移動体計算環境におけるアクティブデータベースの動的トリガグラフ構築機構の実現

寺田 努[†]　塚本昌彦[‡]　西尾章治郎[‡]

[†]大阪大学サイバーメディアセンターサイバーコミュニティ研究部門
[‡]大阪大学大学院工学研究科情報システム工学専攻
[†]tsutomu@cmc.osaka-u.ac.jp　[‡]{tuka, nishio}@ise.eng.osaka-u.ac.jp

無線通信や計算機ハードウェア技術の急速な発展により，ユーザは無線通信機能を持つ携帯端末を用いて，場所を固定せずにネットワークを介して情報を利用することが可能になった．筆者らは，このような環境において移動体がもつデータを統合利用するために，アクティブデータベースを拡張し，移動体の接続，切断，データ交換などを処理するAMDS(Active Mobile Database System)を提案・実装してきた．AMDSの動作言語であるECAルールは，連鎖的に実行させることで複雑な処理が記述できる一方，予期しない異常動作を起こす可能性がある．一般にアクティブデータベースの異常動作検出にはトリガグラフと呼ばれる有向グラフを用いるが，トリガグラフはネットワーク構成に依存するため，ネットワーク構成が動的に変化する移動体計算環境で用いることは困難である．そのため，筆者らは移動体計算環境において動的にトリガグラフを再構築して異常動作を検出する手法を提案してきた．本研究では，動的なトリガグラフ構築機構のAMDS上への実装を行なう．また，シミュレーションにより提案手法の有効性を検証する．

Realizing a Dynamic Construction Mechanism of a Trigger Graph on Active Databases in Mobile Computing Environments

Tsutomu TERADA[†]　Masahiko TSUKAMOTO[‡]　Shojiro NISHIO[‡]

[†]Cybercommunity Division, Cybermedia Center, Osaka University
[‡]Dept. of Information Systems Engineering, Graduate School of Engineering, Osaka University

As a result of rapid development in wireless communications and computer hardware technologies, currently, we can access various information from anywhere using portable terminals with wireless communication capabilities. To support the integration and the use of data held by mobile hosts in this environment, we have proposed and implemented AMDS (Active Mobile Database System) as the kernel system for data and mobile host management in mobile environments. ECA rules, the behavior definition language of AMDS, have high description capability that enables users to define complicated behavior. However, the execution of ECA rules may fall into a chain of unexpected behaviors. In general, a directed graph called trigger graph is used for detecting illegal chains. Unfortunately, trigger graphs highly depend on network topology, thus it is difficult to employ trigger graphs in mobile computing environments. Therefore, we have proposed a method that could reconstruct trigger graph dynamically. In this paper, we implement our method on AMDS and evaluate it by simulation studies.

1　はじめに

近年，無線通信技術や計算機ハードウェア技術の急速な発展により，無線通信機能をもつ携帯端末を用いることで場所を固定せずにネットワーク上のさまざまな資源を利用することが可能になった．この新しい計算環境を移動体計算環境と呼ぶ．移動体計算環境のモデルは図1に示すように，固定ネットワークに無線通信可能な移動端末を含んだ形態である．移動体計算環境においては，以下のような新しいサービスを提供できるようになる．

- 会社において，社員は各自のスケジュールが入力された携帯端末を持ち歩き，本社でスケジュールデータを統合して全社員のスケジュールを管理し，効率的に仕事を割り当てる．
- 遊園地などのアミューズメント施設で入場者に一台ずつ携帯端末を持たせ，各アトラクションの待ち時間等の情報を提供する．
- 美術館や博物館において，入館者は一台ずつ携帯端末を持ち，展示物に近づくと自動的にその

展示物の詳細情報が表示される．

このようなサービスを実現するためには，移動体から，または移動体上でデータを収集する必要がある．しかし，従来の分散データ管理技術では移動するホストを考慮していないため，移動体のネットワークへの接続・切断やデータの収集を自動的に行なう共通の基盤が望まれている．このような要求に対し，筆者らは，イベント駆動型データベースであるアクティブデータベースを拡張することで，移動体計算環境における各種のイベントを容易に扱うことができるアクティブモーバイルデータベース(Active Mobile Database System: AMDS)の研究を行なってきた[4][9]．AMDSでは，イベント，コンディション，アクションの3つを一組として記述するECAルールによって動作を記述する．イベントには従来のアクティブデータベースのイベント(更新・挿入・削除など)に加え，移動体の接続・切断などを扱うイベントも提供している．

