B-035

モバイルエージェントシステム AgentSphere の開発
―分散ハッシュテーブルを用いたエージェント検索機能の実現―
Development of Strong Migration Mobile Agent System AgentSphere
- Implementation of Agent Retrieval Function Using Distributed Hash Table -

黒崎 信清† 鈴木 幸祐† 長塩 征幸‡ 甲斐 宗徳†
Nobukiyo Kurosaki Kousuke Suzuki Masayuki Nagashio Munenori Kai

1. はじめに
1.1 研究概要

分散処理システムは構成しているノードに障害が発生しても処理を続けられるような耐故障性、構成するノードを柔軟に変更できるスケーラビリティ、複数マシンで処理を行うことによる処理速度の向上等様々な利点を見込むことができる。しかしこの利点を十分に発揮するためには、複数マシンを適切に扱うためのアルゴリズム、並列分散処理技術、ネットワークに対する高度な知識や各ノードの状態の把握等が要求される。そこで我々は、インフラとしてのネットワークを準備するだけでユーザが分散処理システムに関する知識、経験が十分に無くてもその恩恵を受けられるようなプラットフォーム、AgentSphere[1]の開発行ってきた。

AgentSphere は自律性を持ったモバイルエージェントシステムを採用している。その理由は、プログラムが稼働しているマシンが AgentSphere で構成されるネットワークから離脱した場合でも、他のマシンにエージェントが移動することで処理を続けられるという利点が存在しているからである。

1.2 現状の課題

AgentSphere ではマシンの参入･離脱の自動検知や性能情報の収集を AgentSphere 間のブロードキャストにより行うので、ブロードキャスト範囲外のマシンは参加不可能となる。また、マシン情報の更新に関しても同様にブロードキャストを使用している。このようにブロードキャストを多用したシステムとなっているために、大量のマシンを接続した場合にネットワークへ負荷がかかる。

また、従来は移動を行うエージェント同士のメッセージングを行う際には、全ノードを巡回するエージェントを生成し、全エージェントを把握するという方法を取っている。しかし、この方法では自律的なエージェントの移動を正しく把握できずに、メッセージングが適切に行えない場合も出てくることが問題となっている。

1.3 本研究の目的

1.2 で挙げた問題の根源は､本システムにおける基礎的なネットワーク機構がサポートすべき機能が十分ではなかったことに起因している。

†成蹊大学理工学研究科理工学専攻 Graduate School of Science and Technology, Seikei University
‡成蹊大学理工学部情報科学科 Department of Computer and Information Science, Seikei University

本システムのネットワークは､AgentSphere 同士を繋ぎ、エージェントが自律的に動作するための基盤とすることが目的である。現在、本研究ではエージェントの自律性を「移動するかしないかの判断を下し、移動する場合には自動で最適な移動先マシンを選定すること」としている。そのため、この機能を実現するためには、自分以外のマシンの状況、エージェントの存在情報の把握が必要となってくる。このような「自分のノードだけではわからない情報」を知るための機構が要求される。このような要求をカバーするために、筆者らはこれまで AgentSphere 間で巡回するエージェントを生成して必要なデータを各ノードに配布する方法や、ブロードキャストで一括送信する方法をとってきた。しかし、これらの方法の場合ノードが「自分以外のノード」について判断材料とする情報は、あくまでそのノードが保持しているデータとなる。本来ならば、必要なデータを必要なときに問い合わせを行い、検索をしたうえで情報を取得し、その情報を使うといった方が自然である。そこで、このような状態に近づけるために、本研究では P2P 技術の一つである､分散ハッシュテーブルを採用することにした。以降では採用した分散ハッシュテーブルについての説明を行う。

2. AgentSphere ネットワークにおける情報共有
2.1 Mercury

本研究では、前項で記した要求をカバーするため、Mercury[2]アルゴリズムを使用することにした。分散ハッシュテーブルとは、P2P 技術の一つで、ネットワークを介して複数のノード間でデータを共有する技術である。特性としては高速に必要とするデータへたどり着けること、データをネットワークに参加しているノード全体で管理することが挙げられる｡本研究で用いる Mercury は、通常の分散ハッシュテーブルアルゴリズムでは不可能な、データ検索の際に複数属性、範囲を指定した検索を可能としたアルゴリズムである。

2.2 Mercury 採用の理由

Mercury を採用した理由は、ネットワークに参加しているマシンが自由にネットワークに存在するデータの検索ができるという点にある。この機能を AgentSphere ネットワークに適応することによって、これまではブロードキャスト等を使いネットワークに存在している全マシンに送信していたマシン情報のデータや、全ノードを逐

Copyright © 2013 by Information Processing Society of Japan and
The Institute of Electronics, Information and Communication Engineers
All rights reserved.

