ディレクトリサービスにおける ソフトステートとハードステートの性能比較

多田知正[†] 今瀬眞[††] 村田正幸[††]

グリッドコンピューティングやユビキタス環境においてディレクトリサービスは重要である．ディレクトリサービスの実装方法はソフトステートとハードステートに大別され，近年はソフトステートが広く用いられている．しかしディレクトリサービスにおけるそれぞれの手法の性能については明らかになっていない．ディレクトリサービスにおいてソフトステートおよびハードステートの性能を評価する場合には，システム障害の影響について考慮する必要がある．本論文ではディレクトリサービスにおいて，障害の発生する状況における，ソフトステートとハートビートによる障害検出を用いたハードステートの性能比較を，マルコフモデル解析を用いて行った．その結果，ハードステートは，パス障害の発生頻度が高い場合にはソフトステートの約 5 倍，受信ノード障害の発生頻度が高い場合にはソフトステートの約 1.4 倍の不一致率を示し，一般に考えられているように，これらの障害に対してソフトステートが有利であることが確認された．またその一方で，送信ノード障害の発生頻度が高い場合には，ハードステートの不一致率がソフトステートの約 1/100 になることがあり，つねにソフトステートの方が障害に対して高い性能を示すとは限らないことも明らかになった．また，実際にディレクトリサービスでどちらの手法を用いるべきかは，計算機や通信パスの信頼性，ディレクトリサーバの数，通信リンクの回線容量，資源情報の更新頻度，誤った資源情報のおよぼす影響の大きさなどといった要因に依存することを示した．

Performance Comparison of Soft State and Hard State in Directory Services

Harumasa Tada,[†] Makoto Imase[††] and Masayuki Murata[††]

Directory services are important in grid computing and ubiquitous environment. Two approaches to implement directory services are soft state and hard state. Though soft state is preferred rather than hard state in recent years, their performance in directory services is not well understood. A performance comparison of soft state and hard state in directory services should consider the effect of system failures. In this paper, we compared the performance of soft state and hard state with heartbeat failure detection under failures using Markov model analysis. The result of analysis is that inconsistency ratio of hard state is about 5 times higher than that of soft state when path failures occur frequently and is about 1.4 times higher when receiver failures occur frequently. The result ensured the popular belief that soft state is preferable for these type of failures. On the other hand, inconsistency ratio of hard state is about 1/100 of that of soft state when sender failures occur frequently. The result showed that soft state was not always better than hard state for failures. Moreover, we showed some actual factors which affect choice between soft state and hard state in directory services. They are host reliability, network reliability, the number of directory servers, network link capacity, frequency of update of resource information, magnitude of the effect of false resource information and so on.

1. はじめに

近年の通信技術の進歩と計算機器の普及により，地理的に分散した資源やサービスを，ローカルな資源と同じように管理することのできる，グリッドやユビキタス環境のようなインフラストラクチャが登場しつつあり，これにより，革新的な広域分散アプリケーションが可能になった．これらのインフラストラクチャにおいては，ディレクトリサービスが重要である[1)]．

本論文で対象とするディレクトリサービスとは，地理的に分散した計算機が保持する資源に関する情報

† 京都教育大学教育学部
Faculty of Education, Kyoto University of Education
†† 大阪大学大学院情報科学研究科
Graduate School of Information Science and Technology, Osaka University

（たとえば，ファイルの位置情報やメタデータ，あるいはCPUの処理能力や負荷など）を計算機間で共有するためのサービスである．資源を保持する計算機は，ディレクトリサーバに対して，資源に関する情報（以下資源情報）を送り，ディレクトリサーバは受け取った資源情報を保持する．資源を利用するクライアントは，ディレクトリサーバに対して資源情報の問合せを行う．ディレクトリサービスの性能にはいくつかの基準が考えられるが，本論文では，問合せによって得られる資源情報の正しさに着目する（詳細は2章で述べる）．

ディレクトリサービスの実装手法にソフトステートとハードステートがある[2)]．ソフトステートでは，ディレクトリサーバはタイムアウトにより資源情報を削除するため，資源を保持する計算機はディレクトリサーバに定期的にメッセージを送信してタイマをリセットする必要がある．一方，ハードステートでは，ディレクトリサーバは資源情報の削除を指示するメッセージを明示的に受信するまで資源情報を保持し続ける（詳細は3章で述べる）．

近年ディレクトリサービスにおいてソフトステートが多く用いられている[2)〜5)]．しかしながら，ソフトステートとハードステートのディレクトリサービスにおける性能については明らかになっていない．ソフトステートが，性能に関する詳細な検討を加えられないまま，おそらく技術開発者らの経験的な知見によって広く採用されているのが現状である．ディレクトリサービスにおけるソフトステートとハードステートの振舞いの違いを明らかにし，それぞれがより適切な状況に用いられるようになることが望ましい．

ディレクトリサービスにおいてソフトステートとハードステートの比較を行うためには，障害発生がシステム性能にどのような影響を与えるかを考慮することが本質である．障害により，ディレクトリサーバと資源を保持する計算機の間に資源情報のずれが生じる場合がある．たとえば，ディレクトリサーバの障害により，ディレクトリサーバが保持する資源情報が失われると，他の計算機がその資源を利用できず，資源の利用効率が低下する．また，資源を保持する計算機の障害やパスの障害により，ディレクトリサーバ上の間違った資源情報が更新されないまま残ってしまうと，存在しない資源への無効なアクセスが発生し，資源へのアクセスにかかる時間が増大する．このように障害発生によって生じる資源情報のずれは，ディレクトリサービスの性能に大きく影響する．ソフトステートとハードステートでは，資源情報のずれが解消されるまでの時間が異なり，これにより両者の障害発生時の性能には大きな差があると考えられる．

通信システムにおけるソフトステートの性能評価についてはいくつかの研究が行われている．文献6)ではソフトステートを待ち行列モデルを用いてモデル化し，ソフトステートの性能やコストについて議論している．文献7)ではマルコフモデル解析により，ソフトステート，ハードステート，ソフトステートにハードステートの特徴を採り入れた拡張手法の性能の比較を行っている．また，文献8)ではネットワークを流れるデータ量やロス率が平常時より著しく高い場合のソフトステートとハードステートの振舞いについて比較を行っている．しかし，これらの評価においては，メッセージ消失による影響は考慮されているが，計算機やネットワークにおける障害の発生による影響については考慮されていない．

これまでの研究[6)〜8)]では通信システムを対象とし，それぞれのノードの通信セッションにおける状態（ステート）を同じに保つシグナリングプロトコルとしてのソフトステートとハードステートの比較を行っているが，本論文ではディレクトリサービスに限定して議論を行う．通信システムでは，ノード障害や伝送路障害といった長時間にわたる障害が発生すると，通信セッションは切断され，その時点で各ノードにおけるステートそのものが無効となるという特徴がある．このため，障害回復後の各ノードにおけるステートのずれについては考える必要がなく，障害の影響はソフトステートでもハードステートでも同様である．一方，ディレクトリサービスにおける資源情報の生存時間は数カ月から数年と非常に長く，障害が発生しても資源情報は破棄されることなく存在し続けるという特徴がある．このため，ディレクトリサービスでは，障害回復後，資源情報のずれをどのように解消するかがシステムの性能に大きく影響を及ぼす．したがって，ディレクトリサービスのこのような特性から，ディレクトリサービスにおいてソフトステートとハードステートの比較を行う際には，過去の研究結果に基づいて評価するだけでは不十分であり，また逆に，通信システムの特性を考慮すると，通信システムにおいて障害発生の影響を評価することには意味がないと考えられる．このため，本論文ではディレクトリサービスに限定して，障害の発生がシステム性能にどのような影響を与えるかについて議論する．

本論文ではマルコフモデル解析を用いてソフトステートとハードステートの比較を行い，障害発生時のソフトステートとハードステートの振舞いの違いを明

らかにした．ただし本論文で対象とするハードステートは，ハートビートによる障害検出を併用している（詳細は 5 章で述べる）．また，その結果をふまえて，実際にディレクトリサービスでどちらの手法を用いるかを決定する際に考慮するべき具体的な要因を示した．以降の構成は以下のとおりである．まず，2 章でディレクトリサービスのモデルを定義する．3 章で本論文で比較する手法を説明し，4 章では本論文で考える障害を定義する．5 章では解析に用いたモデルを説明し，6 章で解析結果について述べ，最後に，7 章でまとめを行う．

2. ディレクトリサービスのモデル

図 1 に本論文において対象とするディレクトリサービスの例を示す．

ディレクトリサービスは，資源情報を計算機間で共有するためのサービスである．資源情報とは，資源の位置（資源を保持する計算機名），資源の属性（ファイルの所有者，更新日時など），資源の特性（CPU の処理速度など），資源の状態（CPU の現在の負荷など）といった情報のことをいう．資源を保持する計算機は，ディレクトリサーバに対して，資源情報を送り，ディレクトリサーバは受け取った資源情報をテーブルの形で保持する．資源を利用するクライアントは，ディレクトリサーバに対して問い合わせることで資源情報を入手し，資源へのアクセスを行う．ディレクトリサーバは一般に複数存在する．

ディレクトリサービスの性能として，

(1) 問合せによって得られる資源情報の正しさ
(2) 問合せに対して返答が返るまでの応答時間の短さ
(3) 資源情報の交換および問合せに用いられるメッセージコストの小ささ

が考えられる．本論文の目的は，ディレクトリサービスにおけるソフトステートとハードステートの障害に対する振舞いの違いを明らかにすることであるが，

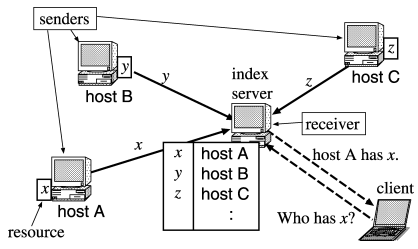


図 1 ディレクトリサービスの例
Fig. 1 An example of directory service.