ECAルールは，アクションの実行によって新たなイベントを引き起こすことができる．ECAルールを連鎖的に実行させることで，複雑な動作が実現できる一方，無限ループなど予期せぬ動作に陥る可能性がある[3]．そのため，ECAルールの実行を監視してシステムの異常動作を回避する必要がある．そこで，筆者らは移動体計算環境において，ECAルールの連鎖によるシステムの異常動作を検出するための動的トリガグラフ構築手法を提案してきた[8]．本研究では，筆者らが提案する動的トリガグラフ構築手法をAMDS上に実装するとともに，シミュレーションによる評価を行ない，本手法の有効性とトラフィックに与える影響を評価する．以下，2章ではAMDSの概要について述べ，3章ではアクティブデータベースの異常動作とその一般的な解決手法について述べる．4章で動的トリガグラフ構築手法について説明し，5章でシステムの実装について述べる．6章で提案手法の評価を行ない，最後に7章で本研究のまとめと今後の課題について述べる．

2 AMDS

AMDSはアクティブデータベースを拡張して移動体計算環境に適合させたシステムである．アクティブデータベースは，データベースの内界・外界で起こる事象の発生に対して，規定された処理を行なうデータベースであり，その動作は，発生する事象(イベント)，ルールの発火条件(コンディション)，実行される操作(アクション)の3つの組みで表わされるECAルールで記述する[2]．

AMDSには，従来のアクティブデータベースで提供されている，データベースに関するイベントに加えて，移動体の動作およびAMDS間の通信に関するイベントが用意されている．アクションにはAMDS間のデータ送信関数，データベース対する問合せ関数などが提供されている[4]．AMDSで提供されているイベント，アクションを表1，2に示す．また，到着したデータの内容を用いて処理を行なうルールなど，イベント対象となったタップル情報が必要になる場合があるため，NEWデータ，OLDデータと呼ぶシステム変数を用意する．イベント発生時にこれらの変数に必要な情報が自動的に格納され，ルール中で自由に使用することができる．各イベントに対するNEWデータ，OLDデータの内容を表3に示す．

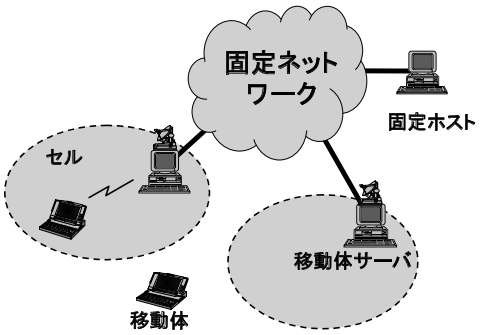

図 1: 移動体計算環境

表 1: AMDSのイベント

名称	内容
CONNECT	移動体のセルへの接続
DISCONNECT	移動体のセルからの切断
SELECT	テーブルに対するデータ参照
INSERT	テーブルに対するタップルの挿入
DELETE	テーブルのタップル削除
UPDATE	テーブルのタップル更新
RECEIVE	データパケットの受信
TIMER	設定したタイマの発火

AMDSにおけるECAルールの記述例を図2に示す．接続ルールは，移動体が接続してきたときに，その移動体に対してデータ要求を行なう移動体サーバ用のルールである．また，返信ルールは，移動体サーバからの要求パケットを受信したとき，スケジュールデータを送り返す移動体用ルールである．移動体サーバでは，移動体が接続してきたときに接続ルールが起動し，接続ルールのアクションにより，移動体の返信ルールが起動する．このように，ECAルールでは，アクションの実行が新たなイベントを発生させられるため，複数のECAルールを連鎖的に実行させて複雑な動作を実現できる．