ーチェックし、どのノードにどのエージェントが存在しているかを探していた従来の方法と違い、目的となるエージェントを検索することが可能となる。さらに、データの検索方法に関しても、通常の Key Mapping 方式（一般的な分散ハッシュテーブルの方式）ではデータは負荷分散のために集合とは関係なく配置され、連続するデータを操作する際に不都合を生じる。エージェントはプログラムであるので､そこで使うデータには関連性が生じ、ある程度データはまとめて管理できるようにする必要がある。そこで、ある程度の効率的なアクセス方法を提供できる Mercury を採用した。これによって、分散環境におけるデータの検索システムが AgentSphere に実装されることになり、問題となっていた移動するエージェントの把握や、マシンの情報等を適切に扱うことが可能となる。

2.3 Mercury の概要

Mercury は前述の通り、分散ハッシュテーブルの一種である。複数属性の検索を可能とするために、ネットワークに検索の Key となる属性ごとに Hub というノードの集合を形成する。Hub は図 1 に示すようにリング状に構成されている。

Hub は他の Hub にアクセスするためのリンクを持ち、他属性の検索の際に使用する。さらに、リング以外に確率密度関数 F(d) = 1/d(log(n)) にしたがって生成されるリンクを持つ。これは、リングをたどる際に一つ一つ回る手間を省くために飛び地のリンクとして使用するためのものである。この確率密度関数によって全ノードが同数のリンクを張ったネットワークはある点からある点までの移動が O(log(n)^2) となる特性を持つ。このようなネットワークで構成されるノード群に対して、保存するデータの担当範囲を割り振り、データを保存する。これが Mercury の概要である。

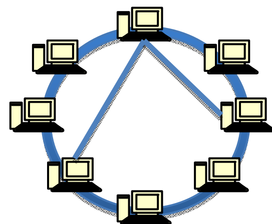

図 1 Hub のリンク

2.4 Mercury のデータ管理

Mercury では、その他の分散ハッシュテーブルと同様に、基本的に保存するデータ（ネットワーク上で共有したいデータ）はそのネットワークに参加しているノード 1 つに保存することとなり、そのデータへのアクセスの到達性を保証するような構造をつくり上げる。

2.3 項で触れた Hub が Mercury においてこのデータへの到達性を保証する重要な要因となっている。図 1 で分かるように全てのノードはリング状に構成される。このリングを一つ一つ渡り歩けば結果的に全ノードにアクセスすることができるので、データへの到達保証はここで満たされることになる。

データへの到達性は以上のように確保されたが、このままでは目的のノードへのアクセスに時間がかかる。そこで 2.3 項で説明した、ある確率密度関数にしたがって生成されるリンクを全ノードに張ることで、アクセススピードを向上させることができる。Mercury では、このようにしてアクセス精度を向上させた上で、データの共有を行う。先に述べた様に基本的には 1 つのデータをどこかの 1 つノードに保存する。また、Mercury ではデータの範囲検索を効率的に実現するために、リング状のネットワークに対して、順番にデータの担当範囲を割り当て、その担当範囲内のデータに対して保存、取得要求が発生した場合に、担当範囲のノードに対してそのデータの保存、取得の操作を行う。

2.5 Mercury の補完

Mercury では、データ管理の手法に焦点が当てられており、ノードに障害が発生した際のデータ復旧の手順、アルゴリズム等に関しては規定されていない。我々のシステムでは、自由にノードがネットワーク上に着脱可能な状態を目指しているので、ノードに障害が発生した際のデータの復旧システムの構築は必至となる。そこで、一番基本的な分散ハッシュテーブルである Chord[3]のバックアップ手順を参考とした。（図 2）

このように、リング状に構成されているネットワークにおいて、自分に直結しているノードのうち、先のノードを Successor、前のノードを Predecessor という。この Successor と Predecessor がそれぞれ自分のデータのバックアップを取るという方法が、Chord では用いられている。実装の際にはその次のノード（Successor の Successor や Predecessor の Predecessor）と数ノードにまたがってこのバックアップをとっていくこととなる。なお、本システムでは何ノード先まで自己のバックアップを保存しておくかはオーバーヘッドとのトレードオフとなるため、ユーザが設定できるようにしてある。

