Research Article

Dynamic Scalable Stochastic Petri Net: A Novel Model for Designing and Analysis of Resource Scheduling in Cloud Computing

Hua He,¹ Shanchen Pang,² and Zenghua Zhao¹

¹School of Computer Science and Technology, Tianjin University, Tianjin 300072, China

²College of Computer and Communication Engineering, China University of Petroleum, Qingdao 266580, China

Correspondence should be addressed to Shanchen Pang; shanchenpang@sohu.com

Received 19 January 2016; Revised 21 June 2016; Accepted 4 July 2016

Academic Editor: Fabrizio Messina

Copyright © 2016 Hua He et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Performance evaluation of cloud computing systems studies the relationships among system configuration, system load, and performance indicators. However, such evaluation is not feasible by dint of measurement methods or simulation methods, due to the properties of cloud computing, such as large scale, diversity, and dynamics. To overcome those challenges, we present a novel Dynamic Scalable Stochastic Petri Net (DSSPN) to model and analyze the performance of cloud computing systems. DSSPN can not only clearly depict system dynamic behaviors in an intuitive and efficient way but also easily discover performance deficiencies and bottlenecks of systems. In this study, we further elaborate some properties of DSSPN. In addition, we improve fair scheduling taking into consideration job diversity and resource heterogeneity. To validate the improved algorithm and the applicability of DSSPN, we conduct extensive experiments through Stochastic Petri Net Package (SPNP). The performance results show that the improved algorithm is better than fair scheduling in some key performance indicators, such as average throughput, response time, and average completion time.

1. Introduction

Cloud computing provides shared configurable resources to users as services with pay-as-you-go scheme [1]. These services that consisted of set of components may be offered by different providers [2]. To meet the needs of customers, cloud service providers have to ensure that their profit and return on investment are not rapidly decreased due to increased costs, while maintaining a desirable level of the quality of service (QoS) of consumers, such as execution time, delay time, and budget restrictions [2–4]. To address this problem, most researches of cloud computing have focused on performance improvement and satisfaction of the QoS requirements and developed some efficient solutions. But for now, little work has been done about finding a convenient method of modeling, analyzing, and evaluating the performance of scheduling algorithms or systems in cloud environment

without spending too much time on comparison and analysis [5, 6].

Performance evaluation is important to cloud computing development. It is primarily aimed at selecting schemes that meet the requirements of consumers, finding out performance defects, predicting performance of the designed systems in future, and discovering better ways to achieve optimal resource allocation. In other words, performance evaluation is important in selecting, improving, and designing systems or scheduling algorithms in cloud computing environment [7]. The methods of performance evaluation are approximately divided into three types: measurement method, simulation method, and model method. However, measurement and simulation methods are only applicable to existing and running systems and might be time consuming. In addition, the two methods are incapable of finding out performance bottlenecks and analyzing large-scale and complicated cloud

computing systems. In this study, we only focus on the model method.

Model method is a kind of analysis method of performance evaluation by studying and describing the relationships among performance, system, and load based on mathematical theories. To facilitate mathematical descriptions and calculation, it usually requires simplification of the system model and making some rational assumptions about the status of the system. Compared to the other two methods, model method is based on a mature theoretical foundation and can clearly describe the relationship among all factors with a lower cost.

Stochastic Petri Net (SPN) is a powerful tool of model method and can be applied to graphic modeling and mathematical analysis of many systems and areas, such as computer science, communication network, and multiprocessor system [8–13]. It can not only easily describe the properties of systems that have concurrency and synchronization characteristics but also clearly depict dynamic behaviors of systems in an intuitive and efficient way. In this way, it is easy to discover performance deficiencies and bottlenecks by using SPN in analysis.

However, SPN is still not entirely suitable for modeling and performance evaluation of cloud computing systems: (i) cloud computing offers scalable infrastructures to consumers on demand by utilizing the virtualization technology. SPN is not capable of adjusting models dynamically when the infrastructure changes [13]. (ii) Different workloads submitted by users, which are simultaneously running on cloud clusters, might have different QoS requirements, such as response time, execution time, and data traffic [14]. However, SPN is incapable of representing the diversity of cloud computing. (iii) The configurable shared resources in cloud computing are usually heterogeneous and geographically distributed, so using SPN to build up models will increase the computational complexity and result in state explosion. Because of the problems mentioned above, SPN does not adequately model and analyze the performance of cloud computing in many situations.

To overcome those challenges, we propose a novel extended form of SPN, which is called Dynamic Scalable Stochastic Petri Net (DSSPN), to conveniently model and analyze the performance of cloud computing systems. In order to support dynamic changes in cloud computing, three kinds of functions are introduced to enhance the dynamics of arcs, transitions, and places in DSSPN. In addition, many cloud service patterns under the same state can be compressed into a simple one by using DSSPN. Therefore, cloud computing systems can be easily modeled and analyzed by using DSSPN without changing the original graph structure. Consumers can easily evaluate performances only by setting the three functions of the DSSPN model without spending too much time on programming. According to the feature of SPN, system decomposition and model compression can be applied to reduce the complexity of the state space in DSSPN models.

The main contributions of this paper include (1) proposing Dynamic Scalable Stochastic Petri Net (DSSPN) and then further demonstrating firing rules and some properties of

DSSPN; (2) presenting classified fair scheduling (CFS) taking into consideration job diversity and resource heterogeneity, which can improve throughput and response time; (3) validating the proposed approach and algorithm, where we conduct and evaluate DSSPN models of fair scheduling and CFS algorithms by using Stochastic Petri Net Package (SPNP) [15, 16] analysis and simulation.

The remainder of the paper is organized as follows. Section 2 describes the related works of this study. Section 3 specifies the novel analytical model called DSSPN and elaborates its dynamics as well as some other properties. In Section 4, we construct a DSSPN model of resource scheduling of fair scheduler and propose the classified fair scheduling (CFS) algorithm taking into consideration the workload and resources diversity. In addition, in order to alleviate the problem of state explosion, we adopt the multiuser multiserver model [17] and analyze some parameters by using equivalent Markov model to refine our original models in Section 4. Section 5 demonstrates the experimental parameters setup and evaluates the system performance of the two scheduling algorithms by using SPNP. Finally, conclusions and future works are given in Section 6.

2. Related Works

Performance evaluation mainly focuses on relationships among system configuration, system load, and performance indicators and has drawn much research attention recently. Due to the complexity of the problem, most studies adopt measurement and simulation methods to quantitatively analyze system performance [18].

By using some measuring devices or measuring programs, measurement and simulation methods can directly obtain the performance indicators of systems or closely related quantities and then work out performance indexes by the corresponding calculation. Ostermann et al. analyze the performance of the Amazon EC2 platform based on measurement at the background of scientific computing [19]. In addition, performance comparisons were made among EC2 and other platforms by using long-term traces of experimental data in some indicators, such as resource acquisition, release overheads, and system workload. Calheiros et al. propose extensible simulation toolkit CloudSim, which can model, simulate, and evaluate the performance of both cloud computing systems and application provisioning environments [20]. CloudSim supports single and internetworked cloud scenarios and is used by several organizations to investigate cloud resource allocation and energy efficiency management of data center resources. Bautista et al. present a performance measurement framework (PMF) for cloud computing systems with integration software quality concepts from ISO 25010 [21]. The PMF defines the requirements, data types, and evaluation criteria to measure “cluster behavior” performance. Mei et al. study performance measurement of network I/O applications in virtualized cloud [22]. This measurement is based on performance impact of coexisting applications in a virtualized cloud, such as throughput and resource sharing effectiveness. In addition, the measurement

can quantify the performance gains and losses and reveal the importance of optimizing for application deployment.

