J. Cent. South Univ. (2014) 21: 3864−3872
DOI: 10.1007/s11771-014-2373-x

Springer

Task scheduling scheme by checkpoint sharing and task duplication in P2P-based desktop grids

Joon-Min Gil, Young-Sik Jeong

1. School of IT Engineering, Catholic University of Daegu, 13-13, Hayang-ro, Hayang-eup, Gyeongsan-si, Gyeongbuk 712-701, Korea;
2. Department of Multimedia Engineering, Dongguk University, 30 Pildong-rol-gil, Jung-gu, Seoul 100-715, Korea

Abstract: A scheduling scheme is proposed to reduce execution time by means of both checkpoint sharing and task duplication under a peer-to-peer (P2P) architecture. In the scheme, the checkpoint executed by each peer (i.e., a resource) is used as an intermediate result and executed in other peers via its duplication and transmission. As the checkpoint is close to a final result, the reduction of execution time for each task becomes higher, leading to reducing turnaround time. To evaluate the performance of our scheduling scheme in terms of transmission cost and execution time, an analytical model with an embedded Markov chain is presented. We also conduct simulations with a failure rate of tasks and compare the performance of our scheduling scheme with that of the existing scheme based on client-server architecture. Performance results show that our scheduling scheme is superior to the existing scheme with respect to the reduction of execution time and turnaround time.

Key words: P2P-based desktop grids; checkpoint sharing; task duplication; embedded Markov chain

1 Introduction

Desktop grids are used in a practical computing paradigm that can process massive computational tasks in various application areas, using the idle cycles of the heterogeneous resources (generally desktop computers) connected over the Internet and owned by different individual users. They are generally suitable for the large-scale applications composed of hundreds of thousands of small-sized tasks for the same computational code. It is well-known that desktop grids make it possible to obtain large-scale computing power with a low cost [1−2]. Since the success of SETI@Home [3−4], a variety of desktop grid platforms, such as BOINC [5−6], XtremWeb [7], Korea@Home [8], SZTAKI [9], QADPZ [10], have been developed. The commercial desktop grid systems, such as Entropia [11] and United Devices [12], are released for enterprise computing, and some practical applications for desktop grids are reported in Refs. [13−14].

An important aspect in desktop grids is that each resource has a volatility property, due to free withdrawal from execution participation even in the middle of task execution. Moreover, each resource has a heterogeneity property as it has a totally different computing environment (e.g., CPU performance, memory capacity, and network speed) [15]. One critical issue of a desktop grid environment is to minimize the execution time of all tasks, even if these two properties affect overall performance adversely [1]. Unexpected failures can be considered degrading factors in the minimization of execution time, which can be partially addressed with the use of a checkpointing mechanism at the application level [16−17]. Another method of minimizing the execution time is to share all of the checkpoints performed on each resource [18]. Checkpoint sharing is a method of reusing the checkpoint, which has been recently performed on a local desktop in another resource (i.e., the intermediate result of a task is transmitted to other resources so that task execution from the last checkpoint position can be restated).

Consequently, the purpose of checkpoint sharing is to reduce the execution time of tasks, leading to a reduction in turnaround time. Most desktop grid systems, however, use a client-server model as their main architecture [6, 11, 19]. Although this model is simple in architecture as well as in the control of resources and tasks, it concentrates all functions on the central server, which heightens the bottleneck phenomenon in the server. Moreover, in the client-server model, checkpoint sharing is based on storing checkpoints in a central stable

Received date: 2013−11−20; **Accepted date:** 2014−01−16
Corresponding author: Young-Sik Jeong; E-mail: ysjeong@dongguk.edu

storage [18]. The checkpoints of all desktops are also concentrated in the central stable storage, which again brings about the bottleneck phenomenon. To overcome this shortcoming of the client-server model in a desktop grid environment, the peer-to-peer (P2P) model [20] is utilized in this work as a fundamental architecture, which has been widely used in Internet services, such as file sharing or content delivery. Compared to the client-server model that completely depends on the central server, the P2P-based desktop grid environment used in this work is based on a three-layered structure (central server, peer groups and peers). In this structure, the central server controls only peer groups, and a representative peer in a peer group controls the peers that belong to the corresponding group, thus dispersing the functions of the central server, and ultimately, reducing the bottleneck phenomenon in the central server.

To cope with task failures in a P2P-based desktop grid environment, in this work, each peer performs checkpointing on its local disk at a periodic cycle. The intermediate result, which is stored in a peer as a checkpoint, is transmitted to another peer requesting a task. Then, the peer continuously executes the intermediate result beginning with the last checkpoint position. This checkpoint sharing leads to the reduction of the execution time. In order to deal with peer volatility and heterogeneity, a task duplication method is used along with the checkpoint sharing. When a peer requests a task, an intermediate result with the last checkpoint among replicas for the task is allocated to the peer. The requesting peer successively executes the task, utilizing the intermediate result. Therefore, it is expected that our scheduling scheme will more significantly reduce the execution time than the existing scheme, where the duplicated tasks are executed from the beginning.

Eventually, this work aims to devise a scheduling scheme to reduce execution time per task using checkpoint sharing and task duplication in a P2P-based desktop grid environment, resulting in providing large-scale applications with fast turnaround time. Contrary to the existing scheme based on a client-server model, our scheduling scheme performs checkpoint sharing and task duplication on the basis of P2P architecture. Thus, our scheme can basically reduce the load of the central server due to the use of P2P architecture. As for checkpoint sharing, it does not need any central storage. Instead, checkpoints in each peer are autonomously transmitted to other peers by the mediation of a peer. To show the superiority of our scheduling scheme, we present a mathematical analysis model with an embedded Markov chain. Based on the model, we compare our scheme with the existing one, in terms of transmission cost and execution time. The simulations with task failures are also conducted and their results are presented.