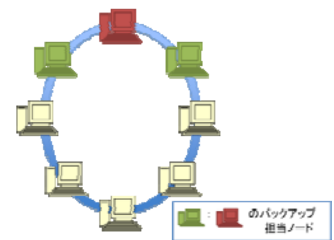

図 2 Chord のバックアップシステム

3. Mercury の改良

本研究では､AgentSphere で Mercury を使用する際に改良を加えた。

3.1 2 層化

Mercury ではネットワークにノードが参加すると、参加を申請したノードから Hub というノードの集合の一部範囲の担当を受け渡され、リング状のネットワークの一部となることでネットワークの参加が完了することとなる。しかし、この方法の場合実際に稼働すると不都合が生じる場合が存在する。

図 3 のように、データ数がノード数と同等の場合は 1 データを 1 ノードで担当することとなる Mercury では O(log(n)^2)で検索ができるが、正

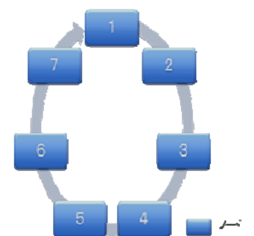

図 3 整数値 1 ～ 7 を 7 ノードで管理する場合

Copyright © 2013 by Information Processing Society of Japan and
The Institute of Electronics, Information and Communication Engineers
All rights reserved.

確には O(log(n)^2)回の通信が発生することとなる。従ってノードと比較して保持するデータが多くない場合には Mercury には不得意なケースとなる。このような状況は、AgentSphere で Mercury を運用すると容易に発生することが想定される。具体例としては、AgentSphere に接続されている各ノードのマシンの状態の情報が挙げられる。これは、エージェントが自動的に移動先マシンを選定するために必須となる情報である。よってネットワークを介して共有したい情報の中でも特に重要となる情報だが、基本的にその情報の数は 1 ノードにつき一つあれば済む。つまり、図 3 に挙げた例がそのまま起きることとなり、Mercury の特性上好ましくない。また、ネットワークを構成した直後にも保存されているデータは存在しないので、Hub が構成できないといった状態も発生することとなる。しかし、本システムにとって Mercury の検索方法は 1.4 項で挙げた必要要件を満たすものであり、検索の手段として見れば有用である。そこで本研究では、Mercury のデータ検索方法はそのままに、ネットワークの参加手順を変更し、リング上の Mercury ネットワークを構築する前段階として、ログインサーバがスーパーノードとして機能し、その後負荷の増加に伴ってサブノードからスーパーノードとなるノードを抽出してスーパーノード同士で Mercury アルゴリズムのネットワークを構築するという 2 層化構造を実現した。

3.2 動的な Hub の追加

複数属性の範囲検索機能は Mercury の特性である。しかし、検索する属性の追加をしようとすると、Java の場合、仕様上ノード同士で通信する際にそれぞれのノードにとって既知の属性である必要がある。既知クラス以外のクラスをロードする手順は図 4 のようになる。

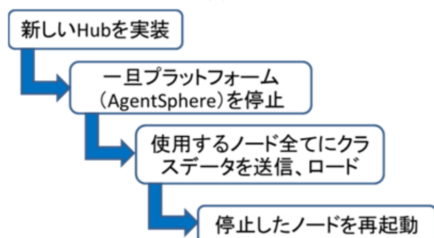

図 4 未知クラスのロード手順

この手順を無視してクラスを読み込もうとした場合にはクラスが解決できずエラーが発生してしまう。これを解消するためには、新たに検索属性、範囲を追加する際にその属性を扱うノードを全てシャットダウンし、それぞれのノードに対してクラスを理解させる手順が必要となる。だがこれではシステムとしての柔軟性に欠け、現実的ではない。そこで、AgentSphere の特性の一つである、システムの動的更新機能を利用し、動的な Hub の追加機能を実現した。

図 5 のように、ユーザは Hub を作成し、（これがコンパイルされ、AgentSphere においての未知クラスとなる）AgentSphere 付属の Sphere Shell を使い未知クラスを解決できるクラスローダ[4]を読み込む。このような手順を踏むことで、図 4 のような煩雑な手順を踏むこと無く未知クラスをロードし、さらには他マシンにそのクラスを転送しても問題なく稼働できるようになる。

