A-027

ランポートタイムスタンプを使ったピュア P2P 型分散チェックポイントアルゴリズムにおける自発的な取得法
Spontaneous method in pure P2P distributed checkpointing using Lamport timestamps

谷本 豊‡　　守屋 宣‡
Yutaka Tanimoto　Sen Moriya

1. はじめに

分散アプリケーションシステムに耐故障性を付加するアプローチには主に、故障プロセスの影響を受けることなく正常なプロセスのみで正しく処理を進める故障排除と、故障プロセスを正しい状態に復帰させる故障復旧とがある。故障排除は、正常なプロセスへのサービスを継続できる点では有用であるが、システムが許容できない大規模な故障が発生した場合には対処できないことも考えられる。一方、故障復旧では、場合によっては継続中のサービスを停止させて復旧させる必要はあるが、どのような故障にも対応可能なアプローチと言える。特に分散アプリケーションシステムで故障復旧を行うには、プロセスが状態保存(チェックポイントの取得)を行い、故障が発生した場合に保存状態に復旧(ロールバック)させることで実現できる。このとき、各プロセスが自律的にチェックポイントを取得するだけにすると、ドミノ効果[1]により状態の一貫性を維持したままチェックポイント状態にロールバックすることができない。そのため、チェックポイントはシステム内のプロセスが連携して取得しなければならない。

これまで様々なチェックポイント取得法が提案されている。主なアプローチによる手法の通信複雑度、付加情報量を表 1 にまとめる。 (a)や(b)のアプローチは、通信計算量または付加情報量がシステム内のプロセス数に依存しており、莫大な数のプロセスが参加するような近年の実システムには適用が困難な手法となる。一方、(c)のアプローチによる手法では、プロセスごとの動作速度の差が大きいシステムでは、ロールバックに利用できるまでにかかる時間(取得時間とよぶ)が長くなりがちとなる。そこで著者らはこの点を考慮し、アプローチ(d)として定数サイズの情報をメッセージに添付しながら、チェックポイント取得のために定数個のメッセージ交換をする手法を提案している[5]。この手法では、ハイブリッド P2P システムのような一般のプロセスとサーバから成るシステムにおいて、論理時計として知られるランポートタイムスタンプ[6]（以下、タイムスタンプ）を使ってチェックポイント取得を行い、チェックポイント取得の取得時間を短縮させる方法について評価している。この手法は、理論上では効率はシステム参加プロセス数には依存しておらず、莫大な数のプロレスが参加するシステムにも適用可能である。ただし、サーバが必要であり、サーバ負荷についても考慮する必要が生じる。

表 1　主なチェックポイント取得法のアプローチ

	通信計算量	付加情報量
(a)特別なメッセージを交換[2]	$O(n^2)$	0
(b)プロセス数サイズ情報を添付[3]	0	$O(n)$
(c)定数サイズの情報を添付[4]	0	$O(1)$
(d)定数サイズの情報を添付、定数個のメッセージを交換[5]	$O(1)$	$O(1)$

n: プロセス数, *: 物理時計の同期手法に依存

‡ 近畿大学 Kindai University

文献[7]では、サーバを利用せずに取得時間を短縮させる方法として、ピュア P2P システムでのオブジェクト探索に使われる乱択フラッディングを使う手法について考察している。ただし、この文献で考察している手法では、プロセスはタイムスタンプに依存したタイミングのみでチェックポイントを取得しており、各プロセスが任意のタイミングで自発的にチェックポイントを取得する必要があるシステムについて対象としていなかった。そこで本稿では、文献[7]の手法に自発的にチェックポイント取得をする機能を加える手法について検討を行い、その効果をシミュレーション実験により評価する。

2. 諸定義

2.1 システム

メッセージ交換により相互に通信を行う複数のプロセス $P=\{P_1,P_2,\cdots,P_i,\cdots\}$、通信リンク$L$から構成される図 1 のようなシステム$(P,L)$を考える。各プロセスでは、アプリケーションが上位、モニタが下位で動作し、モニタはアプリケーションが交換するメッセージや状態の監視、アプリケーションからのチェックポイント取得依頼を受け、モニタ同士でもメッセージ交換を行う。プロセス間で交換するメッセージの受信順序は、FIFO が保証されているとする。本研究で想定するシステムではプロセスの参加、離脱が起こりうるとするが、これをプロセスの状態がスリープとアクティブとに変化することでモデル化する。プロセスは故障することもあるが、故障をするのはアクティブ状態のときのみとする。各プロセスは物理時計を参照しないが、タイマーによりアプリケーションの動作が一定時間σの間なかったことを検知できるとし、アクティブの状態でσ時間動作がなければスリープへ移行する。σはシステムで決められているタイムアウトを想定しており、本稿では全てのプロセスでσは同じであるとする。

