Towards Optimal Collaboration of Policies in the Two-Phase Scheduling of Cloud Tasks

Cong Xu[1,2], Jiahai Yang[1,2], Di Fu[1,2], and Hui Zhang[1,2]

[1] Institute for Network Sciences and Cyberspace, Tsinghua University, Beijing 100084, China
[2] Tsinghua National Laboratory for Information Science and Technology (TNList), Tsinghua University, Beijing 100084, China
xucong10@mails.tsinghua.edu.cn, {yang,hzhang}@cernet.edu.cn, fudi@bupt.edu.cn

Abstract. The use of virtualization technology makes software applications more scalable and cost effective when they are deployed over cloud computing platforms, but virtualization technology also brings challenges to task scheduling over cloud. The commonly used list scheduling schemes split the scheduling process into two phases: ordering and dispatching. However, majorities of recent researches about scheduling of cloud tasks concentrate on optimizing the schedulers' performance in one phase, but seldom consider the collaborations of scheduling policies used in different phases. This paper summarizes some representative ordering and dispatching policies used in list schedulers, models the execution processes of these ordering and dispatching policies using Stochastic Petri Nets (SPN), and simulates the list scheduling process of cloud tasks. Based on the modeling and experimental results, we further evaluate which composition of ordering and dispatching policies provides optimal performance in the two-phase scheduling process of cloud tasks.

Keywords: list scheduling, ordering policy, dispatching policy, Stochastic Petri Net, queuing theory.

1 Introduction

Cloud computing service providers are interested in improving the task scheduling mechanisms to provide better Quality of Service (QoS) to their users. However, with the increasing scale of cloud computing platform, a single cloud may host and provide more and more diverse services, which brings new challenges to the scheduling of cloud tasks. The dynamicity of virtual environment [1], diversity of software applications [2] and elasticity of cloud services [3] exacerbate the dynamicity and complexity of the scheduling mechanisms: a cloud task's processing rate tends to have significant variations since the virtual resources allocated to the same task may be different. Hence, how to optimally dispatch the task requests to the proper virtual resources for processing is an important issue. Moreover, there will also be times

C.-H. Hsu et al. (Eds.): NPC 2014, LNCS 8707, pp. 306–320, 2014.

when a task's arrival rate is greater than its maximum possible processing rate, then how to determine an optimal task execution order to reduce the peak load of a cloud computing platform is another significant issue.

List schedulers [4] are widely used in the dynamic scheduling of cloud tasks. As shown in Fig. 1, a list scheduler splits the scheduling process into two phases: one is the ordering phase, where tasks are sorted according to a specific ordering policy; the other is the dispatching phase, where tasks are dispatched to the allocated virtual resources (VMs) for processing according to a specific dispatching policy. The existing studies on task scheduling over cloud mostly concentrate on improving the schedulers' performance in one phase, such as optimizing the task processing sequence in the first phase to minimize the overall response time [4-11]; or optimizing the assignment of the task requests to the VMs to improve the utilization of cloud resources [12-18] and to ensure the fairness of each task [19-21]. However, only few researches have further studied the optimal collaboration of ordering and dispatching policies in different phases [22].

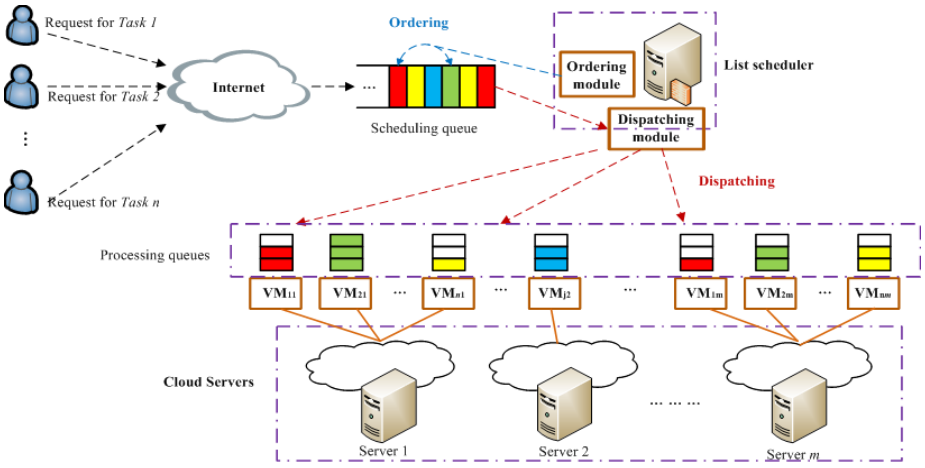

Fig. 1. Two-phase task scheduling process in a cloud computing platform

This paper summarizes some representative ordering and dispatching policies which are widely used in both industry and academic communities. Using Stochastic Petri Nets (SPN), we theoretically and consistently model the execution processes of these ordering and dispatching policies for the first time, and analyze the performance of a list scheduler under different compositions of policies. Based on the modeling and experimental results, we further evaluate which composition of ordering and dispatching policies provide optimal performance in the two-phase scheduling process of cloud tasks.

The rest of this paper is organized as follows. Section 2 summarizes representative ordering and dispatching policies. Section 3 proposes a Stochastic Petri Net model to describe the scheduling process, studies some features of different ordering and dispatching policies and analyzes the performance of a list scheduler under different ordering and dispatching policies. Section 4 evaluates the performance of the list scheduler under different collaborations of ordering and dispatching policies. Section 5 concludes the paper.

2 Scheduling Policies in the Two Phases

This section summarizes some representative ordering and dispatching policies used in the list scheduler of cloud computing platforms. These policies are commonly used in both industry and academic communities.

