(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号
特許第5984355号
(P5984355)
(45)発行日　平成28年9月6日(2016.9.6)　(24)登録日　平成28年8月12日(2016.8.12)

(51)Int.Cl.　　F I
G06F 12/00 (2006.01)　　G06F 12/00 513J
G06F 17/30 (2006.01)　　G06F 17/30 110C
　　G06F 12/00 547Q

請求項の数 30　(全 42 頁)

(21)出願番号　特願2011-229280 (P2011-229280)
(22)出願日　平成23年10月18日 (2011.10.18)
(65)公開番号　特開2012-108889 (P2012-108889A)
(43)公開日　平成24年6月7日 (2012.6.7)
審査請求日　平成26年10月16日 (2014.10.16)
(31)優先権主張番号　特願2010-237102 (P2010-237102)
(32)優先日　平成22年10月22日 (2010.10.22)
(33)優先権主張国　日本国(JP)

(73)特許権者　391021710
株式会社インテック
富山県富山市牛島新町５番５号
(74)代理人　230104019
弁護士　大野　聖二
(74)代理人　100105038
弁理士　田中　久子
(74)代理人　100131451
弁理士　津田　理
(72)発明者　中川　郁夫
東京都江東区新砂１－３－３　株式会社インテック　先端技術研究所内

審査官　打出　義尚

最終頁に続く

(54)【発明の名称】分散型データベースシステムおよび分散型データ処理システム

(57)【特許請求の範囲】
【請求項１】
複数のデータベースノードと一つ以上のクライアントノードとを備え、前記複数のデータベースノードに複数のデータが分散してストアされる分散型データベースシステムであって、
前記クライアントノードは、
オブジェクト指向型プログラミングにおけるオブジェクトであるデータであって任意の構造を有することが可能なデータを、キー情報とペアにして、前記複数のデータベースノードのうち前記キー情報に基づいて特定されるデータベースノードへ送信し、該データベースノードにストアさせる手段と、
オブジェクト指向型プログラミングにおけるオブジェクトであるプログラムによって前記複数のデータベースノードのいずれかにキー情報とペアでストアされている前記データに対する処理を行うプログラム情報を、該キー情報とペアにして、前記複数のデータベースノードのうち前記キー情報に基づいて特定されるデータベースノードへ送信する手段とを備え、
前記プログラム情報は、前記プログラムのオブジェクトが属するクラスを示す情報と、該オブジェクトで用いられる変数であって任意の構造を有することが可能な変数の値の情報とを含むものであり、前記プログラムの定義である前記クラスの情報は、前記クライアントノードと前記データベースノードとで共有可能なものであり、
前記データベースノードは、

前記クライアントノードから送信されたキー情報とペアでストアされている前記データのオブジェクトに対して、前記クライアントノードから送信された前記プログラム情報により定められる前記プログラムのオブジェクトを実行することにより、前記データベースノードで前記データに対する処理を行う手段と、
前記処理を行った結果の情報を、前記クライアントノードへ送信する手段とを備えることを特徴とする分散型データベースシステム。
【請求項２】
前記データベースノードは、各クラスで実行されるメソッドの内容と用いられる変数の定義とを含むクラスライブラリに関する情報を記憶する手段をさらに備え、
前記データに対する処理は、前記クラスライブラリに関する情報を用いて前記プログラムのオブジェクトを実行することにより行われるものである、請求項１に記載の分散型データベースシステム。
【請求項３】
前記クライアントノードは、前記クライアントノードから送信された前記プログラム情報により定められる前記プログラムのオブジェクトを前記データベースノードが実行するのに用いられる前記クラスの情報を作成する手段をさらに備える、請求項１又は２に記載の分散型データベースシステム。
【請求項４】
前記クライアントノードは、前記プログラム情報を作成する手段をさらに備える、請求項１～３のいずれかに記載の分散型データベースシステム。
【請求項５】
前記データに対する処理が、該データに新たなデータを追記する処理である場合、
前記クライアントノードから送信されるプログラム情報は、追記する処理の指示と、前記新たなデータとを含み、
前記データベースノードは、
前記クライアントノードから送信されたキー情報とペアでストアされているデータを読み出し、該データに前記クライアントノードから送信された新たなデータを加えて得られるデータを前記キー情報とペアでストアする処理を、前記データベースノードで行い、
前記追記の成功を示す通知を、前記処理を行った結果の情報として、前記クライアントノードへ送信するものである、請求項１～４のいずれかに記載の分散型データベースシステム。
【請求項６】
前記データに対する処理が、該データのうち少なくとも一部を新たなデータに書き換える処理である場合、前記クライアントノードから送信されるプログラム情報は、書き換える処理の指示と、書き換えられるべきデータの指示と、新たなデータとを含み、
前記データベースノードは、
前記クライアントノードから送信されたキー情報とペアでストアされているデータを読み出し、該データ中に前記クライアントノードから指示された書き換えられるべきデータがあればそれを新たなデータに置き換えて得られるデータを前記キー情報とペアでストアする処理を、前記データベースノードで行い、
前記書き換えの成功を示す通知を、前記処理を行った結果の情報として、前記クライアントノードへ送信するものである、請求項１～５のいずれかに記載の分散型データベースシステム。
【請求項７】
前記データに対する処理が、該データの一部を削除する処理である場合、
前記クライアントノードから送信されるプログラム情報は、削除する処理の指示と、削除すべきデータの指示とを含み、
前記データベースノードは、
前記クライアントノードから送信されたキー情報とペアでストアされているデータを読み出し、該データ中に前記クライアントノードから指示された削除すべきデータがあれば

