複製したエージェントの分散配置による ウェブベースドトレーニングシステムの耐故障性向上について

本村真一[†] 中谷亮介[†]
川村尚生[††] 菅原一孔[††]

分散型非同期ウェブベースドトレーニングシステムを開発している．本システムでは，モバイルエージェント上に学習コンテンツとその採点機能を実現し，複数のコンピュータへ分散させることで，拡張性と頑健性の向上を行っている．学習コンテンツを利用する際は，Content-Addressable Network を応用した P2P ネットワークを用いてエージェントの交換を行うことで実現する．本システムでは，一部のコンピュータが故障により利用できない状態が発生しても，サービス全体が利用不能にはならないが，エージェントの消失に伴い分散している学習コンテンツの一部を消失する問題が生じる．本研究では，この問題の対策としてエージェントのバックアップを他のコンピュータへ分散して配置しておき，コンピュータの故障を検出すると他のコンピュータがエージェントのバックアップを利用してサービスを継続する．エージェントのバックアップを伴った，システムに対するコンピュータの参加，離脱の手順を定め，エージェントのバックアップを利用したサービスの復旧手法を開発した．本手法をシステムに実装することで耐故障性の向上を行った．

Improvement of Fault Tolerance in a Distributed Web based Training System by Using Backup Agents

Shinichi MOTOMURA,[†] Ryosuke NAKATANI,[†]
Takao KAWAMURA[††] and Kazunori SUGAHARA[††]

We have developed a distributed asynchronous Web based training system. In order to improve the scalability and robustness of this system, all contents and a function that marks user's answers are realized on mobile agents. These agents are distributed to computers, and they can obtain using a P2P network that modified Content-Addressable Network. In this system, although whole services do not become impossible even if some computers break down, the problem that contents disappear occurs with an agent's disappearance. In this study, as a solution for this problem, backups of agents are distributed to computers. If a failure of a computer is detected, other computer will continue service using backup of an agent. We have developed new algorithm that recovers services using backups of agents. By implementing the algorithm, we improve fault tolerance in this system.

1. はじめに

e-Learning とは，コンピュータを利用した学習の総称であり，使用するメディアの違い，複数の学習者が一斉に同期的に学習するか，あるいは非同期的に学習するか，教師がシステム中に存在するか否かなどの点から様々に分類できる．本論文では，このうち，非同期型ウェブベースドトレーニング（Asynchronous Web Based Training，以後，AWBT と略す）と呼ばれるものを対象とする．AWBT では，学習者は自分の好きな時間にネットワークを通じて配信される問題に取り組み，その場で採点を受け，正解や解説を得ることができる．AWBT に関しては，これまでに様々な研究が報告されているが[1),2)]，それらはすべて，学習者が Web ブラウザから Web サーバにアクセスするものであり，クライアントサーバモデルに基づいている．このようなクライアントサーバモデルに基づいた AWBT においては，サーバへ負荷が集中するため拡張性や頑健性に問題が発生する可能性がある．AWBT は非同期的であるためサーバへ負荷が集中しにくいが，学習者の利用時間帯が一日において平等に分散するこ

† 鳥取大学大学院工学研究科
The Graduate School of Engineering, Tottori University
†† 鳥取大学工学部
Faculty of Engineering, Tottori University

とは考えにくく，集中的にシステムを利用する時間帯が存在することが想定される．このような時間帯においてはサーバへの負荷集中が発生すると考えられる．

一方，サーバへの負荷集中を避けるために，すべてのクライアントがサーバの役割も兼ねる P2P システムの研究がさかんになっており[3)~5)]，我々は P2P 型の AWBT システムを提案した[6)~8)]．このシステムにおける P2P ネットワークは分散ハッシュテーブルの一種である Content-Addressable Network[3)](以後，CAN と略す) を応用して構築している．このシステムでは，学習者のコンピュータ（以後，ノードと呼ぶ）がシステムに接続すると，学習コンテンツの一部が他のノードから移動し，以後そのノードで管理される．このシステムの１つめの特徴は，各ノードはクライアントとしてだけではなく，他のノードへ適切な学習コンテンツを送り出すサーバとしても機能することである．もう１つの特徴は，学習コンテンツが単なるデータではなくエージェントとして構成されており，答案を採点し，正解や解説を示す機能を持つことである．これらの特徴が示すように，提案したシステムでは問題と機能を複数のノードに分散しているため負荷の集中がなくなる．