表 2: AMDSのアクション

名称	内容
QUERY([クエリー内容])	データベース操作
SEND([宛先], [送信内容])	データの送信
INSERT_ECA([ルール内容])	ECAルール格納
DELETE_ECA([ルール識別子])	ECAルール削除
ENABLE_ECA([ルール抽出条件])	ECAルール有効化
DISABLE_ECA([ルール抽出条件])	ECAルール無効化
SET_TIMER([タイマ条件])	新たなタイマの設定
KILL_TIMER([タイマ識別子])	タイマの削除

```
create rule 接続 on CONNECT
  then do
   SEND( new.from, "Request_" );

create rule 返信 on RECEIVE
  where new.header = 'Request_'
  then do data = QUERY("select s.*
                        from Schedule s");
   SEND( new.from, "result_", data );
```

図 2: ECAルール例

表 3: NEWデータとOLDデータの内容

イベント	NEW	OLD
CONNECT	接続移動体情報	–
DISCONNECT	–	切断移動体情報
SELECT	参照タップル	–
INSERT	挿入タップル	–
DELETE	–	削除タップル
UPDATE	更新後タップル	更新前タップル
RECEIVE	到着パケット内容	–
TIMER	タイマ識別子	–

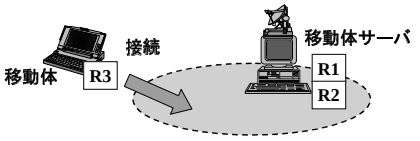

図 3: 異常動作の例

3 異常動作の検出

本章では，ECAルールの連鎖による異常動作の例を示し，アクティブデータベースの異常動作検出手法と，その問題点および解決方法について述べる．

3.1 ECAルールの異常動作

ECAルールは，アクションの実行が新たなイベントを発生させられるため，複数のECAルールを連鎖的に実行できる．そのため，ECAルールの連鎖実行による複雑な動作を実現できる一方，無限ループなど予期しない連鎖を起こす可能性があり，このような異常動作を検出することが必要となる．

図3，表4に，異常動作の例を示す．R1，R2は移動体サーバに格納されているルール，R3は移動体に格納されているルールである．移動体がセルに接続すると，R1が起動され，移動体サーバは移動体に問合せを行なう．問合せにより移動体のR3が起動され，移動体は移動体サーバに移動体サーバの問合せを行なう．これによりR2が起動されるが，このルールは再びR3を起動するため，R2とR3の間に無限ループが発生する．

このような異常動作の解析的な検出は困難であるが，ECAルールを用いたアプリケーションを運用する場合，ECAルールが異常動作を起こさないことを保証する必要がある．したがって，異常動作が起こるかどうかをあらかじめ調べたり，異常が実際に起こったときにそれを検出する仕組みが必要となる．

3.2 異常動作の検出手法

一般に，アクティブデータベースにおける異常動作検出には，次に示す2つの手法が考えられる．

- 実行時検出

  実際にデータベースが動作している状態で，リアルタイムに異常動作の有無を検出する．

- 事前検出

  データベース (ECAルール) が動作していない状態で，トリガグラフと呼ばれる有向グラフを用いて論理的に異常動作の有無を調べる．

実行時検出では，ルールの連鎖カウンタやタイムスタンプを用いて異常動作を検出する [7]．実行時検出は実際に異常が発生してからその検出を行なうので，システムの安全性をあらかじめチェックしておきたい場合などには用いることができない．したがって，異常が実際に発生する前にその可能性を調べる事前検出が必要となる．

事前検出で用いるトリガグラフとは，あるECAルールから次に引き起こすルールに対して有向のパスをはるという作業をすべてのルールに対して行なったもので，出来上がったグラフ中にループが存在していなければ，そのルール群は安全であるとするものである [5]．トリガグラフは，その信頼性を高めるためのさまざまな拡張がされており [1][6],

表 4: ルールの内容

ルール	E	C	A
R1	移動体接続	–	データ要求
R2	パケット到着	身元確認	身元要求
R3	パケット到着	–	身元要求

アクティブデータベースの無限ループ検出においては信頼性が高い．

しかし，AMDSで想定する移動体計算環境においては，移動体の移動によりネットワーク構成が動的に変化する．また，AMDSでは，ECAルールのサイト間でのやりとりも頻繁に起こるため，複数サイト間にまたがるECAルールの相互関係を考慮する必要がある．したがって，移動体計算環境においてトリガグラフを作成するためには，あらかじめすべての移動体や固定ホスト内にあるECAルールを知っておかなければならない．さらに，ネットワーク構成の変化に応じてトリガグラフの構成も変化するため，各移動体があらゆる移動体サーバに接続した場合についてトリガグラフを構築する必要がある[3]．実環境では，どのようなルールをもった移動体が接続してくるかを知ることは困難であり，また，すべてのトポロジについて調べるのは計算量が非常に大きく現実的でない．