それを削って得られるデータを前記キー情報とペアでストアする処理を、前記データベースノードで行い、
前記削除の成功を示す通知を、前記処理を行った結果の情報として、前記クライアントノードへ送信するものである、請求項１～６のいずれかに記載の分散型データベースシステム。
【請求項８】
前記データに対する処理が、該データに含まれるデータ要素を別のデータと比較する処理である場合、
前記クライアントノードから送信されるプログラム情報は、どのデータ要素を比較する処理であるかの指示と、比較対象となる前記別のデータとを含み、
前記データベースノードは、
前記クライアントノードから送信されたキー情報とペアでストアされているデータを読み出し、該データ中の前記クライアントノードから指示されたデータ要素と前記クライアントノードから送信された前記別のデータとを比較する処理を、前記データベースノードで行い、
前記比較により得られた結果を、前記処理を行った結果の情報として、前記クライアントノードへ送信するものである、請求項１～７のいずれかに記載の分散型データベースシステム。
【請求項９】
前記データに対する処理が、該データに含まれる一つ以上のデータ要素を用いて計算を行う処理である場合、
前記クライアントノードから送信されるプログラム情報は、どのデータ要素を用いて何の計算を行う処理であるかの指示を含み、
前記データベースノードは、
前記クライアントノードから送信されたキー情報とペアでストアされているデータを読み出し、該データ中の前記クライアントノードから指示されたデータ要素を用いて前記クライアントノードから指示された計算を行う処理を、前記データベースノードで行い、
前記計算により得られた結果を、前記処理を行った結果の情報として、前記クライアントノードへ送信するものである、請求項１～８のいずれかに記載の分散型データベースシステム。
【請求項１０】
前記クライアントノードは、
複数のデータを対象にして一つの大処理を行うために、キー情報とプログラム情報のペアを複数準備し、各プログラム情報に、前記一つの大処理の前処理となる各データに対する処理の指示を含ませて、前記複数のデータベースノードのうち各キー情報に基づいて特定される一つ以上のデータベースノードへ送信させる手段と、
前記一つ以上のデータベースの各々から、前記各データに対する処理を行った結果の情報を受信し、受信した複数の情報を用いて、前記一つの大処理の結果を生成する手段とをさらに備えるものである、請求項１～９のいずれかに記載の分散型データベースシステム。
【請求項１１】
前記データに対する処理が、該データに含まれるデータ要素を前記クライアントノードが取得するという処理である場合、
前記クライアントノードから送信されるプログラム情報は、どのデータ要素を取得する処理であるかの指示を含み、
前記データベースノードは、
前記クライアントノードから送信されたキー情報とペアでストアされているデータを読み出し、該データ中の前記クライアントノードから指示されたデータ要素を抽出する処理を、前記データベースノードで行い、
前記抽出されたデータ要素を、前記処理を行った結果の情報として、前記クライアント

ノードへ送信するものである、請求項１～１０のいずれかに記載の分散型データベースシステム。
【請求項１２】
前記データに対する処理が、該データを前記クライアントノードが取得するという処理である場合、
前記クライアントノードから送信されるプログラム情報は、取得する処理の指示を含み、
前記データベースノードは、
前記クライアントノードから送信されたキー情報とペアでストアされているデータを読み出す処理を行い、
読み出した前記データを、前記処理を行った結果の情報として、前記クライアントノードへ送信するものである、請求項１～１１のいずれかに記載の分散型データベースシステム。
【請求項１３】
キー情報とデータのペアを、前記複数のデータベースノードのうち前記キー情報に基づいて特定されるデータベースノードにストアさせる手段が、前記キー情報とプログラム情報のペアを、前記クライアントノードから前記データベースノードへ送信することによって実現される場合、
前記クライアントノードから送信されるプログラム情報は、ストアする処理の指示と、ストアすべきデータとを含み、
前記データベースノードは、
前記クライアントノードから送信されたキー情報とペアで、前記クライアントノードから送信されたプログラム情報に含まれるデータをストアする処理を行い、
前記ストアの成功を示す通知を、前記処理を行った結果の情報として、前記クライアントノードへ送信するものである、請求項１～１２のいずれかに記載の分散型データベースシステム。
【請求項１４】
前記クライアントノードから送信されたキー情報とデータのペアをストアしたデータベースノードは、前記複数のデータベースノードのうち自身の代替となり得るデータベースノードに、前記キー情報とデータのペアの複製をストアさせる手段をさらに備える、請求項１～１３のいずれかに記載の分散型データベースシステム。
【請求項１５】
前記データに対する処理は、前記キー情報とペアでストアされるデータの更新を伴うものであるかデータを読み出すだけであるかが区別可能であり、
データを読み出すだけの処理である場合、前記クライアントノードが、前記キー情報とプログラム情報のペアを、前記複数のデータベースノードのうち前記キー情報に基づいて特定されるデータベースノードへ送信する手段は、前記キー情報により指示されるデータベースノード及び当該データベースノードの代替となり得るデータベースノードのうちから選択したデータベースノードへ送信するものである、請求項１４に記載の分散型データベースシステム。
【請求項１６】
前記複数のデータベースノードの各々は、自身の担当する値の範囲を有し、
前記キー情報に基づくデータベースノードの特定は、前記キー情報から算出されたハッシュ値を用いて、いずれのデータベースノードの担当する値の範囲に入るかを判断することにより行われる、請求項１～１５のいずれかに記載の分散型データベースシステム。
【請求項１７】
前記データベースノードには、前記データが、バージョン情報が付されたオブジェクトとしてストアされており、
前記データに対する処理が、前記キー情報とペアでストアされるデータの更新を伴うものである場合、更新されたデータは、前記バージョン情報が更新されたオブジェクトとし