2 P2P-based desktop grid environment

2.1 System model

Figure 1 shows a system model for the P2P-based desktop grid environment assumed in this work. This system model is based on a three-layered structure consisting of a central server, peer groups, and peers. The central server operates minimum functions, such as peer group management, peer authentication management, and metadata management for tasks and peers, rather than various kinds of management for peers and tasks. The peer groups (PGs) consist of peers with identical characteristics under certain conditions. A unique peer within a peer group becomes a representative peer (RP), and the remaining peers become member peers (MPs).

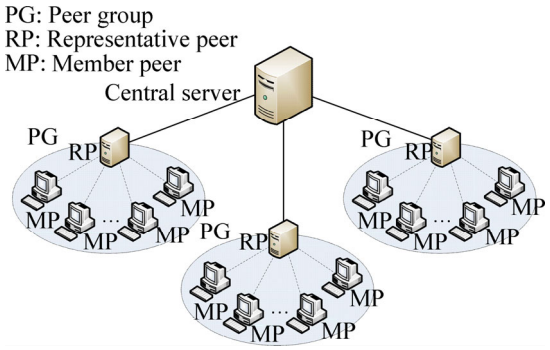

Fig. 1 System model

Generally, a large-scale application in a PC desktop grid environment is divided into hundreds and millions of unit tasks, and each should be suitable for execution in a peer (or a resource). These unit tasks are also structured in such a way that there is no dependency among unit tasks [1].

In our system model, an RP keeps a task list, $\boldsymbol{W}_g=\{w_1, w_2, \cdots, w_m, \cdots, w_M\}$, to manage the tasks to be allocated to the MPs that belong to its PG. Here, g is an index for distinguishing a PG, and M is the number of tasks. An element of a task list, w_m, has the following data structure: $w_m=\{w_{\mathrm{id}}, w_{\mathrm{d}}, w_{\mathrm{c}}\}$ $(1\le m\le M)$, where w_{id} is a unique identifier for a task. For task duplication, each task w_m should recognize how many replicas are being executed on different MPs. The w_m keeps the current number of replicas w_{d} which has one of the following values: $\{0, 1, \cdots, D\}$, where D is the maximum number of duplications. The w_{c} represents the largest number of checkpoints among those of the duplicated tasks. This information is also used by the RP to duplicate the intermediate result that has the largest number of checkpoints when an identical task is being duplicated.

Meanwhile, a member peer in a PG, M_{l}, has the following data structure: $M_{\mathrm{l}}=\{M_{\mathrm{id}}, P_{\mathrm{g}}, w_{\mathrm{id}}, M_c\}$, where M_{id} and P_{g} are the identifiers of the MP and the PG to

which M_{id} belongs, respectively; w_{id} is the identifier of the task allocated from the RP; M_c represents the checkpoint status of the task, which has one of the following values: {0, 1, ⋯, C}, where C represents the maximum number of checkpoints. If M_c=0, this indicates a status that has not yet taken a checkpoint (i.e., either a stand-by status or a status of having executed a task just before having taken the first checkpoint). If M_c has a value of c ($1 \le c < C$), the status has taken the cth checkpoint and executed a task just before having taken the (c+1)th checkpoint. If $M_c=C$, the task execution has ended.

2.2 Checkpointing and task duplication

Figures 2 and 3 show the duplication process and the checkpointing process between RP and MP, respectively. The process of task duplication (Fig. 2) is as follows. First, if M_i sends a task request message to RP ((1) in Fig. 2(a) and (b)), RP searches for the task with the smallest number of duplications from the task list it maintains. Assume that this task is w_m. By examining the w_d value of w_m, RP can recognize how many member peers execute w_m in duplication. If w_d is 0, it means w_m has not yet been duplicated, and therefore, the RP directly transmits the task data of w_m to M_i ((2) in Fig. 2(a)). If w_d is larger than 1, RP recognizes an MP (or MPs) to which w_m has already been allocated (let the MP be M_j). At this point, RP sends M_i the notification message that task data will be received from M_j ((2) in Fig. 2(b)). Then, RP sends an order message to M_j so that M_j will transmit task data to M_i ((3) in Fig. 2(b)). Upon receiving this message, M_j sends M_i its last checkpoint as task data. In other words, it sends the intermediate result, which has been produced by checkpointing, to M_i ((4) in Fig. 2(b)).

Figure 3 illustrates the checkpointing process. As the intermediate result of a task, each checkpoint is saved in the local disk of an MP at a periodic cycle (M_i in Fig. 3). Soon after M_i performs a checkpoint, it notifies RP that its checkpoint has been taken. Since checkpointing is performed at a periodic cycle, M_i can send the RP the checkpoint messages of total C times, from the beginning to the end of a task ((1) in Fig. 3). Meanwhile, by checkpoint sharing, M_i can receive a checkpoint as an intermediate result from another MP. If the checkpoint transmitted to M_i has been performed up to the cth checkpoint, M_i will send the checkpoint messages of $C-c$ times to RP until the task is completed ((2) in Fig. 3). At this time, task execution time can be reduced because M_i executes a task from the cth checkpoint time, not from the beginning.

