).(process, \tau).V_E;$$
$$\boldsymbol{V_I} \stackrel{\text{def}}{=} (req\_I, r_{reqI}).(v\_encrypt, r_{venc}).(tx\_cell\_root, \tau).V_{I'};$$
$$V_{I'} \stackrel{\text{def}}{=} (s\_decrypt, \tau).(process, \tau).V_I;$$
$$\boldsymbol{V_O} \stackrel{\text{def}}{=} (req\_O, r_{reqO}).(v\_encrypt, r_{venc}).(tx\_cell\_neig, \tau).V_{O'};$$
$$V_{O'} \stackrel{\text{def}}{=} (tx\_cell\_root, \tau)(s\_decrypt, \tau).(process, \tau).V_O;$$
$$\boldsymbol{Root} \stackrel{\text{def}}{=} (tx\_cell\_root,\ r_{txcr}).Root;$$
$$\boldsymbol{Neig} \stackrel{\text{def}}{=} (tx\_cell\_neig,\ r_{txcn}).Neig;$$
$$\boldsymbol{Server} \stackrel{\text{def}}{=} (process,\ r_{pros}).Server + (s\_encrypt, r_{senc}).Server + (s\_decrypt, r_{sdec}).Server.$$

The above PEPA model defines communicating behaviors of each group of vehicles. $Server$ component represents the root vehicle behaving as a data receiver and processor. V_N defines the behaviours of vehicles near a root vehicle, which include data encryption and data transmission through a D2D channel modeled with an action $txD2D$. After that, servers decrypt and process the data, which is modeled as two shared actions (i.e. $s\_decrypt$ and $process$) between components V_N and $Server$. Vehicles, defined as V_E, take similar actions to encrypt and transmit data via two rounds of D2D communication and then utilize a beamforming channel to connect the root base station (modeled with an action tx_cell_root) before sending the data to servers. Furthermore, vehicles, located far from the root vehicle, entirely depend on the relay of base stations, either one-hop cellular transmission (e.g., V_I) or multihop (e.g., V_O), which can be modeled as a series of transmitting actions (e.g., tx_cell_root or tx_cell_neig) at base stations before arriving the server vehicle for data processing.

$Root$ and $Neig$ define two different base stations based on the model scheme in Fig. 11. Finally, the entire fog-side communication system can be represented as the cooperation of these components by specifying their cooperative actions. The system model can be formalized as follows:

$$\boldsymbol{Sys} \stackrel{\text{def}}{=} (Root[t_{root}] \underset{\{tx\_cell\_root\}}{\bowtie} (V_N[n]||V_E[m]||V_I[k]||V_O[s]) \underset{\{tx\_cell\_neig\}}{\bowtie} Neig[t_{neig}]) \underset{\{s\_decrypt, process\}}{\bowtie} Server[t_{server}].$$

Additionally, a nonoffloading scheme is defined for comparison with the previous offloading scheme. The nonoffloading scheme is modeled by replacing all D2D communications between vehicles with cellular network communications. As the D2D communication is only used for vehicles near (V_N) and those located on the edge (V_E), the nonoffloading model only needs to change all D2D ($txD2D$) to cellular communications either through the root base station (tx_cell_root) or both types of base stations (tx_cell_root and tx_cell_neig). This is represented by a *Choice* operator in PEPA the *Network Model 2*. All other components are consistent with the offloading model. Thus, the nonoffloading model can be altered as:

Network Model 2: Nonoffloading Model

$$\boldsymbol{V_N} \stackrel{\text{def}}{=} (req\_N, r_{reqN}).(v\_encrypt, r_{venc}).V_{N'};$$
$$V_{N'} \stackrel{\text{def}}{=} (tx\_cell\_root, \tau).(s\_decrypt, \tau).(process, \tau).V_N;$$
$$\boldsymbol{V_E} \stackrel{\text{def}}{=} (req\_E, r_{reqE}).(v\_encrypt, r_{venc}).V_{E'};$$

$$V_{E'} \stackrel{\text{def}}{=} (tx\_cell\_root, \tau).(s\_decrypt, \tau).(process, \tau).V_E + (tx\_cell\_neig,\ \tau).(tx\_cell\_root, \tau).V_{E''};$$
$$V_{E''} \stackrel{\text{def}}{=} (s\_decrypt, \tau).(process, \tau).V_E.$$

Similar to the offloading model, the above PEPA model defines vehicle behaviors of data encryption, transmission and processing via base stations, in which the D2D-based transmission is not performed in terms of the non-offloading scheme. Similar to the offloading model, the above PEPA model defines vehicle behaviors of data encryption, transmission and processing via base stations, in which the D2D-based transmission is not performed in terms of the non-offloading scheme. Regarding the abstraction modeling, since we do not observe protocol behaviors, the details of the protocol are abstracted within a single action, i.e., $txD2D$ or tx_cell_root, and the action rate is calculated from all rates of detailed actions in the protocol. For a high-level model, protocol actions are hidden from abstraction; however, these hidden actions can be obtained by decomposing the abstract actions (e.g., $txD2D$ and tx_cell_root), once they are required to be observed.

The following subsection demonstrates a novel analysis approach, i.e., the fluid-flow analysis based on the abstraction model. All action rates are specified in a parameter table, i.e., Table 2.

4.3 Performance Evaluation with Fluid-flow Analysis

PEPA language offers a compositional function for creating models of large-scale systems. Meanwhile, a novel performance analysis technique, fluid-flow approximation, is provided for large-scale models using PEPA [11]. The fluid-flow approximation avoids the state space explosion issue while solving the underlying Markov chain of PEPA models.

The fluid-flow analysis is conducted on the basis of vector form. The system is inherently discrete, with the entries within the numerical vector form always being nonnegative. With a change in the system state, the numerical vector form is incremented or decremented in steps of one. When each component type in the model is replicated a large number of times, these steps are relatively small. Thus, we can approximate the movement between states as continuous, rather than occurring in discontinuous jumps. The objective of the fluid-flow approximation is to replace the derivative graph of the PEPA model with a continuous model using a set of ordinary differential equations (ODEs).

In the fluid-flow approximation, we need to specify the *exit activity* and *entry activity* of the local derivative of a sequential component. An activity (α, r) is an *exit activity* of D if D enables (α, r), such as $D \xrightarrow{(\alpha,r)} D'$. The set of exit activities of D is denoted by $Ex(D)$. The set of local derivatives for an exit activity (α, r) is denoted by $Ex(\alpha, r)$. Similarly, an activity (β, s) is an *entry activity* if a derivative D' enables (β, s), such as $D' \xrightarrow{(\beta,s)} D$. $En(D)$ denotes the set of entry activities of D.