2.1 Ordering Policies in the First Phase

When the average task arrival rate exceeds the platform's processing rate, the unhandled tasks will be stalled in the scheduling queue, and the platform tends to be overloaded. At this time, an appropriate ordering policy is needed to determine an optimal task processing sequence to slow down the growth rate of the scheduling queue. The commonly used ordering policies are listed below:

Random Ordering Policy: This policy randomly chooses one type of the cloud tasks for processing. All the tasks have equal chances to be scheduled in this policy, which achieves better fairness of tasks.

First in First Out (FIFO) Policy: This is a simple and widely used ordering policy, which makes the tasks to be executed in the order they arrived. This policy ensures all the tasks suffer almost the same waiting delay in the ordering phase.

Shortest Remaining Time First (SRTF) Policy: This policy ensures the smallest task in the scheduling queue to be scheduled first [5], aiming to improve the responsiveness of the scheduler when the load is heavy.

Longest Remaining Time First (LRTF) Policy: This policy ensures the largest task in the scheduling queue to be scheduled first, aiming to improve the throughput of the scheduler.

Myopic MaxWeight Policy: This policy can be viewed as an improvement of SRTF or LRTF policy. Neither SRTF nor LRTF policy is a starvation-free ordering since the large sized tasks under SRTF or the small sized tasks under LRTF are likely to wait forever when the load is heavy [22]. To solve this problem, a weight is assigned to each task in the scheduling queue in this policy, and the value of a task's weight is the task's size (for LRTF) or task's execution rate (for SRTF) multiplies its waiting time. This policy schedules the task with the maximum weight first, which ensures the tasks suffering long waiting time have chances to be processed.

2.2 Dispatching Policies in the Second Phase

The dispatching policies assign tasks to virtual resources for processing. An appropriate dispatching policy optimizes the resource utilization of a cloud platform and further improves the performance of the scheduler. The commonly used dispatching policies are listed below:

Random Routing Policy (RR): This policy randomly dispatches a task to one of the VMs for processing, which ensures each allocated VM has an equal chance to process a task.

Shortest Queue Routing Policy (SQR): This policy dispatches a task to the VM with the shortest processing queue, which achieves better fairness by balancing the load on each VM [19, 20].

Shortest Expected Delay Routing Policy (SEDR): This policy dispatches a task to the VM with the shortest processing delay, which can be viewed as a generalization of the SQR policy for homogeneous servers [12].

Overall Shortest Expected Delay Routing Policy (OSEDR): This policy dispatches a task to the appropriate VM to keep the overall processing delay of a server minimum [21], it optimizes the sum of the execution delays of all the tasks on each server.

3 Modeling and Analysis of Scheduling Policies

In this section, we describe the list scheduling process using the SPN model. By modeling different ordering and dispatching policies, we formulate some performance metrics (e.g. average resource utilization of a service). Furthermore, we evaluate the performance of a list scheduler according to our modeling results.

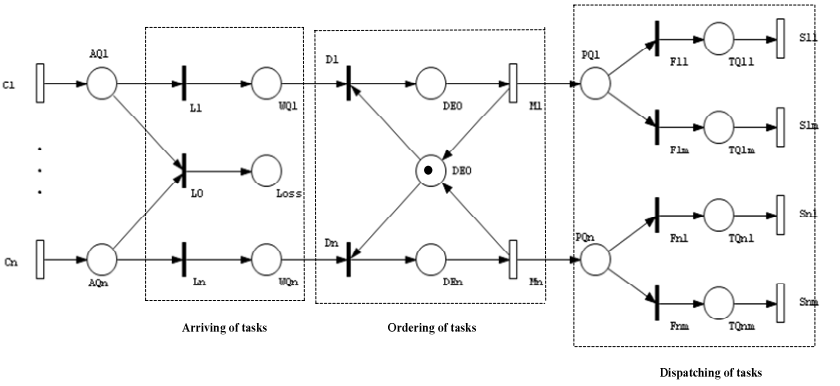

Fig. 2. SPN model of the two-phase task scheduling process

3.1 Description of Scheduling Process Using SPN Model

Fig. 2 shows the SPN model of the two-phase list scheduling process shown in Fig. 1. It describes a scenario that a cloud computing platform comprises *m* servers, and processes *n* types of different tasks. Some important notations and definitions used in the SPN model and subsequent derivations are illustrated in Table 1. The scenario considered here is dynamic or online scheduling of small sized tasks onto a fixed non-preemptive cloud system.

Table 1. Summary of Key Notations and Definitions

Notations	Definitions
C_i.	Timed transition, indicates the arrvial of task i
λ_i	Poisson arrival rate of the task i
L_i.	Free choice conflict transition, decides if task i can be buffered in the global scheduling queue
WQ_i	Place, indicate a logical sub scheduling queue of task i
$WQ_i(t)$	Length of logical sub scheduling queue WQ_i at time t
D_i	Free choice conflict transition, decides if task i can be processed under a specific ordering policy
$P_i[D(t)]$	Probability that a request of task i passes D_i at time t
M_i	Transportation of task i from global scheduling queue to a specific VM for processing
F_{ij}	Free choice conflict transition, decides if task i can be dispatched to the allocated VM located on server j
$P_{ij}[F(t)]$	Probability that requests of task i pass transition F_{ij} at time t
TQ_{ij}	Processing queue of task i on server j
$TQ_{ij}(t)$	Length of processing queue TQ_{ij} at time t
l_{max}	Maximum length of each processing queue
λ_{ij}	Arrival rate of task i to its processing queue on server j
S_{ij}	Timed transition, indicates the processing of task i on server j
μ_{ij}	Processing rate of task i on its allocated VM located on server j
ρ_{ij}	Utilization of the VM allocated to task i on server j
T_{avg}	Average completion time of a task
$\rho_{avg}(i)$	Average resource utilization of task i
$\rho_{worst}(i)$	Worst case resource utilization of task i

For analytic tractability, we assume the arrival of task i satisfies Poisson distribution with mean rate λ_i. After arriving, a token representing task i in the SPN will come to place AQ_i and decide if it can pass through transition L_i to be buffered in the global scheduling queue. For analytic tractability, we assume that the size of the global scheduling queue is properly set to avoid task losses.