(2) の応答時間はソフトステートとハードステートの違いとは無関係であり，また (3) のメッセージコストは障害の発生とは無関係である．したがって本論文ではこれらについては評価の対象とはせず，(1) の問合せによって得られる資源情報の正しさにのみ着目する．

ディレクトリサーバが複数存在する場合，問合せによって得られる資源情報の正しさに直接影響するのは以下の 2 つである．

(1) 各ディレクトリサーバが保持する資源情報と，資源を保持する計算機が保持する資源情報が一致している時間の割合
(2) クライアントのディレクトリサーバに対する問合せ手法．たとえば，最も近い 1 台のディレクトリサーバに問い合わせる，あるいは複数のディレクトリサーバに同時に問い合わせ，得られた結果の多数決をとるなど

このうち，(2) の影響についてはソフトステートとハードステートの違いとは独立に議論できるため，本論文では扱わない．したがって，(1) の資源情報が一致している時間の割合，もしくは一致していない時間の割合を性能評価の基準とすることが妥当と考えられる．

以下では，資源を保持する計算機を送信ノード（sender）といい，ディレクトリサーバを受信ノード（receiver）という．また，両者が共有する資源情報をステートという．送信ノードのステートは，実際の資源の状態を表す資源情報であり，受信ノードのステートは，ディレクトリサーバの保持する資源情報である．送信ノードのステートが変化したとき，送信ノードから受信ノードにメッセージを送信して受信ノードのステートを更新する．目的は，送信ノードと受信ノードのステートが一致している時間の全体に占める割合をできるだけ高くすることである．ステートが一致しているとは，送信ノードと受信ノードの両方にステートが存在し，かつそれが一致している状態，または送信ノードと受信ノードの両方にステートが存在しない状態のいずれかである．性能の評価基準として，文献 7) と同様に不一致率（inconsistency ratio）を用いる．不一致率とは，送信ノードと受信ノードのステートが不一致な状態にある時間の割合である．この時間が長くなるほど，ある資源がアクセスされないまま放置されたり，無効なアクセスが発生したりするといった問題が発生しやすくなるため，不一致率は低いほど望ましい．

3. ソフトステートとハードステート

ここでは本論文で対象とするソフトステートとハードステートの具体的な動作について述べる．

3.1 ソフトステート手法（SS）

図 2 (a) にソフトステートの動作例を示す．送信ノードはステートが生成もしくは更新された場合にトリガメッセージを送信する．その後，ステートが存在する限り，リフレッシュタイマ値として設定した間隔でリフレッシュメッセージを送信する．確認応答は用いず，メッセージが消失しても再送は行わない．受信ノードは，タイムアウト値として設定した時間内にトリガメッセージもリフレッシュメッセージも到着しないと，自分が保持しているステートを削除する．いくつかのリフレッシュメッセージが連続して消失すると，送信ノードがステートを削除していないにもかかわらず，受信ノードが誤ってステートを削除してしまう場合がある．これをステートの誤削除と呼ぶ．

3.2 ハードステート手法（HS）

図 2 (b) にハードステートの動作例を示す．送信ノードはステートの生成，更新および削除が行われたときのみ受信ノードにメッセージを送信する．すべてのメッセージのやりとりは高信頼な方法で行われる．すなわち，受信ノードはメッセージを受信すると確認応答を送信する．送信ノードは，メッセージ送信後，再送タイマ値として設定した時間確認応答が到着しないと，メッセージを再送する．リフレッシュメッセージは用いず，タイムアウトによるステート削除は行わない．

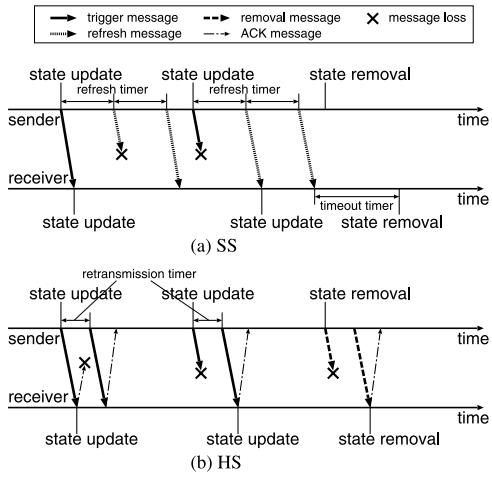


図 2 ソフトステートとハードステートの動作例
Fig. 2 An example of behavior of soft-state and hard-state.

3.3 ソフトステートの拡張手法

文献 7) では，ソフトステートにハードステートの特徴を組み合わせた拡張手法の性能評価が行われ，平常時の性能が向上するという結果が得られている．ここではそれらの手法について簡単に述べる．

明示的削除付き SS（SS+ER） SS に明示的なステート削除メッセージを追加した手法である．送信ノードでステートが削除されると，ステート削除メッセージが送信される．**図 3** (a) に動作例を示す．

高信頼トリガ付き SS（SS+RT） SS におけるトリガメッセージのやりとりを確認応答と再送を用いて高信頼化し，さらにタイムアウト通知を追加した手法である．タイムアウト通知とは，受信ノードがタイムアウトによりステートを削除した場合に，送信ノードにメッセージを送信することであり，これにより送信ノードは新しいトリガメッセージを送信して誤削除から回復できる．図 3 (b) に動作例を示す．

高信頼トリガ/削除付き SS（SS+RTR） この手法は SS+RT と似ているが，SS+RTR はステートの生成と更新だけではなく，ステートの削除の際のメッセージのやりとりも高信頼な方法で行う．図 3 (c) に動作例を示す．

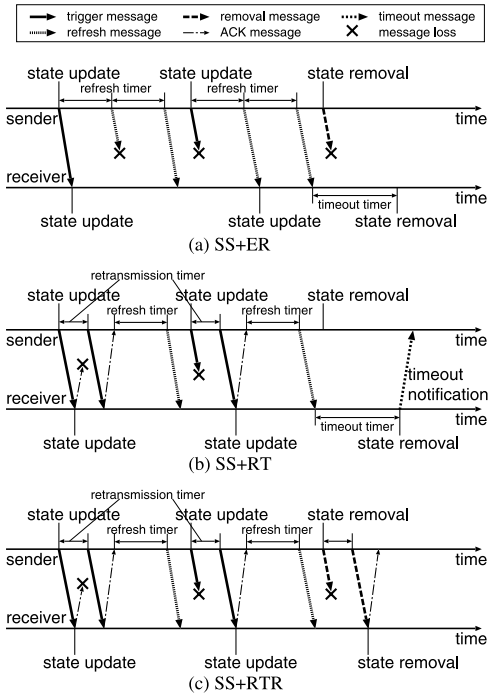


図 3 ソフトステートの拡張手法の動作例
Fig. 3 An example of behavior of extended soft-state.