しかしこのようにエージェントを複数のノードに分散させることで，ノードに故障が発生した際，エージェントの消失に併せて学習コンテンツの一部が消失するという問題が生じる．システムにて対策が必要なノード故障には，単なるノードの故障だけでなく，システムの利用者による，アプリケーションやノードの停止等によるプロセスの強制終了も含まれる．提案したシステムの実用に際しては，このようなノード故障への対策が必要である．そこで我々は，ノードが担当するハッシュテーブル及びデータを他のノードにバックアップし，ノード障害時にはそれらのバックアップを利用して復旧する手法を開発した．本手法では，少なくとも 1 個のノード故障に対してシステムの復旧を保証するが，後で述べる条件を満たす場合，より多くのノード故障が発生してもシステムの復旧を可能とする．

CAN においても，ノード障害への対策として reality の導入が示唆されている．reality とは仮想的に用意された複数の分散ハッシュテーブルのことである．一つのノードは同時に複数の分散ハッシュテーブルに属しており，併せてデータも複製することで冗長化を図る．この手法では，実際には一つのノードがそれぞれの reality 上のハッシュテーブルの一部を担当する．そのため，ノード数が小さい状態では異なる reality に存在するハッシュテーブルの一部を一つのノードが重複して管理する可能性が高くなり，ノード故障時において必要なハッシュテーブルやデータが存在せず，復旧できないことが考えられる．先に我々が提案した AWBT システムのように，ノード数が大きく増減することを前提としたシステムでは，初期状態もしくは終了状態においてノード数が小さい状態が発生する．そのため，reality の導入ではノード故障への対策としては不十分である．

reality に頼らない新しい手法を提案及び実装することで，AWBT システムにおける耐故障性の向上を図る．

2. 提案システム

2.1 概　　要

先に我々が提案したシステムの概要を説明する．学習者は任意の時間に自分が使用するノードを提案システムに接続し，送られてくる問題を解くことにより自習を行う．システム内にはカテゴリに分類された問題が多数存在しており，学習者はカテゴリを指定することで，そのカテゴリの問題が次々に得られる．また，学習者は個々の問題について，答案の採点および正解や解説の表示を求めることができる．それと同時に，ノードはシステムに接続している間いくつかのカテゴリを担当する．すなわち当該カテゴリに属するすべての問題を保持しており，他のノードからの求めに応じて適切な問題を送り出す責任を負う．ここで重要な点は，各ノードは同時にいつくかのカテゴリーを分担して管理しており，それらはそのノード上で学習者が解いている問題のカテゴリとは無関係であるということである．**図 1** は，物理を担当しているノード上の学習者が数学の問題を解こうとしている様子を示している．どのノードがどのカテゴリを担当しているかは各ノードにとっては未知であり，この例においては，一度英語を担当しているノードに送られた要求が，数学を担当しているノードに転送されている．

また，ノード故障の対策として，ノードは自身が保持する問題をコピーし，他のノードにバックアップを保存する．ノードに障害が発生した際は，他のノードがバックアップを利用して消失した問題を復旧する．

2.2 P2P ネットワーク

システムが最初に起動したときには，単一の初期ノードがすべてのカテゴリを担当している．新しいノードは，システム内のある１つのノードに対して参加要求を行い，そのノードが担当しているカテゴリのうち約半数を受け取る．ノードがシステムから離脱す

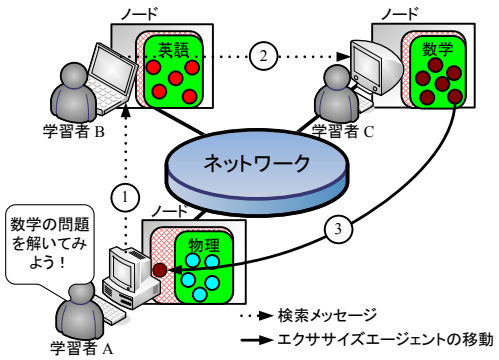

図 1 提案システムの概要
Fig. 1 Overview of proposed system.

