成蹊大学理工学研究報告
J. Fac. Sci.Tech., Seikei Univ.
Vol.50 No.1 (2013)pp.29-36

AgentSphereにおける分散ハッシュテーブルを用いた情報共有機構

鈴木　幸祐*1，甲斐　宗徳*2

An Information Sharing Mechanism using Distributed Hash Table for AgentSphere

Kousuke SUZUKI*1, Munenori KAI*2

ABSTRACT：In recent years, parallel and distributed processing technologies become more and more required. Since parallel and distributed processing systems are, however, very complicated, the users require many knowledge and experience of the target systems in order to get the performance benefit. Therefore we have been developing AgentSphere as a parallel and distributed processing system to reduce the difficulties of the use and to get the performance benefit easily. In this paper, we propose its information sharing mechanism using distributed hash table, which is one of the P2P technologies, and implement the mechanism to share and retrieve the information of each mobile agents which move autonomically in AgentSphere network.

Keywords：Mobile agent system, Distributed hash table

(Received April 8, 2013)

1．はじめに

1.1　研究概要

分散処理システムは，物流，公共分野等様々な場面で普及，応用が進んでいる。これらのシステムは同時に複数のマシンで個別の処理を行い，さらに全体として見ると一つのシステムとして稼働するものである。このような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，負荷分散の実現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，並列処理技術，ネットワークに対する知識等の高度な知識が要求される。これは分散処理システムを導入する上でのハードルとなっている。そこで我々は，この問題を解決すべく，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の開発を行ってきた。

1.2　AgentSphere

様々な分散処理の体系が存在しているが，我々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た。それは，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。自律性に関しては当面の目標として，自動的にネットワークに参加しているマシンを感知し，そのマシン性能の収集を行い，そしてその情報を元に稼働しているプログラムにとって最適なマシンを自動的に選定することを目的としている。

1.3　現状の課題

AgentSphereでは上述のマシン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

*1：理工学研究科理工学専攻博士前期学生
*2：理工学研究科理工学専攻教授 (kai@st.seikei.ac.jp)

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。

2．本研究の目的

1.3 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を未だサポートできていないことに起因している。

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。本研究では，エージェントの自律を「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，これまでのAgentSphereの研究では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がとられてきたのである。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。

3．本研究で開発するネットワーク

3.1　Mercury

本研究では，前項で記した要求をカバーするため，Mercury[1]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。

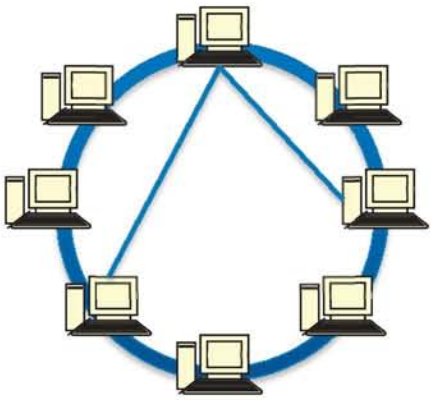

図1　Hubのリンク構成

3.2　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，全ノードを逐一チェックし，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。

3.3　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。

3.4　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。

3.3 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，データへの到達保障性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3.3 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，その担当のノードに保存，取得の操作を行う。

3.5　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離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のバックアップ手順を参考とした。（図２）

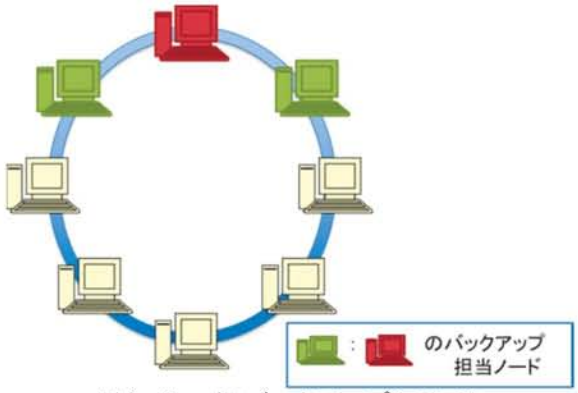

図２　Chordのバックアップシステム

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ゆ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。

４．　Mercuryの改良

本研究では，Mercuryをベースとして，小規模なノード群でAgentSphereを使用する際に備えて改良を加えた。

4.1　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というデータ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際に不都合が生じる場合が存在している。（図３）