てストアされるものである、請求項１～１６のいずれかに記載の分散型データベースシステム。
【請求項１８】
前記データベースノードは、あるクライアントノードから受信したプログラム情報に従って行っている処理が、前記データの更新を伴う処理である場合、該処理を行っている間に、別のクライアントノードから前記データの更新を指示するプログラム情報を受信したら、該別のクライアントノードに処理不能を通知するか、もしくは、該処理の終了を待って該別のクライアントノードから受け取った更新を指示するプログラムの処理を開始する手段をさらに備える、請求項１～１７のいずれかに記載の分散型データベースシステム。
【請求項１９】
前記クライアントノードから送信されるプログラム情報は、前記プログラムのオブジェクトにより実行されるべきメソッドを示す情報と、該オブジェクトで用いられる変数に関する情報とを含み、
前記クライアントノードは、メソッド呼び出しに応じて、呼び出されたメソッドを定義するクラスの情報を参照することにより、前記プログラム情報を作成するものである、請求項１～１８のいずれかに記載の分散型データベースシステム。
【請求項２０】
前記クライアントノードは、前記メソッド呼び出しの際に実行されるプロキシオブジェクトに、キーの情報を含ませることにより、前記キー情報と前記プログラム情報のペアを作成するものである、請求項１９に記載の分散型データベースシステム。
【請求項２１】
前記分散型データベースシステムは、前記クライアントノードから送信されたプログラム情報に従って前記データベースノードが前記プログラムを実行するのに用いられるクラスの情報を記憶しているストレージ又はサーバをさらに備え、
前記複数のデータベースノードの各々は、前記クラスの情報を前記ストレージ又はサーバから取得する手段を備える、請求項１～２０のいずれかに記載の分散型データベースシステム。
【請求項２２】
前記クライアントノードから送信されるキー情報とプログラム情報のペアは、前記データベースノードにおいて行うべき処理を一意に示す処理識別情報を伴うものであり、
前記データベースノードは、
前記処理識別情報に対応させて前記処理が実行済みであることを示す情報を記録する手段をさらに備え、
前記クライアントノードから受信した前記キー情報と前記プログラム情報のペアが伴う処理識別情報が実行済みとして記録されていない場合に、当該処理を実行させるものである、請求項１～２１のいずれかに記載の分散型データベースシステム。
【請求項２３】
前記クライアントノードから送信されるキー情報とプログラム情報のペアは、前記データベースノードにおいて行うべき処理に係るユーザを特定するためのユーザ情報を伴うものであり、
前記ユーザ情報により特定されるユーザに対し、前記処理に関する通知を行う手段を備える、請求項１～２２のいずれかに記載の分散型データベースシステム。
【請求項２４】
前記クライアントノードは、
前記データとペアにするキー情報を、前記データを保存するためのインデックスに基づいて生成する手段をさらに備え、
前記データベースノードは、
インデックスに対応してデータを保存する機能を有する共有ストレージに対し、前記クライアントノードから送信されたキー情報に基づいて求められるインデックスの識別情報に対応させて、前記クライアントノードから送信されたデータを書き出す手段を備えるこ

とを特徴とする請求項１～２３のいずれかに記載の分散型データベースシステム。
【請求項２５】
前記クライアントノードは、
検索条件を含む検索要求を前記プログラム情報として、該プログラム情報とペアにするキー情報を、前記共有ストレージに保存されたデータを検索するためのインデックスに基づいて生成する手段をさらに備え、
前記データベースノードは、
前記クライアントノードから送信されたキー情報に基づいて求められるインデックスの識別情報に対応して保存されているデータを、前記共有ストレージから読み込み、前記プログラム情報により定められる前記プログラムのオブジェクトを実行することにより、前記検索条件に応じて検索されたデータを、前記クライアントノードへ送信する手段をさらに備える、請求項２４に記載の分散型データベースシステム。
【請求項２６】
複数のデータベースノードと一つ以上のクライアントノードとを備え、前記複数のデータベースノードに複数のデータが分散してストアされる分散型データベースシステムで用いられるデータ処理方法であって、
オブジェクト指向型プログラミングにおけるオブジェクトであるデータであって任意の構造を有することが可能なデータが、キー情報とペアで、前記複数のデータベースノードのうち前記キー情報に基づいて特定されるデータベースノードへ前記クライアントノードから送信され、該データベースノードにストアされており、
オブジェクト指向型プログラミングにおけるオブジェクトであるプログラムによって前記複数のデータベースノードのいずれかにキー情報とペアでストアされている前記データに対する処理を行うプログラム情報を、該キー情報とペアにして、前記複数のデータベースノードのうち前記キー情報に基づいて特定されるデータベースノードへ前記クライアントノードから送信し、
前記プログラム情報は、前記プログラムのオブジェクトが属するクラスを示す情報と、該オブジェクトで用いられる変数であって任意の構造を有することが可能な変数の値の情報とを含むものであり、前記プログラムの定義である前記クラスの情報は、前記クライアントノードと前記データベースノードとで共有可能なものであり、
前記データベースノードにおいて、前記クライアントノードから送信されたキー情報とペアでストアされている前記データのオブジェクトに対して、前記クライアントノードから送信された前記プログラム情報により定められる前記プログラムのオブジェクトを実行することにより、前記データベースノードで前記データに対する処理を行い、
前記処理を行った結果の情報を、前記データベースノードから前記クライアントノードへ送信することを特徴とするデータ処理方法。
【請求項２７】
複数のデータベースノードに複数のデータが分散してストアされる分散型データベースシステムに接続されるクライアントノードであって、
オブジェクト指向型プログラミングにおけるオブジェクトであるデータであって任意の構造を有することが可能なデータを、キー情報とペアにして、前記複数のデータベースノードのうち前記キー情報に基づいて特定されるデータベースノードへ送信し、該データベースノードにストアさせる手段と、
オブジェクト指向型プログラミングにおけるオブジェクトであるプログラムによって前記複数のデータベースノードのいずれかにキー情報とペアでストアされている前記データに対する処理を行うプログラム情報を、該キー情報とペアにして、前記複数のデータベースノードのうち前記キー情報に基づいて特定されるデータベースノードへ送信する手段と、
前記プログラム情報は、前記プログラムのオブジェクトが属するクラスを示す情報と、該オブジェクトで用いられる変数であって任意の構造を有することが可能な変数の値の情報とを含むものであり、前記プログラムの定義である前記クラスの情報は、前記クライア