After specifying the concepts of the *exit activity* and *entry activity*, the movements of the numerical state vector of the PEPA model are represented with these concepts. Here, we define $v_{i_j}(t) = N(C_{i_j}, t)$ for the jth entry of the ith subvector at time t; $N(C_{i_j}, t)$ denotes the number of instances of the jth local derivative of sequential component C_i. In a very short time interval δt, the change of the vector entry $v_{i_j}(t)$ can be expressed as:

$$\begin{aligned} N(C_{i_j}, t+\delta t) - N(C_{i_j}, t) = \\ \underbrace{- \sum_{(\alpha,r)\in Ex(C_{i_j})} r \times \min_{C_{k_l}\in Ex(\alpha,r)} (N(C_{k_l}, t))\, \delta t}_{exit\ activities} \\ \underbrace{+ \sum_{(\alpha,r)\in En(C_{i_j})} r \times \min_{C_{k_l}\in Ex(\alpha,r)} (N(C_{k_l}, t))\, \delta t;}_{entry\ activities} \end{aligned} \tag{1}$$

$$\begin{aligned} \frac{dN(C_{i_j}, t)}{dt} = \lim_{\delta t\to 0} \frac{N(C_{i_j}, t+\delta t) - N(C_{i_j}, t)}{\delta t} = \\ - \sum_{(\alpha,r)\in Ex(C_{i_j})} r \times \min_{C_{k_l}\in Ex(\alpha,r)} (N(C_{k_l}, t)) \\ + \sum_{(\alpha,r)\in En(C_{i_j})} r \times \min_{C_{k_l}\in Ex(\alpha,r)} (N(C_{k_l}, t)). \end{aligned} \tag{2}$$

In Eq. 1, the first block represents the impact of exit activities, and the second block records the impact of the entry activities. Now, we can divide Eq. 1 by δt and take a limit. If $\delta t \to 0$, we obtain Eq. 2. In the following analysis, a set of ODEs can be obtained from the PEPA model based on Eq. 2. The quantitative analysis is conducted through solving the ODEs.

According to above PEPA models and the associated ODE equations, numerical analysis is generated by observing average throughput and response time at base stations, which can help analyse the communication overhead of base stations and evaluate the performance of the offloading scheme. Fig. 12 depicts the average throughput of root and neighbour base stations (shown in Fig. 11) by comparing the D2D-based offloading scheme to the non-offloading scheme. It is clear that *Surf 2 & 4*, on behalf of the D2D offloading scheme, represent lower-level throughput in contrast to *Surf 1 & 3* for non-offloading scheme. This indicates the communication overhead of base stations are offloaded to D2D communications. With the increase of communication traffic, the difference between the two schemes grows until the full capacity of communication channels. In addition, Fig. 13 represents the average response time from the root base station. The offloading scheme (*Surf 2*) generates a faster response to vehicle requests in contrast to the non-offloading scheme (*Surf 1*) at the root base station due to the reduced communication overhead.

In conclusion, PEPA languages can generate simplified abstraction models by aggregating model details that are not observed in analysis. The abstraction modeling and

TABLE 2
Related Parameters in 5G Environments

Notations	Values	Comments (Assumed as 100 Mhz BW)
R_{down}	$\geq 1.0\ Gbps$	Downstream (70%) rate of 5G bases.
R_{up}	$\geq 200\ Mbps$	Upstream (30%) rate of 5G bases.
R_{d2d}	$\geq 400\ Mbps$	Duplex D2D (50% Rx/Tx) rate.
D_{d2d}	$[0.5, 1.0]\ Km$	D2D communication range.
D_{base}	$[1.0, 2.0]\ Km$	5G base station coverage radius.

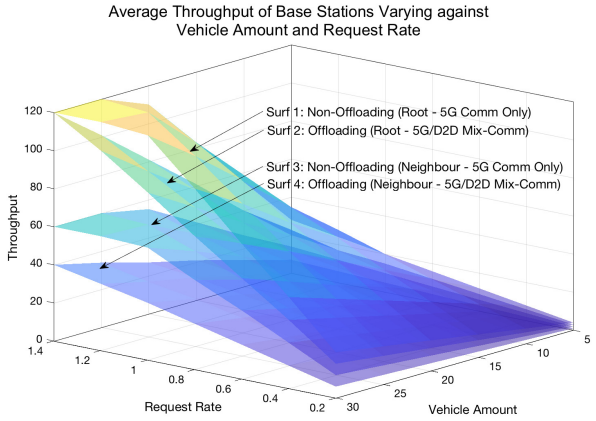


Fig. 12. Average Throughput of Root/Neighbour Base Stations Varying against Number of Vehicles and Request Rates.

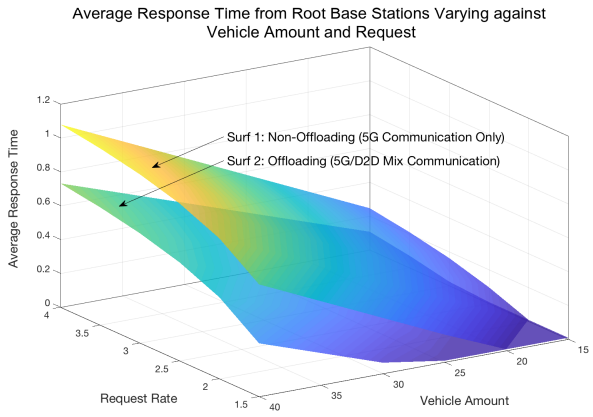


Fig. 13. Average Response Time from Root Base Stations Varying against Number of Vehicles and Request Rates.

fluid-flow approximation allow PEPA to generate efficient performance modeling and analysis, particularly for large-scale modeling systems.

5 Modeling of Smart Scheduling Scheme

With enhanced communication and computation technologies, edge equipment can be utilized as a supplement to central cloud services, which is known as "*Fog Computing Service*". However, a key target is to guarantee the QoS of such fog services. This section aims to explore an efficient smart scheduling scheme to fit the diversity of a fog computing environment. The smart scheduling scheme (SSS) is developed by including two proposed dynamic scheduling algorithms and a decision function for algorithm selection based on changing conditions of the system. To evaluate the performance of the SSS, this section also presents a PEPA-based model prototype to demonstrate how the formal modeling and fluid-flow analysis are used for complex algorithm modeling and analysis, particularly in a large-scale system such as the HCF system.

5.1 Scheduling Model Scenario

According to the scenario mentioned in previous sections, some services can be shifted from conventional cloud servers to more flexible fog-side servers, i.e., service offloading. To generate efficient offloading, scheduling is a crucial issue being investigated in much research. However, most previous research aims to improve the scheduling process through refining an individual scheduling policy (e.g., R-JSQ scheduling [31], Dynamic Task Offloading and Scheduling [32], Dynamic Switching Algorithm [33] and Energy-aware Task Allocation [34]). These research approaches usually provide a more powerful but complex algorithm design, which might not be readily employed for industrial deployment. However, our solution aims to develop a novel smart scheduling scheme with hybrid scheduling algorithms that are designed on the basis of classic scheduling policies. These classic scheduling policies usually benefit from high reliability and usability in practice.