An ordering policy determines the value of $P_i[D(t)]$, which is the probability that a request of task i in the scheduling queue can be handled at time t. To accurately describe the ordering policies, we assume that there exists a sub logical scheduling queue WQ_i which buffers the arrived requests for task i. Suppose the arrival rate of task i changes to $\overline{\lambda_i}$ after the token passes through transition L_i, then $\overline{\lambda_i}=\lambda_i P_i[D(t)]$.

A dispatching policy determines the enabling function of each transition F_{ij}, which further determines the value of $P_{ij}[F(t)]$. $P_{ij}[F(t)]$ is the probability that a request of task i is dispatched to a VM located on server j for processing. Finally, the arrival rate of task i to its processing queue TQ_{ij} on server j is: $\lambda_{ij}=\overline{\lambda_i} P_{ij}[F(t)] =\lambda_i P_i[D(t)] P_{ij}[F(t)]$.

Then the tasks in the processing queue will be serially processed by the program running on the corresponding VM, which is similar to the scenario described in *M/G/*1 model. For analytic tractability, we assume the processing rate of each task *i* on server *j* is exponentially distributed with mean value μ_{ij}, and the tasks' sizes are sorted in ascending order from task 1 to task *n*. Suppose the VMs on the same server have the same configuration, then the execution rates of the tasks dispatched to the same server satisfy: $\mu_{1j} \geq \mu_{2j} \geq \ldots \geq \mu_{nj} \quad (1 \leq j \leq m)$.

Since the utilization metrics and responsiveness metrics are the most frequently used metrics to evaluate the performance of a scheduling policy, we derive the expressions of average resource utilization and average completion time of task *i* based on existing modeling results introduced in queue theory [24]:

$$\rho_{avg}(i) = \sum_{j=1}^{m} \frac{\rho_{ij}}{m} = \sum_{j=1}^{m} \frac{\lambda_i \cdot \lim_{t\to\infty} P_i[D(t)] P_{ij}[F(t)]}{\mu_{ij} m} \tag{1}$$

$$T_{avg} = \sum_{i=1}^{n}\sum_{j=1}^{m} \frac{E[TQ_{ij}(t)]}{\lambda_{ij}(1-\frac{1-\rho_{ij}^{l_{\max}+1}}{1-\rho_{ij}})mn} = \frac{1}{mn}\sum_{i=1}^{n}\sum_{j=1}^{m} \lim_{t\to\infty} \frac{\rho_{ij}(1-\rho_{ij}^{l_{\max}+1}) - (l_{\max}+1)\rho_{ij}^{l_{\max}+1}(1-\rho_{ij})}{\lambda_i P_i[D(t)] P_{ij}[F(t)](1-\rho_{ij}^{l_{\max}+1}) + \rho_{ij} - 1} \tag{2}$$

$$\forall i, j, \ \sum_{i=1}^{n} P_i[D(t)] = 1 \ \ \sum_{j=1}^{m} P_{ij}[F(t)] = 1$$

The value of $\rho_{avg}(i)$ and T_{avg} are determined by the value of $P_i[D(t)]$ and $P_{ij}[F(t)]$. Hence, we need to derive the expressions of $P_i[D(t)]$ and $P_{ij}[F(t)]$ under different ordering and dispatching policies.

3.2 Modeling of Ordering Policies

First, we derive the expressions of $P_i[D(t)]$ in the steady-state under different ordering policies.

Random Ordering Policy. Since this policy randomly chooses one type of the cloud tasks for processing, the probability is time-independent, so the expression of $P_i[D(t)]$ in the steady-state is:

$$P_i[D(t)] = \lim_{t\to\infty} P_i[D(t)] = 1/n$$

FIFO Policy. Using FIFO policy, the value of $P_i[D(t)]$ in the steady-state is positively correlated to the arrival rate of task *i* [24]. Thus:

$$\lim_{t\to\infty} P_i[D(t)] = \lambda_i / \sum_{j=1}^{n} \lambda_j$$

SRTF Policy. This policy ensures the smallest task in the queue to be scheduled first. Using this policy, if task *i* is to be processed, then the logical scheduling queue of task 1 to task *i*-1 should be empty, hence the expression of $P_i[D(t)]$ is:

$$P_i[D(t)] = \prod_{j=1}^{i-1} P[WQ_j(t) = 0] \cdot P[WQ_i(t) > 0]$$

Consider the steady-state, if the logical scheduling queue of task j is not empty, then at least one of this task's processing queue is full. Therefore, we get:

$$\lim_{t\to\infty} P[WQ_j(t) = 0] = \prod_{s=1}^{m} P[\lim_{t\to\infty} TQ_{js}(t) < l_{\max}], \quad \lim_{t\to\infty} P[WQ_j(t) > 0] = 1 - \prod_{s=1}^{m} P[\lim_{t\to\infty} TQ_{js}(t) < l_{\max}]$$

Based on the modeling results of *M/G/*1 model, we derive the expression of $P_i[D(t)]$ in the steady-state:

$$\lim_{t\to\infty} P_i[D(t)] = \lim_{t\to\infty} \prod_{j=1}^{i-1} P[WQ_j(t) = 0] \cdot P[WQ_i(t) > 0]$$

$$= \prod_{s=1}^{m} \prod_{j=1}^{i-1} \frac{(1-\rho_{js})\rho_{js}^{\,l_{\max}}}{1-\rho_{js}^{\,l_{\max}+1}} \cdot [1 - \prod_{s=1}^{m} (\frac{(1-\rho_{is})\rho_{is}^{\,l_{\max}}}{1-\rho_{is}^{\,l_{\max}+1}})]$$