ントノードと前記データベースノードとで共有可能なものであり、
前記データベースノードにおいて、前記クライアントノードから送信されたキー情報とペアでストアされている前記データのオブジェクトに対して、前記クライアントノードから送信された前記プログラム情報により定められる前記プログラムのオブジェクトを実行することにより、前記データベースノードで前記データに対する処理を行ったときに、前記処理を行った結果の情報を、前記データベースノードから受信する手段と、を備えることを特徴とするクライアントノード。
【請求項２８】
複数のデータベースノードに複数のデータが分散してストアされる分散型データベースシステムを構成するデータベースノードであって、
前記分散型データベースシステムに接続される一つ以上のクライアントノードから、
オブジェクト指向型プログラミングにおけるオブジェクトであるデータであって任意の構造を有することが可能なデータが、キー情報とペアで、前記複数のデータベースノードのうち前記キー情報に基づいて特定されるデータベースノードへ送信され、
オブジェクト指向型プログラミングにおけるオブジェクトであるプログラムによって前記複数のデータベースノードのいずれかにキー情報とペアでストアされている前記データに対する処理を行うプログラム情報が、該キー情報とペアで、前記複数のデータベースノードのうち前記キー情報に基づいて特定されるデータベースノードへ送信され、
前記プログラム情報は、前記プログラムのオブジェクトが属するクラスを示す情報と、該オブジェクトで用いられる変数であって任意の構造を有することが可能な変数の値の情報とを含むものであり、前記プログラムの定義である前記クラスの情報は、前記クライアントノードと前記データベースノードとで共有可能なものであり、
前記データベースノードは、
前記クライアントノードから送信された前記キー情報と前記データのペアを受信してストアする手段と、
前記クライアントノードから送信された前記キー情報と前記プログラム情報のペアを受信する手段と、
前記クライアントノードから送信されたキー情報とペアでストアされている前記データのオブジェクトに対して、前記クライアントノードから送信された前記プログラム情報により定められる前記プログラムのオブジェクトを実行することにより、前記データベースノードで前記データに対する処理を行う手段と、
前記処理を行った結果の情報を、前記クライアントノードへ送信する手段と、を備えることを特徴とするデータベースノード。
【請求項２９】
コンピュータを、複数のデータベースノードに複数のデータが分散してストアされる分散型データベースシステムに接続されるクライアントノードとして機能させるためのプログラムであって、
前記コンピュータを、
オブジェクト指向型プログラミングにおけるオブジェクトであるデータであって任意の構造を有することが可能なデータを、キー情報とペアにして、前記複数のデータベースノードのうち前記キー情報に基づいて特定されるデータベースノードへ送信し、該データベースノードにストアさせる手段、
オブジェクト指向型プログラミングにおけるオブジェクトであるプログラムによって前記複数のデータベースノードのいずれかにキー情報とペアでストアされている前記データに対する処理を行うプログラム情報であって、前記プログラムのオブジェクトが属するクラスを示す情報と、該オブジェクトで用いられる変数であって任意の構造を有することが可能な変数の値の情報とを含むプログラム情報を、該キー情報とペアにして、前記複数のデータベースノードのうち前記キー情報に基づいて特定されるデータベースノードへ送信する手段、
前記データベースノードにおいて、前記クライアントノードから送信されたキー情報と

ペアでストアされている前記データのオブジェクトに対して、前記クライアントノードから送信された前記プログラム情報により定められる前記プログラムのオブジェクトを、該プログラムの定義である前記クラスの情報であって前記クライアントノードと共有可能な情報を用いて、実行することにより、前記データベースノードで前記データに対する処理を行ったときに、前記処理を行った結果の情報を、前記データベースノードから受信する手段、

として機能させることを特徴とするプログラム。

【請求項３０】

コンピュータを、複数のデータベースノードに複数のデータが分散してストアされる分散型データベースシステムを構成するデータベースノードとして機能させるプログラムであって、

前記分散型データベースシステムに接続される一つ以上のクライアントノードから、

オブジェクト指向型プログラミングにおけるオブジェクトであるデータであって任意の構造を有することが可能なデータが、キー情報とペアで、前記複数のデータベースノードのうち前記キー情報に基づいて特定されるデータベースノードへ送信され、

オブジェクト指向型プログラミングにおけるオブジェクトであるプログラムによって前記複数のデータベースノードのいずれかにキー情報とペアでストアされている前記データに対する処理を行うプログラム情報であって、前記プログラムのオブジェクトが属するクラスを示す情報と、該オブジェクトで用いられる変数であって任意の構造を有することが可能な変数の値の情報とを含むプログラム情報が、該キー情報とペアで、前記複数のデータベースノードのうち前記キー情報に基づいて特定されるデータベースノードへ送信され、