4. システム障害

本論文では，障害とは計算機や通信機器の故障といった比較的長時間にわたるものをいう．従来の研究でも考慮されているメッセージの消失は一種の障害であると考えられるが，回復のために特別な処理や作業を必要としないため，本論文では，障害ではなく通常の動作の一部であると見なす．また，解析を容易にするために，障害はいずれもフェイルストップ型[9]であるとし，ビザンチン型障害[10]については考えない．このため，プログラムのバグや故意の攻撃，コンピュータウィルスなどによる計算機や通信機器の不正な動作による影響については本論文の評価に含まれない．

4.1 障害の種類

本論文ではシステムの障害として以下のものを考える．

パス障害 通信リンク，ルータの故障などのネットワークの障害により，送信ノードと受信ノード間の通信が不可能な状態である．パス障害の間に送信されたすべてのメッセージは消失するものとする．パス障害が発生した後，以下のいずれかが起こると，送信ノードと受信ノードのステートは不一致な状態となる．

- パス障害から回復する前に，送信ノードにおいてステートが更新，あるいは削除される．
- パス障害から回復する前に，受信ノードにおいてステートが削除される．

ステートの不一致は，以下のいずれかが起こるまで続く．

- パス障害から回復した後，送信ノードから受信ノードにメッセージが送られて受信ノードのステートが更新される．
- 送信ノードのステート，受信ノードのステートがともに削除される．

送信ノード障害 計算機のクラッシュなどにより，送信ノードが利用不可能な状態である．送信ノード障害が発生すると，送信ノードのステートは一時的に消失するが，送信ノード障害からの回復とともにステートも回復する．送信ノード障害から回復すると，送信ノードはただちに受信ノードにトリガメッセージを送信して，ステートを更新するものとする．

送信ノード障害が発生したとき，受信ノードがステートを保持していれば，送信ノードと受信ノードのステートは不一致な状態となる．ステートの不一致は，以下のいずれかが起こるまで続く．

- 送信ノード障害から回復する前に，受信ノードにおいてステートが削除される．
- 送信ノード障害から回復した後，送信ノードから受信ノードにメッセージが送られて受信ノードのステートが更新される．

受信ノード障害 計算機のクラッシュなどにより，受信ノードが利用不可能な状態である．受信ノード障害が発生すると，受信ノードのステートは消失する．

受信ノード障害が発生したとき，送信ノードがステートを保持していれば，送信ノードと受信ノードのステートは不一致な状態となる．また，送信ノードでステートが生成されたとき，受信ノード障害が発生していればステートは不一致であると見なす．ステートの不一致は，以下のいずれかが起こるまで続く．

- 受信ノード障害から回復する前に，送信ノードにおいてステートが削除される．
- 受信ノード障害から回復した後，送信ノードから受信ノードにメッセージが送られて受信ノードに新たにステートが生成される．

5. 性能評価手法

文献 7) では通信システムにおけるメッセージ損失のみを扱っている．本論文では障害発生の影響を考慮するのが主目的であるが，文献 7) の手法を拡張することによってその評価を可能とした．本論文では，送信ノードと受信ノードのステートの変化を文献 7) と同様に通常のマルコフモデルとしてモデル化する．

5.1 仮　　定

モデルを作成するにあたり，障害発生時の各手法の振舞いに関していくつかの仮定をおく．

5.1.1 メッセージの再送

SS+RT，SS+RTR および HS では，送信ノードは，確認応答が到着するまで，トリガメッセージまたは削除メッセージを再送する．しかし実際のシステムでは，ネットワークの輻輳を避けるために，再送回数の上限が設定されており，一定回数再送に失敗するとメッセージの再送は中止される．

本論文では，発生から回復までの時間が長い障害を想定しているため，1 つのメッセージの再送中に障害が発生し，かつ回復することはないと仮定する．すなわち，障害が発生すると必ず，メッセージの再送は中止され，障害から回復しても再送を再開することはない．

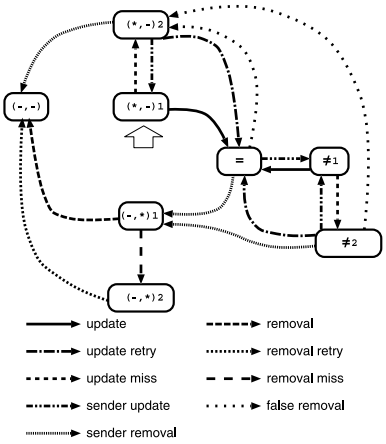


図 4 本モデル（基本的な状態遷移）
Fig. 4 Our model (basic state transitions).

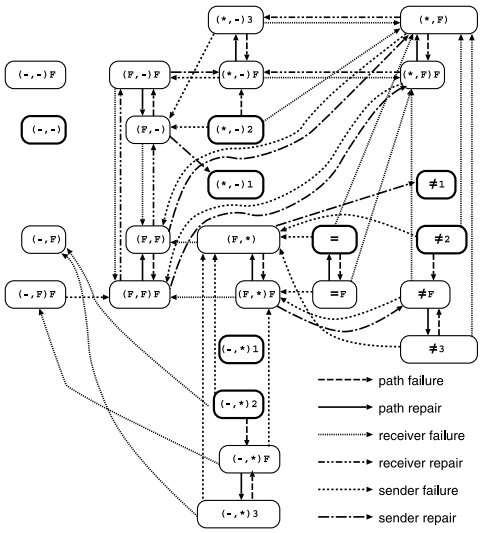


図 5 本モデル（障害に関する状態遷移）
Fig. 5 Our model (state transtions about failures).

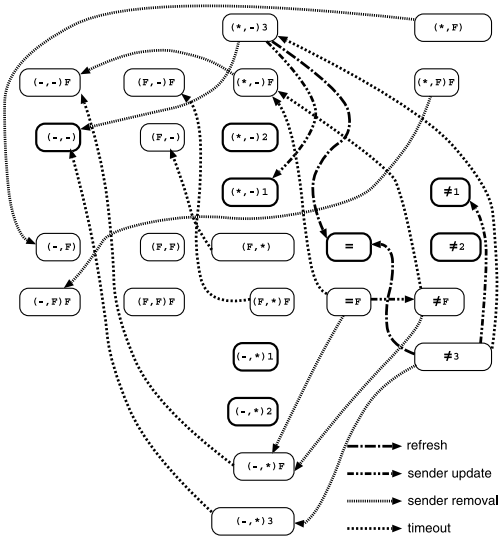


図 6 本モデル（その他の状態遷移）
Fig. 6 Our model (other state transitions).

5.1.2 孤児ステートの削除

送信ノードでステートが消失，あるいは削除された後に，受信ノードで削除されずに残っているステートを孤児ステート（orphaned state）という．

パス障害あるいは送信ノード障害の発生により，受信ノードのステートが孤児ステートになる場合がある．ソフトステートでは孤児ステートはタイムアウトにより自動的に削除されるが，ハードステートでは受信ノードは削除メッセージを受け取るまでステートを削除しないため，障害時に孤児ステートをどのように削除するかが問題となる．

ハードステートでは，ハートビートプロトコル[11]を利用して送信ノードの障害を検出し，孤児ステートを削除するものとする．送信ノードは，受信ノードに定期的にハートビートメッセージを送信し，受信ノードは，一定時間ハートビートメッセージを受け取らないと，何らかの障害が発生したと判断してステートを削除する．

5.2 モデルの概要

本論文で使用するモデル（以下，本モデル）を見やすさのために 3 つの図に分けて示す．**図 4** は本モデルのうち，文献 7) のマルコフモデルであり，基本的な状態遷移のみを表す．**図 5** は本モデルのうち，送受信ノードやパス障害に関する状態遷移に着目し，それらを表している．また，**図 6** は送受信ノードやパスのいずれかに障害が発生している間のステートの更新やタイムアウトにともなう遷移，および障害回復後の状態からの遷移などを考慮したものであり，文献 7) には含まれていないその他の状態遷移を表している．詳細は 5.3 節と 5.4 節で述べる．

5.3 マルコフ状態と状態遷移

本モデルの各状態の意味は以下のとおりである．マルコフ状態 “=” は送信ノードと受信ノードにステートが存在し，かつ一致している状態を表し，“≠” は送信ノードと受信ノードにステートが存在するが，一致していない状態を表す．その他のマルコフ状態は (x_s, x_r) の形をしている．x_s と x_r はそれぞれ送信ノードと受信ノードのステートを表しており，$*$ はステートが存在すること，$-$ はステートが存在しないこと，F は障害が発生していることを表す．たとえば，$(F, *)$ は送

表 1 マルコフモデルの状態遷移を引き起こすイベント
Table 1 Events that trigger state transition of Markov model.