システム内の全てのプロセスの状態と通信リンクを伝播

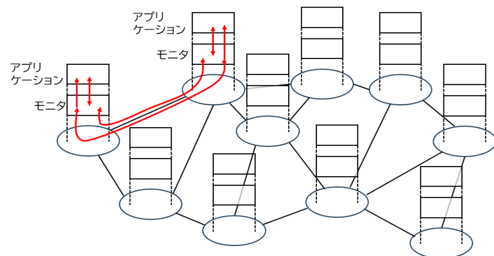

図 1 システムモデル

Copyright © 2025 by
The Institute of Electronics, Information and Communication Engineers and
Information Processing Society of Japan All rights reserved.

中のメッセージから成る集合を状況、プロセスのアプリケーションのメッセージ送受信や内部状態の変化をイベントとよぶ。システムの実行を、状況$C_i(i=0,1,2,\cdots)$とイベント$e_i(i=1,2,3,\cdots)$の無限交替列$E=C_0e_1C_1e_2C_2\cdots e_iC_i\cdots$で定義する。実行に現れるイベント間の前後関係を、以下のように定義する。

[定義 1] 実行Eに現れるイベントの集合を$\mathcal{E}^E$とする。このとき、イベント$e,e'\in\mathcal{E}^E$に対し(1)～(3)のいずれかが成り立つとき、記号→を用いて$e\rightarrow e'$と表記する。

(1) eとe'は同じプロセスでe,e'の順に生起している。

(2) eはメッセージが送信された際の送信イベント、e'は同じメッセージの受信イベントである。

(3) (1)(2)の推移律が成り立つ。■

このとき、実行の無矛盾な切断を次のように定義する。

[定義 2] 実行Eの**無矛盾な切断**cとは、Eに現れる任意のイベント$e,e'\in\mathcal{E}^E$に対して$e\in\mathcal{E}^E_{F(c)}$かつ$e\rightarrow e'$ならば$e'\in\mathcal{E}^E_{F(c)}$が成り立つような、$E$の全イベント$\mathcal{E}^E$の二つの部分集合$\mathcal{E}^E_{P(c)}$と$\mathcal{E}^E_{F(c)}$への分割である。■

無矛盾な切断は、プロセスごとにあるイベントe_tまでのイベントを$\mathcal{E}^E_{P(c)}$に、それよりあとのイベントを$\mathcal{E}^E_{F(c)}$に分割することで構成できる。このとき、$\mathcal{E}^E_{P(c)}$に属する送信イベントで送信されてから$\mathcal{E}^E_{F(c)}$に属する受信イベントで受信されるメッセージを含めるようにすれば、各プロセスP_iのe_t直前の状態に基づいて、必ずしも状況として現れているわけではないがプロセス間のメッセージ送受信の一貫性が維持されている**無矛盾な状況**を構成できる。無矛盾な状況を構成できるならば、あるプロセスが故障したとしても、故障発生前の無矛盾な状況にロールバックできるようになる。

2.2 ランポートタイムスタンプ

任意の実行Eのイベントe,e'に対し、$e\rightarrow e'$ならば$c(e)\rightarrow c(e')$を満たすようにイベントeに割り当てられる値$c(e)$を**論理時計**とよぶ。代表的な論理時計に、ランポートタイムスタンプ[7](以下、**タイムスタンプ**)がある。タイムスタンプは整数値であり、プロセス間で交換されるメッセージに、その送信イベントに割り当てられる値が添付される。各プロセスの送信イベントや内部イベントには、そのプロセスの前のイベントeに対して$c(e)+\alpha(\alpha\geq 1)$が割り当てられる。メッセージ$M$の受信イベントには、メッセージ$M$に添付される値$M.c$に対し、$\max\{c(e),\ M.c\}+\alpha(\alpha\geq 1)$が割り当てられる。通常は$\alpha=1$とするが、必要であれば増加幅$\alpha$を1より大きくする。タイムスタンプをもとにすると、ある値Tに対し、$c(e)<T$なるイベントeは$\mathcal{E}^E_{P(c)}$に、$c(e)\geq T$なるイベントeは$\mathcal{E}^E_{F(c)}$に属する切断cは無矛盾な切断となる。さらに、メッセージのうち、送信イベントで添付されたタイムスタンプがTより小さく、受信イベントのタイムスタンプがT以上であるメッセージを含めるようにすれば、無矛盾な状況を構成できる。