前記プログラムは、前記コンピュータを、

前記クライアントノードから送信された前記キー情報と前記データのペアを受信してストアする手段、

前記クライアントノードから送信された前記キー情報と前記プログラム情報のペアを受信する手段、

前記クライアントノードから送信されたキー情報とペアでストアされている前記データのオブジェクトに対して、前記クライアントノードから送信された前記プログラム情報により定められる前記プログラムのオブジェクトを、該プログラムの定義である前記クラスの情報であって前記クライアントノードと共有可能な情報を用いて、実行することにより、前記データベースノードで前記データに対する処理を行う手段、

前記処理を行った結果の情報を、前記クライアントノードへ送信する手段、

として機能させることを特徴とするプログラム。

【発明の詳細な説明】

【技術分野】

【０００１】

本発明は、分散型データベースシステムを構成するデータベースノードにストアされているデータを処理する技術に関する。

【背景技術】

【０００２】

分散型データベースシステムは、複数のデータベースサーバによって構成され、複数のデータベースサーバに複数のデータが分散してストアされる。従来、分散型データベースシステムとして、ＫＶＳ（キー・バリュー・ストア）が知られており、ＫＶＳを利用した技術には、例えば、ＲＯＭＡ、ｋｕｍｏｆｓ、Ｆｌａｒｅなどがある。ＫＶＳでは、各データベースサーバに、キーとバリューのペアがストアされる。バリューは、文字列やバイト列等のデータであり、キーは、バリュー（データ）を特定するための情報である。

【０００３】

ＫＶＳの長所の一つは、スケーラビリティが高いことにある。したがって、ＫＶＳを利

用してシステムを運用するときに、当初は少ないサーバ台数で低コストで運用しながら、システムが大きくなる（システムで取り扱うデータ量が増える）につれて、後からサーバ台数を増やしてスケールアウトするという運用が可能である。そのため、ＫＶＳでは、膨大なデータが多数のデータベースサーバに適切に分散してストアされる（分散効率を高くする）ことができるように運用されることが多い。

【０００４】

このような従来のシステム（ＫＶＳ）においてデータの処理をする場合には、クライアントが、キーを手がかりにして、処理対象のデータ（バリュー）がストアされているデータベースサーバを特定し、特定されたデータベースサーバへクライアントがキーを送ってデータを要求すると、データベースサーバからクライアントにキーに対応してストアされているデータが送られて、クライアントでその取得したデータに対して処理を実行する。その後、クライアントでデータの処理が完了したら、再び、キーを基にデータベースサーバを特定し、特定されたデータベースサーバへ、クライアントが、処理後のデータをキーとともに送り、ストアさせる。そうすると、データベースサーバに、キーとペアでストアされていたデータが、処理後のデータに置き換わる。このようにして、複数のデータベースサーバに分散してストアされているデータの処理が行われる。

【先行技術文献】

【非特許文献】

【０００５】

【非特許文献１】中田　敦、外１２名、「クラウド大全　第２版　サービス詳細から基盤技術まで」、日経ＢＰ社、平成２２年７月１２日

【発明の概要】

【発明が解決しようとする課題】

【０００６】

従来のシステム（ＫＶＳ）では、データの処理をする際には、処理のたびに、処理されるべきデータをデータベースサーバからクライアントへ送り、処理が完了したら、その処理後のデータをクライアントからデータベースサーバへ再び送る必要がある。このとき、データベースサーバからクライアントが取得するデータは、取得要求に記入されているキーに対応してストアされているデータ（バリュー）の全体となる（部分的に取得することはできない）ため、一つのキーに対応させて、大容量のデータがストアされている場合には、１回の処理あたりのデータ通信量が大きくなる。このようなデータ通信量の問題を考慮すると、従来のシステムでは、各キーに対応させてストアできるデータ量には、自ずと制限が加えられることになる。

【０００７】

そこで、従来のシステム（ＫＶＳ）においては、各キーに対応させてストアするデータ量を小さくする運用が行われていた。しかし、そうすると、一つの処理に必要なデータが複数のキーにまたがってしまい、キーが複数になれば、適度な分散効果により対応するデータがストアされているデータサーバも複数にまたがることになるから、一つの処理を実行するために、複数のデータベースサーバから複数のデータを別々に取得することが必要になる。そのため、一つの処理あたりのデータ通信回数が多くなってしまう。

【０００８】

また、従来のシステム（ＫＶＳ）では、データの処理をする毎に、キーに対応してストアされているデータそのものが、データベースサーバからクライアントへ送られるので、秘匿性の高いデータを扱うのには適しておらず、セキュリティ性を担保するためには、暗号化通信などの何らかの措置を講じる必要があった。

【０００９】

本発明は、上記の課題に鑑みてなされたもので、例えば、データベースノードにストアできるデータ量に、従来のシステム（ＫＶＳ）のような制限が課せられずに、一つのキーに対応させて大容量のデータをストアすることができるとともに、一つの処理に必要なデータの全部が一つのデータベースノードに存在する可能性を高くして、通信回数を減らす

ことができ、また、秘匿性の高いデータを扱うのにも適した分散型データベースシステムを提供することを目的とする。

【課題を解決するための手段】

【００１０】

本発明の原理に従う一つの例に係る分散型データベースシステムは、複数のデータベースノードと一つ以上のクライアントノードとを備え、前記複数のデータベースノードに複数のデータが分散してストアされる分散型データベースシステムであって、前記クライアントノードは、キー情報とデータのペアを、前記複数のデータベースノードのうち前記キー情報に基づいて特定されるデータベースノードへ送信し、該データベースノードにストアさせる手段と、前記複数のデータベースノードのいずれかにキー情報とペアでストアされているデータに対する処理を行うために、該キー情報とプログラム情報のペアを、前記複数のデータベースノードのうち前記キー情報に基づいて特定されるデータベースノードへ送信する手段とを備え、前記データベースノードは、前記クライアントノードから送信されたキー情報とペアでストアされているデータに対して、前記クライアントノードから送信されたプログラム情報に従ってプログラムを実行することにより、前記データベースノード内で前記データに対する処理を行う手段と、前記処理を行った結果の情報を、前記クライアントノードへ送信する手段とを備えている。