状態遷移	イベント
update	ステート更新
update retry	ステート更新（失敗後）
update miss	ステート更新失敗
sender update	送信ノードでのステート更新
sender removal	送信ノードでのステート削除
removal	ステート削除
removal retry	ステート削除（失敗後）
removal miss	ステート削除失敗
false removal	誤削除
path failure	パス障害発生
path repair	パス障害回復
receiver failure	受信ノード障害発生
receiver repair	受信ノード障害回復
sender failure	送信ノード障害発生
sender repair	送信ノード障害回復
refresh	リフレッシュによるステート更新
timeout	タイムアウトによるステート削除

信ノード障害が発生しており，受信ノードにステートが存在する状態を表す．また，添字 F はパス障害の発生している状態を表す．

本モデルにおける状態遷移を引き起こすイベントを**表 1** に示す．表 1 では，特に表記がない限りステートの更新，削除はすべて受信ノードでのものである．いくつかの状態遷移については，手法によって状態遷移を引き起こすイベントが異なる．update retry は，SS と SS+ER ではリフレッシュメッセージによるステート更新，HS ではトリガメッセージの再送によるステート更新によって起こり，SS+RT と SS+RTR ではその両方によって起こる．removal は，SS と SS+RT ではタイムアウトによるステート削除，それ以外の手法では削除メッセージによるステート削除によって起こる．removal retry は，SS+ER ではタイムアウトによるステート削除，HS では削除メッセージの再送によるステート削除によって起こり，SS+RTR ではその両方によって起こる．timeout は，HS 以外の手法では，リフレッシュメッセージが届かないことによるタイムアウトによって起こり，HS ではハートビートメッセージが届かないことによるタイムアウトによって起こる．

5.4 マルコフ状態の追加

マルコフ状態 $(*,-)$，$(-,*)$，“$\neq$” は，添字によって区別される複数のマルコフ状態に分割される．図 4 のモデルでは添字は 1 と 2 の 2 種類である．添字 1 はステートの更新もしくは削除の後，メッセージが送信される前の状態，添字 2 は送信されたメッセージが消失した後の状態を表す．これらの状態では，ハードステートにおいて送信ノードは何らかのメッセージを送信，もしくは再送する．

ここで，ノード障害，またはパス障害が発生すると，5.1.1 項で述べたように，ハードステートにおけるメッセージの再送は中断され，障害から回復した後も再送が行われることはない．すなわち，障害からの回復後，ハードステートにおいて，送信ノードがメッセージを送信しない状態になる．これを表すため，新たに添字 3 を持つマルコフ状態を追加した．受信ノード障害あるいはパス障害から回復すると添字 3 の状態に遷移する．添字 3 の状態ではメッセージの送信は行われないため，ハードステートにおいて状態の遷移をもたらすのはステートの更新，削除およびタイムアウトのみとなる．

5.5 初期状態と吸収状態

マルコフモデルは，新しいステートが生成されてから，完全に削除されるまでの送信ノードと受信ノードにおけるステートの変化を表している．したがって送信ノードでステートが生成された時点が初期状態，送信ノードのステートが削除された後，受信ノードのステートも削除され不一致が解消された状態が吸収状態となる．

文献 7) のマルコフモデル（図 4）では，初期状態は $(*,-)_1$，吸収状態は $(-,-)$ の各 1 つしか存在しないが，障害を考慮すると，複数の初期状態と吸収状態が存在する．送信ノードにステートが生成された時点で，受信ノードおよびパスに障害が発生している場合がある．したがって，初期状態は，$(*,-)$，$(*,-)_F$，$(*,F)$，$(*,F)_F$ の 4 つになる．また，送信ノードでステートが削除された後，受信ノード障害によってステートが消失する場合がある．さらに送信ノードでステートが削除された後，受信ノードのステートが削除された時点でパス障害が発生している場合がある．したがって，吸収状態も，$(-,-)$，$(-,-)_F$，$(-,F)$，$(-,F)_F$ の 4 つとなる．

5.6 状態遷移レート

本モデルの状態遷移レートに関わるパラメータを**表 2** に示す．表 2 において，HB はハートビートの略である．障害を考慮するにあたり，それぞれの障害について，障害の発生間隔の平均である MTBF（Mean Time Between Failure）と障害が発生してから回復するまでの時間の平均である MTTR（Mean Time To Repair）をパラメータとして導入している．また，送信ノードにおいて，ステートが更新または削除されてから，トリガメッセージあるいは削除メッセージを送信するまでの処理時間，および，受信ノードにおいて，

表 3 マルコフモデルの状態遷移レート
Table 3 State transition rates of Markov model.

状態遷移	SS	SS+ER	SS+RT	SS+RTR	HS
update	$(1-p_l)/D$	$(1-p_l)/D$	$(1-p_l)/D$	$(1-p_l)/D$	$(1-p_l)/D$
update miss	p_l/D	p_l/D	p_l/D	p_l/D	p_l/D
update retry	$(1-p_l)/T$	$(1-p_l)/T$	$(1/R+1/T)\cdot(1-p_l)$	$(1/R+1/T)\cdot(1-p_l)$	$(1-p_l)/R$
refresh	$(1-p_l)/T$	$(1-p_l)/T$	$(1-p_l)/T$	$(1-p_l)/T$	–
removal	$1/X$	$(1-p_l)/D$	$1/X$	$(1-p_l)/D$	$(1-p_l)/D$
removal miss	–	p_l/D	–	p_l/D	p_l/D
removal retry	–	$1/X$	–	$1/X+(1-p_l)/R$	$(1-p_l)/R$
timeout	$1/X$	$1/X$	$1/X$	$1/X$	$1/X_b$
false removal	$p_l^{\lfloor X/T\rfloor}/X$	$p_l^{\lfloor X/T\rfloor}/X$	$p_l^{\lfloor X/T\rfloor}/X$	$p_l^{\lfloor X/T\rfloor}/X$	$p_l^{\lfloor X_b/T_b\rfloor}/X_b$

表 2 パラメータ一覧
Table 2 Parameter list.

種類	表記	デフォルト値
ステートの更新間隔	$1/\lambda_u$	10^5(s)（約 1.2 日）
ステートの生存時間	$1/\lambda_d$	10^7(s)（約 116 日）
通信遅延	D	0.03 (s)
メッセージロス率	p_l	0.02
リフレッシュタイマ値	T	86400 (s)（1 日）
タイムアウト値	X	172800 (s)（2 日）
再送タイマ値	R	0.12 (s)
HB タイマ値	T_b	2 (s)
HB タイムアウト値	X_b	30 (s)
パスの MTBF	$1/\lambda_{lf}$	10^8(s)（約 3.2 年）
パスの MTTR	$1/\lambda_{lr}$	10^4(s)（約 2.7 時間）
受信ノードの MTBF	$1/\lambda_{rf}$	10^8(s)（約 3.2 年）
受信ノードの MTTR	$1/\lambda_{rr}$	10^4(s)（約 2.7 時間）
送信ノードの MTBF	$1/\lambda_{sf}$	10^8(s)（約 3.2 年）
送信ノードの MTTR	$1/\lambda_{sr}$	10^4(s)（約 2.7 時間）

トリガメッセージ，リフレッシュメッセージ，削除メッセージのいずれかを受信してから，ステートを更新または削除するまでの処理時間はいずれも 0 であると仮定する．

それぞれの手法におけるマルコフモデルの状態遷移レートを**表 3** に示す．ただし，表 2 のパラメータから自明なものは省略している．表中の “–” は対応する状態遷移が存在しない（状態遷移レートが 0 である）ことを示す．

5.7 不一致率の計算

不一致率は，吸収状態と初期状態を結合した再帰的マルコフモデルを作成し，ステートが不一致であるマルコフ状態の定常確率の和を計算することで求められる．このとき，吸収状態 $(-,-)$，$(-,-)_F$，$(-,F)$，$(-,F)_F$ がそれぞれ初期状態 $(*,-)$，$(*,-)_F$，$(*,F)$，$(*,F)_F$ と結合される．また，この再帰的マルコフモデルにおいてステートの一致している状態は，4.1 節の定義より，“=”，$(F,-)$，$(F,-)_F$，(F,F)，$(F,F)_F$ の 5 つである．したがって，これ以外のすべてのマルコフ状態の定常確率の和が不一致率となる．

ステートが不一致であるとき，受信ノードの状態は以下のいずれかである．

(1) ステートが存在するが，送信ノードのステートと一致していない．
(2) ステートが存在しない，または受信ノード障害が発生している．

文献 7) での評価では (1) と (2) を特に区別していないが，ディレクトリサービスにおいては，(1) と (2) の違いがシステムの性能に影響を及ぼす場合がある．例として，ディレクトリサービスにおいてファイルの複製の位置情報を管理している状況を考える．ファイルの複製を保持する計算機を問い合わせるクエリに対する回答として，(1) の場合は，実際には複製の存在しない計算機の情報を返すため，無駄なアクセスを引き起こす．一方，(2) の場合は，複製を持つ他の計算機へアクセスすることとなり，無駄なアクセスを引き起こすことはない．

そこで本論文では，不一致率のうち，(1) の状態にある時間の割合を不正率，(2) の状態にある時間の割合を喪失率として区別し，それぞれの比較も行う．

6. 性能評価

6.1 パラメータ設定

ディレクトリサービスにおける性能を評価するにあたっては，ディレクトリサービスの特性を考慮したパラメータを設定する必要がある．

デフォルトのパラメータ値を表 2 に示す．この値はデータグリッドにおけるファイル情報を管理するディレクトリサービスを想定して設定している．高エネルギー物理や気候モデリングの分野におけるディレクトリサービスへの要求として，5 千万個のファイルの情報に対して，毎秒 200 回の更新が行えることが必要と考えられている[13]．もしすべてのファイルの情報が均等に更新されると仮定すると，平均更新間隔は $2.5*10^5$（秒）となる．これに基づき，ステートの更新間隔を

表 5 障害の発生頻度が高いとき（MTBF = 10^5）の不一致率
Table 5 Inconsistency ratio with frequent failure (MTBF = 10^5).