Measurement and simulation methods are the most direct and basic ones on performance evaluation, which the model method partly depends on. However, the two methods are only applicable to existing and running systems, and there are a lot of insufficiencies and abuses in evaluating the performance of cloud systems which are under dynamic environments and involve lots of parameters, such as time consuming, low degree of simulation, and quantitative difficulty. In addition, measurement and simulation methods are also incapable of finding out performance bottlenecks. Therefore, how to provide powerful mathematic tool, intuitional description method of models, effective analysis method, and available analysis software is the urgent problem for performance evaluation of cloud systems, which is just the core of analysis technology based on SPN. However, there have been few studies on the application of SPN in cloud computing.

Cao et al. construct stochastic evaluation model based on Queuing Petri Net for “Chinese Cloud” of State Key Laboratory of High-End Server & Storage Technology [23]. They still present three kinds of cloud system architectures, distributed architecture, centralized architecture, and hybrid architecture, and then model the three architectures based on Queuing Petri Net. These models describe the relationships among network, CPU, I/O, and request queue. Finally, system throughputs of the three architectures are compared in different task types and workloads with QPME tool [24].

Targeting the dynamic feature of cloud computing, Fan et al. propose a systematic method to describe the reliability, running time, and failure processing of resource scheduling [25]. In this study, resources scheduling process is abstracted as metaobject by using a reflection mechanism, and Petri Net is introduced to model its components, such as base layer, metalayer, and metaobject protocol. In addition, they present an adaptive resource scheduling strategy described by Computation Tree Logic (CTL) [26], which can realize dynamic reoptimization and distribution of system resources at runtime. Finally, Petri Net and its state space are used to verify the correctness and effectiveness of the proposed algorithm.

In order to evaluate the performance of the Hadoop system, Ruiz et al. introduce Prioritised-Timed Coloured Petri Net (PTCPN) [27] to formally construct its stochastic model of MapReduce paradigm [28]. Tradeoffs are made between processing time and resource cost according to some performance evaluation parameters. In addition, state space and CPNTools [29] auxiliary software will execute the quantitative analysis of the system performance and the accuracy verification of the models.

It is concluded that the above-mentioned methods of performance evaluation of cloud computing can well describe and model the various properties of cloud computing, but there are difficulties in comparative analysis. To overcome these challenges, we present a novel Dynamic Scalable Stochastic Petri Net (DSSPN) to better depict the important properties of cloud systems. Compared to other SPNs, DSSPN has the following advantages: (1) intuitive graphical

representation and model easy to understand; (2) no requirements for strong mathematical background; (3) capability of flexibly depicting characteristics of cloud systems, such as the relationship between network topology and other components; and (4) automatically deriving the steady-state probability of state transitions by using auxiliary software, such as SPNP and SHLPNA.

3. Dynamic Scalable Stochastic Petri Net

Cloud computing is a service-oriented computing model, with the characteristics of large scale, complexity, resource heterogeneity, requirement of QoS diversity, and scalability. Those characteristics make the resource scheduling of cloud computing too complicated to be modeled and analyzed by the traditional Stochastic Petri Net. To overcome the problem, a novel Dynamic Scalable Stochastic Petri Net (DSSPN) is proposed in this study. DSSPN is generated from SPN [9, 30] and Stochastic Reward Net (SRN) [31]. In later sections, we will further introduce the feasibility and applicability in both modeling and performance evaluation of cloud computing systems. To easily understand the definition of DSSPN, we firstly present some notations. Let us suppose that S is a set and e is a number. $|S|$ denotes the number of elements in S . $\hat{S}$ represents the power set of S . $[e]$ indicates the maximal integer that is not larger than e . $\mathbb{N}$ stands for the set of natural numbers, that is, $\mathbb{N} = \{0, 1, 2, \dots\}$, while $\mathbb{N}^+$ means the set of positive integers, that is, $\mathbb{N}^+ = \{1, 2, \dots\}$. Let $M : P \rightarrow \mathbb{N}$ denote a marking of DSSPN. $R(M)$ represents the set of reachable marking of the marking M . For all $x \in P \cup T$, $\cdot x = \{y \mid (y, x) \in P \cup T\}$ indicates the preset of x , while $x^* = \{y \mid (x, y) \in P \cup T\}$ means the postset of x . Φ is an empty set, and ε represents an empty element.

3.1. Definitions of DSSPN

Definition 1. A Dynamic Scalable Stochastic Petri Net is a 12-tuple $(P, T, F, K, W, \lambda, TS, G, E, f, g, M^0)$, where

- (1) $P = \{p_1, p_2, \dots, p_n\}$ is a finite set of places, $n = |P|$;
- (2) $T = \{T_I \cup T_T\}$ is a finite set of transitions, $T_I = \{t_{I1}, t_{I2}, \dots, t_{Im}\}$ is a set of immediate transitions, and $T_T = \{t_{T1}, t_{T2}, \dots, t_{Tl}\}$ is a set of timed transitions; $m = |T_I|$, $l = |T_T|$; note that $T_I \cap T_T = \Phi$;
- (3) $P \cup T \neq \Phi$, and $P \cap T = \Phi$;
- (4) $F \subseteq (P \times T) \cup (T \times P)$ is a set of arcs;
- (5) $K : P \rightarrow \mathbb{N}^+ \cup \{\infty\}$ is a capacity function where $K(p)$ denotes the capacity of the place p ; let $K(p) = k_1$ and $T_1 = \cdot p$; if $M(p) = k_1$, for all $t \in T_1$, t cannot be enabled;
- (6) let $\text{Pr}(M(\hat{P}))$ denote an expression of predicate logic related to the marking of the set $\hat{P}$; $\hat{P}$ means a subset of P ;
- (7) $W : F \rightarrow \mathbb{N} \cup \{\varepsilon\} \cup \{H\} \cup \{\text{Pr}(M(\hat{P}) \rightarrow \mathbb{N})\}$ is a weighted function; $W(p, t)$ and $W(t, p)$ denote the weight of the arc (p, t) and (t, p) , respectively. It may be a natural integer, or a function depending on the marking of

the set $\hat{P}$; if $W(x, y) = \varepsilon$, it can be viewed as the weight of $(x, y) = 1$; assume P_1 is a subset of P , and N_1 is a positive integer; if $W(x, y) = \Pr(M(P_1)) \rightarrow N_1$, it means when $\Pr(M(P_1)) = \text{true}$, the weight of (x, y) is N_1 ;