LRTF Policy. This policy ensures the largest task in the scheduling queue to be scheduled first. Similar to the derivation in SRTF policy, we get the expression of $P_i[D(t)]$ in the steady-state:

$$\lim_{t\to\infty} P_i[D(t)] = \lim_{t\to\infty} \prod_{j=i+1}^{n} P[WQ_j(t) = 0] \cdot P[WQ_i(t) > 0]$$

$$= \prod_{s=1}^{m} \prod_{j=i+1}^{n} \frac{(1-\rho_{js})\rho_{js}^{\,l_{\max}}}{1-\rho_{js}^{\,l_{\max}+1}} \cdot [1 - \prod_{s=1}^{m} \frac{(1-\rho_{is})\rho_{is}^{\,l_{\max}}}{1-\rho_{is}^{\,l_{\max}+1}}]$$

Myopic MaxWeight Policy. This policy determines the weight of a task based on a task's size and waiting time. To approximately model a task's waiting time, we assume that the waiting time of task i is positively correlated to $WQ_i(t)$, (If the first task in WQ_i cannot be handled, all the followed tasks will be blocked. So the queue length reflects the waiting time of the first task). Based on SRTF policy, the expression of $P_i[D(t)]$ under Myopic MaxWeight policy should be modified as:

$$P_i[D(t)] = (\prod_{j=1}^{i-1} P[WQ_j(t) = 0] \cdot P[WQ_i(t) > 0]) \cdot WQ_i(t) / \sum_{k=1}^{n} WQ_k(t)$$

Then the expression of $P_i[D(t)]$ in the steady-state can be derived as follows:

$$\lim_{t\to\infty} P_i[D(t)] = \lim_{t\to\infty} \prod_{j=1}^{i-1} P[WQ_j(t) = 0] \cdot P[WQ_i(t) > 0] \cdot WQ_i(t) / \sum_{k=1}^{n} WQ_k(t)$$

$$= \prod_{s=1}^{m} \prod_{j=1}^{i-1} \frac{(1-\rho_{js})\rho_{js}^{\,l_{\max}}}{1-\rho_{js}^{\,l_{\max}+1}} \cdot [1 - \prod_{s=1}^{m} \frac{(1-\rho_{is})\rho_{is}^{\,l_{\max}}}{1-\rho_{is}^{\,l_{\max}+1}}]$$

$$\cdot \frac{\sum_{s=1}^{m} \rho_{is} / (m - \sum_{s=1}^{m} \rho_{is}) - \sum_{s=1}^{m} [\frac{\rho_{is}}{1-\rho_{is}} - \frac{(l_{\max}+1)\rho_{is}^{\,l_{\max}+1}}{1-\rho_{is}^{\,l_{\max}+1}}]}{\sum_{k=1}^{n} \{\sum_{s=1}^{m} \rho_{ks} / (m - \sum_{s=1}^{m} \rho_{ks}) - \sum_{s=1}^{m} [\frac{\rho_{ks}}{1-\rho_{ks}} - \frac{(l_{\max}+1)\rho_{ks}^{\,l_{\max}+1}}{1-\rho_{ks}^{\,l_{\max}+1}}]\}}$$

Based on the above derivations, we get the specific expressions of $P_i[D(t)]$ under different ordering policies, which is the foundation of further evaluations.

3.3 Modeling of Ordering Policies

In the second phase, we need to formulate $P_{ij}[F(t)]$ in the steady-state under different dispatching policies.

Random Routing Policy (RR). This policy randomly dispatches a task to one of the m allocated VMs for processing, so the probability is time-independent:

$$P_{ij}[F(t)] = \lim_{t\to\infty} P_{ij}[F(t)] = 1/m$$

Shortest Queue Routing Policy (SQR). This policy dispatches a task to the VM with the shortest processing queue. The probability $P_{ij}[F(t)]$ can be formulated as:

$$P_{ij}[F(t)] = \begin{cases} \dfrac{1}{\| SQR(F) \|}, & if \quad j \in SQR(F) \\ 0, & if \quad j \notin SQR(F) \end{cases}$$

$$SQR(F) = \{ j \mid TQ_{ij}(t) = Min(TQ_{i1}(t), ..., TQ_{im}(t)) \}$$

Consider the steady-state, the probability that task *i* is dispatched to VM located on server *j* is positively correlated to its expected processing rate on server *j*. Therefore, the expression of $P_{ij}[F(t)]$ in the steady-state is:

$$\lim_{t\to\infty} P_{ij}[F(t)] = \mu_{ij} / \sum_{s=1}^{m} \mu_{is}$$

Shortest Expected Delay Routing Policy (SEDR). This policy dispatches a task to the VM with the shortest processing delay. $P_{ij}[F(t)]$ can be formulated as:

$$P_{ij}[F(t)] = \begin{cases} \dfrac{1}{\| SEDR(F) \|}, & if \quad j \in SEDR(F) \\ 0, & if \quad j \notin SEDR(F) \end{cases}$$

$$SEDR(F) = \{ j \mid TQ_{ij}(t)/\mu_{ij} = Min(TQ_{i1}(t)/\mu_{i1}, ..., TQ_{im}(t)/\mu_{im}) \}$$