図 5 動的な Hub の追加機能

3.3 2 層化 Mercury の AgentSphere への適応

今回作成したネットワークを AgentSphere へ適応することによって、AgentSphere、エージェントどちらも使用できる情報共有機構が構築されることとなった。この機構を使うことによって、これまでの方式とは違い「必要なときに必要なデータ」を問い合わせ、検索を行える環境が整った。一例としては、これまでブロードキャストで全マシンに対して各自のノードが送信していたマシンの情報を、この機構を使うことによって、情報検索、共有のための Mercury ネットワークに対して保存要求を行うだけで共有することができるようになった。

4 移動を行うエージェントの位置情報の共有

本研究の自律型エージェントは､自律的に移動を行い､さらにはその移動を行うエージェント自身が他のエージェントに対してメッセージを送信することにより、協調動作を実現することができる。それにより、分散処理システムとしての性能をより向上させることをコンセプトの一つとしている。しかし、それぞれがユーザの管理下を離れ自律的に移動するエージェントの位置を把握し続けることは困難である。さらにそれが稼働するエージェントの数が増加した場合にはなおさらである。これまでの AgentSphere における解決方法は、1.4 項で説明を行ったように、全てのマシンがそれぞれに自分の情報をネットワークに存在しているマシン全てに送信するというものであった。これまで説明してきた通り、本研究ではこの手法を改善し、より効率的に複数マシンでの情報の共有を行うために、分散ハッシュテーブルを基盤として新しいネットワーク機構を導入した。移動を行うエージェント自身も、共有されるべきデータであると考えると今回構築したシステムを使うことにより、存在位置の把握が可能となるからである。

4.1 エージェントの位置情報の必要性

エージェント自身もデータであり、エージェント名やエージェント ID 等のエージェント自身のデータをそのまま Mercury ネットワークに対して投げ、共有することも可能である。しかし、この方法ではエージェント、もしくはエージェントが存在しているマシン（以降ではエージェントとしてまとめて記述）が能動的にデータを保存した時の情報しか共有されないこととなる。つまり、エージェントが存在しており、かつそのエージェントが

Copyright © 2013 by Information Processing Society of Japan and
The Institute of Electronics, Information and Communication Engineers
All rights reserved.

意図的に自身の存在位置を共有した場合の情報のみ共有されることとなる。これは、エージェントが正常に動作している場合には問題なく稼働することとなるが、エージェントに何かしらの障害が発生した場合に問題が起こる。分散処理システムの特性として、耐故障性が高いことはよく知られている。AgentSphere もこの特性をユーザがより簡単に享受できる環境の構築を目指しており、エージェントの存在位置を把握することでエージェントの停止、生存の判定を行う必要があった。そこで、本研究ではエージェントの存在位置、さらには生存情報を確認できるようなシステムを構築した。

4.2 エージェントの位置情報を共有するための手法

今回作成した Mercury ネットワークを利用することで、基本的なデータ共有の仕組みは整った。そこで、エージェント検索用にエージェントの情報を保持する Hub(Agent Hub)を新たに作成することとした。この割り振りに関しては、Mercury で情報を保存する際のアルゴリズムを転用することでノードに障害が発生した場合に関しても堅牢性を維持できることとなる。

ここで取り扱うデータの情報は下記とする。

・エージェントの名称
・各エージェントが固有に持つ ID（エージェント ID）
・エージェントが作成された IP アドレス
・エージェントが現在滞在している IP アドレス（CurrentIP）

また、ここではエージェントの送信側がデータを送信することでデータを共有することとなるが、エージェントの生存確認も兼ねることを目的としているので、ここで作成する機構はエージェントの存在情報（保存されるデータ）とデータを管理するノードそれぞれがお互いに相互確認を行うこととする。

通常の Mercury のデータ管理システムである、データ管理側へのアクセスだけの場合には、データ管理側ノードがエージェントの移動を監視することができるが、逆にいうとそれ以上のことはできない。そこで、データを管理するノードからもデータ（この場合はエージェント）の状態を確認することで、データ管理ノードの障害発生、エージェントの障害発生等が検知できるようになる。

エージェントの生存確認を行うノードとして割り当てられたノードは、Mercury のアルゴリズムにしたがって割り当てられることになるので、Mercury のデータ検索アルゴリズムにしたがって自由に検索可能となる。つまり、ネットワークに参加しているノードから特定のエージェントを管理しているノードへ通信を行うことが可能となるのである。