- (8) $H : M(\hat{P}) \rightarrow \mathbb{R}^+ \cup \{0\}$ is a function which indicates the mapping from the marking of $\hat{P}$ to a positive integer; let $P_1 \subseteq P$ and $Z = \{M(p) \mid p \in P_1\}$; then $H(Z) \in \mathbb{N}$;
- (9) $\lambda = \{\lambda_1, \lambda_2, \dots, \lambda_l\} * (1 \vee H)$ is a finite set of average transition enabling rates, where $l = |T|$;
- (10) $\text{TS} = \{\text{TS}_1, \text{TS}_2, \dots, \text{TS}_{n1}\}$ is a finite set of types, where $n1 = |\text{TS}|$;
- (11) $G : P \rightarrow \text{TS}$ is a function denoting the type assigned to place p ;
- (12) $E : P \rightarrow \{\Pr(M(\hat{P})) \rightarrow \mathbb{N}\} \cup \{H * \mathbb{N}\} \cup \mathbb{N}$ is a function indicating the values of types; if $G(p) = \text{TS}_i$, then $E(\text{TS}_i)$ stands for the value of tokens with type TS_i in place p ; note that $E(\text{TS}_i)$ may be time-variant; it generally denotes the value of current period of time when a process is executed;
- (13) $f : T \rightarrow \{\Pr(M(\hat{P}))\} \cup \{\varepsilon\}$ is a function of enabling predicate, where $f(t)$ represents the enabling predicate of transition t . When $f(t) = \varepsilon$, it means that the enabling condition of transition t is the same as in SPN;
- (14) $g : (T \rightarrow \{\varepsilon \cup \mathbb{R}^+ \cup H\}) \cup (P \rightarrow \mathbb{N}^+)$ is a function of random switch, where $g(t)$ denotes the enabling priority of transition t , while $g(p)$ means the priority of place p ; if there is a transition without a random switch, that is, $g(t) = \varepsilon$, it represents that its enabling priority is 1 and $g(p) = \varepsilon$ has the same meaning of place p ;
- (15) $M_0 : P \rightarrow \mathbb{N}$ is the initial marking which models the initial status of a system and satisfies: for all $p \in P$, $M_0(p) \leq K(p)$.

As described above, DSSPN is a novel extended form of SPN. The major differences lie in that the weight of an arc (or the transition enabling rate) not only is a constant but also is a function depending on the marking of a subset of P . The weights of arcs and the transition enabling rates can be defined by customers. In addition, these values may actually change during the whole process. These features will increase the dynamic flexibility of SPN and allow the modeling process to automatically adjust.

Definition 2. The transition firing rule of DSSPN is elaborated as follows.

- (1) For all $t \in T$, if $\forall p \in P$,

$$M(p) \geq W(p, t), \quad \text{if } (p \in \bullet t) \wedge \{\text{true}, \varepsilon\}$$

$$M(p) + W(t, p) \geq W(p, t),$$

$$\text{if } (p \in t^\bullet - \bullet t) \wedge (f(t) \in \{\text{true}, \varepsilon\})$$

$$M(p) + W(t, p) - W(p, t) \leq K(p),$$

$$\text{if } (p \in t^\bullet \cap \bullet t) \wedge (f(t) \in \{\text{true}, \varepsilon\})$$

$$M(p), \quad \text{otherwise.}$$

(1)

It is said that transition t with the marking M is enabled, which is denoted as $M[t]$.

(2) If $M[t]$ and $g(t) = \max\{g(s) \mid s \text{ belongs to } P \text{ and satisfies (1)}\}$, then transition t can fire. After t fired, a new subsequent marking M' is generated from M , which is denoted as $M[t]M'$ or $M \xrightarrow{t} M'$. For all $p \in P$,

$$M'(p)$$

$$= \begin{cases} M(p) - W(p, t), & \text{if } (p \in \bullet t) \wedge \{\text{true}, \varepsilon\} \\ M(p) + W(t, p), & \text{if } (p \in t^\bullet - \bullet t) \\ M(p) + W(t, p) - W(p, t), & \text{if } (p \in t^\bullet \cap \bullet t) \\ M(p), & \text{otherwise.} \end{cases} \quad (2)$$

In the marking M , there may be multiple transitions being enabled simultaneously. In this case, a transition is randomly chosen out from the set T' to be fired, where $T' = \{s \mid g(s) = \max\{g(t), t \in T''\}\}$, and $T'' = \{t \mid M[t], t \in T\}$.

In order to formalize the dynamics of DSSPN, incidence matrix is introduced to depict its structure and behaviors.

Definition 3. The structure of a DSSPN can be expressed by using a matrix (called the incidence matrix of DSSPN) with n rows and m columns, where $n = |P|$ and $m = |T|$:

$$A = [a_{ij}^+ - a_{ij}^-]_{n \times m}. \quad (3)$$

$$\text{For } 1 \leq i \leq n, 1 \leq j \leq m,$$

$$a_{ij}^+$$

$$= \begin{cases} W(t_j, p_i), & \text{if } ((t_j, p_i) \in F) \wedge (W(t_j, p_i) \neq \varepsilon) \\ 1, & \text{if } ((t_j, p_i) \in F) \wedge (W(t_j, p_i) = \varepsilon) \\ 0, & \text{otherwise} \end{cases} \quad (4)$$

$$a_{ij}^-$$

$$= \begin{cases} W(p_i, t_j), & \text{if } (p_i, (t_j) \in F) \wedge (W(p_i, t_j) \neq \varepsilon) \\ 1, & \text{if } ((p_i, t_j) \in F) \wedge (p_i, W(t_j) = \varepsilon) \\ 0, & \text{otherwise.} \end{cases}$$

Because $W(t_j, p_i)$ or $W(p_i, t_j)$ can be a constant or a function depending on the marking of a subset of P , we firstly divide the set of transitions into two subsets: T_c and T_v . Consider

$$T_c = \{t \mid \forall p \in \bullet t \cup t^\bullet, M(p) \text{ is related to } a_{ij}\} \quad (5)$$

$$T_v = T - T_c.$$

That is, if any transition in T_c fired, the incidence matrix will be unchanged in current marking. Otherwise, a new marking will be generated and the value(s) of some element(s) will change. Suppose σ is a firing sequence of transitions. σ is firstly divided into two subsequences according to (6), σ_c and σ_v , where σ_c (or σ_v) only includes transitions in T_c (or T_v), and the orders of these transitions in σ_c and σ_v are the same as that in σ . Suppose C (an m -dimensional column vector) only counts the firing number of the transitions included in σ_c , and $\sigma = t_1 t_2 \cdots t_k$. Consider $M \xrightarrow{\sigma_c} M_1 \xrightarrow{t_1} M_2 \cdots M_k \xrightarrow{t_k} M_{k+1}$; then a fundamental equation [30] is obtained. The markings in the sequence change as follows:

$$\begin{aligned} M_1 &= M + A \cdot C \\ M_{j+1} &= M_j + A_{*j}, \end{aligned} \quad (6)$$

where $1 \leq j \leq k$. A_{*j} denotes the j th column vector of A . Note that if $t \in T_v$, the values of these elements in incidence matrix A , which are related to $\{M(p) \mid p \in t^* \cup t^*\}$, should be updated after t fired.