そこで，筆者らはこれらの問題点を解決するため，新たに直前検出と呼ぶ検出手法を提案した．直前検出手法では，ネットワーク構成の変化に応じてトリガ情報をホスト間で送受信してトリガグラフを再構築する．本手法は，トリガグラフを用いて異常動作を検出するため，実際に異常が発生する前にその可能性を調べることができる．また，ネットワーク構成の変化に応じて必要な情報だけをやり取りするため，新たな移動体の接続にも対処でき，計算量も削減できる．

4 直前検出手法

本章では，筆者らが提案する直前検出手法について述べる．まず，トリガグラフの構築アルゴリズムについて述べ，次にトリガグラフの更新処理について説明する．そして，最後に異常動作を検出したときの処理方法について述べる．

4.1 トリガグラフの構築

トリガグラフの構築は以下のステップで行なう．

1. 自サイトのトリガグラフ生成と無限ループ検出.
2. RSパスの検出.
3. RSパスのマージ.
4. RSパスの送信.
5. 他サイトでのRSパスの受信処理.

まず，各ホストがそれぞれ自サイトのトリガグラフを作成し，ローカルな無限ループの検出を行なう．ここで異常が検出されなかった場合，サイト間にまたがった連鎖を引き起こす可能性のあるルールから構成される部分トリガグラフを送信する．この部分グラフを**RS(Receive-Send)パス**と呼ぶ．他のサイトではこのRSパスを受け取ったら，その情報を含めて再び無限ループの検出処理を行なう．

以下，それぞれのステップについて説明する．

4.1.1 自サイトのトリガグラフ作成

自サイトのトリガグラフを構築する作業は次の手順で行なう．

1. **連鎖パスの作成**.

   自サイトのすべてのルールについて，そのアクションが発火させる可能性のあるルールにパスを張り，あるルールから始まった連鎖が再び同じルールを発火させた場合ループでとして検出する．ループとなるパスが存在しなければトリガパス生成は正常終了する．

2. **コンディションのチェック**.

   ループが存在した場合，それが無限ループになるかどうかを判断するために以下の手順でコンディションのチェックを行なう．

   (a) NEW，OLD変数の置換.
   コンディションに，NEW変数やOLD変数が用いられていた場合，NEWやOLDを具体的な値に変換する．

   (b) コンディションのマージ.
   連鎖パスに沿ってコンディションをANDで連結する．コンディションを連結する際にはそのルールのアクションもチェックし，そのアクションがアクセスしているテーブルに関するコンディションは，無効になっている可能性があるのでそれまでの連結コンディションから取り除く．

   (c) コンディションのチェック.
   マージされたコンディションに明らかな矛盾がなければ，無限ループと判断．矛盾があればその連鎖パスは無効とする．

このような手順でトリガグラフを作成し，ホスト内での無限ループを検出する．図4にトリガグラフの構築例を示す．システム内にR1からR5までの5つのルールが存在する．ルールxか