Having received a checkpoint message for task w_m

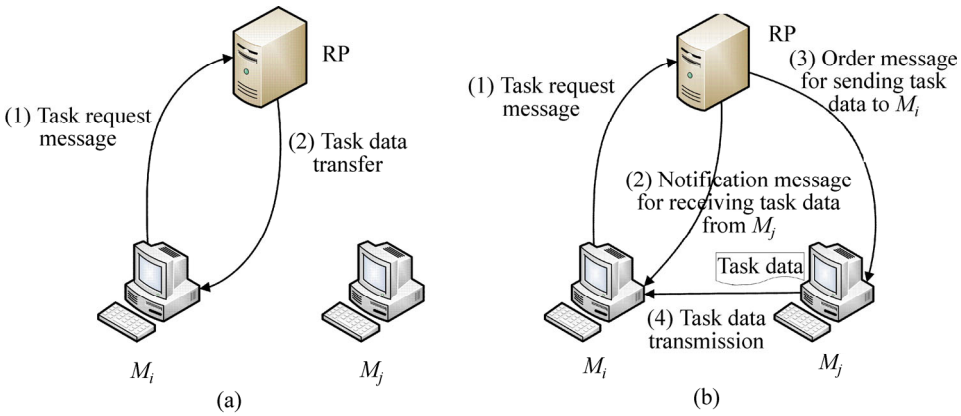

Fig. 2 Duplication process between RP and MP: (a) RP transmitting task data directly; (b) Task data transmitted by mediation of RP

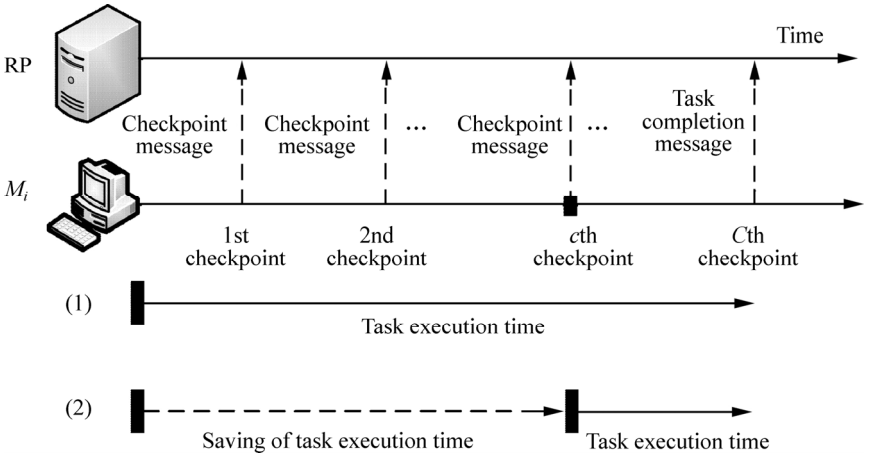

Fig. 3 Checkpointing process between RP and MP

from M_i, RP updates the w_c value of w_m in the task list. At this point, w_c is compared to the M_c of M_i, which is included in the checkpoint message; if $w_c<M_c$, w_c is replaced with M_c. This case indicates that in a state where task w_m is being executed on several MPs in duplication, the M_i with the largest number of checkpoints sends its checkpoint message to RP. If $w_c \geq M_c$, w_c is kept without any change. Meanwhile, if RP receives a task completion message from M_i, it confirms that task execution has been completed. Then, RP receives a final result from M_i.

3 Analytical model

In this section, we analyze our scheme using an embedded Markov chain model in terms of message/data transmission cost and the reduction of execution time.

3.1 Analysis modeling with an embedded Markov chain

Figure 4 shows the state transition diagram established when MPs execute, at most, two checkpoints (C=2) until completing each replica, permitting three duplications per task (D=3). As shown in Fig. 4, when D and C have the values of 3 and 2, respectively, each state is expressed as a state vector (a, c_1, c_2, c_3, r). The first element, a ($0 \leq a \leq D$), represents the number of replicas for the task w_m. The second to the fourth elements, c_1, c_2, c_3, come in the order of the number of checkpoints of the MPs executing the task, which is $0 \leq c_3 \leq c_2 \leq c_1 < C$. That is to say, c_1 is the number of checkpoints of the MP that has the largest number of checkpoints among the MPs executing the task. On the other hand, c_3 is the number of checkpoints of the MP that has the smallest number of checkpoints. Because the checkpoint status for all the MPs executing the task should be expressed in the state vector, the total number of c_i elements is required as many as D (i.e., the number of duplications). The latest element r represents the number of the completed tasks among the duplicated ones.

In this work, we assume that a task request and a checkpoint report follow the Poisson process [21] with the ratio of λ_1 and λ_2, respectively. λ_1 indicates the mean number of events generated for a unit of time when the RP receives a task request from an MP and allocates a task to the MP, and λ_2 indicates the mean number of events generated for a unit of time when the MP executing a task performs checkpointing and reports to the RP that it has taken the checkpoint. From Fig. 4, we can observe that the state transition of a task by checkpoint sharing and task duplication is caused by an MP's task request or checkpoint report. Whenever either a task request event or a checkpoint event occurs in an MP, the RP receives messages for the event from the MP. Thus, if such messages are observed on the RP, the state transition of a task can be modeled. Let $p_{(a,c_1,c_2,c_3,r),(a',c_1',c_2',c_3',r')}$ be the one-stop transition probability from a state (a, c_1, c_2, c_3, r) to a state (a', c_1', c_2', c_3', r'). It can be calculated based on the state transition shown in Fig. 4. The one-step transition matrix $\boldsymbol{P}=(p_{(a,c_1,c_2,c_3,r),(a',c_1',c_2',c_3',r')})$ can be expressed as follows:

$$\boldsymbol{P}=\frac{1}{L}\cdot$$

$$\begin{pmatrix}
0 & L & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & \lambda_1 & 0 & \lambda_2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & \lambda_1 & 0 & \lambda_2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & \lambda_1 & 0 & 0 & \lambda_2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & \lambda_2 & 0 & \lambda_2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{\lambda_2}{2} & \frac{\lambda_2}{2} & 0 & \lambda_1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & \lambda_1 & 0 & 0 & \frac{\lambda_2}{2} & \frac{\lambda_2}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & L & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \lambda_1 & 0 & \lambda_2 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \lambda_2 & 0 & \lambda_1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \lambda_1 & 0 & 0 & \lambda_2 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \lambda_1 & 0 & \frac{\lambda_2}{2} & \frac{\lambda_2}{2} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \lambda_2 & 0 & \lambda_1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \lambda_1 & 0 & 0 & \frac{\lambda_2}{2} & \frac{\lambda_2}{2} & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \lambda_1 & 0 & 0 & \lambda_2 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & L & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \lambda_1 & 0 & \lambda_2 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \lambda_1 & \lambda_2 \\
\lambda_2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \lambda_1
\end{pmatrix}$$

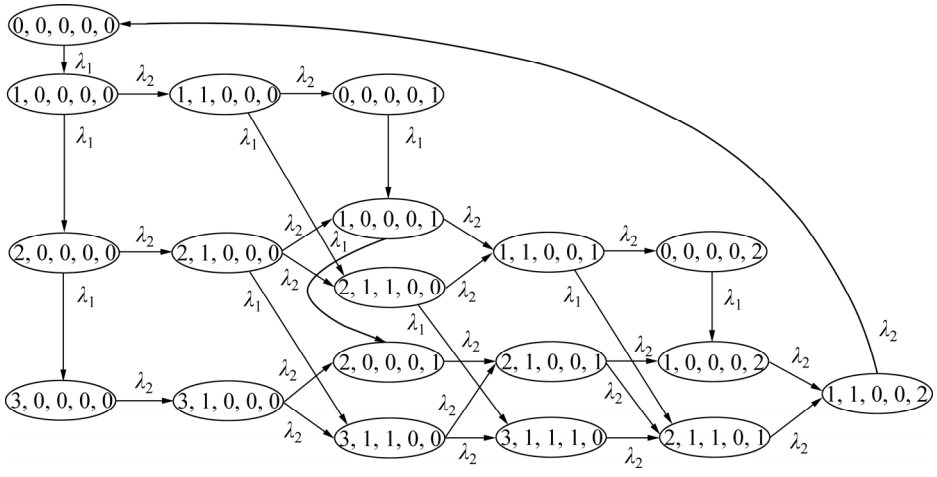

Fig. 4 State transition diagram (D=3, C=2)

In this transition matrix, λ_1 and λ_2 indicate the task request rate and the checkpoint arrival rate per unit of time, respectively, and $L=\lambda_1+\lambda_2$. The elements of each column and row in the transition matrix are listed in the following order (0, 0, 0, 0, 0), (1, 0, 0, 0, 0), (2, 0, 0, 0, 0), (3, 0, 0, 0, 0), ···, (0, 0, 0, 0, 2), (1, 0, 0, 0, 2), (2, 1, 1, 0, 2), and (1, 1, 0, 0, 2). Next, let N be the total number of states and $\pi_{(a, c_1, c_2, c_3, r)}$ the steady-state probability of a state (a, c_1, c_2, c_3, r). The unique steady-state probability distribution vector $\boldsymbol{\pi}$ for these states can be obtained from the following formula [19]:

$$\boldsymbol{\pi}=\boldsymbol{e}(\boldsymbol{I}+\boldsymbol{E}-\boldsymbol{P})^{-1} \tag{1}$$

where $\boldsymbol{E}$ and $\boldsymbol{e}$ are the matrix ($N\times N$) and the row ($1\times N$), whose element has a value of 1, respectively. Equation (1), a variation of $\boldsymbol{\pi}=\boldsymbol{\pi P}$, is used very significantly for the numerical calculation; using this formula, we can compute each $\pi_{(a, c_1, c_2, c_3, r)}$.

3.2 Analysis of transmission cost

Here, we describe the transmission cost required by task requests and checkpointing when D=3 and C=2. First, consider the transmission cost related to task requests. Let m_r and d_r denote message cost and data cost, respectively. If we assume that initial task data and intermediate result data have the same size, the transmission cost for the two data will be identical (i.e., transmission cost for each data is d_r). Then, the unit cost required by a task request event can be calculated by

$$\begin{aligned} U_1 &= m_\mathrm{r} + d_\mathrm{r} \\ U_2 &= 3m_\mathrm{r} + d_\mathrm{r} \end{aligned} \tag{2}$$

where U_1 is the unit cost made when the RP directly transmits task data, and U_2 is the unit cost made when task data are transmitted by the mediation of the RP. Using Eq. (2) and the transition probability for the task request, the total cost R_c required by a task request event is calculated by

$$\begin{cases} R_\mathrm{c} = R_1\cdot U_1 + R_2\cdot U_2 \\ R_1 = \displaystyle\sum_{i=0}^{D-1}\pi_{(0,0,0,0,i)} + \left(\sum_{i=1}^{D-1}\pi_{(i,0,0,0,0)} + \pi_{(1,0,0,0,1)}\right)\cdot\left(\frac{\lambda_1}{\lambda_1+\lambda_2}\right) \\ R_2 = \left(\displaystyle\sum_{i=0}^{D-1}\pi_{(0,0,0,0,i)} + \sum_{j=0}^{C-1}\pi_{(2,1,j,0,0)}\right)\cdot\left(\frac{\lambda_1}{\lambda_1+\lambda_2}\right) \end{cases} \tag{3}$$

where R_1 is the total sum of the transition probabilities that the RP directly sends task data to an MP for a task request and R_2 is the total sum of the transition probabilities that the checkpoint of another MP as task data is sent to the MP making a task request by the mediation of the RP.