Fig. 14 depicts a vehicular fog environment, in which a group of vehicles can be organized to perform as a *Vehicular Fog Server (VFS)* by sharing their available physical resources. In each VFS group, these vehicle nodes can be either static nodes (parking vehicles) or mobile nodes (moving vehicles) according to their mobility feature. Therefore, VFSs usually suffer from unstable capability in computation caused by the changing physical resources of vehicular nodes. For this reason, this section proposes a novel smart scheduling scheme (SSS) that can adaptively conduct scheduling operations based on the real-time VFS conditions. In the scheme, a novel time-preemptive scheduling (TS) algorithm is developed for the unstable VFS groups, along with a smart scheduling scheme with both TS and a queue-preemptive scheduling (QS) algorithm that is derived from the classic join-the-shortest-queue (JSQ) algorithm [30]. In addition, a decision module is designed to perform selection of scheduling algorithm (TS or QS) along with real-time VFS conditions.

The main goal of smart scheduling scheme is to combine two atomic scheduling algorithms (i.e. TS and QS) to a hybrid scheme and apply the optimal algorithm along with a changing system environment. Such a design can take full advantage of each scheduling algorithm based on the system condition and improve the adaptivity of the scheduling scheme, particularly in diverse system conditions. As a result, the end-users can have improved quality of service such as faster system response.

5.2 Definition of Smart Scheduling

With the aim of improving resource utilization and QoS, a smart scheduling scheme is proposed for such a diverse fog environment. Thus, the scheduling scheme can be formally represented as a 4-tuple M in Definition 1:

Definition 1. *Smart Scheduling Scheme*

$$\begin{aligned} \boldsymbol{M} &=< S,\ F,\ D(f), q_0 >; \\ \boldsymbol{S} &:\ is\ a\ scheduling\ algorithm\ set; \\ \boldsymbol{F} &:\ is\ a\ transiant\ fog\ server\ status; \\ \boldsymbol{D(f)} &:\ f \rightarrow s,\ f \in F,\ s \in S;\ is\ a\ decision\ function; \\ \boldsymbol{q_0} &:\ is\ the\ system\ environment. \end{aligned}$$

In the smart scheduling scheme, the algorithm set S includes different atomic scheduling algorithms that can be

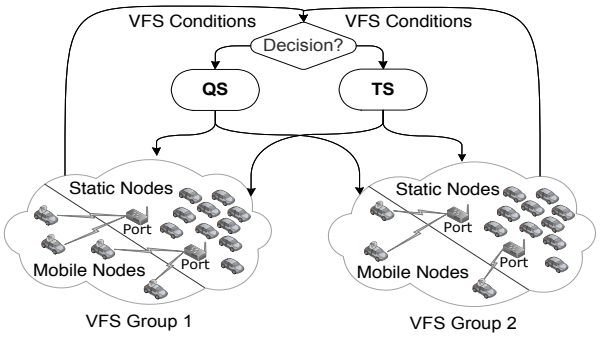


Fig. 14. Smart Scheduling Framework in Fog Computing

selected by the scheme. Join-the-Shortest-Queue (JSQ) is a classic scheduling algorithm that assigns a new arrival to a queue with the least number of unfinished jobs. However, in the fog computing environment, both incoming task stream and server capability encounter unstable situations, and thus it is difficult to determine whether the classic JSQ policy still performs well in such a diverse fog environment. Meanwhile, the recent task stream has an obvious heavy-tail distributed feature. A larger number of waiting jobs in the queue does not represent longer waiting time, as a large-sized job may consume much more time than many other small-sized tasks. In the situation, the classic JSQ algorithm maybe not the most efficient solution in scheduling. Thus, the smart scheduling scheme aims to combine the classic JSQ algorithm with a new time-preemptive algorithm to improve scheduling adaptivity by selecting the optimal scheduling algorithm based on the system environment through a decision function $D(f)$. The $D(f)$ takes the status of current servers ($f \in F$) as input and makes a decision to select a scheduling algorithm ($s \in S$) that is considered as the output of function $D(f)$.

With this in mind, the classic JSQ algorithm is formalised to a *Queue-preemptive Scheduling* algorithm - i.e. QS algorithm. Meanwhile, a new scheduling algorithm is built based on the predicted response time of jobs in the queue, which is named the *Time-preemptive Scheduling* algorithm - i.e. TS algorithm. The decision function, $D(f)$, is designed to select the optimal algorithm from QS and TS on the basis of server status (F), which indicates a real-time system environment.

First, the definitions of QS and TS must be specified, and the performance of each scheduling algorithm needs to be evaluated in various system conditions to confirm the best operating conditions.

Definition 2. *Queue-preemptive Scheduling (QS)*

$$CF_{QS} = \frac{1}{L_{trans}(t)} \cdot C. \tag{3}$$

Generally, the queue-preemptive scheduling (QS) algorithm means that the task scheduling is conducted on the basis of queue length. In other words, the server with the shortest waiting queue will receive the next scheduled task.

In Definition 2, the value of the QS algorithm is defined to be a controlling factor, CF_{QS}. The CF_{QS} value is governed by the transient queue length ($L_{trans}(t)$) of the VFS group. Longer queue length means a smaller value of CF_{QS}, which indicates the rate of sending tasks to a target server. Moreover, C is specified as a constant value used for adjusting the sending rate. The value of CF_{QS} can be directly obtained by solving a PEPA model in which the QS process is defined.

Definition 3. *Time-preemptive Scheduling (TS)*

$$CF_{TS} = \frac{R_{avg}(N)}{R_{trans}(t)} \cdot C. \tag{4}$$

The time-preemptive scheduling (TS) algorithm controls task scheduling based on the response time, which means that if the transient response time (i.e., $R_{trans}(t)$) of a server increases, the rate of scheduling tasks (i.e., CF_{TS}) to the server must be reduced. To control the rate of task scheduling, the TS algorithm designs a controlling factor represented in Definition 3, which is formed by a ratio of average response time (i.e., $R_{avg}(N)$) and transient response time (i.e., $R_{trans}(t)$) of a server. As a controlling factor, the ratio quantifies the variation of specific transient response time of a task at the server compared to the average response time at the server. Usually, the average response time is a fixed but predicted value for each server, while the transient response time can be directly calculated from the system observation. The following section will introduce how to obtain the transient response time and predict the average response time.

In Definition 3, the transient response time $R_{trans}(t)$ represents the value of response time based on the tasks in a server queue, which can be obtained by calculating the summation of transient waiting time ($W_{tans}(t)$) of all tasks in the queue and the transient service time ($S_{trans}(t)$) of the task in the server. In other words, $R_{trans}(t)$ indicates the duration for a task to wait before gaining access to the server.

As the value of $R_{trans}(t)$ can be easily calculated with $L_{trans}(t)$, we have the formula of calculating $R_{trans}(t)$ as:

$$\begin{aligned} R_{trans}(t) &= W_{tans}(t) + S_{trans}(t) \\ &= \frac{l}{\mu} + \frac{1}{\mu} = (l+1) \cdot \omega(t), \end{aligned} \tag{5}$$

in which $l = L_{trans}(t)$ and μ is the average service rate of a fog server ($\omega(t) = 1/\mu$).