【００１１】

上記の構成では、キー情報により特定されるデータベースノードに、そのキー情報とペアをなすデータが送信されてストアされる。つまり、あるデータをストアするデータベースノードは、そのデータとペアをなすキー情報により特定できるようになっている。そして、クライアントノードから、データがストアされたデータベースノード（キー情報により特定されるデータベースノード）に、そのキー情報とペアをなすプログラム情報が送信されると、プログラム情報に従ってプログラムが実行され、データベースノード内でデータに対する処理が行われる。そして、その処理を行った結果の情報が、データベースからクライアントノードへ送信される。このように、データに対する処理がデータベースノード内で行われるので、データの処理が行われるときに、データベースノードからクライアントノードへ、処理されるべきデータが送信されることがない。この場合、データベースノードからクライアントノードへ送信されるのは、結果の情報であり、これは、処理されるデータよりはるかにデータ量が少ない。

【００１２】

そのため、上記の構成では、一つのデータベースノードに大容量のデータがストアされている場合であっても、データに対する処理を１回行ったときにデータベースノードとクライアントノードとの間で通信されるデータ量（１回の処理あたりのデータ通信量）は、プログラム情報と処理を行った結果の情報という小さなデータ量だけで済む。したがって、データベースノードにストアできるデータ量に、従来のシステム（ＫＶＳ）のような制限が課せられることがない。そして、上記の構成では、一つのデータベースノードに大容量のデータをストアすることができるので、一つの処理に必要なデータが、一つのデータベースノードにまとめて存在する可能性を高くすることができる。したがって、データベースノードがクライアントノードとの間で通信を行う回数（一つの処理あたりのデータ通信回数）が少なくて（例えば、１回で）済む。また、上記の構成では、データそのものがデータベースノードからクライアントノードへ送信されることがない（結果の情報だけが送信される）ので、秘匿性の高いデータを扱うのに適している。

【００１３】

上述した分散型データベースシステムにおいて、前記データベースノードで実行されるプログラムは、オブジェクト指向型プログラミングにおけるオブジェクトであり、前記クライアントノードから送信されるプログラム情報は、前記オブジェクトが属するクラスを示す情報と、該オブジェクトで用いられる変数の値の情報とを含んでもよい。

【００１４】

データベースノードで実行されるプログラムが、オブジェクト指向型プログラミングに

おけるオブジェクトであり、クライアントノードから送信されるプログラム情報には、オブジェクトが属するクラスを示す情報と、そのオブジェクトで用いられる変数の値の情報が含まれるので、データベースノードでプログラム（オブジェクト）の実行が可能になる。このように、オブジェクト指向型プログラミングを利用することにより、データベースノード内でのデータの処理（プログラムの実行）が実現できる。

【００１５】

上述した分散型データベースシステムにおいて、前記データベースノードは、各クラスで実行されるメソッドの内容と用いられる変数の定義とを含むクラスライブラリに関する情報を記憶する手段をさらに備え、前記データに対する処理は、前記クラスライブラリに関する情報を用いて前記プログラムを実行することにより行われるものであってもよい。

【００１６】

データベースノードに記憶されるクラスライブラリに関する情報には、例えば、各クラスについて、クラスで実行されるメソッドの内容と用いられる変数の定義が含まれているので、クライアントノードからプログラム情報（クラスを示す情報と変数の値の情報が含まれる）が送信されると、このクラスライブラリに関する情報を用いて、データベースノードでプログラム（オブジェクト）の実行が可能になる。このように、オブジェクト指向型プログラミングを利用することにより、データベースノード内でのデータの処理（プログラムの実行）が実現できる。

【００１７】

上記の場合には、そのクラス情報を必要とするノードの各々に、クラス情報（オリジナルまたはそのコピー）を登録してもよい。また、別の例として、共有ストレージや共有ウェブサーバを利用してもよく、その場合、まず、本システムの利用者（例えば、データベースノードにストアされているデータを処理するサービスを提供する者）が、サービス開始前に、共有ファイルサーバ（または共有ウェブサーバ）にクラス情報を登録し、各データベースノードに対しては、保存場所（またはＵＲＬ）を指示し、各データベースノードが、サービス開始時に、記憶された保存場所（またはＵＲＬ）により示される共有ファイルサーバ（または共有ウェブサーバ）からクラス情報を取得するようにしてもよい。

【００１８】

上述した分散型データベースシステムにおいて、前記クライアントノードからキー情報とペアで送信され、前記データベースノードにストアされるデータが、オブジェクト指向型プログラミングにおけるオブジェクトで表現され、任意の構造を有することが可能なものであってもよい。

【００１９】

データベースノードにストアされるデータが、オブジェクト指向型プログラミングにおけるオブジェクトであるので、オブジェクト指向型プログラミングを利用することにより、データベースノード内でのデータの処理が実現できる。また、データベースノードにストアされるデータが、任意の構造を有することが可能なものであるので、クライアントノードから送信されるプログラム情報が含む変数の値の情報を、変数が任意の構造を有するものとして記述することができ、該プログラム情報に従って実行されるオブジェクトで用いられる変数を、記述された構造に沿って指定することができるようになる。例えば、データの構造が、複数のデータオブジェクト内キーと各キーに対応するバリューである場合、クライアントノードから送信されるプログラム情報によって、処理対象のデータの構造内に存在するあるキー（データオブジェクト内キー）に対応してストアされているバリューに対して、選択的に処理を行うことが可能になる。なお、ここでのデータ構造は、クライアントノードだけでなくデータベースノードでも扱うことのできるものであり、データベースノードでプログラムを実行する際に、文字列や数値だけではなく、ある一定以上の構造やサイズを持ったデータに対する処理を一括で行うことが可能になる。