Second, consider the transmission cost related to checkpointing. Let m_c and d_c denote message cost and data cost for checkpointing, respectively. The cost of one checkpoint message is incurred only when RP receives a checkpoint notification message from an MP. However, there is an exception when an RP is notified of the completion of a task. In this case, the transmission costs of both the task completion message and the final result data are incurred once each. Also, if the checkpoint message and the task completion message have the same cost, the unit cost required by a checkpoint event can be calculated by

$$\begin{aligned} N_1 &= m_\mathrm{c} \\ N_2 &= m_\mathrm{c} + d_\mathrm{c} \end{aligned} \tag{4}$$

where N_1 represents the unit cost required when an MP notifies the RP that it has acquired a checkpoint, and N_2 represents the unit cost required when the MP notifies the RP that it has completed the allocated task. Using Eq. (4) and the transition probability for checkpointing, the total cost of a checkpoint event can be calculated by

$$\begin{cases} C_\mathrm{c} = C_1\cdot N_1 + C_2\cdot N_2 \\ C_1 = \left(\displaystyle\sum_{i=0}^{D-2}\pi_{(1,0,0,0,i)} + \pi_{(2,0,0,0,0)} + \frac{1}{2}\pi_{(2,1,0,0,0)}\right)\cdot\left(\frac{\lambda_1}{\lambda_1+\lambda_2}\right) + \\ \quad \displaystyle\sum_{i=0}^{D-1}\pi_{(D-i,0,0,0,i)} + \frac{1}{2}\left(\pi_{(2,1,0,0,1)} + \sum_{j=0}^{C-1}\pi_{(3,1,j,0,0)}\right) \\ C_2 = \left(\displaystyle\sum_{i=0}^{D-2}\pi_{(1,1,0,0,i)} + \pi_{(2,1,1,0,0)} + \frac{1}{2}\pi_{(2,1,0,0,0)}\right)\cdot\left(\frac{\lambda_1}{\lambda_1+\lambda_2}\right) + \\ \quad \pi_{(3,1,1,1,0)} + \pi_{(2,1,1,0,1)} + \pi_{(1,1,0,0,2)} + \\ \quad \frac{1}{2}\left(\pi_{(2,1,0,0,1)} + \displaystyle\sum_{j=0}^{C-1}\pi_{(3,1,j,0,0)}\right) \end{cases} \tag{5}$$

where C_1 is the total sum of the transition probabilities that an MP notifies the RP of the execution of checkpointing, and C_2 is the total sum of the transition probabilities that the final result of the completed task is transmitted to the RP.

Thus far, using Eqs. (3) and (5), we have calculated the average cost for one task request event and one checkpoint report event. Until a task is completed going through checkpoint sharing and task duplication, on average, D times of task request reports and $D\cdot C$ times of checkpoint reports occur. Thus, the total cost of our scheduling scheme, C_T, is the sum of the task request cost and the checkpoint report cost, as follows:

$$C_\mathrm{T}=D\cdot R_\mathrm{c}+(D\cdot C)\cdot C_\mathrm{c} \tag{6}$$

3.3 Analysis of execution time reduction cost

Here, we examine how much execution time our scheme can reduce as compared to the client-server model. The average execution time ($\overline{t_\mathrm{e}}$) reducing per task request in our scheduling scheme can be calculated

by

$$\overline{t_e} = \left(\sum_{i=0}^{D-2} \pi_{(1,1,0,0,i)} + \sum_{j=0}^{C-1} \pi_{(2,1,j,0,0)} \right) \cdot \left(\frac{1}{C} \cdot T \right) \left(\frac{\lambda_1}{\lambda_1 + \lambda_2} \right) \tag{7}$$

where T is the execution time required when one task is performed from the beginning to the end without the use of the intermediate result, and C is the total number of checkpoints. For analytical convenience, we assume that all MPs have the same performance. Accordingly, all checkpoints will be taken at a periodic cycle. Then, we can see that a mean execution time between two consecutive checkpoints is $\frac{1}{C} \cdot T$. The reduction of the execution time is determined by how few checkpoints the MP has performed after it receives an intermediate result from another MP (i.e., as a checkpoint in an intermediate result is close to a final result, the execution time becomes less, as much as any times of $\frac{1}{C} \cdot T$). On the other hand, tasks in the client-server model are not executed using an intermediate result; rather, even a replica is executed from the beginning. As a result, in the client-server model, if a task is duplicated D times, the total execution time becomes $D \cdot T$. Therefore, the execution time reduction ratio (R_e) of our scheduling scheme to the client-server model is as follows:

$$R_e = \frac{D \cdot \overline{t_e}}{D \cdot T} = \frac{\overline{t_e}}{T} \tag{8}$$

4 Performance evaluation

In this section, the performance of our scheduling scheme is compared to that of the scheduling scheme based on the client-server model. Using the analytical model described in the previous section, the total cost generated by task request and checkpoint events for one task is calculated. For analytical convenience, we assume that the message cost generated in the task request and checkpoint events is identical and that the data cost in the two events is also identical (i.e., $m_r = m_c$ and $d_r = d_c$). Generally, data transmission incurs more costs than message transmission, and so we define a relationship between the two costs as follows:

$$d_r = \alpha \cdot m_r \tag{9}$$

where $\alpha \geq 1$.