However, the average response time $R_{avg}(N)$ is defined as an average value for all tasks in a server system with N users. Actually, it is impossible to obtain the accurate value of $R_{avg}(N)$ before completing all tasks. Thus, an approximation will be considered to estimate the value of $R_{avg}(N)$. The fog server system can be theoretically considered as a closed $M|M|1||K$ terminal model, in which system users can be thought of as sitting behind their *terminals*, and their sending tasks queue for the *server system*. These users send their task requests in every thinking timeslot t that is exponentially distributed with a mean $\phi(t) = 1/\lambda$. The more users in the thinking state, the higher the effective completion rate of the thinkers. In this situation, we can consider that the actual arrival rate in the system must be proportional to the number of thinking users. Hence, in such a system model, users are represented as infinite servers with each user configured with its own server. Once the user submits a task request, it only needs to wait for a response. The task can be completed within a mean time $\omega(t) = 1/\mu$.

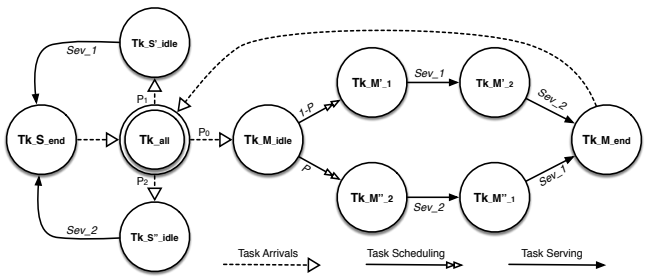


Fig. 15. State Diagram of Smart Scheduling Framework

According to the terminal model, every task has its own server in an infinite-server queueing station; thus, there will be no queueing or waiting. As a result, the average response time at terminals ($\bar{R}_{ter}$) equals $\phi(t)$, which is independent of the number of users (N). However, the average response time of the server system ($\bar{R}_{ser} = R_{avg}(N)$) depends on the number of users (N). Hence, the $R_{avg}(N)$ can be obtained by calculating the average circle time, which is defined as:

$$\bar{C}(N) = \bar{R}_{ter} + \bar{R}_{ser} = \phi(t) + R_{avg}(N). \tag{6}$$

Hence, $\bar{C}(N)$ represents the mean time of a customer going through the cycle of *terminal-server* once. The throughput $T(N)$ can be represented as $N/\bar{C}(N)$. Based on the above two equations, we can obtain:

$$T(N) = \frac{N}{\bar{C}(N)} = \frac{N}{\phi(t) + R_{avg}(N)}. \tag{7}$$

Thereafter, from Eq. (7), the average response time of the server system can be represented as:

$$R_{avg}(N) = \frac{N}{T(N)} - \phi(t). \tag{8}$$

As we known, the throughput $T(N)$ can be represented with the product of the server-busy probability and the system service rate, which is formulated as:

$$T(N) = (1 - p_0) \cdot \mu = \frac{1 - p_0}{\omega(t)}, \tag{9}$$

in which p_0 is the probability of the server being in the idle state. As the value of p_0 changes against N, p_0 can be considered a function of N, namely, $p(N)$. Therefore, Eq. (8) can be converted to a new expression:

$$R_{avg}(N) = \frac{N \cdot \omega(t)}{1 - p(N)} - \phi(t). \tag{10}$$

In a large-scale system, the value of N is set with a large value, which means that the probability of server-idle state is quite small. As a result, $1 - p(N)$ in Eq. (10) can be approximated as 1. Therefore, Eq. (10) is now approximated to the following expression:

$$R_{avg}(N) \approx N \cdot \omega(t) - \phi(t). \tag{11}$$

Finally, according to Eqs. (5) and (11), the TS algorithm in Definition 3 (CF_{TS}) can be represented as:

$$CF_{TS} = \frac{R_{avg}(N)}{R_{trans}(t)} \cdot C = \frac{N \cdot \omega(t) - \phi(t)}{(l+1) \cdot \omega(t)} \cdot C. \tag{12}$$

In Eq. (12), all parameters have been attained except the transient queue length $l = L_{trans}(t)$ that is measured from real-time model operation.

Lastly, for the decision function $D(f)$, it is designed to select an atomic scheduling algorithm (either QS or TS) through comparing the dispersion of the probability distributions representing varying arrivals and server capabilities. In the $D(f)$, the coefficient of variation (i.e. CV) is used to estimate the dispersion of probability distributions, which equals the ratio of the standard deviation to its absolute mean value. The $D(f)$ first calculates the CVs of both parameters - i.e. arrival rates (i.e. CV_λ) and service rates (i.e. CV_μ) representing the dynamical arrivals and servers; then compares the two CV values to determine which one is larger. The parameter with larger CV value can make a greater impact on the system changing. Thus, the selection of an algorithm is based on the comparison results, which will be confirmed after introducing the first stage of performance evaluation in Section 5.4.

5.3 PEPA-based Scheduling Model

In this subsection, PEPA language is used to model QS and TS algorithms in order to evaluate their performances in various system conditions. As described in Section 7.1, a diverse fog computing environment is supposed with respect to the following three conditions: varying arrival condition that indicates the unstable incoming task stream with varying arrival rates defined as $\lambda = f(t)$; varying service condition that indicates the unstable fog server capability with a varying service rate $\mu = g(t)$; varying both arrival and service rates.

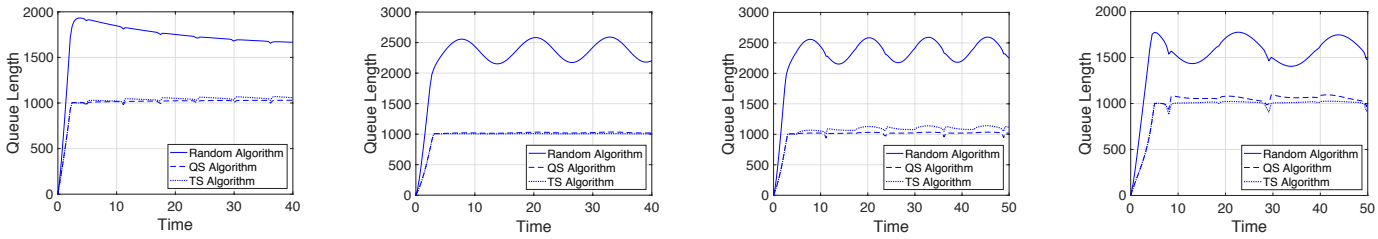
A general scheduling framework is modeled as shown in Fig. 15. In the model, the system is defined with two types of serving activities which are specified as Sev_1 and Sev_2, representing serving activities of two VFS groups previously shown in Fig. 14. According to the serving types, user tasks can be specified in three groups: Tk'_S, indicating a simple-task group that needs Sev_1 service only, Tk''_S, indicating another simple-task group that needs Sev_2 service only, and TK_M, a complex task requiring both Sev_1 and Sev_2. The scheduling process can be considered as the selection operation on candidate fog servers. For TK_M, the serving action flow determined by the scheduling algorithm is either in the order $Sev_1 \rightarrow Sev_2$ or $Sev_2 \rightarrow Sev_1$. Thereafter, the framework will be defined in PEPA as follows:

Step 1: Task generation is achieved by a branching process with a given probability $p \in < p_0, p_1, p_2 >$. Thus, the PEPA model of the task-generation is specified with a $sending$ action and its corresponding rate λ multiplied by an associated probability p, and then behaves as different types of tasks in their initial idle states, such as Tk_M_{idle}, $Tk\_S'_{idle}$, and $Tk\_S''_{idle}$, respectively.

$$Tk_{all} \stackrel{\text{def}}{=} (sending,\ p_0 \cdot \lambda).Tk\_M_{idle} + (sending,\ p_1 \cdot \lambda).Tk\_S'_{idle} + (sending,\ p_2 \cdot \lambda).Tk\_S''_{idle}.$$

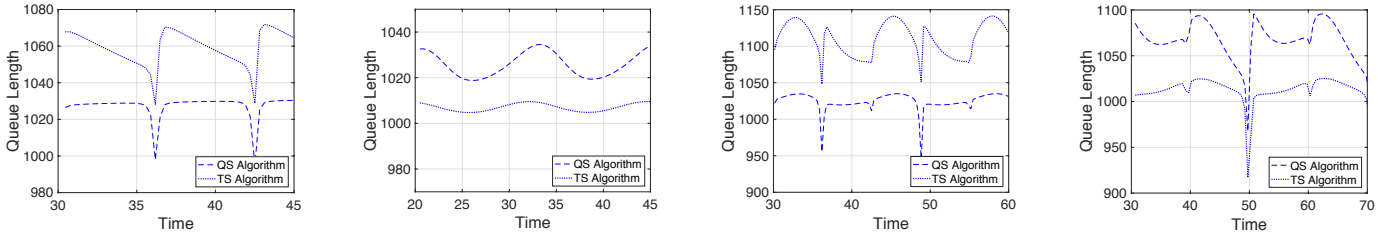
Step 2: VFS groups are defined as two components:

$$FS_{grp\_1} \stackrel{\text{def}}{=} (Sev_1,\ \mu_1).FS_{grp\_1};$$
$$FS_{grp\_2} \stackrel{\text{def}}{=} (Sev_2,\ \mu_2).FS_{grp\_2},$$



(a) Varying Arrival Only (b) Varying Service Only (c) Varying Service & Arrival: service varying mainly (d) Varying Arrival & Service: arrival varying mainly

Fig. 16. Queue Length of Servers Based on Three Scheduling Algorithms under Three Different System Conditions



(a) Varying Arrival Only (b) Varying Service Only (c) Varying Service & Arrival: service varying mainly (d) Varying Arrival & Service: arrival varying mainly

Fig. 17. Queue Length of Servers Based on Two Dynamic Scheduling Algorithms under Three Different System Conditions

in which Sev_1 and Sev_2 are defined to be cooperatively used by both server components (FS_{grp_1} and FS_{grp_2}) and the task component (Tk_{all}).

Step 3: With the state diagram Fig. 15, the task serving process can be modeled as the cooperative interaction between the task component and the fog server component.

Simple-task Group Serving Model:

$$Tk\_S'_{idle} \stackrel{\text{def}}{=} (Sev_1,\ \mu_1).Tk\_S'_{end};$$
$$Tk\_S''_{idle} \stackrel{\text{def}}{=} (Sev_2,\ \mu_2).Tk\_S''_{end}.$$

Each simple-task reaches its end state ($Tk\_S'_{end}$ or $Tk\_S''_{end}$) after a server processing action (Sev_1 or Sev_2).

Complex-task Group Serving Model:

$$Tk\_M_{idle} \stackrel{\text{def}}{=} (Sev_1,\ \mu_1).(Sev_2,\ \mu_2).Tk\_M_{end} + (Sev_2,\ \mu_2).(Sev_1,\ \mu_1).Tk\_M_{end}.$$

Each complex-task completes its processing on fog servers in either action sequence $Sev_1 \rightarrow Sev_2$ or $Sev_2 \rightarrow Sev_1$.

Step 4: For the complex-tasks, a scheduling decision is made to select the first VFS-group to determine its processing action sequence. Thus, the scheduling algorithm needs to be formally represented as the following expression:

$$P = H\ (s) \circ D\ (c), \tag{13}$$

in which the function D indicates the decision function for selecting an optimal scheduling algorithm based on the current server conditions (c), and the function H is a composite scheduling function based on the selected scheduling algorithm $s = D(c)$. Finally, Eq. (13) outputs the selected server for the current task given a probabilistic value P.

Step 5: To model a scheduling process, we need to design a separate scheduler component (SCH) in the PEPA model, which can be represented as:

$$SCH \stackrel{\text{def}}{=} (Sch\_{QS},\ p\_{QS}).SCH + (Sch\_{TS},\ p\_{TS}).SCH,$$

in which Sch_{QS} is for the execution of queue-preemptive scheduling (QS) and Sch_{TS} is to implement time-preemptive scheduling (TS). p_{QS} and p_{TS} are the rates of scheduling processes that are obtained from Eq. (13) for QS and TS algorithms, respectively.

Hence, the previous complex-task serving model should be cooperatively implemented with the scheduler component SCH. Thus, the defined Tk_M_{idle} component needs to be updated by adding a new scheduling action:

$$Tk\_M_{idle} \stackrel{\text{def}}{=} (Sch\_{QS},\ p\_{QS}).(Sev_1,\ \mu_1).(Sev_2,\ \mu_2).Tk\_M_{end} + (Sch\_{QS},\ p\_{QS}).(Sev_2,\ \mu_2).(Sev_1,\ \mu_1).Tk\_M_{end},$$

which implements the QS algorithm; however, an alternative Tk_M_{idle} using the TS algorithm is denoted as:

$$Tk\_M_{idle} \stackrel{\text{def}}{=} (Sch\_{TS},\ p\_{TS}).(Sev_1,\ \mu_1).(Sev_2,\ \mu_2).Tk\_M_{end} + (Sch\_{TS},\ p\_{TS}).(Sev_2,\ \mu_2).(Sev_1,\ \mu_1).Tk\_M_{end}.$$

Step 6: Finally, a complete scheduling model based on a VFS system can be represented by joining these components as Sys, which is represented as:

$$Sys \stackrel{\text{def}}{=} Tk_{all}[n] \underset{L}{\bowtie} (FS_{grp\_1}[k_1] \| FS_{grp\_2}[k_2] \ \| SCH[m])$$
$$L = \{pros1,\ pros2,\ Sch\_{QD},\ Sch\_{TD}\},$$

in which $< n, m, k_1, k_2 >$ denotes the number of instances of each component.