図 2 に、タイムスタンプと、$T=10,20$として構成した無矛盾な切断の例を示す。

3. チェックポイントアルゴリズム

チェックポイントアルゴリズムを、ロールバックに利用できる状況で初期状況でない無矛盾な状況を構成できるよう、すべてのプロセスがチェックポイントを取得するアルゴリズムと定義する。

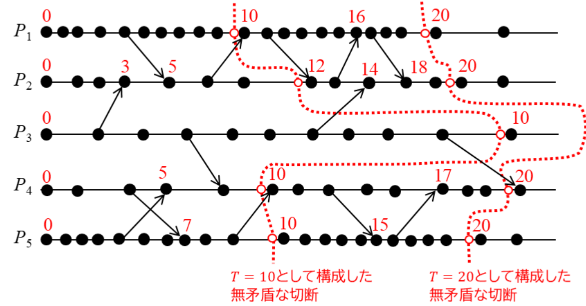

図 2 無矛盾な切断の例

前節で述べたとおり、タイムスタンプをもとに無矛盾な切断を設定でき、ロールバックに利用できる無矛盾な状況を構成できる。この考えにもとづいた手法で本稿で検討するチェックポイントの取得法について述べる。

3.1 定期チェックポイントの取得

あらかじめ、無矛盾な状況を構成するタイムスタンプの間隔T_δを決めておき、タイムスタンプが初めて$k\cdot T_\delta$以上$(k=0,1,2,\cdots)$となるイベントの直前にチェックポイントを取得する。このとき、同じkに対して取得されたチェックポイントにより、プロセスが一貫性を維持したロールバックに利用するためのチェックポイントを取得できる。無矛盾な状況に含めるメッセージについては、タイムスタンプが$k\cdot T_\delta$以上のイベントで受信するメッセージのうち、送信時のイベントのタイムスタンプが$k\cdot T_\delta$より小さいメッセージを含めるようにする。以下、タイムスタンプ$k\cdot T$で取得されるチェックポイントを**ラウンドkのチェックポイント**とよぶ。プロセスがラウンドkのチェックポイントを取得したのち、送信元プロセスがラウンドkのチェックポイントを取得する前に送信されたメッセージを受信することがある。このようなメッセージの情報はロールバックのときに必要となるため、ラウンドkのチェックポイントの状態に紐づけして保存しておく。なお、送信がラウンドkのチェックポイント取得前かどうかは、メッセージに添付されているタイムスタンプにより判断できる。

この手続きにより取得されたチェックポイントを使ったロールバックは次のように行う。あるプロセスの故障などにより過去の状態にロールバックする必要が生じた場合は、その影響がないチェックポイントのラウンドkをロールバック先として決定する。ラウンドkのチェックポイントを取得済みの各プロセスP_iは、ラウンドkのチェックポイント状態を復元し、送信元プロセスがラウンドkのチェックポイント取得前に送信したメッセージでP_iがラウンドkのチェックポイント取得後に受信したメッセージの受信処理を復元状態に反映させて、動作を再開させる。このとき、ラウンドkのチェックポイントを取得していないプロセスは、その時点の状態を無矛盾な状況に含められるため、ロールバック処理をする必要はない。

チェックポイントを本小節で述べた定期チェックポイントのみで取得するのみとすると、問題点がある。タイムスタンプはプロセスが動作しない限り増加しないため、各プロセスがいつラウンドkのチェックポイントを取得するかわからない。そのため、タイムスタンプにより定期チェックポイントを取得するのみとすると、プロセスごと動作速度の差が大きい場合に、動作の遅いプロセスの故障が発生

Copyright © 2025 by
The Institute of Electronics, Information and Communication Engineers and
Information Processing Society of Japan All rights reserved.