In the proposed scheme, an RP can be located either in the identical network where each peer is located, or in a network not far away from each peer; message/data transmission is performed within one hop on average because an RP is located between the central server and MPs. On the other hand, since there is no a special peer such as an RP in the client-server model, more than two hops are needed to transmit messages or data. Thus, it is assumed that message/data transmission of the client-server model costs twice as much as that of the proposed scheme. Table 1 shows the parameters used for performance evaluation.

Table 1 Parameters for performance evaluation

Parameter	Value
Number of duplications per task, D	2, 4, 6
Number of checkpointing per task, C	3, 4, 5
Ratio of message transmission cost to data transmission cost, α	1:100
Ratio between our scheme and client-server model for message/ data transmission cost	1:2
Ratio of task request to checkpoint report, ρ	0.1, 0.2, ⋯, 6.0

The effect of D and C on total costs is now examined. Towards this end, D and C are divided into three cases (D=2, 4, 6 and C=3, 4, 5), respectively. The total costs for the nine cases based on all combinations of D and C are calculated and compared.

Figure 5 shows a relative cost for message/data transmission between the two schemes. The relative cost is defined as the ratio of the message/data transmission cost of the proposed scheme to that of the client-server model. A relative cost of more than 1.0 means that the proposed scheme costs less than the client-server model. From Fig. 5, we can observe that in all combinations of D and C, the relative cost is larger than 1.0. This result indicates that the message/data transmission cost of the proposed scheme is relatively lower than that of the client-server model. As a result, the proposed scheme can distribute the load of message/data transmission as compared to the client-server model.

Now, let us examine a relative cost for the ratio ρ of the task request ratio λ_1 to the checkpoint report ratio λ_2. Figure 5 shows that for all cases of C, the relative cost rapidly declines in a region where $\rho \leq k$. Here, k represents the lowest relative cost in each graph of Fig. 5. For C=3, 4, 5, k is 1.1, 1.5, 1.8, respectively. This sharp decline is because the number of task requests is relatively more than that of the checkpoint reports. Since few MPs receive intermediate results when they request a task to be executed, most task executions are performed from the beginning (i.e., there is a high probability that MPs perform lots of checkpointing without a reduction in the number of checkpoint reports). As a result, the increase in the checkpoint reports causes transmission costs to increase. On the other hand, in the region where $\rho > k$, the relative cost gradually increases. In this region, the number of checkpoint reports is relatively more than that