3.2. Properties of DSSPN. The major motivation to model systems or processes by DSSPN is the simplicity and dynamic expressions in representing systems with multiple users and dynamic environments. In some situations, there may be redundant transitions in DSSPN models. In order to precisely and concisely describe systems, we offer the following theorems.

Theorem 4. *If there are some transitions with the same meaning in a DSSPN model, these transitions can be merged into one so that each transition is unique in a DSSPN model; that is, transition redundancy can be eliminated.*

Proof. Assume transitions t_2 and t'_2 have the same meaning. The preset and postset of t_2 are *t_2 and t_2^* , respectively. Meanwhile the preset and postset of t'_2 are ${}^*t'_2$ and t'^*_2 . Their enabling predicates and random switches are $f(t_2)$, $f(t'_2)$, $g(t_2)$, and $g(t'_2)$, respectively. Let us suppose t_1 is a forerunner transition of t_2 and t'_1 is a forerunner transition of t'_2 . The two transitions can be merged as follows:

- Transitions t_2 and t'_2 are merged into one transition t .
- The preset of t is ${}^*t = {}^*t_2 \cup {}^*t'_2$. For all $p, s \in {}^*t \cup t^*$, $p \neq s$; if their types and values are the same, that is, $G(p) = G(s)$ and $E(p) = E(s)$, then places p and s will be merged into one place, denoted by p' . Moreover, the type and the corresponding value remain the same.
- The enabling predicate is $f(t) = f(t_2) \vee f(t'_2)$, and the random switch is $g(t) = g(t_2) \wedge g(t'_2)$.
- Assume p and s will be merged; if $p \in {}^*t_2$ and $s \in {}^*t'_2$ or $p \in t_2^*$ and $s \in t'^*_2$, the weights of arcs relating to merged transition t and place p' are set as follows:

$$W(p', t) = \begin{cases} f(t_2) \longrightarrow W(p, t_2) \\ f(t'_2) \longrightarrow W(s, t'_2) \end{cases}$$

$$\text{or } W(t, p') = \begin{cases} f(t_2) \longrightarrow W(t_2, p) \\ f(t'_2) \longrightarrow W(t'_2, s). \end{cases} \quad (7)$$

Figure 1 shows an example to merge transitions t_2 and t_3 with the same meaning. For places p_2, p_3, p_4 , and p_5 , assume $G(p_2) = G(p_3)$, $E(p_2) = E(p_3)$, $G(p_4) = G(p_5)$, and $E(p_4) = E(p_5)$. Note that the weights of some arcs relating to merged transitions and places will be changed, where

$$\begin{aligned} w'_1 &= \begin{cases} f(t_2) \longrightarrow w_1 \\ f(t_3) \longrightarrow w_2 \end{cases} \\ w'_2 &= \begin{cases} f(t_2) \longrightarrow w_3 \\ f(t_3) \longrightarrow w_4 \end{cases} \\ w'_3 &= \begin{cases} f(t_3) \longrightarrow w_5 \\ 0. \end{cases} \end{aligned} \quad (8)$$

As illustrated in Theorem 4, a DSSPN model can eliminate redundant transitions. In DSSPN, each service or activity only corresponds to one transition that models a dynamic process or a system including multiple customers on a more convenient way. $\square$

Theorem 5. *A DSSPN can be transformed into a simple net [17], such that, for all $x, y \in P \cup T$, the preset of x is equal to that of y while the postset of x is equal to that of y only if x equals y ; that is,*

$$({}^*x = {}^*y) \wedge (x^* = y^*) \longrightarrow x = y, \quad \forall x, y \in P \cup T. \quad (9)$$

Proof. First, we consider the case of two places with the same preset and postset, as shown in Figure 2. If $G(p) = G(s)$ and $E(p) = E(s)$, we can easily transform it into a simple net just as illustrated in Theorem 4. Otherwise, we insert two new immediate transitions and two new places into the original model. Then the original net transforms into a simple one. Two things to note here are the settings of new arcs and places, that is, $W(p_1, d_1) = W(d_1, p_3) = W(p_3, t_2) = W(p_1, t_2)$ and $W(p_2, d_2) = W(d_2, p_4) = W(p_4, t_2) = W(p_2, t_2)$, while the settings of p_3 and p_4 are the same as those of p_1 and p_2 . Similarly, the case of two transitions with the same preset and postset can be proven, just as shown in Figure 3. $\square$

4. System Model Based on DSSPN

Nowadays, numerous cloud computing platforms are commercially available, such as Eucalyptus, Hadoop, and Amazon EC2 [31–33]. In this study, we take a typical cloud system by adopting fair scheduling algorithm as an example to construct a DSSPN model. Figure 4 illustrates the basic working process of tasks on a cloud platform in the light of the characteristics of a typical cloud system architecture. In the cloud system, jobs submitted by different customers may have different QoS requirements on computing time, memory

FIGURE 1: Equivalent transformation of merging two transitions with the same meaning.

FIGURE 2: Equivalent transformation of two places with the same preset and postset.

space, data traffic, response time, and so forth. That is, a typical cloud platform can be viewed as a multiuser multitask system involving multiple data sets with different types of processing jobs at the same time [32]. In a cloud platform, tasks are the basic processing units in the executive process. Dispatchers firstly select tasks according to a certain rule from the waiting queues and then assign them to appropriate resources adopting some scheduling policies. However, the properties of cloud computing, such as large scale, dynamics, heterogeneity, and diversity, present a range of challenges for performance evaluation of cloud systems and cloud optimization problem [34]. In order to verify the applicability and feasibility of DSSPN, we will model and analyze the performance of a typical cloud system based on DSSPN in this section.

4.1. Modeling Abstract. Without loss of generality, let us make the following assumptions for a typical cloud system:

- (1) There are n clients, denoted by c_i . Client i submits jobs into a waiting queue (i.e., pool i) with a capacity of b_i .
- (2) The minimum share of pool i is denoted by ms_i .
- (3) In fair scheduling, the set of priorities of each pool is $\{\text{VERY_HIGH}, \text{HIGH}, \text{NORMAL}, \text{LOW}, \text{VERY_LOW}\}$. In order to facilitate the analysis, the set of priorities are set to $\{5, 4, 3, 2, 1\}$.
- (4) The arrival process of tasks, submitted by client i , obeys the Poisson distribution with rate of λ_i . When the number of tasks submitted by client i exceeds b_i , the job submission is rejected.
- (5) In each waiting queue, the scheduling discipline is First Come First Served (FCFS).
- (6) There are m servers (denoted by s_j), each of which has r_j virtual machines (VMs) shared by n clients.

- (7) The service rate of each VM on s_j is μ_j with exponent distribution. In addition, the service rates are generally independent of each other. Note that the sum of ms_i is equal to or smaller than the total number of resources; that is, $\sum_{i=1}^n ms_i \leq \sum_{j=1}^m r_j$.

4.2. DSSPN Model of Fair Scheduling. Based on DSSPN, we model a typical cloud system adopting fair scheduling as a multiserver multiqueue system with n clients and m servers. The DSSPN model and involved notations are shown in Figure 5 and Notations. In order to simplify the description of the DSSPN model, we would not show the shared structures of servers.