5.4 Performance Evaluation with Scheduling Model

In performance analysis, to clear the advantages of two atomic scheduling algorithms, we first compare the proposed time-preemptive scheduling algorithm to the conventional JSQ algorithm - i.e. the formulated queue-preemptive scheduling given in Definition 2. As per the general service-level agreement between customers and service providers, the total resolution time (TRT) becomes a primary concern

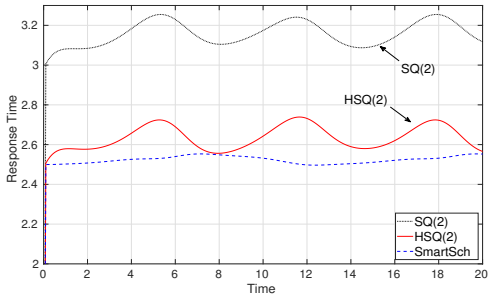


Fig. 18. Response Time Comparison Based on Fluid-flow Analysis

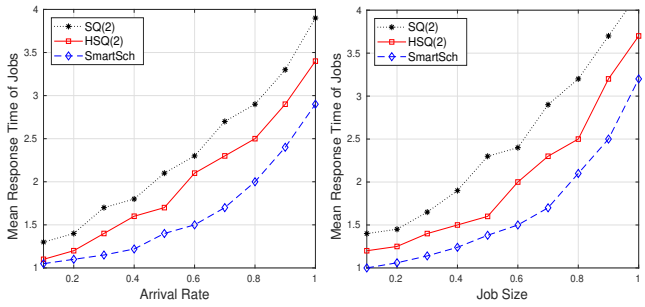


Fig. 19. Response Time Comparison Against Arrival Rate or Job Size

in representing the quality of service. In performance engineering, TRT is similar to the response time that defines the amount of time between when a client sends a request and when the request is answered. Hence, the following analysis is based on the performance metric - i.e. response time.

To conduct prformance evaluation, a set of parameters needs to be specified under the defined PEPA model defined in the previous section. Each task is modelled as a text message, and the workload is represented by the message size (e.g. megabytes). The performance metric - i.e. response time is defined by the sum of transmission delay and processing delay that are measured by altering workload and task arrival rates. Thereafter, the number of task senders is set to 4,000 for each fog server unit. As there are three types of task senders (i.e., $Tk\_S'_{idle}$, $Tk\_S''_{idle}$ and Tk_M_{idle}) defined in the model, 1,000 sender instances are set for each simple-task component, i.e., $Tk\_S'_{idle}$ and $Tk\_S''_{idle}$, respectively, with 2,000 senders for the complex-task component, i.e., Tk_M_{idle}. The system assumes that there are two independent server groups (i.e., FS_{grp_1} and FS_{grp_2}) in the model, and the number of server instances based on each group is set to one. The arrival rates of all types of tasks are set to the same value of 400 (tasks/time unit). The service rates are set to 120 and 80 for FS_{grp_1} (μ_1) or FS_{grp_2} (μ_2), respectively. Furthermore, to achieve the diverse system conditions, trigonometric functions (e.g., sin and cos) are used to govern the varying means of arrival rates and service rates. Three sets of experiments are conducted against different system conditions: 1). varying task arrivals only, 2). varying server capability only, and 3). a comprehensive condition with both varying task arrivals and server capability. In addition, a classic random scheduling algorithm is applied for comparison with QS and TS algorithms.

Figs. 16(a) and 17(a) represent the average queue length of three scheduling algorithms. As shown in the figures, both QS and TS exhibit substantially better performance than the random algorithms due to their shorter waiting-queue length. In detail, the QS algorithm generates more stable and shorter queue length in contrast to TS. Hence, this means that the QS is a better choice under the above system condition 1).

Figs. 16(b) and 17(b) depict the performances of three scheduling algorithms under varying service capability conditions. It is clear that the random algorithm has inferior performance due to the large movement of queue length; furthermore, the TS algorithm has better performance than QS with a more stable and shorter queue length. It is concluded that the TS algorithm becomes better under system condition 2).

To validate these conclusions, a complex system condition 3) is used to observe the scheduling performance. Figs.16(c) and 17(c) represent the performance under a complex condition with varying arrival rates and service rates, in which the varying arrival rates dominate the influence to a greater extent than service rates. In this case, the QS is better than the TS with both shorter average queue length and reduced fluctuation of queue length. Conversely, when the service rates vary larger than the arrival rates, as shown in Fig. 16(d) and Fig. 17(d), the TS algorithm becomes superior to the QS.

Based on the above analysis, we find that the QS algorithm performs better under an intensive varying-arrival environment; however, the TS algorithm is more suitable for the intensive change of server condition. Therefore, if we combine two basic scheduling algorithms to a hybrid scheduling algorithm and apply a decision component to support the selection of two algorithms, such smart scheduling scheme should gain improved performance under dynamic environments with both varying arrivals and servers. According to the design principles of $D(f)$ and the analysis conclusion, if $CV_\lambda > CV_\mu$ representing an intensive varying-arrival environment, the $D(f)$ selects the QS algorithm; conversely, it selects the TS algorithm.

Next, to verify the performance of the smart scheduling scheme, we will compare it to two randomized join-the-shortest-queue scheduling algorithms defined in [31], which are SQ(2) and HSQ(2) schemes. In the SQ(2) scheme, a subset of two servers is selected from the set of N servers uniformly at random at each arrival instant. The job is scheduled to the server with the least number of unfinished jobs among the two chosen servers. The SQ(2) assigns jobs to any of the two servers with equal probability. Nevertheless, the HSQ(2) first choose a capacity value C_j with a probability p_j upon arrival of a new job. Then two servers having the selected value of capacity are chosen uniformly from a set of available servers having that capacity. Finally, the job is assigned to the server with the least number of unfinished jobs among the two chosen servers. Comparing to the conventional JSQ algorithm, these two schemes do not need to observe all servers to select that with the shortest waiting queue, which can reduce the observation cost, particularly for systems with large node sets. In this experiment, we

will compare our smart scheduling scheme with the two randomized JSQ schemes under a heterogeneous system environment (i.e., varying rates of arriving tasks and VFS capability). All scheduling schemes are modeled in PEPA and analyzed with fluid-flow approximation. Moreover, to verify the results of fluid-flow approximation, we also build equivalent simulation models to enable a comparison.

Fig. 18 shows that our smart scheduling scheme yields less response delay than both SQ(2) and HSQ(2); moreover, under the complex system environment, the smart scheduling scheme has stable performance in response time, which is caused by the joint use of two algorithms - i.e. QS and TS. This hybrid scheme allows the scheduling algorithm to be altered with the changing system environment, which can minimize the influence on system performance. However, both SQ(2) and HSQ(2) algorithms are affected by the varying environment, which causes a fluctuation of response time, as shown in Fig. 18. To build a simulation-based analysis, a Java-based modeling tool, named SimJava [36], is applied to generate a discrete event simulation model that is equivalent to the PEPA model design. In the simulation model, system components/behaviors and action rates are defined along with the PEPA model, such as the number of instances for request senders and servers as well as their rates (e.g. arrival rates and service rates) are sent the same to that used in PEPA models - i.e. as stated before 4000 task senders per server including three types, one instance for each fog server group, 400 task arrival rates, and 120 and 80 service rates for each server group. Moreover, the network parameter is also consistent with PEPA parameters: 10 and 5 (simulation time units) mean delays from scheduling node to senders and fog servers, respectively; 1 Gbps bandwidth and zero packet loss rate.