るときは，隣接ノードのいずれかに担当していたカテゴリを渡す．このようにして，カテゴリはシステムに参加しているノード間で動的に分散される．カテゴリの授受には，そのカテゴリに属している問題の授受も含まれる．

Napster，Gnutella，Freenet などの既存の P2P 型ファイル共有システムは，各ノードが所有しているファイルを共有することが目的であり，ファイルは元々分散しているのに対し，提案システムにおいては，全てのカテゴリが最初は単一ノードに集中管理されていることに注意を要する．すなわち，新規参加ノードに対して担当すべきカテゴリを保持するノードの IP アドレスだけを渡すのでは不十分で，カテゴリそのものを渡す必要がある．この点を考慮すれば，本システムの P2P ネットワークは分散ハッシュテーブルの一種である，CAN を用いて構築することができる．

CAN では（キー K，値 V）のペアを格納するための仮想座標空間がいくつかのゾーンに分割され，ゾーンはノードによって所有される．(K, V) を格納する際には，ハッシュ関数によってキー K から得られるハッシュ値を仮想座標空間上の点とし，その点が含まれるゾーンを担当しているノードに (K, V) を格納する．キー K を持つ値 V を取り出すときも同じハッシュ関数によって対応する点を求め，その点が含まれるゾーンを担当しているノードから値 V を取り出す．本システムでは，カテゴリをキーとしそのカテゴリに属する問題の集まりを値とする．

本システムにおける P2P ネットワークは [0,1] × [0,1] の 2 次元トーラスで構築している．図 2 の例は当初ノード A のみが存在しており，その後ノード B とノード C がシステムに参加したものである．その結果，ノード A が全体の半分のゾーン，ノード B とノード C が全体の 1/4 のゾーンをぞれぞれ担当している．この時，ノード A が英文法のカテゴリに属する問題を要求することを考える．“英文法” という文字列をキーとしてハッシュ関数から対応する座標を求める．そして，その座標を含むゾーンを担当するノード B から要求する問題を取り出すことができる．

本システムではノード障害時における復旧のために，ノードが所有するカテゴリを隣接ノードにバックアップするが，その際，隣接ノードの選択や検索には P2P ネットワークを用いていない．これらの動作については後の章で詳しく述べる．なお，ここで言う隣接ノードとは，ノードが保持するゾーンの辺を共有しているものを指す．例えば，図 2 におけるノード A の隣接ノードはノード B とノード C になる．

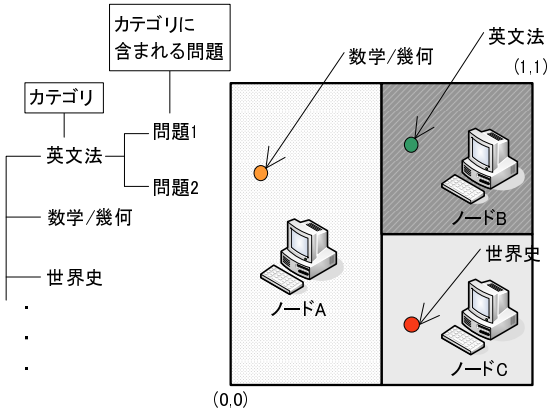

図 2 P2P ネットワーク
Fig. 2 Peer to peer network.

2.3 システムを構成するモバイルエージェント

AWBT システムを分散システムとして実現するには，問題を分散させるだけでは不十分で，その問題に対する答案の採点機能，正解や解説の表示機能も分散させる必要がある．例えば，英語の選択問題と数学等の数式を解答として入力させる問題では，その採点機能が異なる．また，カテゴリーや問題の種類が増えると，それらを採点するために多様な採点機能を持たせることが望まれる．このような，問題やその採点機能を随時追加及び更新するシステムでは，それらを利用者へ配布する際にかかる作業負担の軽減も重要である．そこで，問題とその採点等の機能を併せて分散管理し，またノード間を移動できるようにモバイルエージェントとして実現することで，利用者への配布や更新作業の軽減を図った．加えて，カテゴリもモバイルエージェントとすることで，カテゴリをノード間で受け渡すことをカテゴリに対応するエージェントの移動で実現する．これらのエージェントは我々の開発しているモバ

イルエージェントフレームワーク Maglog[9)~11)] 上に実現している．

各ノードには下記に示すエージェント及びユーザインターフェースが存在する．

ノードエージェント 各ノードに 1 つ存在し，CAN のゾーン情報及びバックアップゾーンを管理する．バックアップカテゴリエージェント及びバックアップエクササイズエージェントの生成を行う．