手法	デフォルト	(a) パス障害	(b) 送信ノード障害	(c) 受信ノード障害
SS	0.027	0.096 (0.069)	0.104 (0.077)	0.392 (0.365)
SS+RT	0.018	0.087 (0.069)	0.092 (0.074)	0.387 (0.369)
SS+ER	0.011	0.083 (0.072)	0.098 (0.087)	0.386 (0.375)
SS+RTR	0.001	0.074 (0.073)	0.086 (0.085)	0.380 (0.379)
HS	0.002	0.543 (0.541)	0.001 (−0.001)	0.543 (0.541)

表 4 サーバ計算機および通信機器の MTBF の例
Table 4 MTBF of servers and network devices.

メーカおよび機種名	MTBF
Sun Fire V40z サーバ	約 4 年
HP ProLiant ML570 G4 サーバ	約 14 年
Huawai-3com S6502 ルータ	約 26 年
Cisco 12410 ルータ	約 27 年
Foundry FastIron SuperX スイッチ	約 31 年
Cisco Catalyst 2960-24TT スイッチ	約 32 年

10^5 秒（約 1.2 日）に設定した．ステートの生存時間については，データグリッドのファイルは一度生成されると削除されることはほとんどないため，10^8 秒を超えるような非常に長い生存時間も考えられる．しかし先行研究である文献 7) の結果より，ステートの生存時間が長くなるほど，SS とその拡張手法の不一致率の差が小さくなることが分かっている．そこで，障害発生時の SS とその拡張手法の振舞いの違いを明らかにするため，ファイル情報を管理するディレクトリサービスとしては比較的短い生存時間である 10^7 秒（約 116 日）に設定している．通信遅延，メッセージロス率，再送タイマ値については文献 7) と同じ値に設定している．文献 7) のパラメータ設定はインターネット上の P2P アプリケーションである Kazaa を想定したものであり，インターネット環境の特性を反映したものとして妥当と考えられる．また，リフレッシュタイマ値は，グリッドの代表的ミドルウェアである Globus Toolkit に含まれるディレクトリサービスである RLS（Replica Location Service）[12),13)] におけるデフォルト値に設定し，タイムアウト値は，文献 7) において SS が最も高い性能を示した値である，リフレッシュタイマ値の 2 倍に設定している．ハートビートタイマ値およびハートビートタイムアウト値については High Availability Linux Project の heartbeat パッケージ[14)] におけるデフォルト値に設定している．MTBF については実際の機器の MTBF に基づいて設定している．**表 4** に示すように，現在のサーバ計算機は数年から十数年の MTBF を持つのが一般的である．これをふまえて，送信ノードおよび受信ノードの MTBF を 10^8 秒（約 3.2 年）に設定している．また，表 4 に示すように，現在の通信機器は 20 年から 30 年といった MTBF を持つのが一般的である．しかし，1 つのパス上には複数のネットワーク機器が存在するために，パスの MTBF は個々の通信機器の MTBF よりも短くなると考えられる．これを考慮して，パスの MTBF を 10^8 秒（約 3.2 年）に設定している．また，送信ノード，受信ノード，およびパスの MTTR は，システム管理者が機器の障害を検知して，機器の交換などの適切な処置を行うまでの時間を想定して 10^4 秒（約 2.7 時間）に設定している．

6.2 解析結果

パス障害，送信ノード障害，受信ノード障害のそれぞれについて，平均発生間隔を変化させて，それぞれの手法の不一致率の変化を調べた．また，SS と HS については不正率と喪失率の変化も調べた．障害発生の頻度が高くなっても低い不一致率を維持できる手法ほど，障害に対して優れているといえる．

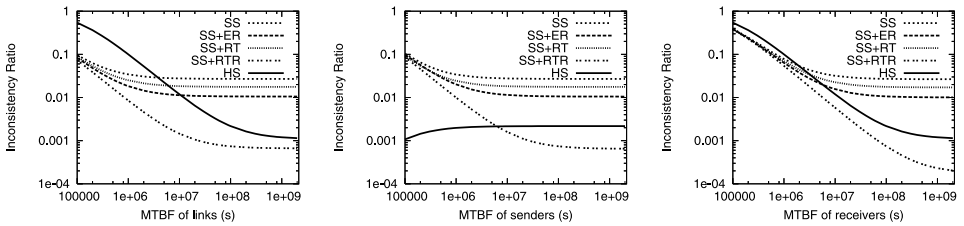
以降では，SS，SS+ER，SS+RT，SS+RTR の 4 つの手法をまとめて SS*と略記する．

6.2.1 障害の不一致率への影響

パス障害，送信ノード障害，受信ノード障害それぞれの平均発生間隔を変化させた場合の不一致率を**図 7** に示す．また，パス障害，送信ノード障害，受信ノード障害それぞれの MTBF が非常に短いとき（10^5 秒）の不一致率，およびデフォルトの状態の不一致率を**表 5** に示す．表 5 で，() 内はデフォルトの状態の不一致率との差を表す．

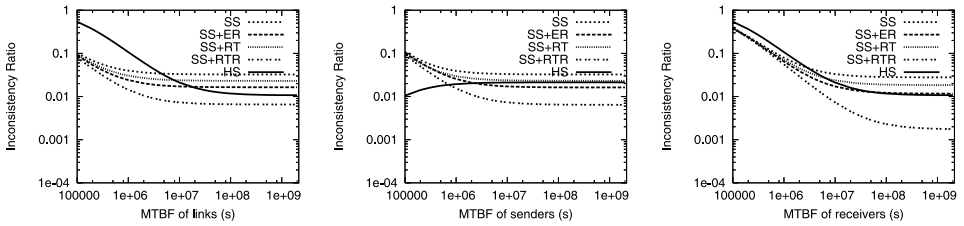
(a) パス障害の影響 図 7 (a) および表 5 から，SS+RTR が，SS，SS+ER，SS+RT と比べてパス障害の影響を大きく受けているが，パス障害の発生頻度が高いときの，SS*の不一致率はほぼ同じであることが分かる．また，HS が SS*と比べてパス障害の発生頻度の影響を非常に大きく受けていることが分かる．これより，パス障害に対してはソフトステートの方が有利である．

(b) 送信ノード障害の影響 図 7 (b) および表 5 から，SS*はパス障害の場合と同程度に送信ノード障害の影響を受けるが，HS は送信ノード障害の影響



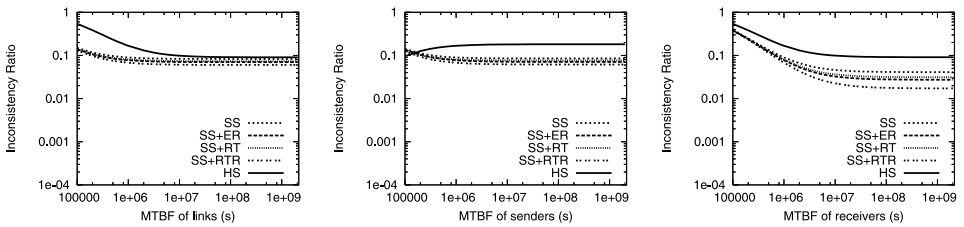
(a) パス障害 (b) 送信ノード障害 (c) 受信ノード障害

図 7 不一致率（MTBF のデフォルト値 = 10^8）
Fig. 7 Inconsistency ratio (default MTBF = 10^8).



(a) パス障害 (b) 送信ノード障害 (c) 受信ノード障害

図 8 不一致率（MTBF のデフォルト値 = 10^7）
Fig. 8 Inconsistency ratio (default MTBF = 10^7).



(a) パス障害 (b) 送信ノード障害 (c) 受信ノード障害

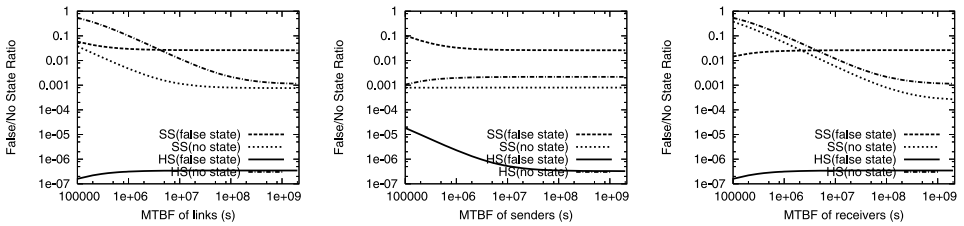
図 9 不一致率（MTBF のデフォルト値 = 10^6）
Fig. 9 Inconsistency ratio (default MTBF = 10^6).