Consider the steady-state, the probability that task *i* is dispatched to VM located on server *j* is positively correlated to the square of its processing rate on server *j*. Therefore, the expression of $P_{ij}[F(t)]$ in the steady-state is:

$$\lim_{t\to\infty} P_{ij}[F(t)] = \mu_{ij}^2 / \sum_{s=1}^{m} \mu_{is}^2$$

Overall Shortest Expected Delay Routing Policy (OSEDR). This policy dispatches a task to the appropriate VM to keep the overall processing delay of a server minimum. The probability $P_{ij}[F(t)]$ can be formulated as:

$$P_{ij}[F(t)] = \begin{cases} \dfrac{1}{\| OSEDR(F) \|}, & if \quad j \in OSEDR(F) \\ 0, & if \quad j \notin OSEDR(F) \end{cases}$$

$$OSEDR(F) = \{ j \mid \sum_{i=1}^{n} TQ_{ij}(t)/\mu_{ij} = Min(\sum_{i=1}^{n} TQ_{i1}(t)/\mu_{i1}, ..., \sum_{i=1}^{n} TQ_{im}(t)/\mu_{im}) \}$$

This policy minimizes the sum of all the tasks' processing delays on a server, aiming to improve the overall execution time of a server. However, in the scenario described in Fig. 1, the VMs located on the same server will process different tasks

simultaneously; hence the optimization goal should be the processing delay of the slowest task, but not the sum of processing delays. This policy may be not applicable in the scheduling of cloud tasks, hence will not be modeled in the following context.

3.4 Performance Analysis of Scheduling Policies

Substitute the expressions of $P_i[D(t)]$ and $P_{ij}[F(t)]$ into formula (1), we can calculate the value of $\rho_{avg}(i)$, which can be further used to evaluate the performance of different collaborations of ordering and dispatching policies. Some useful conclusions are illustrated below:

Based on the modeling results, we prove that SEDR and SQR optimize the utilization of each task in average and worst cases respectively, no matter what ordering policy they are collaborated with.

Optimizing the average resource utilization of task i is to minimize the value of $\rho_{avg}(i)$. Substitute the expression of $P_{ij}[F(t)]$ into (1), we get the expressions of $\rho_{avg}(i)$ under different dispatching policies:

SQR:
$$\rho_{avg}(i) = \frac{\overline{\lambda_i}}{m} \cdot \sum_{j=1}^{m} \frac{\mu_{ij}^2}{\mu_{ij} \sum_{j=1}^{m} \mu_{ij}^2} = \frac{\overline{\lambda_i}}{m} \cdot \sum_{j=1}^{m} \mu_{ij} / \sum_{j=1}^{m} \mu_{ij}^2 \tag{3}$$

SEDR:
$$\rho_{avg}(i) = \frac{\overline{\lambda_i}}{m} \cdot \sum_{j=1}^{m} [\mu_{ij} / (\mu_{ij} \sum_{j=1}^{m} \mu_{ij})] = \overline{\lambda_i} / \sum_{j=1}^{m} \mu_{ij} \tag{4}$$

Calculate the value of (4)-(3):

$$\Delta\rho_{avg}(i) = \overline{\lambda_i} \cdot \frac{1}{\sum_{j=1}^{m} \mu_{ij}} - \frac{\overline{\lambda_i}}{m} \cdot \frac{\sum_{j=1}^{m} \mu_{ij}}{\sum_{j=1}^{m} \mu_{ij}^2} = \overline{\lambda_i} \cdot \frac{m \sum_{j=1}^{m} \mu_{ij}^2 - (\sum_{j=1}^{m} \mu_{ij})^2}{m \sum_{j=1}^{m} \mu_{ij}^2 \sum_{j=1}^{m} \mu_{ij}} = \overline{\lambda_i} \cdot \frac{\sum_{i=1}^{m-1} \sum_{k=i+1}^{m} (\mu_{ij} - \mu_{ik})^2}{m \sum_{j=1}^{m} \mu_{ij}^2 \sum_{j=1}^{m} \mu_{ij}} \geq 0$$

Thus, SEDR policy provides better resource utilization of a task than SQR policy in average case. Similarly, we can prove that SEDR policy provides better resource utilization of a task than the other dispatching policies shown in this paper. Due to the space limitation, we omit the proof details here.

On the other hand, in the worst case, a request of task i is likely to be blocked in the scheduling queue when task i's processing queue with the heaviest load is full. Hence the worst case resource utilization of task i is determined by the utilization of task i's VM on the heaviest load:

$$\rho_{worst}(i) = Max\{\frac{\overline{\lambda_i} \lim_{t \to \infty} P_{i1}[F(t)]}{\mu_{i1}}, ..., \frac{\overline{\lambda_i} \lim_{t \to \infty} P_{im}[F(t)]}{\mu_{im}}\}$$

An optimal scheduling policy should minimize the value of $\rho_{worst}(i)$. We know $\rho_{worst}(i)$ gets the minimum value when:

$$\lim_{t\to\infty} P_{i1}[F(t)]/\mu_{i1} = \lim_{t\to\infty} P_{i2}[F(t)]/\mu_{i2} = \ldots = \lim_{t\to\infty} P_{im}[F(t)]/\mu_{im} \tag{5}$$

To satisfy (5), the expression of $P_{ij}[F(t)]$ should be:

$$\lim_{t\to\infty} P_{ij}[F(t)] = \mu_{ij} / \sum_{s=1}^{m} \mu_{is} \tag{6}$$

We find that (6) is the same as the expression of $P_{ij}[F(t)]$ under SQR policy. Hence SQR dispatching policy provides optimal utilization of tasks in the worst case.

Next, we prove that, when the requests of different tasks are arriving uniformly, SRTF is the optimal ordering policy to collaborate with SEDR in the average case, while LRTF is the worst ordering policy to collaborate with SEDR.