All the places and transitions included in Figure 5 are described as follows ($1 \leq i \leq n$, $1 \leq j \leq m$):

- (1) c_j : a timed transition denotes client i submitting tasks with the firing rate of λ_i . The enabling predicate f_i of c_i is

$$f_i(M) : M(p_i) \leq b_i, \quad 1 \leq i \leq n. \quad (10)$$

That is, client i can submit tasks when the number of tasks is smaller than its capacity.

- (2) p_i : a place indicates the pool storing these tasks submitted by client i , and $K(p_i) = b_i$. In addition, $G(p_i) = MS$, $E(p_i) = ms_i$, and $g(p_i) = pl_i$, where ms_i means the guaranteed minimum share of pool i , pl_i represents the priority of pool i , and $pl_i \in \{5, 4, 3, 2, 1\}$ (just as elaborated in previous section).

- (3) R_j : a place stands for the status of server j ; for simplicity, it is not shown in Figure 5. $M(R_j)$ is the number of idle VMs of server j . $K(R_j) = r_j$, which means the total number of VMs on server j .

- (4) d_{ij} : an immediate transition indicates the execution of some scheduling or decision. The scheduling or decision is expressed by the enabling predicate f_{ij} and random switch g_{ij} associated with d_{ij} :

$$f_{ij} = \left(((AR_i < E(p_i)) \vee (\text{dem}_i \geq E(p_i))) \right. \\ \wedge \left(\sum_{j=1}^m M(q_{ij}) = 0 \right) \wedge (|SIDS(M)| > 0) \Big) \\ \vee ((\text{dem}_i \geq E(p_i)) \\ \wedge (\text{for } \forall h \neq i, E(p_h) \leq E(p_i)) \\ \wedge (|SIDS(M)| > 0)) \quad (11)$$

$$g_{ij} = \begin{cases} 5 \times \frac{1}{|UDLMS(M)|}, & \text{if } i \in UDLMS(M) \\ 4 \times \frac{1}{|UDGMS(M)|}, & \text{if } i \in UDGMS(M) \\ 3 \times \frac{1}{|ULMS(M)|}, & \text{if } i \in DLMS(M) \\ 2 \times \frac{1}{|MMS(M)|}, & \text{if } i \in MMS(M) \\ 0, & \text{otherwise.} \end{cases}$$

In this scheme, the highest priority is firstly given to the unallocated pools whose demand is smaller than its

FIGURE 3: Equivalent transformation of two transitions with the same preset and postset.

FIGURE 4: Basic working process of tasks on a cloud platform.

FIGURE 5: The refined DSSPN model of a typical cloud system adopting fair scheduling.

minimum share. Secondly, a higher priority is assigned to the unallocated pools whose demands are equal to or greater than its minimum share. Then, a normal priority is given to allocated pools included in $DLMS(M)$. Finally, if there are any unallocated VMs, these idle resources will be assigned to the pools included in $MMS(M)$.

(5) q_{ij} : a place indicates the queue receiving tasks with the capacity of r_{ij} ; that is, $K(q_{ij}) = r_{ij}$.

(6) s_{ij} : a timed transition stands for a VM on server j with the firing rate of μ_{ij} . The server j is shared by VM s_{ij} , where $1 \leq i \leq n$ and $1 \leq j \leq m$.

4.3. DSSPN Model of Classified Fair Scheduling. Although fair scheduling can share a cluster among different users as fair as possible, it does not make good use of resources without considering various workload types or resource diversity. Various types of workload with different requirements of resources consequently launch different kinds of tasks, usually including CPU intensive tasks and I/O intensive tasks. Hence, it is beneficial for improving hardware utilization to distinguish types of tasks and resources. For example, the processing time of a CPU intensive task in resources with stronger computing power would be shorter than that in other resources. Let d_{ik} denote the demand with type of k , and R_k represent the total number of VMs with type of k . Because of limited space, we only illustrate the improved part in classified fair scheduling (CFS) algorithm, shown in Algorithm 1. The remaining part

```

(1) Initialize the classification of all available resources;
(2) Initialize the classification of tasks when they are submitted to pools;
(3) for each pool  $i$  whose demand  $\leq$  its minimum share do
(4)   for each type  $k$  do
(5)     if  $d_{ik} \leq R_k$  then
(6)       allocate the  $d_{ik}$  resources with type of  $k$ ;
(7)        $R_k - = d_{ik}$ ;
(8)     else
(9)       allocate the  $R_k$  resources with the type of  $k$ ;
(10)       $d_{ik} - = R_k$ ;
(11)      allocate  $d_{ik}$  resources with other types, while satisfying  $R_j \geq d_{ik}$ ,  $j \in \{1, 2, \dots, l\}$ ;
(12)    end if
(13)  end for
(14) end for
(15) for (each pool  $i$  whose demand  $>$  its minimum share)  $\wedge$  (remaining idle unallocated VMs) do
(16)   add the similar process as described above in light of the assigning decision of each pool;
(17) end for

```

ALGORITHM 1: The improved part of fair scheduling in CFS.

of CFS is similar to that of fair scheduling presented by Zaharia et al. [35].

The descriptions of places and transitions in Figure 6 are similar to that in Figure 5. We will not reiterate them here. In order to facilitate understanding, we only emphasize the meaning of the subscripts for places and transitions. The subscript i denotes client i , the subscript k represents tasks with type k , and the subscript j describes server j . There are some differences on the values of some notations between Figures 5 and 6. The enabling rate of c_{ik} is λ_{ik} , and $K(q_{ikj}) = b_{ikj}$, where $\sum_{i=1}^n \sum_{k=1}^l b_{ikj} = b_j$. The enabling rate of s_{ikj} is μ_{ikj} , where $\sum_{i=1}^n \sum_{k=1}^l \mu_{ikj} = \mu_j$. In addition, the servers are classified; that is, $g(p_{ikj}) \in \{1, 2, \dots, l\}$. The differences on the values between Figures 5 and 6 are described as follows:

$$\begin{aligned}
 AR_i &= \sum_{k=1}^l \sum_{j=1}^m M(q_{ikj}) \times \mathbb{Z} \\
 dem_i &= \sum_{k=1}^l M(p_{ik}) + AR_i \\
 SIDS(M) &= \left\{ h \mid \sum_{i=1}^n \sum_{k=1}^l M(q_{ikh}) \leq b_h \right\}.
 \end{aligned} \tag{12}$$

Let y_{ikj} denote the service rate of s_{ikj} provided for the tasks in queue q_{ikj} :

$$y_{ikj} = \begin{cases} pl \times \mu_{ikj}, & \text{if } g(p_{ikj}) = k \\ pl' \times \mu_{ikj}, & \text{otherwise.} \end{cases} \tag{13}$$

Note that $pl > pl'$. The scheme would ensure tasks whose types are the same as that of servers served at a higher priority.

The major difference between fair scheduling (FS) and CFS is that tasks and resources diversity are taken into

account. Without loss of generality, assume tasks and resources can be divided into l categories. The refined DSSPN model of CFS is shown in Figure 6. Note that Algorithm 1 only describes the improved part of FS [35], that is, the decision procedure to allocate resources with various types to different kinds of tasks.