をほとんど受けないことが分かる．また，HS の不一致率が送信ノード障害の発生頻度が高くなるほど逆に低下している．これは，パス障害または受信ノード障害が発生すると受信ノードのステートが削除されてステートの不一致が生じるが，この状態でさらに送信ノード障害が発生するとステートの不一致が解消されてしまうためである．これより，送信ノード障害に対してはハードステートの方が有利である．

(c) 受信ノード障害の影響 図 7 (c) および表 5 から，すべての手法が受信ノード障害の影響を大きく受けているが，なかでも HS が SS*と比べて受信ノード障害の影響をより受けやすいことが分かる．これより，受信ノード障害に対しては，ソフトステートの方が有利である．

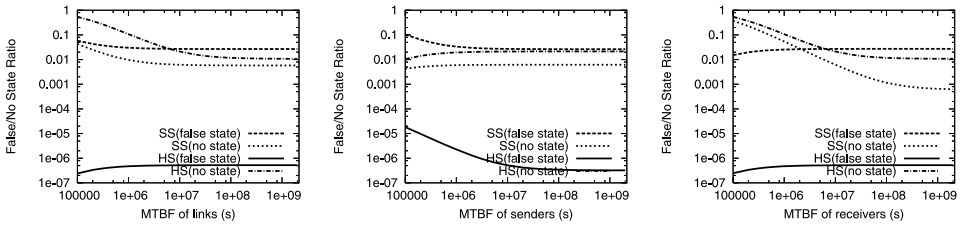
6.2.2 障害に対する拡張手法の効果

文献 7) では，平常時における SS，SS+ER，SS+RT，SS+RTR，HS の不一致率を比較しており，パラメータによって異なる場合があるものの，一般的には SS+RTR，SS+ER，SS+RT，SS の順に不一致率が低く，HS は SS+RTR とほぼ同じ不一致率という結果が得られている．本論文でのパラメータ設定においても，表 5 より，デフォルトの状態において，SS と SS+RT の不一致率には約 1.5 倍，SS と SS+ER の不一致率には約 2.5 倍，SS と SS+RTR の不一致率には約 27 倍の差が見られ，拡張手法によって SS の不一致率が大きく改善されることが確認できる．しかしながら，障害の種類によらず，発生頻度が高くなるにつれて，SS+ER，SS+RT，SS+RTR の不一致率と SS の不一致率の差は小さくなり，MTBF = 10^5 のときにはたかだか 20%以内の差にとどまった．このこと



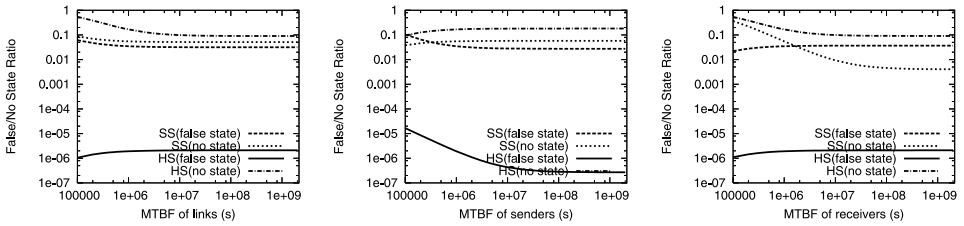
(a) パス障害 (b) 送信ノード障害 (c) 受信ノード障害

図 10 不正率と喪失率（MTBF のデフォルト値 = 10^8）
Fig. 10 False state ratio and no state ratio (default MTBF = 10^8).



(a) パス障害 (b) 送信ノード障害 (c) 受信ノード障害

図 11 不正率と喪失率（MTBF のデフォルト値 = 10^7）
Fig. 11 False state ratio and no state ratio (default MTBF = 10^7).



(a) パス障害 (b) 送信ノード障害 (c) 受信ノード障害

図 12 不正率と喪失率（MTBF のデフォルト値 = 10^6）
Fig. 12 False state ratio and no state ratio (default MTBF = 10^6).

表 6 障害の発生頻度が高いとき（MTBF = 10^5）の不正率と喪失率
Table 6 False state ratio and no state ratio with frequent failure (MTBF = 10^5).

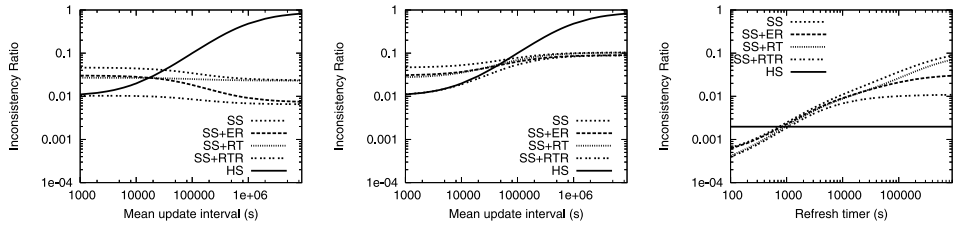
手法	デフォルト	(a) パス障害	(b) 送信ノード障害	(c) 受信ノード障害
SS（不正率）	0.026	0.058 (0.032)	0.103 (0.077)	0.015 (−0.011)
SS（喪失率）	0.001	0.038 (0.037)	0.001 (0.000)	0.378 (0.377)
HS（不正率）	$3.5 * 10^{-7}$	$1.6 * 10^{-7}$	$1.8 * 10^{-5}$	$1.6 * 10^{-7}$
HS（喪失率）	0.002	0.543 (0.541)	0.001 (−0.001)	0.543 (0.541)

から，ソフトステートの拡張手法は，平常時には有効であるが，障害によって発生したステートのずれの解消にはほとんど寄与しないと考えられる．

6.2.3 障害の不正率，喪失率への影響

パス障害，送信ノード障害，受信ノード障害それぞれの平均発生間隔を変化させた場合の HS と SS の不正率と喪失率を**図 10** に示す．また，パス障害，送信ノード障害，受信ノード障害それぞれの MTBF が非常に短いとき（10^5 秒），およびデフォルトの状態の不正率と喪失率を**表 6** に示す．表 6 で，() 内はデフォルトの状態の不一致率との差を表す．

図 10 (a) および表 6 から，パス障害は主に喪失率に影響しており，特に HS ではその影響が非常に大きいことが分かる．その半面，HS の不正率は SS と比べて非常に低いという特徴が見られる．また図 10 (b) および表 6 から，送信ノード障害は主に不正率に大きな影響を与えることが分かる．しかし HS はもともと不正率が非常に低いため，送信ノード障害の頻度が高



(a) ステートの更新間隔（パスの MTBF = 10^6）
(b) ステートの更新間隔（受信ノードの MTBF = 10^6）
(c) リフレッシュタイマ値（送信ノードの MTBF = 10^6）

図 13 システムパラメータ値の不一致率に対する影響
Fig. 13 Effect of system parameters on inconsistency ratio.

表 7 システムパラメータ値を変化させたときの不一致率
Table 7 Inconsistency ratio with various system parameters.

手法	(a) ステートの更新間隔（パスの MTBF = 10^6）			(b) ステートの更新間隔（受信ノードの MTBF = 10^6）			(c) リフレッシュタイマ値（送信ノードの MTBF = 10^6）		
	1000	10^6	差	1000	10^6	差	100	10^5	差
SS	0.046	0.025	−0.021	0.047	0.100	0.053	0.001	0.037	0.036
SS+RT	0.027	0.023	−0.003	0.028	0.098	0.070	0.000	0.026	0.026
SS+ER	0.030	0.009	−0.021	0.031	0.086	0.055	0.001	0.021	0.020
SS+RTR	0.010	0.007	−0.003	0.011	0.084	0.073	0.000	0.010	0.010
HS	0.011	0.482	0.471	0.011	0.482	0.471	0.002	0.002	0.000

い場合でも低い不正率を示す．さらに図 10 (c) および表 6 から，受信ノード障害は主に喪失率に大きな影響を与えていることが分かる．HS の喪失率の上昇の仕方は SS とほぼ同様であるが，HS は SS よりも高い喪失率を示す．

これらのことから，喪失率を含めた不一致率では概してソフトステートの方が有利であるが，不正率の低さが重要な用途においてはハードステートの方が有利であるといえる．

6.2.4 パラメータ値の影響

障害に対する振舞いに影響するパラメータを調べるため，それぞれのパラメータ値を変化させて解析を行った結果，ステートの更新間隔とリフレッシュタイマ値が大きな影響を与えていることが分かった．