Average resource utilization of all the tasks can be expressed as follows:

$$\rho_{total} = \frac{1}{n}\sum_{i=1}^{n} \rho_{avg}(i) = \sum_{i=1}^{n}\sum_{j=1}^{m} \frac{\lambda_i \cdot \lim_{t\to\infty} P_i[D(t)]P_{ij}[F(t)]}{\mu_{ij} mn} \tag{7}$$

Since the dispatching policy SEDR is chosen, the value of $P_{ij}[F(t)]$ is determined. The requests of different tasks are uniformly arrived, so the value of λ_i is basically identical. And we have assumed $\mu_{1j} \geq \mu_{2j} \geq \ldots \geq \mu_{nj}$, hence we get:

$$\lambda_1 \sum_{j=1}^{m} \frac{1}{\mu_{ij}} \leq \lambda_2 \sum_{j=1}^{m} \frac{1}{\mu_{2j}} \leq \ldots \leq \lambda_n \sum_{j=1}^{m} \frac{1}{\mu_{nj}}$$

And the value range of ρ_{total} should be:

$$\lambda_1 \sum_{j=1}^{m} \frac{1}{\mu_{1j}} \cdot \sum_{j=1}^{m} \frac{\lim_{t\to\infty} P_{ij}[F(t)]}{mn} \leq \rho_{total} \leq \lambda_n \sum_{j=1}^{m} \frac{1}{\mu_{nj}} \cdot \sum_{j=1}^{m} \frac{\lim_{t\to\infty} P_{ij}[F(t)]}{mn}$$

At this time, the value of ρ_{total} is minimum when:

$$\lim_{t\to\infty} P_1[D(t)] \to 1, \quad \lim_{t\to\infty} P_i[D(t)] \to 0, \quad \forall 2 \leq i \leq n$$

And the value of ρ_{total} is maximum when:

$$\lim_{t\to\infty} P_n[D(t)] \to 1, \quad \lim_{t\to\infty} P_i[D(t)] \to 0, \quad \forall 1 \leq i \leq n-1$$

In the average case, the processing queue is not full, hence:

$$1 - \rho_{ij} >> 0 \quad \forall 1 \leq i \leq n, \quad \forall 1 \leq j \leq m$$

By calculating the value of $P_i[D(t)]$ under different ordering policies, we get the following conclusions:

- The value of $P_1[D(t)]$ in the steady-state is closest to 1 under the SRTF ordering policy, which indicates that SRTF is the optimal ordering policy to cooperate with SEDR
- The value of $P_n[D(t)]$ in the steady-state is closest to 1 under the LRTF ordering policy, which indicates that LRTF is not an appropriate ordering policy to achieve optimal resource utilization.

- The value of $P_i[D(t)]$ is fixed under the FIFO ordering policy, which indicates that FIFO is a moderate ordering policy.

- $P_1[D(t)]$ also gets the maximum value under the Myopic MaxWeight ordering policy, but the value is not so close to 1 compared with the value of $P_1[D(t)]$ under the SRTF policy. Thus, using Myopic MaxWeight policy, we can also achieve relatively good resource utilization in the average case, but it is not the best choice.

If the requests of different tasks are not uniformly arrived, similarly we can prove that the optimal ordering policy to collaborate with SEDR should satisfy:

$$\lim_{t\to\infty} P_i[D(t)] \to 1 \quad \forall \lambda_i \sum_{j=1}^{m} \frac{1}{\mu_{ij}} = Min\{\lambda_1 \sum_{j=1}^{m} \frac{1}{\mu_{1j}}, \lambda_2 \sum_{j=1}^{m} \frac{1}{\mu_{2j}}, \ldots \lambda_n \sum_{j=1}^{m} \frac{1}{\mu_{nj}}\}$$

However, in the worst case, the processing queue is nearly full, and when a request of task i is blocked in the scheduling queue, it will not be processed. So, there is a probability that:

$$\lambda_1 \sum_{j=1}^{m} \frac{1}{\mu_{1j}} \to \infty$$

At this time, SRTF is not the optimal ordering policy since:

$$\sum_{i=1}^{n} \{\lambda_i \cdot \lim_{t\to\infty} P_i[D(t)] \sum_{j=1}^{m} \frac{1}{\mu_{ij}}\} \to \infty, \quad if \quad \lim_{t\to\infty} P_1[D(t)] \to 1$$

And Myopic MaxWeigtht is an optimal ordering policy to cooperate with SQR, since it reduces the value of $P_1[D(t)]$, but also keep it maximum. The collaboration of Myopic MaxWeight ordering and SQR dispatching has been proved to be throughput optimal elsewhere [22, 25].

Using our model, we can also evaluate the average responsiveness of the tasks under different collaborations of ordering and dispatching policies, by substituting the expressions of $P_i[D(t)]$ and $P_{ij}[F(t)]$ into (2) and comparing the value of T_{avg}. Due to the space limitation, we omit the evaluation of other performance metrics here.

4 Experiments and Evaluations

In this section, we realize different collaborations of ordering and dispatching policies in the scheduling process of cloud tasks. Based on the experimental results, we evaluate the performance of the list schedulers and validate our modeling results.

We simulate the scenario where a cloud computing platform comprises N servers, and processes tasks with 5 different sizes (i.e. 5 types of tasks). The tasks simulated here are the simple character counting tasks, thus we can approximately predict the processing time of a task according to the task's size. The task scheduling process simulated in our experiment is the same as shown in Fig. 1. The arrival rate of task i satisfies Poisson distribution with mean rate λ_i, and the requests of different tasks are

uniformly arrived, thus $\lambda_1 = \lambda_2 = \ldots = \lambda_5$. The tasks' sizes are sequenced in ascending order, hence the processing rate of each task on the platform satisfies: $\mu_{1j} \geqslant \mu_{2j} \geqslant \ldots \geqslant \mu_{5j}$ ($1 \leq j \leq N$). The simulation time is set long enough (1000 time units) to make the scheduling process reach its steady-state.