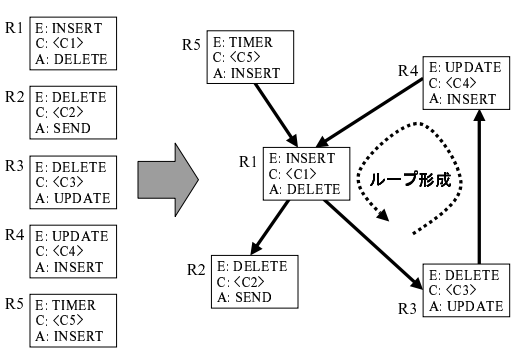

図 4: トリガグラフ構築例

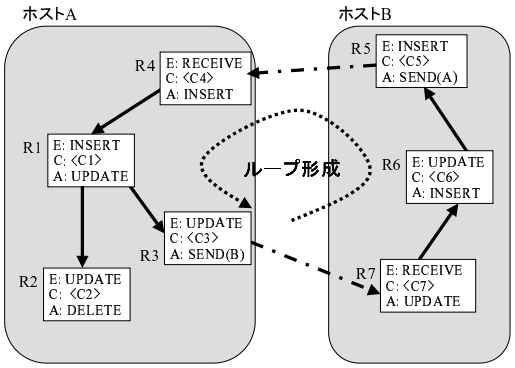

図 5: サイト間にまたがった連鎖

ら y へのパスを (x, y) と表現すると，図 4 におけるパスは $\{(R1, R2), (R1, R3), (R3, R4), (R4, R1), (R5, R1)\}$ となる．このパス集合から連鎖パス $(R1, R3, R4, R1)$ が検出され，もしマージされたコンディションが成り立つ可能性があるならループとして検出される．

4.1.2 RSパスの作成

AMDSにおいて，図5に示すような，サイト間にまたがった連鎖を引き起こすためには，無限ループを構成するサイトにRECEIVEイベントで始まってSENDアクションで終わる連鎖パスが存在する必要がある．そこで，RECEIVEイベント→SENDアクションのパスを**RS(Receive-Send)パス**と呼ぶ．自分のサイトにRSパスが存在した場合，それがサイト間にまたがった無限ループを構成する可能性があるため，関連するサイトにRSパスの情報を送信することでサイト間にまたがった無限ループを検出する．

RSパスは，4.1.2節でトリガグラフを構築する手順と同様に，自サイトの連鎖パスからRECEIVEイベントで始まりSENDアクションで終わるものを抽出して作成する．

4.1.3 RSパスのマージ

RSパスは送信先サイトでそのままルールの連鎖パスとして使われ，さらに他のサイトに伝播することがあるため，出来上がったRSパスをそのまま送信すると，通信量が多くなってしまう．そこで，以下の手順でRSパスから不要な情報を削除する．

1. **連鎖パス内でのコンディションのマージ**.

   RSパスを構成するすべてのルールの情報のうち，最終的に利用されるのは先頭のルールのイベント部分と最後のルールのアクション部分のみとなる．そこで，RSパスを単一ルールの形に縮退する．まず，先頭ルールのイベントと最終ルールのアクションをそれぞれマージ後のルールのイベント，アクションとし，4.1.2節の方法を用いてコンディションをマージした結果をマージ後のコンディションとする．ここで，ローカルデータベースに関するコンディションは，RSパス送信先のサイトには関係しないのですべて取り除く．

2. **複数の縮退RSパスのマージ**.

   縮退したRSパスのうち，SENDアクションの宛先が同じものを，コンディションをOR条件で統合して，1つのルールにマージする．

4.1.4 RSパスの送信

生成したRSパスを送信する．送信先サイトは以下のどちらかとなる．

- **対象のRSパスのSENDアクションに宛先が指定されている場合**.

  宛先が固定されている場合は，そのサイトにしかルールの連鎖が起こらないため，対象サイトのみにRSパスを送信する．

- **SENDアクションに宛先がない場合**.

  宛先を指定しない場合は，無線通信可能な範囲にいるすべてのサイトが対象になるため，RSパスもすべてのサイトに送信する．

4.2 トリガグラフの更新処理

ネットワーク構成やECAルール構成が変化した場合，その時点までのトリガグラフは無効になる．

また，サイト間にまたがるトリガグラフの場合，ローカルの更新を他のサイトに伝播する必要がある．そのため，状況の変化に応じてトリガグラフの再構築処理を行なう．具体的には，ECAルールが追加・削除された場合，操作後のルール集合に再びアルゴリズムを適用する．移動体が接続したときには，移動体に対してRSパス生成要求を送信し，移動体上でアルゴリズムを開始させる．

4.3 異常動作検出後の処理

本研究で提案する手法を用いて異常が検出されなかった場合，そのルール構成が安全であることが保障できる．一方，トリガグラフによる異常の検出は，異常動作を起こす可能性を示すものであり，必ず異常が起こることを保証するものではない．したがって，トリガグラフにより異常を検出した場合も単純にシステムを停止させるだけではなく，柔軟な処理が行なえることが望ましい．

そこで，本手法では異常動作を検出した場合，以下の3種類の処理を選択的に，または組み合わせて利用できるようにする．

- **ユーザへの通知とルール監視**.