4.4. Analysis and Solution of DSSPN Models. Although the problem of state explosion is improved to some extent in DSSPN compared to other forms of Petri Nets, it is still difficult to analyze the performance of large-scale cloud systems. Model refinement techniques elaborated by Lin [17] can develop compact models and expose the independence as well as the interdependent relations between submodels of an original model. Model refinement can lay a foundation for the decomposition and analysis of models. Consequently, the refinement of models has become a necessary step of the model design. The refinement methods have been applied to the performance evaluation of high speed network and shared resources systems [17, 36].

4.4.1. Equivalent Refinement Model and Markov Model. In this section, we will make further use of enabling predicates and random switches of transitions to refine the model proposed above. Figure 7 shows the equivalent model for models in Figures 5 and 6, while Figure 8 describes the equivalent Markov model of Figure 7.

Comparing Figure 7 with Figures 5 and 6, it can be found that the refined model is easier to understand and significantly reduces the state space by deleting any unnecessary vanishing states. In addition, refined model greatly decreases the complexity in performance evaluation because of structural similarities of submodels.

In Figure 7, immediate transitions and place p_i (or p_{ik}) and related arcs are removed from Figure 5 (or Figure 6), where $1 \leq i \leq n$ and $1 \leq k \leq l$. The enabling predicates

FIGURE 6: The refined DSSPN model of a typical cloud system adopting CFS algorithm.

FIGURE 7: The refined DSSPN model of Figures 5 and 6.

and random switches associated with d_{ij} and c_{ij} (or d_{ikj} and c_{ikj}) have changed, while others are remaining the same. The random switch of transition c_{ij} is defined as follows:

$$g_{ij}(M) : \lambda_i \times g_{ij}(M). \quad (14)$$

The enabling switch of transition c_{ikj} is

$$g_{ikj}(M) : \lambda_{ik} \times g_{ikj}(M). \quad (15)$$

4.4.2. Parameters Analysis. In order to obtain the steady-state probabilities of all states, a state transition matrix can be constructed based on the state transition rate and Markov chain illustrated in Figure 8. Then, the performance parameters of the modeled cloud system can be discussed. Let $P[M]$ denote the steady-state probability of M .

The throughput of transition t is denoted as T_t :

$$T_t = \sum_{M \in H} P(M) \times \lambda_t, \quad (16)$$

where H is a set of all markings under which transition t is enabled with the enabling rate of λ_t in marking M .

The average number of tokens in place p is denoted as N_p :

$$N_p = \sum_j j \times P[M(p) = j]. \quad (17)$$

The throughput is a crucial indicator of the system performance. Let T_{ij} (or T_{ikj}) indicate the throughput of

subsystem A_{ij} (or A_{ikj}). According to the illustration in [16], the throughput of the model can be calculated as follows:

$$T = \sum_{i=1}^n \sum_{j=1}^m T_{sij} \quad (18)$$

$$\text{or } T = \sum_{i=1}^n \sum_{k=1}^l \sum_{j=1}^m T_{sijk}.$$

Another important indicator is response time. R_{ij} (or R_{ikj}), R_i , and R denote the response time of subsystem A_{ij} (or A_{ikj}), client i , and the system, respectively:

$$\begin{aligned} R_{ij} &= \frac{D_{qij}}{T_{sij}} \\ R_i &= \sum_{j=1}^m \left(T_{sij} \times R_{ij} \setminus \sum_{h=1}^m T_{siah} \right) \\ R &= \sum_{i=1}^n \left(R_i \times \sum_{h=1}^l \frac{T_{siah}}{T} \right) \\ R_{ikj} &= \frac{D_{qikj}}{T_{sijk}} \\ R_{ik} &= \sum_{j=1}^m \left(T_{sijk} \times R_{ikj} \setminus \sum_{h=1}^m T_{sikh} \right) \\ R_i &= \sum_{k=1}^l \left(T_{sik} \times R_{ik} \setminus \sum_{h=1}^l T_{siah} \right) \\ R &= \sum_{i=1}^n \left(R_i \times \sum_{h=1}^l \frac{T_{siah}}{T} \right). \end{aligned} \quad (19)$$

FIGURE 8: The equivalent Markov model of Figure 7.

The average rejection rate of tasks in the cloud system with FS at time t is expressed by $AER(t)$:

$$AER(t) = \frac{\sum_{i=1}^n \left(\sum_{j=1}^m P(M(p_{ij})) > b_i \right)}{n \times t}. \quad (20)$$

The average rejection rate of tasks in the cloud system with CFS at time t is expressed by $AER'(t)$:

$$AER'(t) = \frac{\sum_{i=1}^n \left(\sum_{k=1}^l \sum_{j=1}^m P(M(p_{ikj})) > b_i \right)}{n \times t}. \quad (21)$$

The average idle rate of servers in the cloud system with FS at time t is expressed by $AUR(t)$:

$$AUR(t) = \frac{\sum_{j=1}^m \left(\sum_{i=1}^n \sum_{y=0}^t (1 - P(\text{enabled}(s_{ij}(y)))) \right)}{m \times t}, \quad (22)$$

where $P(\text{enabled}(s_{ij}(y)))$ means the probability that transition $s_{ij}(y)$ can fire at time y .

The average idle rate of servers in the cloud system with CFS at time t is expressed by $AUR'(t)$:

$$AUR'(t) = \frac{\sum_{j=1}^m \left(\sum_{i=1}^n \sum_{k=1}^l \sum_{y=0}^t (1 - P(\text{enabled}(s_{ikj}(y)))) \right)}{m \times t}, \quad (23)$$

where $P(\text{enabled}(s_{ikj}(y)))$ means the probability that transition $s_{ikj}(y)$ can fire at time y .

In a multiuser multiserver cloud system, the performance parameters include the state changes of waiting queues and the service rates of shared servers. The improvement of throughput and the decrease of response time can be realized by furthest parallelizing the operations of n servers. In other words, load balance should be maintained.

5. Case Study and Evaluation

In this section, we provide a case to study the performance of the DSSPN model based on steady-state probabilities. To verify the applicability and feasibility of DSSPN, we

TABLE 1: Number of states and fired transitions.

	1 machine	2 machines	3 machines	4 machines
Reachable states	283	569	1088	1594
Fired transitions	923	1977	3928	5842

only study some performance indicators of FS and CFS by means of the above method. In addition, Stochastic Petri Net Package (SPNP) is applied to automatically derive the analytic solution of performance for the DSSPN model. This is beneficial in modeling and evaluating the performance of cloud systems, because the number of states might reach thousands even only including few machines, shown in Table 1. Table 2 describes the parameter settings in the simulation.

The simulation was conducted to the cloud system consisting of 3 servers, 2 customers, and 2 categories. That is, there are 4 waiting queues in FS, while 8 waiting queues are existing in CFS. Assume $g(s1) = 1$ and $g(s2) = 2$. The task submitted by each client can be classified into 2 groups. In the simulation scenario, there are 4 VMs that can be running on server 1 simultaneously, while 5 VMs are running on server 2.