Fig. 19 presents the mean response time of three scheduling algorithms against varying arrival rate and job size. In the experiment, both arrival rate and job size are set from 0.1 to 1.0 with step size 0.1, and the service is also set to fluctuated values as stated before. The left side sub-figure shows the change of mean response time with varying arrival rate while setting the job size to 0.1. Comparing to SQ(2) and HSQ(2), the line representing the smart scheduling algorithm indicates a smoother and low-level change of mean response time with the growth of arrival rates. The right side sub-figure indicates the mean response time with varying job size while the arrival rate is 0.1. Similarly, the smart scheduling scheme presents a relatively more stable change and slower increase of response time with the growth of job size. These analysis results demonstrate the better performance of our smart scheduling scheme under a complex system environment.

In summary, this section illustrates a scheduling model prototype based on a proposed smart scheduling scheme and indicates that PEPA exhibits a powerful model specification capability and accurate performance analysis precision, which can be applied to support the performance evaluation of large-scale and complex systems.

6 Conclusion

This paper proposed a novel modeling framework for HCF systems based on a formal modeling technique, i.e., performance evaluation process algebra. We developed the framework with the aim of providing a cost-effective formal modeling solution for supporting the design of large-scale systems such as HCF systems. In comparison with conventional simulation and other modeling techniques (e.g., queueing modeling and Petri-net), the PEPA-based framework offers the following crucial advantages: 1). strong capability in compositional and abstract modeling, which help to quickly build models from an elaborate design scheme; 2). various approaches (e.g., stochastic simulation and fluid-flow analysis) that can support performance evaluation by quickly yielding reliable analysis results. In detail, this work presented how to use the proposed framework by introducing three core model prototypes, in which the *compositional model* illustrated how to build models of system structures and components as well as system behaviors; the second *abstraction model* introduced how to aggregate model details in order to focus on the components and behaviors that need be observed. Finally, we proposed a *scheduling model* that defined a novel smart scheduling scheme. The scheme integrated two atomic scheduling algorithms and used a decision support module to select the right algorithm on the basis of changing system environments. This design can improve scheduling performance and service quality by making scheduler smarter. Furthermore, this model also demonstrated how to achieve algorithm modeling based on the PEPA model framework. In our future work, we will apply this framework to more modeling and analyzing missions and explore other techniques to make the framework more adaptable.

Acknowledgments

The research is supported by the National Natural Science Foundation of China (NSFC) under Grants 61702233, 61472343, 61773220 and 61976117, and State Key Laboratory of Software Development Environment under Grant SKLSDE-2017KF-03, and the National Science Foundation of Jiangsu Province under Grants BK20191409 and 19KJA360001.

References

[1] M. Giordani, M. Polese, M. Mezzavilla, S. Rangan and M. Zorzi, *Toward 6G Networks: Use Cases and Technologies*, IEEE Commun. Mag., vol.58, no.3, pp.55-61, Mar. 2020.

[2] L. Chettri and R. Bera, *A Comprehensive Survey on Internet of Things (IoT) Toward 5G Wireless Systems*, IEEE Internet Things J., vol.7, no.1, pp.16-32, Jan. 2020.

[3] X. Wang and Y. Li, *Content Retrieval Based on Vehicular Cloud in Internet of Vehicles*, IEEE Trans. Comput. Social Syst., vol.6, no.3, pp.582-591, Jun. 2019.

[4] Y. Yao, X. Chang, J. Misic and V. Misic, *Reliable and Secure Vehicular Fog Service Provision*, IEEE Internet Things J., vol.6, no.1, pp.734-743, Feb. 2019.

[5] Z. Zhao et al., *On the Design of Computation Offloading in Fog Radio Access Networks*, IEEE Trans. Veh. Technol., vol.68, no.7, pp.7136-7149, Jul. 2019.

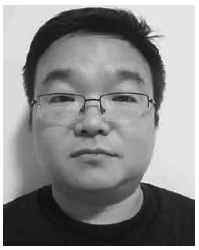
[6] T. T. Nguyen, V. N. Ha, L. B. Le and R. Schober, *Joint Data Compression and Computation Offloading in Hierarchical Fog-Cloud Systems*, IEEE Trans. Wireless Commun., vol.19, no.1, pp.293-309, Jan. 2020.

[7] N. Choi, D. Kim, S. J. Lee, and Y. Yi, *A Fog Operating System for User-oriented IoT Services: Challenges and Research Directions*, IEEE Commun. Mag., vol.55, no.8, pp.44-51, 2017.