Since the utilization metrics reflect the throughput of a system, we record the percentage of processed tasks at each time point to evaluate the throughput of the platform. To make our simulation more convincible, we simulate dynamic scheduling of tasks onto a large scale cloud system with 500 servers (N=500). Twenty-five list schedulers work together to order the arrived tasks and dispatch them to 2500 VMs located on the 500 cloud servers.

Fig. 3 shows the throughput of the cloud computing platform at each time point under different collaborations of ordering and dispatching policies. We simulated the collaborations of all the dispatching policies introduced in this paper and ordering policies FIFO (shown in Fig. 3A), SRTF (shown in Fig. 3B), LRTF (shown in Fig. 3C) and Myopic MaxWeight (shown in Fig. 3D).

We can see from Fig. 3A-3D that, no matter what ordering policy is used, SQR and SEDR dispatching policies achieve better throughput to the cloud platform among the four dispatching policies. This phenomenon validates our modeling results in Section 3 that SEDR and SQR achieve better resource utilization of a task in the scheduling process.

We also notice that the scheduling results using OSEDR is not effective: sometimes, the scheduling results of OSEDR are even no better than Random policy. The reason of this phenomenon is that optimizing the sum of execution delays on a server is not reasonable in cloud computing platforms just as previously discussed in Section 3, since the VMs co-located on the same cloud server can process different tasks simultaneously, the applicable dispatching policy is to optimize the execution delay of the slowest task on a server.

Although the platform achieves better throughput using SEDR and SQR policies, there are still some differences in the scheduling results. Using SEDR policy, the throughput is the best in the initial and middle periods of the scheduling (typically from time unit 1-600), but the throughput will then drop down below SQR policy; on the other hand, using SQR policy, the throughput of the platform rises to the best in the final period of the scheduling. This fact can be explained using the conclusions presented in Section 3: SEDR is the optimal dispatching policy to collaborate with in the average case, hence it provides best throughput in the initial and middle periods when the task loads are not heavy enough. However, when it comes to the final periods of the scheduling, the accumulated unhandled tasks will affect the scheduling results, and the scenario will be changed from the average case scheduling to the worst case scheduling. According to our conclusions in Section 3, SQR is the utilization optimal dispatching policy in the worst case; it explains the phenomenon that using SQR, the platform gets the best throughput in the final period of the scheduling.

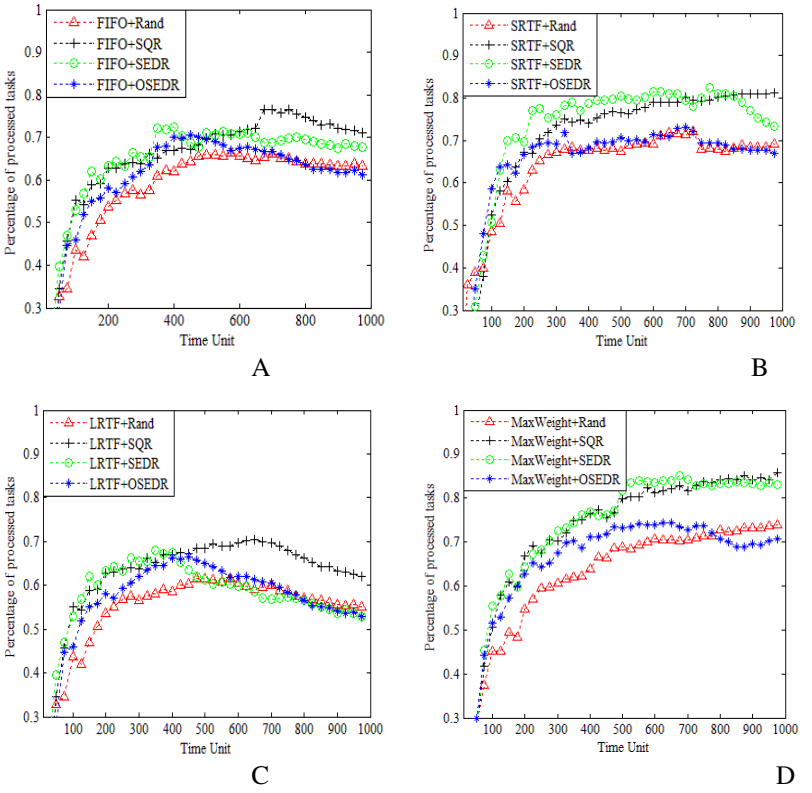

Fig. 2. Throughput of the cloud platform under different collaborations of ordering and dispatching policies

Comparing the results shown in Fig. 3A-3C, we find that the collaboration of SRTF ordering and SEDR dispatching achieves best throughput in the initial and middle scheduling periods; the collaboration of FIFO and SEDR performs moderately; and the collaboration of LRTF and SEDR performs worst. This fact confirms to our modeling results that SRTF is the optimal ordering policy to cooperate with SEDR in the average case when the task requests are uniformly distributed, while LRTF is the worst policy to collaborate with.

We notice from Fig. 3B and 3C that, using SRTF and LRTF policy, the throughput will suffer attenuation at a specific time point (around time unit 400-500 for LRTF, 700-800 for SRTF). This is because SRTF and LRTF are not starvation-free orderings: some tasks have already been stalled in the scheduling queue even if their processing queues are not full; to make things worse, if the shortest (for SRTF) or longest (for LRTF) remaining time tasks are blocked, all the arrived tasks at this time point will not be processed, which will dramatically attenuate the system's throughput close to zero at this time point. However, we can see in Fig. 3D that Myopic MaxWeight policy solves the throughput attenuation problem by improving the SRTF policy. The collaboration of Myopic MaxWeight and SQR provides best throughput

in the final period of scheduling (after time unit 700), which theoretically and experimentally validate the conclusion that the collaboration of Myopic MaxWeight ordering and SQR dispatching is the utilization optimal list scheduling policy in the worst case.