【００２０】

上述した分散型データベースシステムにおいて、前記データに対する処理が、該データに新たなデータを追記する処理である場合、前記クライアントノードから送信されるプロ

グラム情報は、追記する処理の指示と、前記新たなデータとを含み、前記データベースノードは、前記クライアントノードから送信されたキー情報とペアでストアされているデータを読み出し、該データに前記クライアントノードから送信された新たなデータを加えて得られるデータを前記キー情報とペアでストアする処理を、前記データベースノード内で行い、前記追記の成功を示す通知を、前記処理を行った結果の情報として、前記クライアントノードへ送信するものであってもよい。

【００２１】

例えば、あるデータベースノードに「ａｂｃ」というデータがストアされており、そのデータに「ｘｙｚ」を追記する場合には、クライアントノードから、追記する処理の指示と「ｘｙｚ」のデータを含んだプログラム情報が、「ａｂｃ」のデータのキー情報とともに送信される。データベースノードでは、キー情報に基づいて「ａｂｃ」のデータを読み出して、それに「ｘｙｚ」のデータを加える。このようにして得られた新たなデータ「ａｂｃｘｙｚ」がキー情報とともにデータベースノードにストアされる。この追記の処理が完了すると、データベースノードからクライアントノードへ通知（追記の成功を示す通知）が送信される。このようにして、データベースノードにストアされたデータに新たなデータを追記する処理を、そのデータベースノード内で実現することができる。

【００２２】

上述したの分散型データベースシステムにおいて、前記データに対する処理が、該データのうち少なくとも一部を新たなデータに書き換える処理である場合、前記クライアントノードから送信されるプログラム情報は、書き換える処理の指示と、書き換えられるべきデータの指示と、新たなデータとを含み、前記データベースノードは、前記クライアントノードから送信されたキー情報とペアでストアされているデータを読み出し、該データ中に前記クライアントノードから指示された書き換えられるべきデータがあればそれを新たなデータに置き換えて得られるデータを前記キー情報とペアでストアする処理を、前記データベースノード内で行い、前記書き換えの成功を示す通知を、前記処理を行った結果の情報として、前記クライアントノードへ送信するものであってもよい。

【００２３】

例えば、あるデータベースノードに「ａｂｃ」というデータがストアされており、そのデータの２文字目を「ｂ」から「ｘ」に書き換える場合には、クライアントノードから、書き換える処理の指示と２文字目を示すデータと「ｘ」のデータを含んだプログラム情報が、「ａｂｃ」のデータのキー情報とともに送信される。データベースノードでは、キー情報に基づいて「ａｂｃ」のデータを読み出して、２文字目のデータを「ｘ」に書き換える。このようにして得られた新たなデータ「ａｘｃ」がキー情報とともにデータベースノードにストアされる。この書き換えの処理が完了すると、データベースノードからクライアントノードへ通知（書き換えの成功を示す通知）が送信される。このようにして、データベースノードにストアされたデータのうち少なくとも一部を新たなデータに書き換える処理を、そのデータベースノード内で実現することができる。

【００２４】

上述した分散型データベースシステムにおいて、前記データに対する処理が、該データの一部を削除する処理である場合、前記クライアントノードから送信されるプログラム情報は、削除する処理の指示と、削除すべきデータの指示とを含み、前記データベースノードは、前記クライアントノードから送信されたキー情報とペアでストアされているデータを読み出し、該データ中に前記クライアントノードから指示された削除すべきデータがあればそれを削って得られるデータを前記キー情報とペアでストアする処理を、前記データベースノード内で行い、前記削除の成功を示す通知を、前記処理を行った結果の情報として、前記クライアントノードへ送信するものであってもよい。

【００２５】

例えば、あるデータベースノードに「ａｂｃ」というデータがストアされており、そのデータの２文字目の「ｂ」を削除する場合には、クライアントノードから、削除する処理の指示と２文字目を示すデータを含んだプログラム情報が、「ａｂｃ」のデータのキー情

報とともに送信される。データベースノードでは、キー情報に基づいて「ａｂｃ」のデータを読み出して、２文字目の「ｂ」のデータを削除する。このようにして得られた新たなデータ「ａｃ」がキー情報とともにデータベースノードにストアされる。この削除の処理が完了すると、データベースノードからクライアントノードへ通知（削除の成功を示す通知）が送信される。このようにして、データベースノードにストアされたデータの一部を削除する処理を、そのデータベースノード内で実現することができる。

【００２６】

上述した分散型データベースシステムにおいて、前記データに対する処理が、該データに含まれるデータ要素を別のデータと比較する処理である場合、前記クライアントノードから送信されるプログラム情報は、どのデータ要素を比較する処理であるかの指示と、比較対象となる前記別のデータとを含み、前記データベースノードは、前記クライアントノードから送信されたキー情報とペアでストアされているデータを読み出し、該データ中の前記クライアントノードから指示されたデータ要素と前記クライアントノードから送信された前記別のデータとを比較する処理を、前記データベースノード内で行い、前記比較により得られた結果を、前記処理を行った結果の情報として、前記クライアントノードへ送信するものであってもよい。