したときに動作の速いプロセスが古いチェックポイントに戻らなければならなくなる。さらに、プロセスの始動をタイムスタンプ0からとすると、プロセスの始動と故障が繰り返し発生することにより、ラウンド0のチェックポイントにロールバックしうる状況が無限に続くことになる。そこで、3.2 節以降で述べるとおり、定期チェックポイント以外によるチェックポイント取得を行う。

3.2 乱択フラッディングによりタイムスタンプを調整したチェックポイントの取得

3.2.1 定期チェックポイント前のタイムスタンプ調整

動作の速いプロセスがラウンドkのチェックポイントを取得してから動作の遅いプロセスが大幅に遅れてチェックポイントを取得することを防ぐため、定期チェックポイントを取得する前に乱択フラッディングにより他のプロセスにチェックポイント取得済みのラウンドを問い合わせる。本稿のモデルでは、プロセスはタイムアウトの時間σを計測する以外に実時間を測ることはできない。そこで、定期チェックポイントのタイムスタンプ間隔T_δに対し、乱択フラッディングを起動するタイミングT_{fld}を決めておき、タイムスタンプが各kに対して初めて$(k \cdot T_\delta - T_{fld})$以上となるイベント時に転送プロセス数$c$、ホップ数$h$の取得済みラウンドを問い合わせる乱択フラッディングを起動する。フラッディングされた問い合わせメッセージを受信したプロセスは、大きいラウンドのチェックポイントを取得済みならば、そのラウンドをフラッディング起動プロセスへ応答する。その後、フラッディング起動プロセスはタイムスタンプが$k \cdot T_\delta$になるまでフラッディング先のプロセスからの応答を受信し、ラウンドkより大きいラウンドの応答を受信した場合は、その最大値$k'(> k)$に対してラウンドkからk'のチェックポイントを 1 つのチェックポイントでまとめて取得する。なお、取得済みのラウンドを問い合わせる乱択フラッディングのメッセージとその応答には送信時のタイムスタンプの添付はせず、それらの受信時もタイムスタンプの更新は行わない。

実行例として、図 3 に定期チェックポイント前のタイムスタンプ調整をせずに、図 4 に定期チェックポイント前にタイムスタンプ調整をしてチェックポイントを取得する例を示す。両方の図でT_δは 10 としている。また、図 4 におけるT_{fld}を 2 としており、ラウンドを問い合わせる乱択フラッディングはP_3のもののみ表記している。この例では、図 3 でP_3がラウンド 1 のチェックポイントを取得したときに、図 4 ではラウンド 1 と 2 のチェックポイントをまとめて取得でき、ラウンド 2 の取得時間が短縮される。

3.2.2 アクティブ化チェックポイントでのタイムスタンプ調整

プロセスが初めて動作するときや、プロセスでタイマーの時間σ以上イベントが生起しなかったあとで初めてイベントが生起したときなど、プロセス状態がスリープからアクティブに移行したときは、動作を開始／再開する前にまず、定期チェックポイント前のタイムスタンプ調整と同様に、乱択フラッディングにより取得済みラウンドの問い合わせを行いタイムスタンプの調整を行う。アクティブ化チェックポイントでのタイムスタンプ調整では、1 つのプロセスから応答を受信したら、その応答のラウンドに従ってチェックポイントを取得する。ラウンドkまでのチェックポイント取得済みのプロセスがラウンド$k'(> k)$の応答を受信した場合は、ラウンド$k+1$からk'のチェックポイントを 1 つのチェックポイントでまとめて取得する。これにより、小さいラウンドのチェックポイント取得により速度の速いプロセスが他のプロセスの故障により古いチェックポイントへのロールバックを強いられるリスクを低減する。応答のラウンドk'がk以下の場合は、チェックポイント取得は行わない。

3.3 自発チェックポイントの取得

チェックポイントは、定期的なバックアップを取るだけでなく、高額な決済などアプリケーションが何か重要な計算をした結果の状態を保存したいために取得したいこともある。定期チェックポイントを基本としたチェックポイントだけでは、そのような特に保存したい状態をすぐに保存できるとは限らない。そこで、アプリケーションが特に保存したい状態に到達したとき、タイムスタンプの増加幅αを 1 より大きくすることで、次のチェックポイントを取得できるようにする。すなわち、最新のチェックポイントのラウンドをkとすると、タイムスタンプの増加幅αを 1 より大きくして$k' \cdot T_\delta (k' > k)$とすることで、新たなチェックポイントを取得する。このように、アプリケーションが特に保存したい状態に到達して取得されたチェックポイントを**自発チェックポイント**とよぶ。k'の値は、乱択フラッディングに対する応答の受信でラウンドがkより大きいものがあればその最大値に、kより大きいラウンドの応答がなければ$k' = k + 1$にする。