  この処理手法は，異常動作を検出した場合ユーザにアプリケーションの継続か停止かを選択させる手法である．実行を継続する場合，ループを形成している連鎖パスを危険パスとし，危険パスが実際に発火した場合にはそれ以降のイベント発火とルール連鎖をトレースしてログに記録する．イベントログを利用することで原因の解決を図ることは一般的であるが，本手法では危険性がある場合のログだけが記録されるため，ログ記録処理のオーバーヘッドが少なくなる．

- **異常発生原因の除去**.

  この処理手法は，異常動作の原因となったルールやホストを切り離すことで無限ループを回避する方法である．例えば移動体が接続してきたことにより，トリガグラフがループを検出した場合，その移動体との通信を拒否することでループの原因を除去する．

- **エラーイベントを発生させる**.

  異常動作を検出したときの状況をNEWデータにもつERRORイベントを発生させる．したがって，ERRORイベントを処理するECAルールを記述しておくことで柔軟なエラー処理が可能となる．

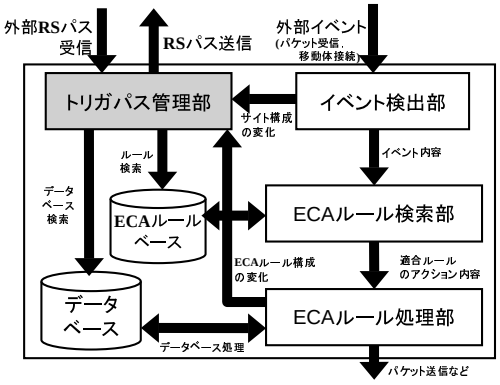

図 6: AMDSのシステム構成

5 実装

以上で述べた動的トリガグラフ構築機構を，AMDS上に実装した．実装はWindows2000 Professional上でVisualBasic6.0を用いて行ない，無線通信には赤外線を用いた．動的トリガグラフ構築機構を含むAMDSの構成は図6のようになる．従来のAMDSの各モジュールに加え，新たにトリガパス管理部を実装した．トリガパス管理部では，ECAルールベース，データベースの内容をもとにホスト内のトリガパスの作成およびRSパスの作成を行ない，関連サイトにRSパスの送信を行なう．イベント検出部でCONNECT/DISCONNECTイベントを検出したとき，およびECAルール処理部でECAルールの追加/削除アクションが実行されたとき，または他のサイトからRSパスを受け取ったときにはトリガパス管理部に通知され，トリガパスの再構築が行なわれる．実装したシステム上でいくつかのルールを実際に動作させ，本手法が正常に動作していることを確認した．

また，多数のホストを想定したシミュレーション実験を行なうため，筆者らが開発したAMDSシミュレータ上にも同様に本機構を実装した．

6 評価

本章では，提案手法を用いることで増加するシステムのトラフィック量を，動的トリガグラフ構築機構を実装したAMDSシミュレータを用いて評価する．まず，本評価で用いたAMDSシミュレータについて説明し，次に実験結果について述べる．