さらに、このノードがエージェントの存在位置を把握しているので、特定のエージェントの検索機能が構築できたこととなる。

このようなエージェントの検索機能が実現できるようになったため、移動を行うエージェント同士がメッセージングを行う際に、すべてのノードを総当たりで探すことがなくなり、エージェント間通信がより円滑に行われるようになった。

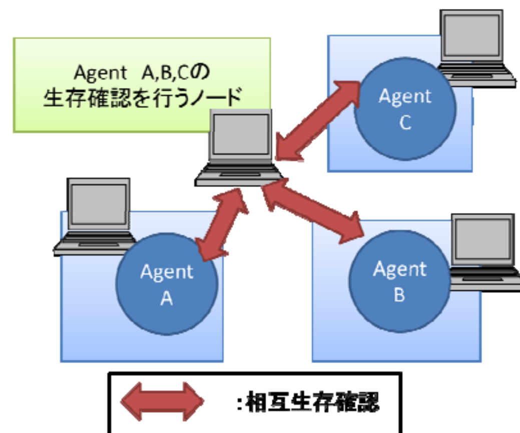

図 6　エージェント情報共有への **Mercury** の適用

5. 評価

今回作成した情報共有機構が正常に動作しているのか確認を行った。まずは、データを保存するための MercuryHub の作成を行う。ここでは、整数値１～４００を Key とする Hub を作成する。（図 7）

```
Range OF HUB : Areas of responsivility ::upper bound is 400: lower bound is 50:
```

図 7　**Hub** の作成

次に、実際に保存するデータを保存する。この例の場合保存するデータは一つだけであり、Key の値は 100 となっている。（図 8）

```
[Integer, Areas of responsivility ::upper bound is 100: lower bound is 100, tequila.data.KeyValuePair@7e5a9de6]
```

図 **8** データの保存

その後、保存したデータを取得する。そのために、取得する範囲をここで指定している。（図 9）

```
Trying to get Entry Areas of responsivility ::upper bound is 100: lower bound is 100
```

図 **9** データの取得

さらに、取得したデータが正確かどうかをチェックしているのが図 10 である。

```
ENTRY ARRIVED! integer key is 100
```

図 **10** 取得したデータのチェック

ここまでが、ネットワークが全て正常に動作している場合の動作実験である。この後、データを保存してあるノードを意図的に落とした場合、バックアップデータを適切に取得できるかどうか実験を行った。(図 11)

Copyright © 2013 by Information Processing Society of Japan and
The Institute of Electronics, Information and Communication Engineers
All rights reserved.

```
7406 [RequestHandler_192.168.11.4:5122] DEBUG tequila.nodefunction.impl.RequestHandler – Exception occurred while receiving a request. Maybe socket has been closed.
7406 [RequestHandler_192.168.11.4:5122] INFO tequila.nodefunction.impl.RequestHandler – Disconnecting.
7407 [RequestHandler_192.168.11.4:5122] INFO tequila.nodefunction.impl.RequestHandler – Closing socket Socket[addr=/192.168.11.5, port=51094, localport=5122]
7407 [RequestHandler_192.168.11.4:5122] INFO tequila.nodefunction.impl.RequestHandler – Socket Closed
7407 [RequestHandler_192.168.11.4:5122] DEBUG tequila.nodefunction.impl.RequestHandler – Disconnected.
7409 [SocketProxy_Thread_192.168.11.5:5122] WARN tequila.nodefunction.impl.SocketProxy – Could not read response from stream!
java.io.EOFException
  at java.io.ObjectInputStream$BlockDataInputStream.peekByte
  (ObjectInputStream.java:2553)
  at java.io.ObjectInputStream.readObject0
  (ObjectInputStream.java:1296)
  at java.io.ObjectInputStream.readObject
  (ObjectInputStream.java:350)
  at tequila.nodefunction.impl.SocketProxy.run
  (SocketProxy.java:340)
  at java.lang.Thread.run
  (Thread.java:662)
7410 [SocketProxy_Thread_192.168.11.5:5122] INFO tequila.nodefunction.impl.SocketProxy – Connection broken down!
```

図 **11** データを保存しているノードが切断した場合

このように、コネクションが切断されたため、エクセプションが投げられている。

しかし、バックアップデータを別のノードが保持しているため、問題なく取得が完了した。(図 12)

```
ENTRY ARRIVED! integer key is 100
```