カテゴリエージェント (以後，CA と略す) カテゴリごとに存在し，エクササイズエージェントを管理する．

バックアップカテゴリエージェント カテゴリエージェントの複製であり，一度生成されると更新は行われない．ノード故障が発生した際その復旧に利用される．バックアップエクササイズエージェントを管理する．

エクササイズエージェント (以後，EA と略す) 問題ごとに存在し，問題データと，その採点を行い正解や解説を表示するプログラムからなる．問題や解説などのデータは HTML で保持されている．

バックアップエクササイズエージェント エクササイズエージェントの複製であり，バックアップカテゴリエージェントと同時に生成される．

ログエージェント 学習者の記録を保持するエージェントであり，学習の数だけ存在している．

インタフェースエージェント 各ノードに 1 つ存在し，ユーザインタフェースプログラムと他のエージェント間の通信を実現する．

3. ノード故障を考慮しない場合のシステムの動作

はじめに，ノード故障を考慮しない場合のシステムに対するノードの参加や離脱の手順を示す．以下で述べる手順は CA 及び EA(以後，CA+EA と略す) の移動を除けば，CAN で提案されている手順と同じである．

3.1 ノードの参加

ノード参加時の手順を以下に示す．隣接ノードに対してノードのアドレスとゾーン情報を転送することで，ルーティングテーブルの更新を要求する関数を update と定義し，以下で用いる．

Step1. 参加するノード N は，区間 [0,1) の一様乱数を 2 個発生させ，それぞれを x 座標，y 座標とする点を含むノード Np の検索を既知のノードに要求し，Np のアドレスを入手する．

Step2. N は Np に対してゾーンを半分割するよう要求する．

Step3. Np は半分割したゾーン Znp を自身が保持するゾーンとし，もう片方のゾーン Zn を N へ割り当てる．

Step4. N は Np から Zn とそのゾーンに含まれる CA+EA，Znp，Np の隣接ノードのリスト Adj(Np) を受け取る．

Step5. N は Adj(Np) に含まれるノードのなかで，N と隣接しているノードと Np を自身の隣接ノードのリスト Adj(N) に登録し，upate を実行する．

Step6. Np は Adj(Np) に含まれるノードのなかから Np と隣接していないノードを削除して，N を Adj(Np) に追加し，update を実行する．

3.2 ノードの離脱

ノード離脱時の手順を以下に示す．離脱するノードは，隣接するノードのうち同じ大きさのゾーンを保持するノードにゾーン及び CA+EA を渡す．受け取ったノードは，受け取ったゾーンと自身が保持するゾーンを結合する．しかし，隣接ノードに同じ大きさのゾーンを保持するノードがない場合，ゾーンを結合することができないため，ノードの入れ替えを行う必要がある．これは，以下の Step.3 から Step.4 が相当する．隣接ノードのリストから同じ大きさのゾーンを持つノードを検索する関数を SSVZ(SearchSameVolumeZone の略) と定義し，以下で用いる．

Step1. 離脱するノード N は SSVZ を実行し，N と同じ大きさのゾーンを持つノード Ns を見つける．

Step2. Ns が見つかれば Step.5 へ．

Step3. Ns が見つからなければ，隣接ノードのリスト Adj(N) から最も小さいノード Nm を取り出し，Nm に対して SSVZ を実行する．この Step を Ns が見つかるまで再帰的に実行する．

Step4. Ns を見つけた Nm は N とゾーン及び CA+EA を交換し，それぞれ update を実行する．

Step5. N は保持するゾーン Zn，Zn に含まれる CA+EA，Adj(N) を Ns へ渡し，システムを離脱する．

Step6. Ns は保持するゾーンと Zn を結合して新たなゾーンを生成し，自身が保持するゾーンとする．

Step7. Ns は Adj(Ns) に Adj(N) を併合し，update を実行する．

4. ノード故障を考慮した場合のシステムの動作

ノード故障への対策として，ノードが保持するゾーン及び CA+EA をノード参加時に他のノードへバックアップを行う．そこでまず，バックアップ動作について述べる．

4.1 ノードのバックアップ

ノード故障が発生すると，ゾーンの一部とそのゾーンに含まれる CA+EA が消失する．その対策として，ノードが保持するゾーンに含まれる CA+EA のバックアップ (以後，バックアップゾーンと略す) とゾーンの領域情報，すなわち仮想座標空間の対角点を隣接ノードに保持しておく．ノード故障の際はゾーンとそのゾーンに含まれる CA+EA の復旧を行う．