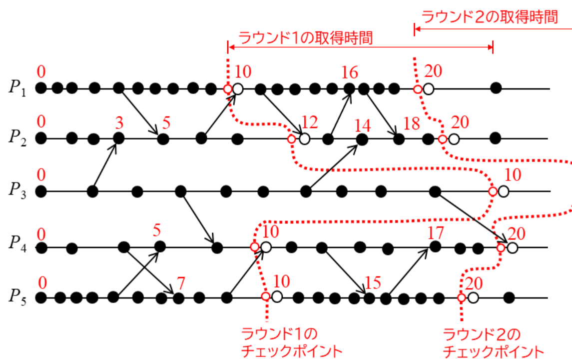

図 3 定期チェックポイントのみの実行例

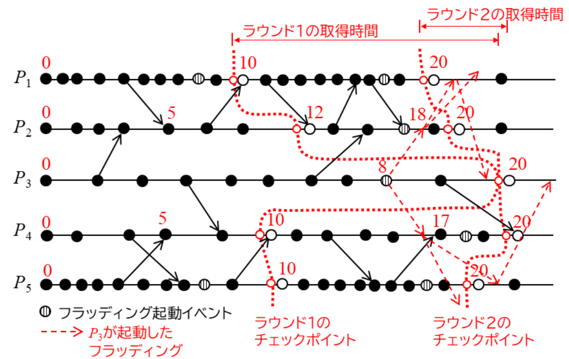

図 4 タイムスタンプ調整がある実行例

Copyright © 2025 by
The Institute of Electronics, Information and Communication Engineers and
Information Processing Society of Japan All rights reserved.

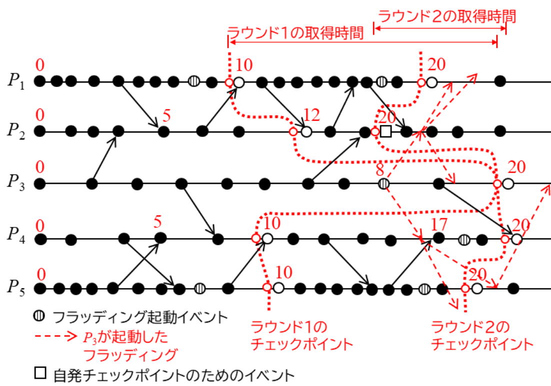

図 5 自発チェックポイントがある実行例

自発チェックポイントを含む実行例を図 5 に示す。図 5 では、P_2がラウンド 2 のチェックポイントを自発チェックポイントにより取得している。自発チェックポイントの取得がある場合、自発チェックポイントがない場合と比べてチェックポイント取得回数が多くなる。

3.4 チェックポイントアルゴリズムの性質

本節で述べているチェックポイントアルゴリズムの性質をまとめる。

プロセスが遅くとも次に動作したときにラウンドkのチェックポイント取得済みとなる状態を、**ラウンドk確定状態**とよぶ。先述したとおり、3.1 節で述べた定期チェックポイントのみとすると、ラウンド0のチェックポイントにロールバックしうる状況が無限に続く場合がある。しかし、3.2.2 節で述べたアクティブ化チェックポイントにより、ある状況c_tにおけるアクティブ状態のプロセスの最新のチェックポイントラウンドの最小値を$minr_t$とすると、c_tでスリープ状態のプロセスはラウンド$minr_t$確定状態となっている。以上より、任意のkに対して、やがてラウンドkより前のラウンドのチェックポイントにロールバックしなくてよい状況に到達することが保証される。

計算量は次のとおりである。アプリケーションのメッセージに添付する情報はサイズ$O(1)$のタイムスタンプのみなので、付加情報量は$O(1)$である。また、1 プロセスが 1 回チェックポイントを取得するために必要なメッセージ数は乱択フラッディングに依存し、転送プロセス数c、ホップ数hに対して$O(c^h)$となる。したがって、本節で述べたチェックポイントアルゴリズムの通信計算量は$O(c^h)$となる。