As shown in Figure 9, when the configuration parameters are identical, the values of system average throughput in steady state of CFS are significantly greater than that of fair scheduling. Figure 10 describes the average delay, which is depicted by average response time in DSSPN models, in steady state of CFS and FS. Apparently, the average delay of CFS is prominently smaller than that of fair scheduling. That is, CFS is a powerful way to decrease waiting time for users. As can be seen from Figure 9, the difference of average throughput between CFS and FS can reach 14.8 when $\lambda_1 = 6/\text{sec}$, while the maximal difference of average delay between CFS and FS is 5.75 sec when $\lambda_1 = 6/\text{sec}$.

Figure 11 illustrates that average completion time of CFS is significantly better than that of FS. The simulation results present that the novel scheme (CFS) can efficiently increase the average system throughput and thus can improve utilization of resources. This means that it can realize economic benefits in the commercial cloud services.

Moreover, Figures 9, 10, and 11 also show that the performance of CFS is generally better than that of fair scheduling

TABLE 2: Parameter settings in simulation.

Algorithm	λ_2/λ_1	ms_1	ms_2	b_{1j}	b_{2j}	b_{1kj}	b_{2kj}	μ_{1j}	μ_{2j}	μ_{1kj}	μ_{2kj}	pl	pl'	b
FS	2/3	3	4	10	8			3	2					30
CFS	2/3	3	4			10	8			3	2	2	1	30

FIGURE 9: Average throughput when $\lambda_1 = \{3, 4, 5, 6, 7\}$. FIGURE 11: Average completion time when $\lambda_1 = \{3, 4, 5, 6, 7\}$. FIGURE 10: Average response time when $\lambda_1 = \{3, 4, 5, 6, 7\}$. FIGURE 12: Average rejection rate at different service time t .

across all circumstances, especially at heavy load. However, queues cannot be simulated efficiently, because these schemes are only based on the current state of queues but ignore the dynamics of task in the queues. The simulation results are different by setting different input rates due to incapability of predicting the future state of the waiting queues.

Figure 12 shows how the average rejection rate of the cloud system changes as service time goes on. When the task request in one waiting pool is up to 30, the system will reject new requests submitted by the corresponding user. When $1 \leq t \leq 10$, the average rejection rate of FS is higher than that of CFS. The differences between FS and CFS in the average rejection rate are up to 40.08% at service time of 5 seconds. In addition, Figure 12 also illustrates that, along with the operation of the cloud system, the average reject rate increases with the accumulation of backlogs in waiting queues.

Figure 13 illustrates how the scheduling strategies affect the average resource utilization of the system. When $0 \leq t \leq 10$, the average idle rate of servers in FS is lower than that

in CFS. The maximal differences between FS and CFS in the average idle rate of servers at different service times are 4%. It means that there is potential to achieve higher utilization rate with CFS algorithm by increasing the system throughput.

6. Conclusion

In this paper, we propose the definition of DSSPN that can easily describe the multiple clients systems based on cloud services, such as a typical cloud platform. The major motivation to model systems or processes by DSSPN is its simplicity and dynamic expressions to represent systems with multiple users and dynamic environments. Moreover, we further elaborate dynamic property of DSSPN and analyze some properties of DSSPN. In the following section, for some shortcomings of fair scheduling, the classified fair scheduling (CFS) algorithm is proposed taking into consideration jobs and resources diversity.

In the real world, a typical cloud system is shared by multiple applications including production applications, batch

FIGURE 13: Average idle rate of servers at different service time t .

jobs, and interactive jobs. Meanwhile, different applications have different requirements on hardware resources and QoS parameters. Therefore, we adopt the multiuser multiserver model to analyze the performance analysis and design DSSPN models for FS and CFS. In order to avoid the state space explosion, the analysis techniques and model refinement techniques are applied to performance evaluation of their DSSPN models. Finally, SPNP is used to obtain some key indicators of QoS; that is, system average throughput, response time, and average completion time are compared between the two schemes. Just as shown from Figures 9–11, the performance of CFS is generally better than that of fair scheduling across all circumstances, especially at heavy load.

The following topics are of high interest for future work:

- (1) Other quality metrics, such as energy consumption and cost, should be analyzed.
- (2) The proposed model is without considering local task migrations among servers in the same data center.
- (3) The theoretical derivations between simulation results and actual cloud systems will be studied.

Notations

Involved Notations and Equations in Figure 5

- AR_i : The VMs allocated to pool i ; $AR_i = \sum_{j=1}^m M(q_{ij})$
- sms : The smallest minimum share among some pools; $sms = \min\{E(p_h), h \in DGMS(M)\}$
- dem_i : The demand of pool i ; $dem_i = M(p_i) + AR_i$
- $sdem$: The smallest demand among some pools; $sdem = \min\{dem_h, h \in DGMS(M)\}$
- def_i : The deficit between dem_i and ms_i ; $def_i = E(p_i) - AR_i$
- SIDS: The set of all servers that has idle slot waiting to be assigned; $SIDS(M) = \{h \mid \sum_{i=1}^n M(q_{ih}) \leq b_h\}$
- DLMS: The set of all pools whose demand is less than its minimum share; $DLMS(M) = \{h \mid dem_h < E(p_i), 1 \leq h \leq n\}$
- UDLMS: The set of all unallocated pools whose demand is less than its minimum share; $UDLMS(M) = \{i \mid \sum_{j=1}^m M(p_{ij}), i \in DLMS(M)\}$

- DGMS: The set of all pools whose demand is equal to or larger than its minimum share; $DGMS(M) = \{h \mid dem_h \geq E(p_i), 1 \leq h \leq n\}$
- UDGMS: The set of all pools in DGMS without any allocated resources at the current status; $UDGMS(M) = \{i \mid \sum_{j=1}^m M(q_{ij}) = 0, i \in DGMS(M)\}$
- MMS: The set of pools with the smallest minimum share in DGMS; $MMS(M) = \{h \mid E(p_i) = sms, h \in DGMS\}$.

Competing Interests

The authors declare that they have no competing interests.

Acknowledgments

This work was partially supported by the National Natural Science Foundation of China (nos. 61172063, 61272093, and 61572523) and special fund project for work method innovation of Ministry of Science and Technology of China (no. 2015IM010300).