HS の障害に対する性能はステートの更新間隔が長いほど低下する．**図 13** (a) および図 13 (b) はそれぞれパス障害および受信ノード障害の発生頻度が高い状態で，ステートの更新間隔を変化させた場合の結果である．また**表 7** (a) および表 7 (b) はそれぞれの場合における不一致率を数値で示している．SS*はステートの更新間隔の影響をそれほど受けていないのに対し，HS はステートの更新間隔が長くなるにつれて不一致率が大きく上昇している．したがって，HS がパス障害および受信ノード障害の影響を SS* よりも大きく受けているのは，ステートの更新間隔が長いことが原因であると考えられる．

一方，SS*の障害に対する性能はリフレッシュタイマ値が大きいほど低下する．図 13 (c) は送信ノード障害の発生頻度が高い状態で，リフレッシュタイマ値を変化させた場合の結果である．また表 7 (c) は不一致率を数値で示している．ただし，タイムアウトタイマ値はつねにリフレッシュタイマ値の 2 倍に設定している．また，HS はリフレッシュメッセージを用いないが比較のために並べて示している．リフレッシュタイマ値が大きくなるにつれて SS*の不一致率が大きく上昇している．したがって，SS*が送信ノード障害の影響を HS より大きく受けているのは，リフレッシュタイマ値が大きいことが原因であると考えられる．

ソフトステートにおいて，リフレッシュタイマ値を小さくすることは，メッセージ数の増大を意味する．このため，通信リンクの回線容量に対するメッセージの占める割合が小さい場合には，ソフトステートを用い，リフレッシュタイマ値を十分小さく設定することにより，ハードステートより高い性能が得られる．逆に通信リンクの回線容量に余裕がなく，リフレッシュタイマ値を大きくせざるをえない場合は，ハードステートが有利となる．ハードステートでも，ハートビートメッセージの送信間隔によってメッセージ数は変化するが，ハートビートメッセージのサイズは小さいため，ネットワークを流れるデータ量はソフトステートより小さい．

6.2.5 デフォルトの障害発生頻度が高い場合

ある障害の MTBF を変化させたときの不一致率の挙動は，MTBF のデフォルト値によって変化すると考えられる．そこで，MTBF のデフォルト値を変化させて解析を行った．

MTBF のデフォルト値を 10^7 秒，10^6 秒としたときの不一致率をそれぞれ**図 8** と**図 9** に示す．MTBF のデフォルト値が小さくなるにつれて，不一致率が全体的に高くなっているが，特定の障害の発生頻度が高いとき（MTBF が 10^5 秒付近）の不一致率はほとんど同じである☆．このことから，それぞれの障害が不一致率に及ぼす影響は互いに独立しており，ある障害の発生頻度が高いときに別の障害の影響が特に大きくなるといった関係はないことが分かる．

また，文献 7) では，ソフトステートおよびその拡張手法とハードステートの間には大きな差異はないという結論を得ているが，図 9 より，障害の発生頻度が高くなるについて，SS とその拡張手法の不一致率の差が小さくなり，ソフトステートとハードステートに二分される傾向があることが分かる．このことは，ソフトステートとハードステートの障害発生時の振舞いに本質的な差異があることを示しているといえる．

MTBF のデフォルト値を 10^7 秒，10^6 秒としたときの SS と HS の不正率と喪失率をそれぞれ**図 11** と**図 12** に示す．不正率より喪失率の方が MTBF のデフォルト値の変化による影響が大きい．これは喪失率の方が障害の影響をより強く受けることを表している．

6.3 考　　察

各手法の障害発生時の不一致率にどのパラメータが影響するかということは論理的に導くことができるが，それぞれの要因が及ぼす影響の大きさについてはただちには分からない．そこで解析により，現実的なパラメータ設定の下でそれぞれの要因が及ぼす影響の大きさを求めた結果，どの要因が支配的であるかを特定することができた．これにより，ある状況においてソフトステートとハードステートのどちらを用いるべきかを決定する際に考慮すべき要因が明らかになり，より適切な手法を選択することが可能となる．ここでは，グリッドコンピューティングを例として，ディレクトリサービスにおいて，ハードステートまたはソフトステートのどちらを用いるかを決定する際に考慮すべき要因について考察する．

☆ 例外として，送信ノード障害の発生頻度を変化させたときの HS の不一致率が MTBF のデフォルト値が小さいほど高くなっているが，これは HS が送信ノード障害の影響をほとんど受けないために，他の障害の影響のみが反映された結果である．

まず，グリッドを構成する計算機とネットワークにおける障害の発生頻度を考慮する必要がある．

6.2.1 項 (a) の結果より，ネットワークにおけるパス障害は，ハードステートの不一致率に特に大きな影響を与える．したがってネットワークの信頼性が低く，パス障害が 1 カ月に 1 度以上発生するような場合には，ソフトステートが適している．また，近年無線ネットワークを用いたグリッドに関する研究が行われている[15]．無線ネットワークでは電波状況によって計算機間の通信ができなくなる状況が頻繁に起こるため，数分おきにパス障害が発生することがある[16]．このような状況ではハードステートは現実的ではなく，必然的にソフトステートを用いることになる．

6.2.1 項 (b) の結果より，資源を保持する計算機の障害は，ソフトステートの不一致率に影響を与える一方，ハードステートの不一致率にはほとんど影響しない．グリッドでは，ある計算機に障害が発生してもグリッド全体としては動作するため，各計算機にそれほど高い信頼性は要求されない[17]．このため，グリッドに信頼性の低い計算機が含まれている場合がある．もし，資源を保持する計算機の信頼性が非常に低く，障害が数日に 1 度以上発生するといった場合には，ハードステートが適している．

6.2.1 項 (c) の結果より，ディレクトリサーバの障害は，ソフトステート，ハードステートの両方の不一致率に大きな影響を与える．たとえば，ディレクトリサーバの障害が 1 カ月に 1 度以上発生するような場合，ソフトステートの方が若干有利ではあるが，どちらの手法でも不一致率は大きく上昇してしまう．ディレクトリサーバの信頼性が低い場合，複数のディレクトリサーバを設置し，クライアントが，同じ資源情報を保持する複数のディレクトリサーバに問い合わせて多数決をとる手法がある[18]．これにより，いくつかのディレクトリサーバが資源情報を返さなかったり，誤った資源情報を返したりしたとしても，クライアントは正しい資源情報を選択できるため，ディレクトリサービスの信頼性が向上する．このような状況においてどちらの手法が適しているかは，ディレクトリサーバの台数に依存する．ディレクトリサーバの数が 2，3 台といった少数であれば，あるディレクトリサーバが誤った資源情報を返したときに，それが最終的な結果として採用されてしまう確率が高い．この場合は，それぞれのディレクトリサーバの不一致率は高くても，不正率の低い手法が望ましく，6.2.3 項の結果より，ハードステートが適しているといえる．一方，ディレクトリサーバの数が 4 台以上であれば，ある 1 台のサーバ

が誤った資源情報を返すことによる影響はほぼ無視できるため，喪失率も含めた不一致率がより低いソフトステートが適している．

さらに，対象となるグリッドの特性の違いも考慮する必要がある．

たとえば，データグリッド[19]におけるディレクトリサービスの場合，資源情報はファイルのメタデータやファイルの複製の場所などである．資源情報の更新間隔に着目すると，メタデータの更新やファイルの複製の移動が数分おきに発生するといったことはまれであり，ファイルによっては長期間資源情報が更新されない場合もある．よって，6.2.4 項の結果よりソフトステートが適している．また，不正率が高いと，無効なアクセスが発生し，資源へのアクセスに時間がかかるが，ギガバイト単位の容量のファイルを扱うデータグリッドでは，ファイルの転送に時間がかかるために，無効なアクセスによる遅れは相対的に大きな問題にならない．以上のことから，データグリッドにおいては一般にソフトステートが適している．

一方，計算グリッド[20]におけるディレクトリサービスでは，資源情報は各計算ノードの処理速度や負荷などである．資源情報の更新間隔に着目すると，計算ノードの処理速度は数分あるいは数時間の単位で大きく変動することがある[21]ので，資源情報は頻繁に更新する必要がある．また，不正率が高いと，スケジューラは計算ノードの負荷を正しく把握できないため，特定の計算ノードの負荷が大きく上昇してしまう場合がある．これはジョブの実行時間に大きな影響を与える．このため，ソフトステートとハードステートの不一致率に大きな差がなければ，より不正率の低い方が望ましく，6.2.3 項の結果より，ハードステートが有利である．以上のことから，計算グリッドにおいては一般にハードステートが適している．