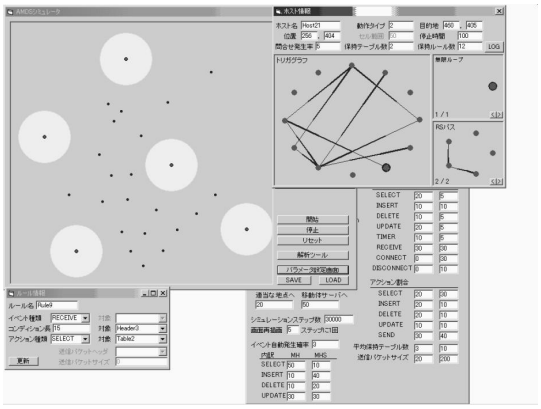

図 7: AMDSシミュレータの実行画面

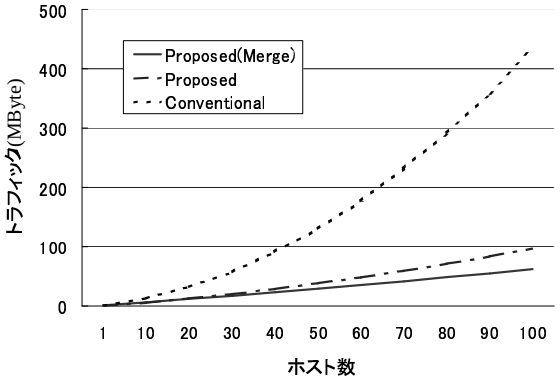

図 8: ホスト数に対する総トラフィックの変化

6.1 AMDSシミュレータ

AMDSは，携帯端末上での利用を想定しているため，その評価を行なうためには多数の端末を用意し，それぞれにECAルールを設定した上で環境内をアプリケーションを実行しながら移動させる必要がある．しかし，大量の実機を用いてアプリケーションのテストを行なうことは現実的でないため，筆者らはPC上に擬似的に実験環境を再現するAMDSシミュレータを開発した．AMDSシミュレータの実行画面を図7に示す．

AMDSシミュレータは以下の特徴をもつ．

- 多数の移動体を設置し，自由に移動させることができるので，大規模なアプリケーションのシミュレーションが可能．
- それぞれのホストごとに実際にAMDSのインスタンスを実行しているため，シミュレーションの信頼性が高く，実環境への適用が容易である．
- RSパスの使用/不使用だけでなく，RSパスのマージの有り無しなど，手法の細かい部分ごとに利用するかしないかを決定できるため，手法の各部分が性能に与える影響を測定できる．
- グラフィカルなインタフェース上で自由にホスト情報の閲覧およびホストがもつECAルールを書き換えられるため，アプリケーションを修正しながらシミュレーションが行なえる．
- 移動体の移動パターンは，ランダムや特定の移動体サーバに向かうものの他に，自由にシナリオを書いて動作させることができるため，さまざまな移動モデルをシミュレートできる．

6.2 シミュレーション結果

前節で説明したAMDSシミュレータを用いて，シミュレーション評価を行なった．シミュレーションは，遊園地における待ち時間取得アプリケーションをシミュレータ上に構築して行なった．各施設を表す移動体サーバを環境内に6台設置し，同時に存在する移動体の数を1〜100まで変化させたときに，提案手法がシステムのトラフィックに与える影響を調べた．評価の比較対象として，RSパスを用いるが，4.1.3節で述べたRSパスのマージを行なわない手法，およびRSパスを用いずに，すべてのサイトに更新されたトリガ情報を伝播する手法のトラフィックを調べた．また，各ホスト上では，一定の確率で保持するデータベースに対する問合せや更新が発生するとした．

移動体は，500×500マスのフィールド上に配置され，1ステップにつき1マス移動できるとした．本シミュレーションでは，移動体はランダムに選んだ移動体サーバに向かって進み，たどり着いたら一定時間休む，という動作を繰り返すようにした．上記の条件において10万ステップのシミュレーションを行ないその結果を調べた．結果のグラフを図8，9に示す．

図8は，環境内の移動体数が変化したときの総トラフィック量の変化を示している．総トラフィック量とは，実際にアプリケーションが送受信するデータ量に，異常動作検出に用いるパス情報の通信量を加えたものである．したがって，この総トラフィック量は，実環境において実際に異常動作検出機構を備えたアプリケーションを動作させるときにシステムに発生するトラフィック量を表している．

グラフより，すべてのサイトでトリガグラフをもつ従来手法では，ホスト数の増加に対してトラ

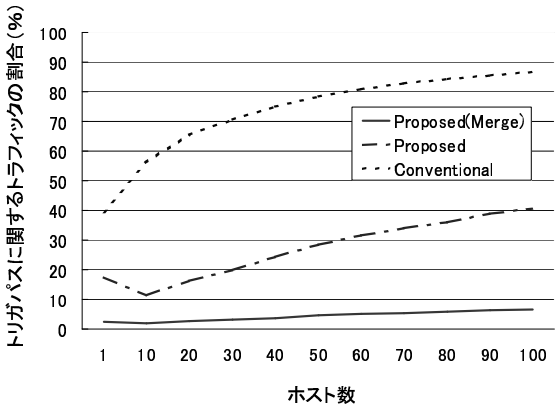

図 9: 総トラフィックに占めるパス情報の割合の変化

フィック量が指数関数的に増加していることがわかる．これは，従来手法ではトリガの更新情報を接続しているすべてのサイトに対して送信しており，ホスト数の増加が，トリガ情報の更新の発生頻度および送信相手の増加の両方に影響するためであると考えられる．提案手法では，RSパスを用いることによるパス情報量の減少だけでなく，関連するサイトにしかトリガ情報が送信されないため，トラフィックの増加はほぼ一次関数的増加になっている．ホスト数100の場合で見ると，従来手法のトラフィック量は提案手法の7倍以上になっており，提案手法の有効性は明らかである．