バックアップゾーンを保持するノード (以後，バックアップノードと略す) は隣接ノードのうち一つのノードで保持することとする．バックアップノードの選択は，近隣間のノードにおいてバックアップを含めた CA の数を均等に保ちノードの負荷を平均化するため，隣接ノードのなかからバックアップも含めて最も保持する CA が少ないノードを選ぶ．

以下にバックアップ処理，すなわちバックアップゾーンの生成とバックアップノードを決定する手順を示す．

Step1. ノード N は隣接ノードのリスト Adj(N) に含まれる全てのノードに対して，バックアップを含めて保持する CA の数 Num を返すよう要求する．

Step2. N は最も小さい Num を返したノードをバックアップノード Nbk に選出する．

Step3. N はバックアップゾーンを生成し Nbk へ転送する．

Step4. N は Nbk へ Adj(N) を転送する．

この手順の動作を図 3 の例を用いて説明する．例ではノード A がバックアップ処理を実行しようとしている．

(1) ノード A は隣接ノードであるノード B，ノード C，ノード D に対して，バックアップも含めて保持する CA の数を返すよう要求する．

(2) ノード B，ノード C，ノード D はノード A に CA の数を返答する．

(3) 最も CA の数が少ないノードがノード B であった場合，ノード A はノード B をバックアップノードに選出し，バックアップゾーンを生成して転送する．

(4) ノード A はノード B に隣接ノードのリスト (ノード B，ノード C，ノード D) を転送する．

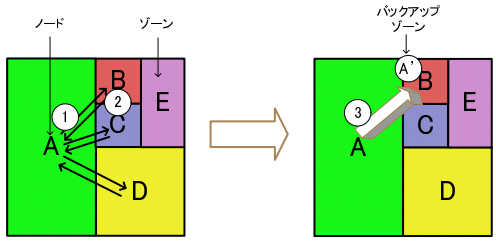

図 3 ノードのバックアップ処理
Fig. 3 A backup process of Node A.

本手法ではバックアップを一つのノードで保持しているため，あるノードとそのバックアップノードに対して同時に故障が発生した場合は復旧できないが，それ以外の組み合わせであれば複数台の故障時においても復旧できる．

次にノードの参加，離脱の手順とその動作例について示した後，ノード障害が発生した際の復旧について示す．

4.2 ノードの参加

バックアップ処理を含めたノードの参加の手順は，3.1 で示した手順の最後に以下の手順を追加したものである．

Step7. Np はバックアップノードに Adj(Np) を転送する．

Step8. N はバックアップ処理を行う．

この動作を図 4 の例を用いて説明する．例において，参加するノード F が一様乱数により発生した座標がノード A の保持するゾーンに含まれていたとすると，ノード A に対してシステムへの参加要求を行う．

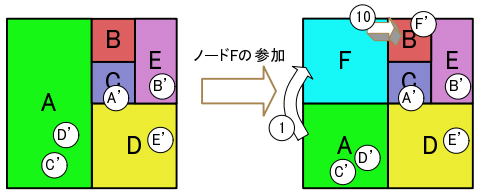

図 4 ノードの参加
Fig. 4 Node F joins the system and Node A splits own zone in half, retaining half and handing the other half to Node F.

(1) ノード F はノード A が保持するゾーンの半分とそのゾーンに含まれる CA+EA，ノード A に残るゾーンの領域情報及びノード A の隣接ノードのリストであるノード B，ノード C，ノード D を受け取る．

(2) ノード F は受け取ったリストのうち，隣接しているノード B 及びノード C と分割前のノードであるノード A を隣接ノードとして登録し，ノード B とノード C に対してルーティングテーブルの更新要求を行う．
(3) ノード A は隣接ノードのリストから，既に隣接していないノード B とノード C を削除して，ノード F を隣接ノードのリストに追加し，ノード D に対してルーティングテーブルの更新要求を行う．
(4) ノード A はバックアップノードであるノード C に隣接ノードのリストを転送する．
(5) ノード F はバックアップ処理を行う．ここでは，ノード B がノード F のバックアップノードとなっている．

ここで，ノード C はノード A に隣接していないが継続してノード A のバックアップノードとなっている．これにより，ノードの離脱やノード故障時における動作に不具合が発生することはない．後の例で正常に動作することを示す．