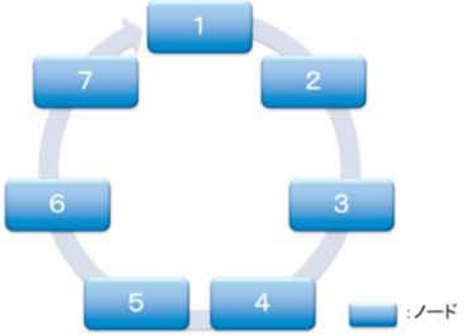

図３　整数値１～７を７ノードで管理する場合

この例のように，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で運用した場合に容易に発生することが想定される。具体例としては，AgentSphereに接続されているマシンの情報が挙げられる。これは，各ノードのマシンの状態を見るための物であり，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得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２章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し，2層化構造を実現する。

4.2　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。

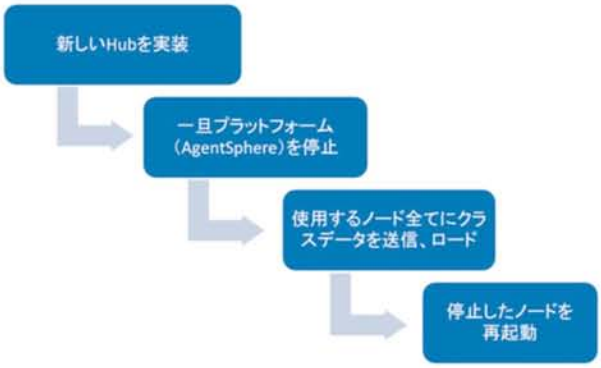

図４　未知クラスのロード手順

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）

図５　動的なHubの追加機能

このように，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Shellを使い未知クラスを解決できるクラスローダを読み込む。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するようになる。このクラスローダに関しては論文"Design and Implementation of Serializing Method for Non-procedural Object Transfer between JavaVMs":2011 Pacrim" [4]を参照されたい。

4.3　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，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。

５．移動を行うエージェントの存在位置の把握

本研究の自律型エージェントは，自動的に移動を行い，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で，協調動作を実現することで，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，2 章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で，存在位置の把握が可能となるからである。

5.1　要求と目的

エージェント自身もデータであり，こ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として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。

5.2　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ている。そこで，ノードに管理するエージェントを割り振り，分散してエージェントを管理することとした。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。

また，本来はデータを送信する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，図３の状況だと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，過去の研究で問題となっていた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存在位置がわからないという問題が解決できた。

6．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う。まずは，データを保存するための

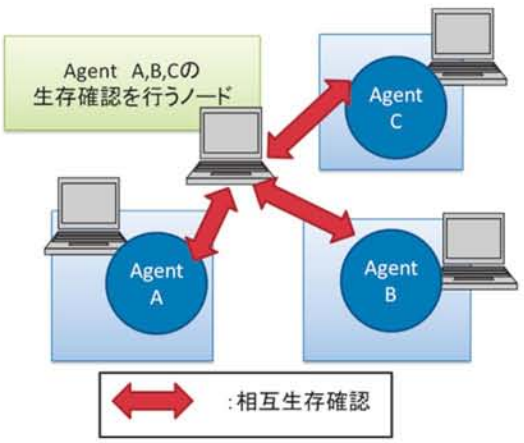

図 6　エージェント情報共有へのMercuryの適用

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)

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 8：①)

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 9：①)

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（**エラー! 参照元が見つかりません。**）である。これは，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 11:①)

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 12：②)

7．まとめ

今回の研究で，AgentSphereの問題点であった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なデータを共有が可能となった。

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して移動を行うエージェントの存在位置の把握という問題に対して，それぞれのエージェントの生存管理を行うノードをネットワーク状に配置することで，それぞれのエージェントの存在位置をネットワーク全体と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。今後の課題としては，これまで構築をおこなってきたシステムを最大限活用するためのエージェントの自律挙動の調査，実際にユーザが使用するための，ユーザ補助機能の実現等が挙げられる。

謝辞

本研究は科研費（基盤研究(C)21500041）の助成を受けたものであることをここに記し，謝意を表します。

参考文献

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

[2] Gurmeet Singh Manku, Mayank Bawa,Prabhaker Raghavan
“Symphony: Distributed Hashing In A Small World”, USITS 2003 Proceedings of the 4th conference on USENIX Symposium on Internet Technologies and Systems”.