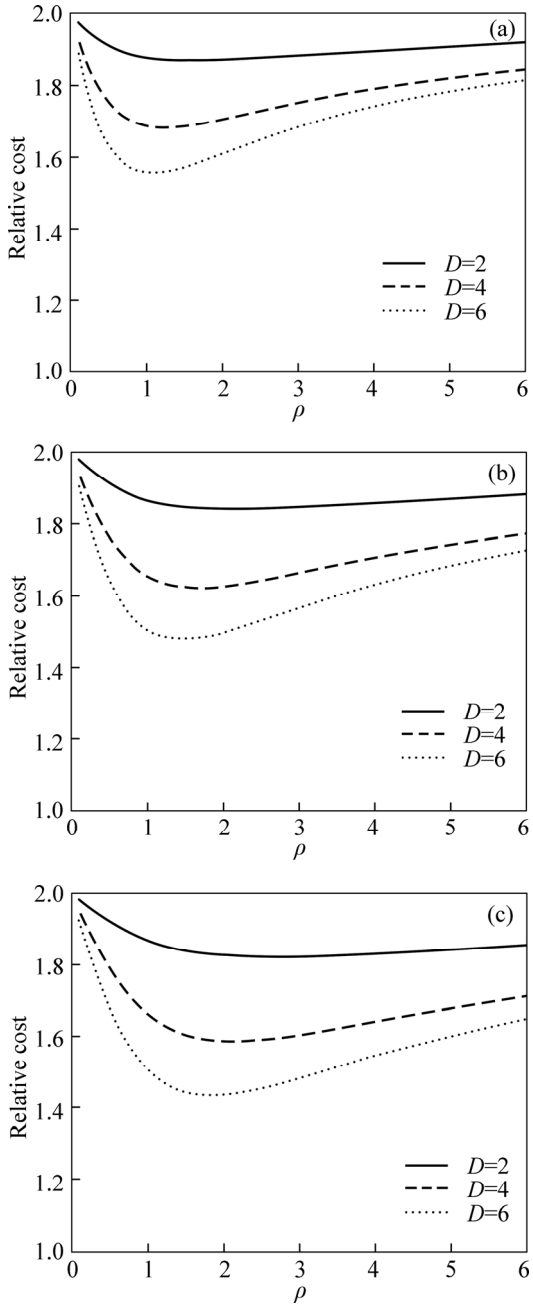

Fig. 5 Message/data transmission cost: (a) C=3; (b) C=4; (c) C=5

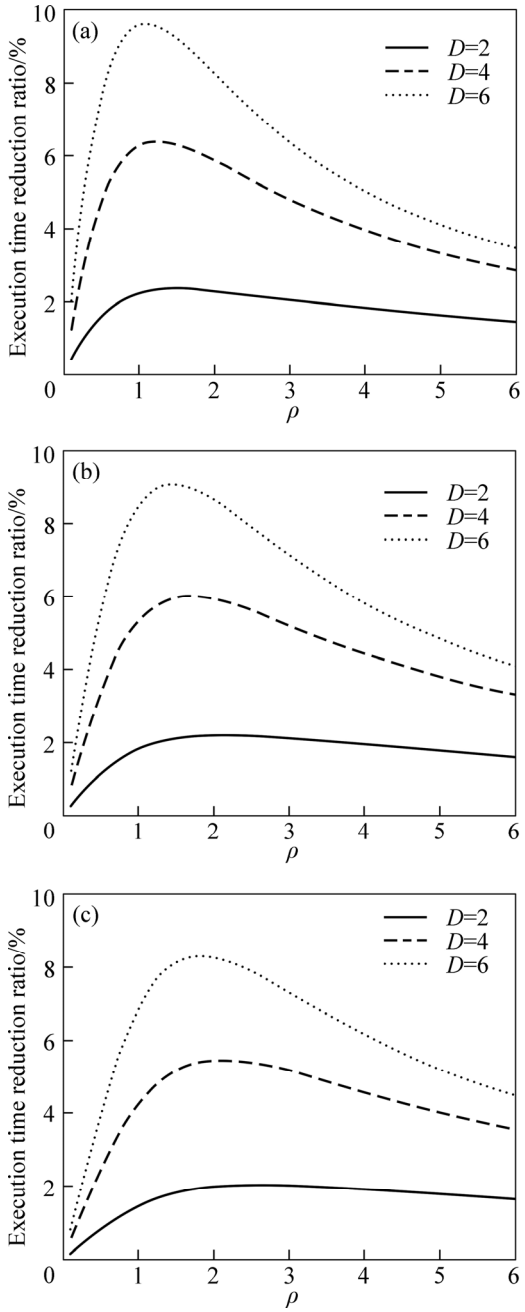

Fig. 6 Execution time reduction ratio: (a) C=4; (b) C=5; (c) C=6

of task requests, and most MPs receive intermediate results from other MPs when they request the task to be executed. Accordingly, the MPs can perform checkpoint reports fewer than C times; therefore, message costs in this region are lower than that in the remaining region.

Figure 6 shows the execution time reduction ratio (ETRR) according to the variations of D and C. From Fig. 6, we can observe that as D increases in relation to each C, the ETRR becomes higher. When D=6 for each C, the execution time of the proposed scheme is reduced as much as approximately 9%–10%, compared to that of the client-server model. This is because when an MP requests a task, the frequency of task executions based on the checkpoint of other MPs becomes more as D increases. It should be noted that as the checkpoint becomes closer to a final result, the reduction of execution time per task becomes higher. Consequently, by means of checkpoint sharing and task duplication, we can see that the proposed scheme has a shorter execution time than the client-server model.

To evaluate the effect of task failures on our scheduling scheme, we conduct the simulations with task failure rates. For performance comparison, the scheduling scheme based on the client-server model is simulated under the same conditions as those used in our scheduling scheme. The failure event in each MP is

generated by Poisson process with a mean rate of f (i.e., each MP can encounter the task failures artificially generated), which will eventually bring about the events that the deadline of task execution is exceeded or intermediate execution results are not delivered to other MPs. As a result, the failed tasks are re-executed on other MPs.

As a performance measure, we use the turnaround time, which is defined as the total time taken between the submission of the first task for execution and the return of the complete result of the last task. The total number of tasks used in the simulations is 1000.

Figure 7 shows the simulation results of the two scheduling schemes according to a variation of the number of duplications (D) when the number of checkpoints (C) and a failure rate (f) are 5 and 0.001, respectively. As shown in Fig. 7, our scheduling scheme has less turnaround time with an increase in the number of duplications than the scheduling scheme based on the client-server model. This is because as the number of duplications is higher, our scheduling scheme can restart more failed tasks from the last checkpoint position of the task, instead of restarting from the beginning.

Figure 8 shows the simulation results of the two scheduling schemes according to a variation in the number of checkpoints (C) when the number of duplications (D) and a failure rate (f) are 5 and 0.001, respectively. The results in Fig. 8 also indicate that our scheduling scheme has less turnaround time than the scheduling scheme based on the client-server model. When an MP requests a task with many checkpoints, it can receive an intermediate result that is close to the final result. This results in fast task completion, leading to a reduction in turnaround time.

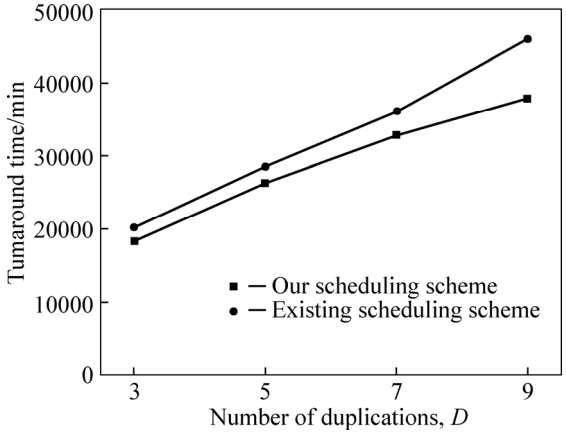

Fig. 7 Performance comparison according to a variation in number of duplications (D)