図9は，移動体数を変化させたときに総トラフィックに占めるトリガ情報の割合を示している．従来手法では，かなりホスト数の少ない段階からトラフィック量の大部分をトリガ情報が占めており，異常動作検出のために大量のトラフィックを発生させていることがわかる．RSパスのマージを用いない手法では，従来手法に比べて大幅に効率が改善されているが，ホスト数100の場合でトラフィックの4割程度をトリガ情報が占めており，さらにホスト数が増大したときに対応できなくなる恐れがある．RSパスのマージを用いる手法では，常に総トラフィックの1割以下となっており，本手法を用いることで発生するトラフィック量がシステム全体にそれほど影響を与えていないことがわかる．

7　おわりに

本研究では，AMDSにおける異常動作検出手法である直前検出手法とその評価について述べた．本手法を用いることで，移動体計算環境においてもECAルールの異常動作が検出でき，AMDSをより安全に運用できるようになる．また，シミュレーション評価により，本手法が従来手法に比べて大幅にトラフィック量を削減でき，本手法がシステム全体のトラフィックに与える影響が小さいことを確認した．今後の課題としては，さまざまなタイプのアプリケーションにおける本手法の効率の評価および実機での実測評価が挙げられる．

謝辞

本研究は，日本学術振興会基盤研究(B)(2) (12480095)，奨励研究(A) (13780331) および日本学術振興会未来開拓学術研究推進事業における研究プロジェクト「マルチメディア・コンテンツの高次処理の研究」(Project No. JSPS-RFTF97P00501) の研究助成によるものである．ここに記して深謝の意を表す．

参考文献

[1] A. P. Karadimce and S. D. Urban, “Refined Trigger Graphs: A Logic-Based Approach to Termination Analysis in an Active Object-Oriented Database,” *ICDE’96*, pp. 384–391 (1996).

[2] 石川 博, “アクティブデータベース,” 情報処理, vol. 35, no. 2, pp. 120–129 (1994).

[3] 村瀬 亨, 塚本昌彦, 西尾章治郎, “移動体環境におけるアクティブデータベースの安全性について,” 情報処理学会研究報告 96-DBS-106, vol. 96, no. 11, pp. 33–40 (1996).

[4] T. Murase, M. Tsukamoto, and S. Nishio, “Active Mobile Database Systems for Mobile Computing Environments,” *IEICE Trans. Info. and Syst.*, pp.427–433, Vol. E81-D, Vol.5 (1998).

[5] S. Ceri and J. Widom, “Deriving Production Rules for Constraint Maintenance,” *16th VLDB Conf*, pp. 566–577 (1990).

[6] S. Y. Lee, T. W. Ling, “A Path Removing Technique for Detecting Trigger Termination,” *EDBT*, pp. 341–355 (1998).

[7] 寺田 努, 莫 君, 村瀬 亨, 塚本昌彦, 西尾章治郎, “移動体計算環境におけるアクティブデータベースのECAルール実行監視機構の設計と実装,” 情報処理学会研究報告 99-DBS-119，Vol. 99，No. 7，pp. 369–374 (1999).

[8] 寺田 努，塚本昌彦，西尾章治郎，“移動体計算環境におけるアクティブデータベースの動的トリガグラフ構築手法,” 情報処理学会研究報告 2000-DBS-122, Vol. 2000，No. 69，pp. 191–198 (2000).

[9] T. Murase, M. Tsukamoto, and S. Nishio, “A system Platform for Mobile Computing base on Active Database,” *International Symposium on Cooperative Database Systems for Adavanced Applications*, vol. 2, pp. 424–427 (1996).