[8] X. Chen, J. Ding, and Z. Lu, *A Decentralized Trust Management System for Intelligent Transportation Environments*, IEEE Trans. Intell. Transp. Syst., 2020, Accepted.
[9] Y. Jiao, P. Wang, D. Niyato and K. Suankaewmanee, *Auction Mechanisms in Cloud/Fog Computing Resource Allocation for Public Blockchain Networks*, in IEEE Transactions on Parallel and Distributed Systems, vol. 30, no. 9, pp. 1975-1989, 1 Sept. 2019.
[10] J. Hillston, *A compositional approach to performance modelling*, Cambridge University Press, 1996.
[11] J. Hillston, *Fluid Flow Approximation of PEPA Models*, in Proc. of QEST'05, Torino, Italy, 2005, pp.33-42.
[12] R. Yu, G. Xue, and X. Zhang, *Application Provisioning in Fog Computing-enabled Internet-of-Things: A network Perspective*, in Prof. of INFOCOM'18, Honolulu, HI, USA, 2018.
[13] J. He, J. Wei, K. Chen, Z. Tang, Y. Zhou and Y. Zhang, *Multitier Fog Computing With Large-Scale IoT Data Analytics for Smart Cities*, IEEE Internet Things J., vol.5, no.2, pp.677-686, Apr. 2018.
[14] F. A. Silva et al., *Mobile Cloud Performance Evaluation Using Stochastic Models*, IEEE Trans. Mobile Comput., vol.17, no.5, pp.1134-1147, May 2018.
[15] R. Ghosh, F. Longo, F. Frattini, S. Russo, and K. S. Trivedi, *Scalable Analytics for IaaS Cloud Availability*, IEEE Trans. on Cloud Comput., vol.2, no.1, pp. 57-70, 2014.
[16] X. Chang, B. Wang, J. K. Muppala, and J. Liu, *Modeling Active Virtual Machines on IaaS Clouds Using an M/G/m/m+K Queue*, IEEE Trans. on Serv. Comput., vol.9, no.3, pp.408-420, 2016.
[17] X. Li, L. Pan, J. Huang, S. Liu, Y. Shi, L. Cui, and C. Pu, *Performance Analysis of Cloud Computing Centers Serving Parallelizable Rendering Jobs Using M/M/c/r Queuing Systems*, in Proc. of ICDCS'17, Atlanta, GA, 2017, pp. 1378-1388.
[18] Z. Su, Y. Hui, and S. Guo, *D2D-based Content Delivery with Parked Vehicles in Vehicular Social Networks*, IEEE Wireless Commun., vol.23, no.4, pp.90-95, 2016.
[19] X. Chen, and J. Zhang, *When D2D Meets Cloud: Hybrid Mobile Task Offloadings in Fog Computing*, in Proc. of ICC'17, Paris, France, 2017, pp.1-6.
[20] X. Wang, Z. Ning and L. Wang, *Offloading in Internet of Vehicles: A Fog-Enabled Real-Time Traffic Management System*, IEEE Trans. Ind. Informat., vol.14, no.10, pp.4568-4578, Oct. 2018.
[21] F. Mehmeti, and T. Spyropoulos, *Performance Modeling, Analysis, and Optimization of Delayed Mobile Data Offloading for Mobile Users*, IEEE/ACM Trans. on Netw., vol.25, no.1, pp.550-564, 2017.
[22] N. Cheng, N. Lu, N. Zhang, X. S. Shen and J. W. Mark, *Opportunistic WiFi Offloading in Vehicular Environment: A Queueing Analysis*, in Proc. of IEEE GLOBECOM'14, Austin, TX, 2014, pp.211-216.
[23] H. Wu and K. Wolter, *Stochastic Analysis of Delayed Mobile Offloading in Heterogeneous Networks*, IEEE Trans. Mobile Comput., vol.17, no.2, pp.461-474, Feb. 2018.
[24] H. Tan, Z. Han, X. Y. Li, and F. C. M. Lau, *Online Job Dispatching and Scheduling in Edge-clouds*, in Proc. INFOCOM'17, Atlanta, GA, 2017, pp.1557-1566.
[25] D. Zeng, L. Gu, S. Guo, Z. Cheng, and S. Yu, *Joint Optimization of Task Scheduling and Image Placement in Fog Computing Supported Software-defined Embedded Systems*, IEEE Trans. on Comput., vol.65, no.12, pp.3702-3712, 2016.
[26] J. Wan, B. Chen, S. Wang, M. Xia, D. Li and C. Liu, *Fog Computing for Energy-Aware Load Balancing and Scheduling in Smart Factory*, IEEE Trans. Ind. Informat., vol.14, no.10, pp.4548-4556, Oct. 2018.
[27] L. F. Bittencourt, J. Diaz-Montes, R. Buyya, O. F. Rana, and M. Parashar, *Mobility-aware Application Scheduling in Fog Computing*, IEEE Cloud Comput., vol.4, no.2, pp.26-35, 2017.
[28] T. Wang, L. Qiu, A. K. Sangaiah, G. Xu and A. Liu, *Energy-Efficient and Trustworthy Data Collection Protocol Based on Mobile Fog Computing in Internet of Things*, IEEE Trans. Ind. Informat., vol.16, no.5, pp.3531-3539, May 2020.
[29] S. Zeadally, J. Guerrero and J. Contreras, *A Tutorial Survey on Vehicle-to-vehicle Communications*, Telecommunication Systems, vol.73, pp.469-489, 2020.
[30] Hwa-Chun Lin and C. S. Raghavendra, *An Approximate Analysis of the Join the Shortest Queue (JSQ) Policy*, IEEE Transactions on Parallel and Distributed Systems, vol. 7, no. 3, pp. 301-307, 1996.
[31] A. Mukhopadhyay and R. R. Mazumdar, *Analysis of Randomized Join-the-Shortest-Queue (JSQ) Schemes in Large Heterogeneous Processor-Sharing Systems*, IEEE Trans. Control Netw. Syst, vol.3, no.2, pp.116-126, Jun. 2016.
[32] H. A. Alameddine, S. Sharafeddine, S. Sebbah, S. Ayoubi and C. Assi, *Dynamic Task Offloading and Scheduling for Low-Latency IoT Services in Multi-Access Edge Computing*, in IEEE Journal on Selected Areas in Communications, vol. 37, no. 3, pp. 668-682, March 2019.
[33] T. Wang et al., "An Intelligent Dynamic Offloading from Cloud to Edge for Smart IoT Systems with Big Data," in IEEE Transactions on Network Science and Engineering, doi: 10.1109/TNSE.2020.2988052.
[34] T. Liu, J. Li, F. Shu and Z. Han, *Optimal Task Allocation in Vehicular Fog Networks Requiring URLLC: An Energy-Aware Perspective*, in IEEE Transactions on Network Science and Engineering,doi: 10.1109/TNSE.2019.2955474.
[35] X. Chen and L. Wang, *Exploring Fog Computing-Based Adaptive Vehicular Data Scheduling Policies Through a Compositional Formal Method—PEPA*, in IEEE Communications Letters, vol. 21, no. 4, pp. 745-748, April 2017.
[36] SimJava Tutorial: http://www.dcs.ed.ac.uk/home/simjava

XIAO CHEN received the M.Sc. and Ph.D. degrees in computing science from Newcastle University in 2009 and 2013, respectively. He is currently a research fellow (Marie Sklodowska-Curie) at School of Informatics in the University of Edinburgh, UK. His research interests include performance evaluation and stochastic optimization for large-scale/distributed systems, e.g., IoT systems, cloud/fog systems, and Blockchain systems.

JIE DING received the B.S. degree in mathematical education from Yangzhou University, Yangzhou, China, in 2001, the M.S. degree in mathematical statistics from Southeast University, Nanjing, China, in 2004, and the Ph.D. degree in communication from The Edinburgh University, U.K., in 2010. He is currently a professor of Shanghai Maritime University. His research interests include performance modelling for communication and computer systems.



ZHENYU LU received the B.Sc. degree in electricity and the M.Sc. degree in information and communication from the Nanjing Institute of Meteorology, Nanjing, China, in 1999 and 2002, respectively, and the Ph.D. degree in optics engineering from the Nanjing University of Science and Technology, Nanjing, in 2008. He was a Research Associate with the Department of Mathematics and Statistics, university of Strathclyde, Glasgow, U.K., from 2012 to 2013. He is currently a Professor with the School of AI, Nanjing University of Information Science and Technology. He has published seven international journal papers. His current research interests include neural networks, stochastic control, and artificial intelligence.

TIANMING ZHAN received the Ph.D. degree in Pattern Recognition and Intelligence System from Nanjing University of Science and Technology (NUST), Nanjing, Jiangsu, China, in 2013. He is currently an Associate Professor at the School of Information and Engineering of Nanjing Audit University (NAU). His current research interests include image processing and data analysis.