図 **12** データを保存しているノードが切断した場合の実行結果

次に、エージェントの様子を視覚的に確認することが出来る AgentMonitor というアプリケーションを使用してエージェントの検索機能が正常に動作しているかどうかの確認を行った。

エージェントの検索方法はエージェントごとに割り当てられているエージェント ID による検索と、エージェントの種類による検索がある。

エージェント ID による検索では調べたいエージェント ID を入力し、該当するエージェントがあれば緑色の円として表示される。(図 13)

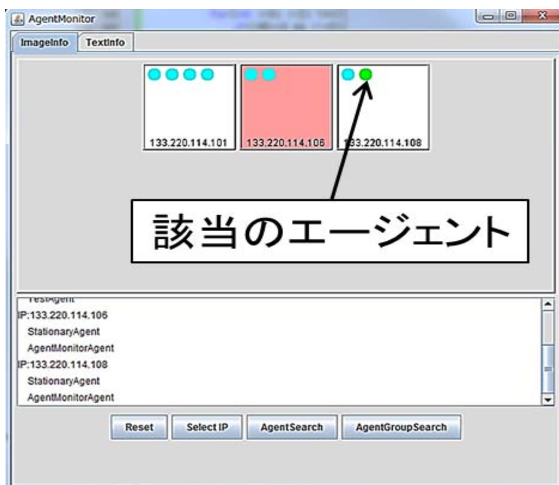

図 **13** エージェント **ID** による検索結果

エージェントの種類による検索の場合も同様に、検索したいエージェントの種類を入力し、該当するエージェントがあれば青色の円として表示される。（図 14）

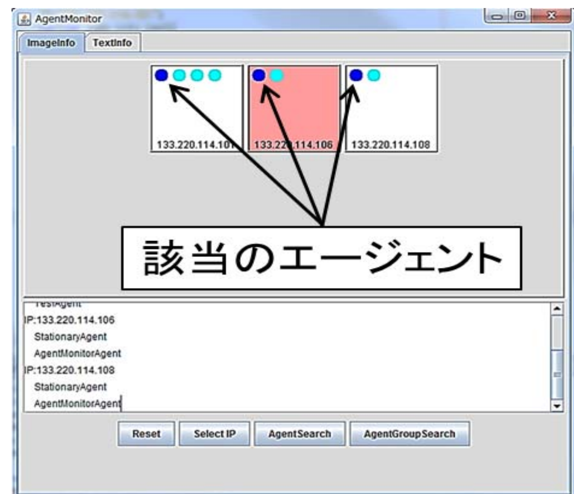

図 **14** エージェントの種類による検索結果

6. まとめ

今回の研究で、AgentSphere 間のネットワークを介したデータの共有方法を分散ハッシュテーブルの一種である Mercury を使うことで改めた。その際に、AgentSphere にとって適した形での導入となるように改良を行い、Mercury の弱点であった保存するデータが少数であった場合のデータ共有の仕組みや、検索用のモジュールである Hub を動的に追加する仕組みを整えた。

この結果、複数ノードで効率的にデータ共有することが可能となった。

また、これまでの研究で解決が急務となっていた、それぞれが自律的に移動を行うエージェントの位置情報の共有という問題に対して、それぞれのエージェントの位置情報を Mercury ネットワークを用いて全ノードで分担して管理できる体制が整った。

本研究の結果、AgentSphere においてエージェントが自律的に稼働し、またその稼働に合わせたメッセージングを行う機構が整ったと考えている。

参考文献

[1] 赤井雄樹・横内　貴・若尾一晃・甲斐宗徳「強マイグレーションモバイルエージェントシステム AgentSphere の開発」,FIT2009, B-011,Sep.2009

[2] Ashwin R. Bharambe, Mukesh Agrawal, Srinivasan Seshan “Mercury: Supporting Scalable Multi-Attribute Range, SIGCOMM 2004”.

[3] I Stoica, R Morris, D Karger, M Kaashoek, H Balakrishnan. “Chord: A scalable peer-to-peer lookup service for Internet applications”, in Proc. ACM SIGCOMM 2001.

[4] Daisuke YAMAGUCHI, Yuuki AKAI, Munenori KAI. ” Design and Implementation of Serializing Method for Non-procedural Object Transfer between JavaVMs”:Proc. of IEEE PACRIM’11 on Communications , Computers and Signal Processing , 2011

Copyright © 2013 by Information Processing Society of Japan and
The Institute of Electronics, Information and Communication Engineers
All rights reserved.