References

- [1] P. Mell and T. Grance, *The NIST Definition of Cloud Computing*, Recommendations of the National Institute Standards and Technology-Special Publication 800-145, NIST, Washington, DC, USA, <http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800>.
- [2] S. Singh and I. Chana, "QRSF: QoS-aware resource scheduling framework in cloud computing," *Journal of Supercomputing*, vol. 71, no. 1, pp. 241–292, 2014.
- [3] J. Baliga, R. W. A. Ayre, K. Hinton, and R. S. Tucker, "Green cloud computing: balancing energy in processing, storage and transport," *Proceedings of the IEEE*, vol. 99, no. 1, pp. 149–167, 2011.
- [4] B. P. Rimal, A. Jukan, D. Katsaros, and Y. Goeleven, "Architectural requirements for cloud computing systems: an enterprise cloud approach," *Journal of Grid Computing*, vol. 9, no. 1, pp. 3–26, 2011.
- [5] A. L. Bardsiri and S. M. Hashemi, "A review of workflow scheduling in cloud computing environment," *International Journal of Computer Science and Management Research*, vol. 1, no. 3, pp. 348–351, 2012.
- [6] Y. Chawla and M. Bhonsle, "A study on scheduling methods in cloud computing," *International Journal of Emerging Trends and Technology in Computer Science*, vol. 1, no. 3, pp. 12–17, 2012.
- [7] L. Chuang, *Stochastic Petri Net and System Performance Evaluation*, Tsinghua University Press, Beijing, China, 2005.
- [8] M. K. Molloy, "Discrete time stochastic Petri nets," *IEEE Transactions on Software Engineering*, vol. 11, no. 4, pp. 417–423, 1985.
- [9] M. A. Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis, "Modelling with generalized stochastic petri nets," *ACM SIGMETRICS Performance Evaluation Review*, vol. 26, no. 2, p. 2, 1998.
- [10] W. M. P. van der Aalst, "The application of Petri nets to workflow management," *Journal of Circuits, Systems and Computers*, vol. 8, no. 1, pp. 21–66, 1998.

- [11] K. Jensen, *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use*, Springer, New York, NY, USA, 2013.
- [12] K. Jensen and G. Rozenberg, *High-Level Petri Nets: Theory and Application*, Springer Science and Business Media, Berlin, Germany, 2012.
- [13] N. Ferry, A. Rossini, F. Chauvel, B. Morin, and A. Solberg, "Towards model-driven provisioning, deployment, monitoring, and adaptation of multi-cloud systems," in *Proceedings of the IEEE 6th International Conference on Cloud Computing (CLOUD '13)*, pp. 887–894, IEEE, Santa Clara, Calif, USA, June 2013.
- [14] B. P. Rimal, E. Choi, and I. Lumb, "A taxonomy and survey of cloud computing systems," in *Proceedings of the 5th International Joint Conference on INC, IMS and IDC*, pp. 44–51, Seoul, Republic of Korea, August 2009.
- [15] M. Llorens and J. Oliver, "Marked-controlled reconfigurable workflow nets," in *Proceedings of the 8th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC '06)*, pp. 407–413, Timisoara, Romania, September 2006.
- [16] L. Lei, C. Lin, J. Cai, and X. Shen, "Performance analysis of wireless opportunistic schedulers using stochastic Petri nets," *IEEE Transactions on Wireless Communications*, vol. 8, no. 4, pp. 2076–2087, 2009.
- [17] C. Lin, "On refinement of model structure for stochastic Petri Nets," *Journal of Software*, vol. 1, p. 017, 2000.
- [18] Y. Xia, M. Zhou, X. Luo, S. Pang, and Q. Zhu, "Stochastic modeling and performance analysis of migration-enabled and error-prone clouds," *IEEE Transactions on Industrial Informatics*, vol. 11, no. 2, pp. 495–504, 2015.
- [19] S. Ostermann, A. Iosup, N. Yigitbasi, R. Prodan, T. Fahringer, and D. Epema, "A performance analysis of EC2 cloud computing services for scientific computing," in *Cloud Computing*, D. R. Avresky, M. Diaz, A. Bode, B. Ciciani, and E. Dekel, Eds., vol. 34 of *Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering*, pp. 115–131, Springer, Berlin, Germany, 2010.
- [20] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, "CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and Experience*, vol. 41, no. 1, pp. 23–50, 2011.
- [21] L. Bautista, A. Abran, and A. April, "Design of a performance measurement framework for cloud computing," *Journal of Software Engineering and Applications*, vol. 5, no. 2, pp. 69–75, 2012.
- [22] Y. Mei, L. Liu, X. Pu, and S. Sivathanu, "Performance measurements and analysis of network I/O applications in virtualized cloud," in *Proceedings of the IEEE 3rd International Conference on Cloud Computing*, pp. 59–66, Miami, Fla, USA, July 2010.
- [23] Y. Cao, H. Lu, X. Shi, and P. Duan, "Evaluation model of the cloud systems based on Queuing Petri net," in *Algorithms and Architectures for Parallel Processing*, pp. 413–423, Springer International, Cham, Switzerland, 2015.
- [24] S. Kounev and C. Dutz, "QPME: a performance modeling tool based on queueing Petri Nets," *ACM SIGMETRICS Performance Evaluation Review*, vol. 36, no. 4, pp. 46–51, 2009.
- [25] G. Fan, H. Yu, and L. Chen, "A formal aspect-oriented method for modeling and analyzing adaptive resource scheduling in cloud computing," *IEEE Transactions on Network and Service Management*, vol. 13, no. 2, pp. 281–294, 2016.
- [26] M. Reynolds, "An axiomatization of full computation tree logic," *The Journal of Symbolic Logic*, vol. 66, no. 3, pp. 1011–1057, 2001.
- [27] K. Jensen and L. M. Kristensen, *Colored Petri Nets: Modelling and Validation of Concurrent Systems*, Springer, 2009.
- [28] M. C. Ruiz, J. Calleja, and D. Cazorla, "Petri nets formalization of map/reduce paradigm to optimise the performance-cost tradeo," in *Proceedings of the IEEE Trustcom/BigDataSE/ISPA*, vol. 3, pp. 92–99, 2015.
- [29] A. V. Ratzer, L. Wells, H. M. Lassen et al., "CPN tools for editing, simulating, and analysing coloured Petri nets," in *Applications and Theory of Petri Nets 2003*, pp. 450–462, Springer, 2003.
- [30] C. Lin and D. C. Marinescu, "Stochastic high-level Petri nets and applications," in *High-Level Petri Nets*, pp. 459–469, Springer, Berlin, Germany, 1991.
- [31] D. Nurmi, R. Wolski, C. Grzegorzczak et al., "The eucalyptus open-source cloud-computing system," in *Proceedings of the 9th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID '09)*, pp. 124–131, Shanghai, China, May 2009.
- [32] T. White, *Hadoop: The Definitive Guide*, O'Reilly Media, 2012.
- [33] J. Peng, X. Zhang, Z. Lei, B. Zhang, W. Zhang, and Q. Li, "Comparison of several cloud computing platforms," in *Proceedings of the 2nd International Symposium on Information Science and Engineering*, pp. 23–27, IEEE, Shanghai, China, December 2009.
- [34] J. Xu, J. Tang, K. Kwiat, W. Zhang, and G. Xue, "Enhancing survivability in virtualized data centers: a service-aware approach," *IEEE Journal on Selected Areas in Communications*, vol. 31, no. 12, pp. 2610–2619, 2013.
- [35] M. Zaharia, D. Borthakur, J. S. Sarma et al., "Job scheduling for multiuser mapreduce clusters," Tech. Rep. UCB/EECS-2009-55, EECS Department, University of California, Berkeley, Calif, USA, 2009.
- [36] C. Lin, "A model of systems with shared resources and analysis of approximate performance," *Chinese Journal of Computers*, vol. 20, pp. 865–871, 1997.