7. おわりに

本論文ではディレクトリサービスにおいて，障害の発生する状況における，ソフトステートおよびその拡張手法と，ハートビートによる障害検出を用いたハードステートの性能比較を行った．その結果，HS は，パス障害の発生頻度が高い場合には SS の約 5 倍，受信ノード障害の発生頻度が高い場合には SS の約 1.4 倍の不一致率を示し，これらの障害に対しては，一般に考えられているように，ハードステートよりもソフトステートが有利であることが確認された．またその一方で，送信ノード障害の発生頻度が高い場合には，HS が SS の約 1/100 の不一致率を示すことがあり，つねにソフトステートの方が障害に対して高い性能を示すとは限らないということも明らかになった．ただしこの結果は，ディレクトリサービスを想定したパラメータの下での解析により得られたものであり，一般的に成り立つものではない．たとえば，リフレッシュタイマ値を数秒といった非常に小さい値に設定すると，ソフトステートはつねにハードステートよりも高い性能を示すことになり，逆にステートの更新間隔が数秒から数十秒といった非常に短い時間の場合は，ハードステートがつねにソフトステートよりも高い性能を示すこともある．また，文献 7) では，ソフトステートの拡張手法である SS+ER，SS+RT，SS+RTR が，平常時の不一致率の改善に有効であるとの結果が得られているが，障害発生頻度が高い場合には，これらは SS に近い不一致率を示すことが分かった．このことから，拡張手法は障害発生時の性能の改善にはほとんど寄与しないと考えられる．また，ディレクトリサービスにおいて，ソフトステートとハードステートのどちらを用いるべきかは，計算機や通信パスの信頼性，ディレクトリサーバの数，通信リンクの回線容量といった環境の要因のほかに，資源情報の更新頻度，誤った資源情報の及ぼす影響の大きさなどといったアプリケーションの特性を考慮して決定するべきであることを示した．

今後の課題として，ステートの不正率と喪失率に着目した，アプリケーションレベルでのソフトステートとハードステートの評価があげられる．5.7 節で述べたように，不一致率が同じであっても，不正率および喪失率の違いがシステムの性能に影響を及ぼすことがある．これは，ディレクトリサービスで提供される情報をどのように利用するかというアプリケーションの特性のほか，ディレクトリサーバが複数存在する場合には，クライアントがどのように問合せを行い，得られた答えをどのように選択するかといった方針にも依存する．本論文ではグリッドコンピューティングを例として，いくつかの場合について考察を行ったが，今後より詳細な評価を行いたいと考えている．

また別の課題として，スケーラビリティに着目したソフトステートとハードステートの評価があげられる．直観的には，タイムアウトタイマ値やリフレッシュタイマ値の調整により，規模が大きくなっても問題なく動作するソフトステートの方がスケーラビリティが高いと考えられるが，ソフトステートのスケーラビリティは性能とのトレードオフであり，実用上十分な性能が得られるかどうかに着目した場合には両者の優劣は明らかではない．こういった点を考慮し，ハードス

テートとソフトステートのスケーラビリティの差を明らかにしたいと考えている．

参 考 文 献

1) Bustamante, F., Widener, P. and Schwan, K.: Scalable Directory Services using Proactivity, *Proc. Supercomputing 2002* (2002).
2) Chandy, K.M., Rifkin, A. and Schooler, E.: Using Announce-Listen with Global Events to Develop Distributed Control Systems, *Proc. ACM Workshop on High Performance Java Network Computing*, Palo Alto, CA (1998).
3) Hoschek, W.: A Database for Dynamic Distributed Content and its Application for Service and Resource Discovery, *Proc. Int'l. IEEE Symp. on Parallel and Distributed Computing* (*ISPDC 2002*), Iasi, Romania (2002).
4) Ripeanu, M. and Foster, I.: A Decentralized, Adaptive Replica Location Mechanism, *Proc. HPDC 2002* (2002).
5) Czajkowski, K., Fitzgerald, S., Foster, I. and Kesselman, C.: Grid Information Services for Distributed Resource Sharing, *Proc. 10th IEEE Intl. Symp. on High Performance Distributed Computing* (*HPDC-10*), San Francisco, Los Alamitos, CA, USA, IEEE (2001).
6) Raman, S. and McCanne, S.: A Model, Analysis, and Protocol Framework for Soft State-based Communication, *Proc. ACM SIGCOMM* (1999).
7) Ji, P., Ge, Z., Kurose, J. and Towsley, D.: A Comparison of hard-state and soft-state Signaling Protocols, *Proc. 2003 Conf. on Applications, technologies, architectures, and protocols for computer communications*, Karlsruhe, Germany, pp.251–262 (2003).
8) Lui, J.C.S., Misra, V. and Rubenstein, D.: On the Robustness of Soft State Protocols, *Proc. 12th IEEE Intl. Conf. on Network Protocols* (*ICNP'04*), pp.50–60 (2004).
9) Schlichting, R. and Schneider, F.: Fail-stop processors: An Approach to Designing fault-tolerant Computing Systems, *ACM Trans. Computer Systems*, Vol.1, No.3, pp.222–238 (1983).
10) Lamport, L., Shostak, R. and Pease, M.: The Byzantine Generals Problem, *ACM Trans. Prog. Lang. Syst.*, Vol.4, No.3, pp.382–401 (1982).
11) Aguilera, M.K., Chen, W. and Toueg, S.: Heartbeat: A timeout-free Failure Detector for Quiescent Reliable Communication, *Proc. 11th Intl. Workshop on Distributed Algorithms* (*WDAG'97*), Saarbrucken, Germany, pp.126–140 (1997).
12) The Globus Alliance: Globus RLS (Replica Location Service). http://www.globus.org/toolkit/data/rls/
13) Chervenak, A., et al.: Giggle: A Framework for Constructing Scalable Replica Location Services, *Proc. IEEE Supercomputing 2002* (2002).
14) The High Availability Linux Project. http://www.linux-ha.org/
15) Ahuja, S.P. and Myers, J.R.: A Survey on Wireless Grid Computing, *The Journal of Supercomputing*, Vol.37, No.1, pp.3–11 (2006).
16) Sadagopan, N., Bai, F., Krishnamachari, B. and Helmy, A.: PATHS: Analysis of PATH Duration Statistics and Their Impact on Reactive MANET Routing Protocols, *Proc. 4th ACM Intl. Symp. on Mobile Ad Hoc Networking and Computing* (*MobiHoc'03*), Annapolis, USA (2003).
17) Czajkowski, K., Foster, I. and Kesselman, C.: Resource Co-Allocation in Computational Grids, *Proc. 8th IEEE Int'l Syrup. on High Performance Distributed Computing*, Redondo Beach, CA, USA, pp.219–228 (1999).
18) Cachin, C. and Samar, A.: Secure Distributed DNS, *Proc. Intl. Conf. on Dependable Systems and Networks* (*DSN-2004*), pp.423–432 (2004).
19) Venugopal, S., Buyya, R. and Ramamohanarao, K.: A Taxonomy of Data Grids for Distributed Data Sharing, Management, and Processing, *ACM Computing Surveys*, Vol.38, No.1 (2006).
20) Foster, I. and Kesselman, C.: *The Grid: Blueprint for a Future Computing Infrastructure*, Morgan Kaufmann, San Francisco, CA, USA (1999).
21) Dobber, M., Koole, G. and van der Mei, R.: Dynamic Load Balancing for a Grid Application, *Proc. High Performance Computing* (*HiPC 2004*), Springer, pp.342–352 (2004).

(平成 18 年 10 月 3 日受付)
(平成 19 年 2 月 1 日採録)

多田 知正（正会員）

平成 5 年大阪大学基礎工学部情報工学科卒業．平成 7 年同大学大学院博士前期課程修了．平成 10 年同博士後期課程退学．同年大阪大学大学院基礎工学研究科助手．平成 14 年同大学大学院情報科学研究科助手．現在，京都教育大学教育学部講師．博士（工学）．情報ネットワークおよび分散システムに関する研究に従事．電子情報通信学会会員．

今瀬 眞（正会員）

昭和 50 年大阪大学基礎工学部情報工学科卒業．昭和 52 年同大学大学院修士課程修了．同年日本電信電話公社武蔵野研究所入所．NTT マルチメディアネットワーク研究部長等を歴任．現在，大阪大学大学院情報科学研究科教授．工学博士．情報ネットワーク，ネットワーク理論等の研究開発に従事．電子情報通信学会，応用数理学会各会員．

村田 正幸

昭和 57 年大阪大学基礎工学部情報工学科卒業．昭和 59 年同大学大学院博士前期課程修了．同年日本アイ・ビー・エム（株）入社．同社東京基礎研究所を経て，昭和 62 年大阪大学大型計算機センター助手．平成元年同大学基礎工学部助手．平成 3 年同講師．平成 4 年同助教授．平成 11 年同教授．平成 12 年大阪大学サイバーメディアセンター教授．現在，大阪大学大学院情報科学研究科教授．工学博士．システムのモデル化と性能評価，情報ネットワークアーキテクチャ等の研究に従事．IEEE, ACM，電子情報通信学会各会員．