4.3 ノードの離脱

バックアップを含めたノードの離脱の手順は，3.2 で示した手順に対して以下の追加と変更を行う．また，Step5. の手順を次の Step5'. に変更する．

Step5'. N は保持するゾーン Zn，Zn に含まれる CA+EA，Adj(N)，保持するバックアップゾーンの集合 S(BKz) を Ns へ渡す．

上記 Step5'. の後に以下の 3 つの手順を追加する．

Step5.1. N はバックアップノードに対してバックアップゾーンの破棄を要求し，システムを離脱する．
Step5.2. Ns はバックアップノードに対してバックアップゾーンの破棄を要求する．
Step5.3. Ns は S(BKz) の各ノードに対してバックアップノードが Ns へ更新された旨通知する．

3.2 で示した Step7. の後に以下の手順を追加する．

Step8. Ns はバックアップ処理を行う．

この動作を図 5 の例を用いて説明する．例において，ノード F がシステムから離脱しようとしている．

(1) ノード F はゾーン及び CA+EA を受け渡す相手に，隣接ノードのなかからゾーンの大きさが同じノード A を選び，通知する．
(2) ノード F はノード A にゾーンと CA+EA，隣接ノードのリストを転送する．バックアップゾーンは保持していないため転送しない．
(3) ノード F はバックアップノードであるノード B にゾーンの破棄を要求し，システムを離脱する．
(4) ノード A はバックアップノードであるノード C にゾーンの破棄を要求する．
(5) ノード A は保持するゾーンと受け取ったゾーンを結合し，受け取った CA+EA を管理する．
(6) ノード A は隣接ノードのリスト (ノード D) に受け取った隣接ノードのリスト (ノード B，ノード C) を追加し，それぞれのノードに対してルーティングテーブルの更新を要求する．
(7) ノード A はバックアップ処理を行う．この例では，ノード B にバックアップゾーンを転送している．

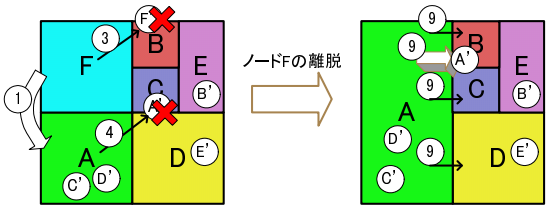

図 5 ノードの離脱
Fig. 5 Node F leaves the system and Node A merges Node F's zone with own zone.

4.4 ノード故障からの復旧

全てのノードはバックアップノードに対して定期的に ping メッセージを送信している．バックアップノードは ping メッセージが一定時間以上届かなくなるとノードに故障が発生したと判断し，故障が発生したノードの隣接ノードのなかから同じ大きさを持つノード，存在しなければ一番小さいノードを選出してバックアップゾーンを渡し，ゾーン及び CA+EA の復旧を指示する．

Step1. ノード N からの ping が一定時間以上届かなくなると，バックアップノード Nbk は N に故障が発生したと判断する．
Step2. Nbk は N の隣接ノードに故障発生メッセージを送信する．
Step3. 故障発生メッセージを受信したノードは保持するゾーンの大きさを返信する．
Step4. Nbk は全ての返信を受け取ると，Nbk が保持するゾーンと同じ大きさのノード Nr を選出し，復旧を担当するノードとする．
Step5. Nr がなければ 4.3 の Step.3 から Step.4 と同様の処理を実行して Nr を選出する．
Step6. Nbk は Nr にバックアップゾーンを転送する．
Step7. Nr はバックアップノードに対してバックアップゾーンの破棄を要求する．
Step8. Nr は保持するゾーンとバックアップゾーンに

含まれるゾーンを結合して新たなゾーンを生成し，自身が保持するゾーンとする．

Step9. Nr は update を実行し，バックアップ処理を行う．

この動作を図 6 の例を用いて説明する．ノード B にて故障が発生した際，ノード B のバックアップノードであるノード E がその故障を検出し，下記手順でそのゾーンと CA+EA の復旧を行う．

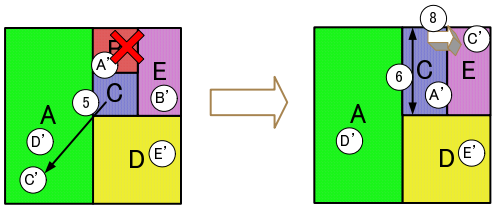

図 6 ノード故障からの復旧
Fig. 6 Node C recovers from Node B's failure.