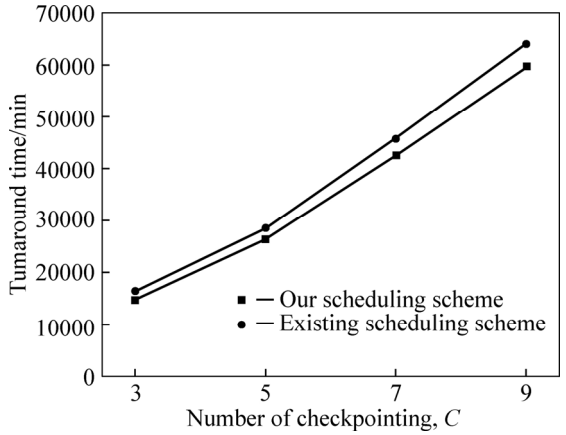

Fig. 8 Performance comparison according to a variation in number of checkpoint (C)

5 Conclusions

1) The analysis of transmission cost and execution time by the embedded Markov chain shows that the transmission cost of the proposed scheme somewhat increases when the number of duplications per task increases.

2) However, our scheduling scheme shows that when checkpointing is executed more and more, it can considerably reduce the execution time compared to the scheduling scheme based on the client-server model.

3) Our scheduling scheme also has less turnaround time than the scheduling scheme based on the client-server model, even if task failures occur.

4) Consequently, our scheduling scheme can reduce the time it takes for an application user to obtain a final task result. It is expected that when our scheduling scheme is implemented in an actual desktop grid system, it will be useful in minimizing the turnaround time of large-scale applications.

Acknowledgments

This research was supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (2012R1A1A4A0105777) and supported by the MSIP (Ministry of Science, ICT and Future Planning), Korea, under the ITRC (Information Technology Research Center) support program (NIPA-2013-H0301-13-4007) supervised by the NIPA (National IT Industry Promotion Agency).

References

[1] EMMANUEL U. Cloud, grid and high performance computing [M]. IGI Global, 2011: 135−154.

[2] ZHAO H, LIU X, LI X. A taxonomy of peer-to-peer desktop grid paradigms [J]. Cluster Computing, 2011, 14(2): 129−144.

[3] SETI@Home Project [EB/OL]. http://setiathome.ssl.berkeley.edu. 2014.

[4] CESARIO E, de CARIA N, MASTROIANNI C, TALIA D. Distributed data mining using a public resource computing framework [M]// Grids, P2P and Services Computing. Springer, 2010: 33−44.

[5] Berkeley Open Infrastructure for Network Computing (BOINC)

[EB/OL]. http://boinc.berkeley.edu/. 2014.

[6] PATNI J C, ASWAL M S, PRAKASH O M, GUPTA A. Load balancing strategies for grid computing [C]// International Conference on Electronics Computer Technology. Kanyakumari, 2011: 239−243.

[7] URBAH E, KACSUK P, FARKAS Z, FEDAK G, KECSKEMETI G, LODYGENSKY O, MAROSI A, BALATON Z, CAILLAT G, GOMBAS G, KORNAFELD A, KOVACS J, HE H, LOVAS R. EDGeS: Bridging EGEE to BOINC and XtremWeb [J]. Journal of Grid Computing, 2009, 7(3): 335−354.

[8] Korea@Home [EB/OL]. http://www.koreaathome.org/eng/. 2010.

[9] KACSUK P, KOVACS J, FARKAS Z, MAROSI A C, GOMBAS G, BALATON Z. SZTAKI Desktop grid (SZDG): A flexible and scalable desktop grid system [J]. Journal of Grid Computing, 2009, 7(4): 439−461.

[10] VLADOIU M, CONSTANTINESCU Z. Development journey of QADPZ–A desktop grid computing platform [J]. International Journal of Computers, Communications & Control, 2009, 44(1): 82−91.

[11] Entropia [EB/OL]. http://enterthegrid.com/. 2014.

[12] United devices [EB/OL]. http://www.univa.com/. 2014.

[13] International desktop grid federation [EB/OL]. http:// desktopgridfederation. org/applications/. 2014.

[14] PATAKI M, MAROSI A C. Searching for translated plagiarism with the help of desktop grids [J]. Journal of Grid Computing, 2013, 11(1): 149−166.

[15] GIL J M, KIM M. A log analysis system with REST Web services for desktop grids and its application to resource group-based task scheduling [J]. Journal of Information Processing Systems, 2011, 7(4): 707−716.

[16] BOUGUERRA M S, KONDO D, TRYSTRAM D. On the scheduling of checkpoints in desktop grids [C]// 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing. Newport Beach, CA, USA, 2011: 305−313.

[17] WANG D, GONG B. On the checkpointing strategy in desktop grids [J]. Lecture Notes in Computer Science, 2012, 7648: 217226.

[18] DOMINGUS P, SILVA J G., SILVA L. Sharing Checkpoints to improve turnaround time in desktop grid computing [C]// 20th International Conference on Advanced Information Networking and Applications. Vienna, Austria, 2006: 6−11.

[19] XtremWeb [EB/OL]. http://www.xtremweb.net/index.html. 2014.

[20] KWOK Y K R. Peer-to-peer computing: Applications, architecture, protocols, and challenges [M]. CRC Press, 2011: 8−9.

[21] FELDMAN R M, VALDEZ-FLORES C. Applied probability and stochastic processes [M]. Springer, 2010: 115−132.

(Edited by YANG Bing)