4. チェックポイントアルゴリズムの評価

3 節で述べたチェックポイントアルゴリズムの性能を確認するため、プロセス数 1000 個が完全グラフのトポロジーで接続し、プロセスごとの平均動作速度が$2 \le x \le 18$の一様分布となる確率変数xに対してイベントの生起間隔が平均x秒の指数分布とするシステムを想定したシミュレーション実験を行った。システムのモデルやチェックポイントアルゴリズムに現れるパラメータは、σ：40(秒)、T_δ：30、T_{fld}: 15 とする。今回の評価実験では、乱択フラッディングにおける転送プロセス数cは$4, 8, 12, 16$に変化させ、ホップ数hは1、すなわち乱択したプロセス集合に 1 回マルチキャストするのみとした。比較のため、乱択フラッディングを行わない場合（調整なし）の実験も実施した。自発チェックポイントについては、取得しない場合（自発 CP なし）と 1/1000 の割合のイベントで取得する場合（自発 CP あり）の評価を行った。このシミュレータで、同一条件で 60 分動作させる実験を 20 回行い、取得時間の平均を評価した。取得時間は、チェックポイントの各ラウンドkに対し、あるプロセスが最初にラウンドkのチェックポイントを取得してから、すべてのプロセスがラウンドkのチェックポイントを取得するかラウンドk確定状態となるまでとする。

実験結果を図 6 に示す。実線は実験における取得時間を表し、点線はその線形近似である。乱択フラッディングによる問い合わせ先を増加させると取得時間は短くなる傾向はあったが、期待していたほどの効果を見ることはできなかった。自発 CP ありの場合、自発 CP なしに比べて取得時間がやや長くなった。これは、自発 CP ありの場合はチェックポイントを取得するタイミングが早くなること、速度の遅いプロセスがチェックポイントを取得するタイミングも早くなるため同一時間内で取得するチェックポイント数が多くなることなどが理由として挙げられる。

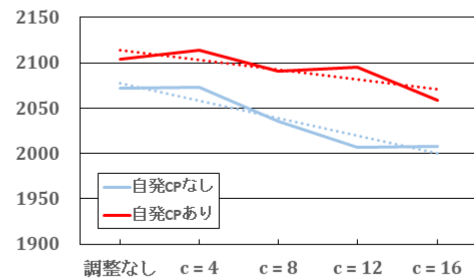

図 6 実験結果

5. おわりに

本稿では、タイムスタンプを利用する分散チェックポイントアルゴリズムとそのアルゴリズムで自発的なチェックポイント取得をする手法について述べたうえで、評価実験の結果を示した。今後の課題として、乱択フラッディンにおける通信計算量を最適化する、手順に関する理論的な解析などが挙げられる。

謝辞

本研究の一部は、JSPS 科研費 24K14907 の助成を受けた。

参考文献

[1] B. Randell, “System structure for software fault tolerance,” Proc. International Conference on Reliable Software, pp.437-449, (1975).
[2] K.M. Chandy and L. Lamport, “Distributed snapshots: Determining global states of distributed systems,” ACM Trans. Comput. Syst., vol.3, no.1, pp.63-75, (1985).
[3] C.J. Fidge, “Timestamps in message-passing systems that preserve partial ordering,” Proc. the 11th Australian Computer Science Conference, pp.56-66, (1988).
[4] T,H, Lai and T.H. Yang, “On distributed snapshots,” Information Processing Letters, vol.25, no.3, pp.153-158, (1987).
[5] 東 瞭斗, 守屋 宣, “分散アプリケーションのためのハイブリッド P2P 型分散チェックポイント・ロールバックアルゴリズムとその評価”, 電子情報通信学会論文誌 D, Vol.J103-D, No.1, pp.1-12, (2020).
[6] L. Lamport, “Time, clocks, and the ordering of events in a distributed system”, Commun. ACM, vol.21, no. 7, pp.558-565, (1978).
[7] 谷本 豊, パン・ジュンジェ, 守屋 宣, “ランポートタイムスタンプを使ったチェックポイント取得における乱択フラッディングの利用”, 第 87 回情報処理学会全国大会, pp.1-401--1-402, (2025).

Copyright © 2025 by
The Institute of Electronics, Information and Communication Engineers and
Information Processing Society of Japan All rights reserved.