(1) ノード E はノード B の隣接ノードであるノード A，ノード C に故障発生メッセージを送信する．
(2) ノード A，ノード C は自身が保持するゾーンの大きさを返信する．
(3) ノード E はノード B と同じ大きさのゾーンを保持するノード C を復旧を担当するノードとして選出する．
(4) ノード E はノード C に対してノード B のバックアップゾーンを転送する．
(5) ノード C はバックアップノードであるノード A にバックアップゾーンの破棄を要求する．
(6) ノード C は保持するゾーンとバックアップゾーンに含まれるゾーンを結合して新たなゾーンを生成し，自身が保持するゾーンとする．
(7) ノード C は隣接ノードであるノード A，ノード D，ノード E にルーティングテーブルの更新を要求し，バックアップ処理を行う．この例では，ノード E にバックアップゾーンを転送している．

5. 実　　験

1Gbps の LAN にて接続した 70 台程度のコンピュータを用いて，本システムの試験利用を行っている．本システムの有効性を確認するため次の実験を行った．表 1 に示す構成のコンピュータを用いて，エージェントの生成を 1,000 回行い，1 個のエージェントを生成するために必要な平均時間を求めた．また，同条件のコンピュータ 2 台を 1Gbps の LAN で接続し，片方のコンピュータからもう一方のコンピュータへエージェントの移動を 1,000 回行い，1 個のエージェントが 1 回移動するために必要な平均時間を求めた．これらの結果を表 2 に示す。

表 1 実験に使用したコンピュータの構成
Table 1 Components of the computer used for the experiments

CPU	Intel Xeon 3.4GHz
メモリ	1GB
OS	TurboLinux Server10
Java の実行環境	JRE1.4.2_04

表 2 エージェントの生成と移動に要する時間
Table 2 Experimental results of agents creation and agents migration

実験項目	空のエージェント (1.7KB の JAR ファイル)	EA (120KB の JAR ファイル)
1 個のエージェントの生成に要する時間 (msec)	10.5	43.3
1 個のエージェントが移動に要する時間 (msec)	1.2	2.6

6. おわりに

既存の AWBT システムはクライアントサーバモデルに基づいているため，拡張性や頑健性に問題がある．この問題を解決するため，先に，P2P ネットワークとエージェント技術を用いた分散型 AWBT システムを，モバイルエージェントシステム構築用プラットフォーム Maglog を用いて開発した．

このシステムでは一部のノードに故障が発生した際も，サービス全てが利用不能にはならないが，分散している学習コンテンツの一部が消失する可能性がある．そこで今回我々はノード故障に備えて，学習コンテンツのバックアップを伴った新しい復旧アルゴリズムを開発した．本手法ではバックアップを一つのノードで保持しているため，あるノードとそのバックアップノードに対して同時に故障が発生した場合は復旧できないが，それ以外の組み合わせであれば複数台の故障時においても復旧できる．

本手法によるシステムへのノードの参加，離脱の手順を示し，ノード故障が発生した際の復旧手順を示した．本手法を実装することでシステムの耐故障性向上を行った．

謝辞 プログラムの作製にあたり協力いただいた，鳥取大学工学部の木下俊吾氏に感謝します．

参 考 文 献

1) Implementing Project-Based Learning in WBT Systems, pp. 2189–2196 (2003).
2) Creation of WBT Server on Digital Signal Processing (2003). Marrakech, Morocco.
3) A Scalable Content-Addressable Network, ACM Press, pp. 161–172 (2001).
4) PeerDB: A P2P-based System for Distributed Data Sharing, IEEE Computer Society, pp. 633–644 (2003).
5) EDUTELLA: A P2P Networking Infrastructure Based on RDF, ACM Press, pp. 604–615 (2002).
6) モバイルエージェントに基づく P2P 型 e-Learning システム, 情報処理学会論文誌, Vol. 46, No. 1, pp. 222–225 (2005).
7) P2P e-Learning System and Its Squeak-based User Interface, IEEE Computer Society, pp.57–63 (2005).
8) : .
9) A Logic-based Framework for Mobile Multi-Agent Systems, pp. 754–759 (2003). Boston, Massachusetts, USA.
10) Implementation of a Mobile Agent Framework on Java Environment, pp.589–593 (2004). MIT, Cambridge, USA.
11) Logic-Based Mobile Agent Framework Using Web Technologies, pp. 198–201 (2005). Miami, USA.