5 Conclusion

We briefly summarize some representative ordering and dispatching policies which are commonly used in both industry and academic communities, model the execution processes of these ordering and dispatching policies using the SPN model, and apply different collaborated scheduling policies to schedule cloud tasks in simulation environment. Based on the modeling and experimental results, we have evaluated that SEDR and SQR are utilization optimal dispatching policies to collaborate with in the average case and worst case scheduling respectively; the collaboration of SRTF ordering and SEDR dispatching achieves optimal throughput in the average case, while the collaboration of Myopic MaxWeight ordering and SQR dispatching achieves optimal throughput in the worst case.

References

1. Marty, M., Hill, M.: Virtual Hierarchies to Support Server Consolidation. In: Proceedings of the 34th Annual International Symposium on Computer Architecture, pp. 46–56 (2007)
2. Speitkamp, B., Bichler, M.: A mathematical programming approach for server consolidation problems in virtualized data centers. IEEE Transactions on Services Computing, 266–278 (2010)
3. Beloglazov, A., Buyya, R.: Energy efficient allocation of virtual machines in cloud data centers. In: 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, pp. 577–578 (2010)
4. Burkimsher, A., Bate, I., Indrusiak, L.S.: A survey of scheduling metrics and an improved ordering policy for list schedulers operating on workloads with dependencies and a wide variation in execution times. In: Future Generation Computer Systems (2012, 2013)
5. Topcuoglu, H., Hariri, S., Wu, M.Y.: Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Transactions on Parallel and Distributed Systems, 260–274 (2002)
6. Lu, X., Sitters, R.A., Stougie, L.: A class of on-line scheduling algorithms to minimize total completion time. Operations Research Letters 31(3), 232–236 (2003)
7. Laili, Y., et al.: A Ranking Chaos Algorithm for dual scheduling of cloud service and computing resource in private cloud. Computers in Industry, 448–463 (2013)
8. Oracle: N1 grid engine 6 administration guide—configuring the share-based policy (2010), `http://docs.oracle.com/cd/E19080-01/n1.grid.eng6/817-5677/i999588/index.html`
9. Östberg, P.O., Danie, E., Erik, E.: Decentralized scalable fairshare scheduling. Future Generation Computer Systems, 130–143 (2013)
10. Chang, H., Kodialam, M., Kompella, R.R., et al.: Scheduling in mapreduce-like systems for fast completion time. In: Proceedings of IEEE INFOCOM Conference, pp. 3074–3082 (2011)

11. Chen, F., Kodialam, M., Lakshman, T.V.: Joint Scheduling of Processing and Shuffle Phases in MapReduce Systems. In: Proceedings of IEEE INFOCOM Conference (2012)
12. Lui, J.C.S., Richard, R.M., Don, T.: Bounding the mean response time of the minimum expected delay routing policy: an algorithmic approach. IEEE Transactions on Computers 44(12), 1371–1382 (1995)
13. Albers, S.: Better bounds for online scheduling. In: Proceedings of the 29th Annual ACM Symposium on Theory of Computing (STOC 1997), New York, USA, pp. 130–139 (1997)
14. Bender, M.A., Chakrabarti, S., Muthukrishnan, S.: Flow and stretch metrics for scheduling continuous job streams. In: Proceedings of the 9th Annual ACM–SIAM Symposium on Discrete Algorithms, SODA 1998, Philadelphia, PA, USA, pp. 270–279 (1998)
15. Kong, X., Lin, C., Jiang, Y., Yan, W., Chu, X.: Efficient dynamic task scheduling in virtualized data centers with fuzzy prediction. Journal of Network and Computer Applications 34(4), 1068–1077 (2011)
16. Calheiros, R.N., Ranjan, R., Buyya, R.: Virtual machine provisioning based on analytical performance and qos in cloud computing environments. In: IEEE ICPP (2011)
17. Yuan, Y., Wang, H., Wang, D.: On Interference-aware Provisioning for Cloud-based Big Data Processing. In: IEEE 20th International Workshop on Quality of Service 2013 (IWQoS 2013) (June 2013)
18. Jung, G., Hiltunen, M.A., Joshi, K.R., Schlichting, R.D., Pu, C.: Mistral: Dynamically Managing Power, Performance, and Adaptation Cost in Cloud Infrastructures. In: IEEE ICDCS 2010 (June 2010)
19. Mitzenmacher, M.: The power of two choices in randomized load balancing. Ph.D. dissertation, University of California at Berkeley (1996)
20. He, Y.T., Down, D.G.: Limited choice and locality considerations for load balancing. Performance Evaluation 65(9) (2008)
21. Lin, C., Shan, Z., Yang, Y.: Integrated schemes of request dispatching and selecting in Web server clusters. In: Proceedings of Conference on Software: Theory and Practice, 16 th World Computer Congress 2000 (WCC 2000), Beijing, China (August 2000)
22. Maguluri, S.T., Srikant, R., Ying, L.: Stochastic models of load balancing and scheduling in cloud computing clusters. In: Proceedings of IEEE INFOCOMM Conference (2012)
23. Khan, A.A., McCreary, C.L., Jones, M.S.: A comparison of multiprocessor scheduling heuristics. In: IEEE International Conference on Parallel Processing (ICPP) (1994)
24. Kleinrock, L.: Queueing Systems. Theory, vol. I, p. 187. John Wiley & Sons, New York (1975)
25. Maguluri, S.T., Srikant, R.: Scheduling Jobs with Unknown Duration in Clouds. In: IEEE INFOCOM Conference (2013)