図 7　Hubの作成

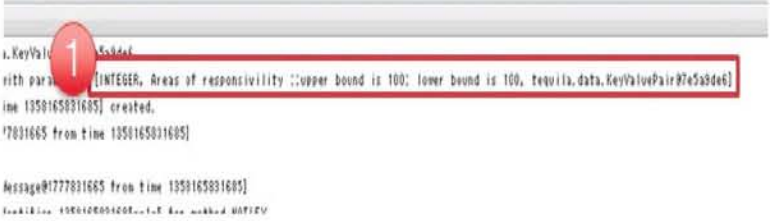

図 8　データの保存

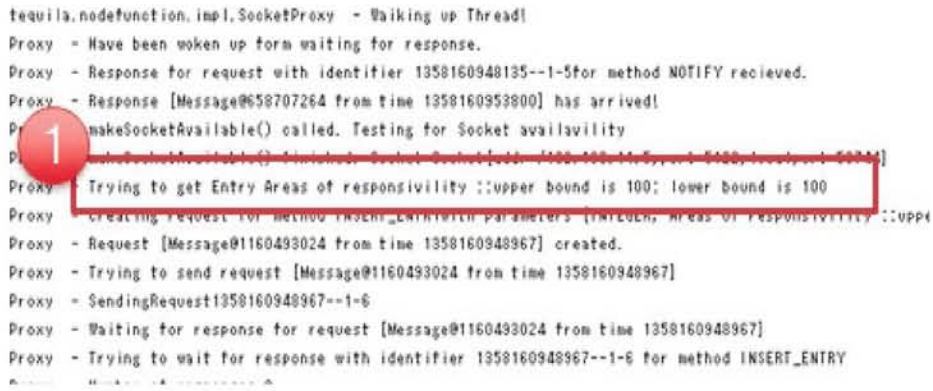

図 9　データの取得

図 10　取得したデータのチェック

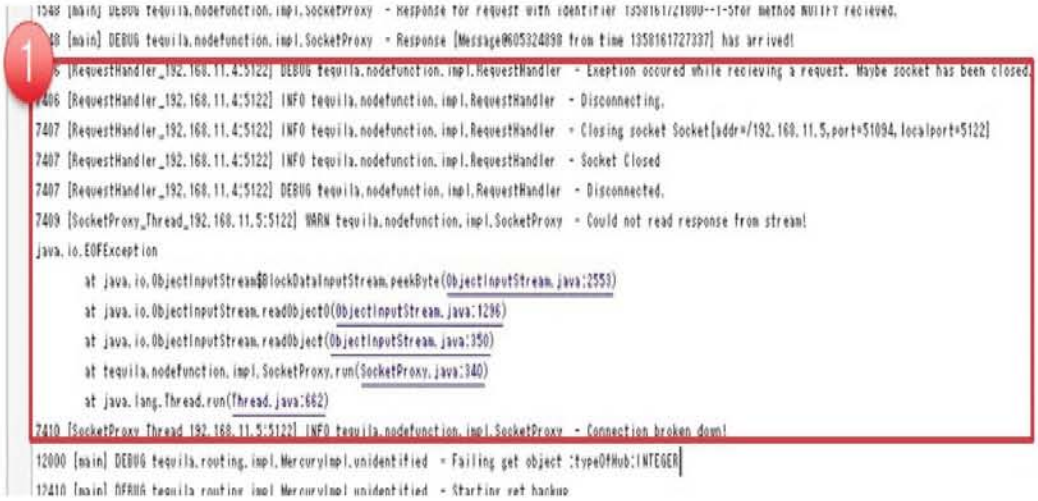

図 11　データを保存しているノードを切断した場合

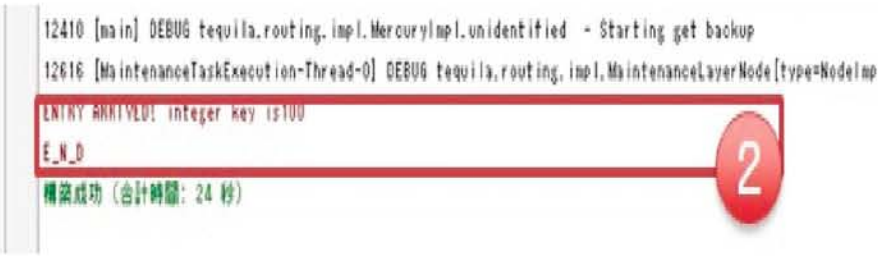

図 12　データを保存しているノードが切断した場合の実行結果