【００２７】

例えば、あるデータベースノードに、データ（キー情報とペアをなすデータ）に含まれるデータ要素として「パスワード」がストアされており、クライアントノードの利用者が入力した「パスワード」との比較を行って、クライアントノードの利用者のユーザ認証を行う場合を考える。この場合には、クライアントノードから、「パスワード」のデータ要素を比較する処理であることの指示と、比較対象となる「パスワード」（クライアントノードの利用者が入力した「パスワード」）を含んだプログラム情報がキー情報とともに送信される。データベースノードでは、キー情報に基づいてデータを読み出し、そのデータ中に含まれる「パスワード」と比較対象となる「パスワード」を比較する。この比較の処理が完了すると、比較によって得られた結果（認証ＯＫまたは認証ＮＧ）がデータベースノードからクライアントノードへ送信される。このようにして、データベースノードにストアされたデータに含まれるデータ要素を別のデータと比較する処理を、そのデータベースノード内で実現することができる。

【００２８】

上述した分散型データベースシステムにおいて、前記データに対する処理が、該データに含まれる一つ以上のデータ要素を用いて計算を行う処理である場合、前記クライアントノードから送信されるプログラム情報は、どのデータ要素を用いて何の計算を行う処理であるかの指示を含み、前記データベースノードは、前記クライアントノードから送信されたキー情報とペアでストアされているデータを読み出し、該データ中の前記クライアントノードから指示されたデータ要素を用いて前記クライアントノードから指示された計算を行う処理を、前記データベースノード内で行い、前記計算により得られた結果を、前記処理を行った結果の情報として、前記クライアントノードへ送信するものであってもよい。

【００２９】

例えば、あるデータベースノードに、データ（キー情報とペアをなすデータ）に含まれる一つ以上のデータ要素として「過去のテストの点数」が幾つかストアされており、それらのデータ要素を用いた計算として「過去のテストの平均点の算出」を行う場合を考える。この場合には、クライアントノードから、「過去のテストの点数」のデータ要素を用いて「過去のテストの平均点の算出」を行う処理であることの指示を含んだプログラム情報がキー情報とともに送信される。データベースノードでは、キー情報に基づいてデータを読み出し、そのデータ中の「過去のテストの点数」を用いて「過去のテストの平均点の算出」を行う。この計算の処理が完了すると、計算によって得られた結果（算出された平均点）がデータベースノードからクライアントノードへ送信される。このようにして、データベースノードにストアされたデータに含まれる一つ以上のデータ要素を用いて計算を行う処理を、そのデータベース内で実現することができる。

【００３０】
上述した分散型データベースシステムにおいて、前記クライアントノードは、複数のデータを対象にして一つの大処理を行うために、キー情報とプログラム情報のペアを複数準備し、各プログラム情報に、前記一つの大処理の前処理となる各データに対する処理の指示を含ませて、前記複数のデータベースノードのうち各キー情報に基づいて特定される一つ以上のデータベースノードへ送信させる手段と、前記一つ以上のデータベースの各々から、前記各データに対する処理を行った結果の情報を受信し、受信した複数の情報を用いて、前記一つの大処理の結果を生成する手段とをさらに備えるものであってもよい。
【００３１】
例えば、ある通信事業者のメールサービスの利用者のメールのデータが、複数のデータベースノードに分散してストアされている場合に、全利用者の全メール数の平均を計算したい場合を考える。この場合、複数のデータベースノードに分散してストアされている複数のデータを対象として「全利用者の全メール数の平均の計算」という大処理を行うことになる。この場合には、クライアントノードから、大処理の前処理として「（キー情報で指定される）利用者のメール数の平均の計算」という処理の指示を含んだ各プログラム情報が各キー情報とともに各データベースノードにそれぞれ送信される。それぞれのデータベースノードでは、自身にストアされている各利用者についてメール数の平均の計算の処理が行われ、その処理を行った結果の情報（各利用者のメール数の平均）が、クライアントノードへ送信される。クライアントノードでは、各データベースノードから受信した情報を用いて、全利用者の全メール数の平均の計算を行うことができる。
【００３２】
上述した分散型データベースシステムにおいて、前記データに対する処理が、該データに含まれるデータ要素を前記クライアントノードが取得するという処理である場合、前記クライアントノードから送信されるプログラム情報は、どのデータ要素を取得する処理であるかの指示を含み、前記データベースノードは、前記クライアントノードから送信されたキー情報とペアでストアされているデータを読み出し、該データ中の前記クライアントノードから指示されたデータ要素を抽出する処理を、前記データベースノード内で行い、前記抽出されたデータ要素を、前記処理を行った結果の情報として、前記クライアントノードへ送信するものであってもよい。
【００３３】
例えば、あるデータベースノードに、データ（キー情報とペアをなすデータ）として「過去１０年分のテストの点数」がストアされており、そのデータに含まれるデータ要素として「最近１年分のテストの点数」を取得する場合には、クライアントノードから、「最近１年分のテストの点数」というデータ要素を取得する処理であることの指示を含んだプログラム情報がキー情報とともに送信される。データベースノードでは、キー情報に基づいて「過去１０年分のテストの点数」のデータを読み出し、そのデータ中の「最近１年分のテストの点数」のデータ要素を抽出する。そして、抽出されたデータ要素（最近１年分のテストの点数）が、クライアントノードへ送信される。このようにして、データベースノードにストアされたデータに含まれるデータ要素（データの一部）を、クライアントノードが取得する処理を実現できる。
【００３４】
上述した分散型データベースシステムにおいて、前記データに対する処理が、該データを前記クライアントノードが取得するという処理である場合、前記クライアントノードから送信されるプログラム情報は、取得する処理の指示を含み、前記データベースノードは、前記クライアントノードから送信されたキー情報とペアでストアされているデータを読み出す処理を行い、読み出した前記データを、前記処理を行った結果の情報として、前記クライアントノードへ送信するものであってもよい。
【００３５】
あるデータベースノードにストアされているデータを、クライアントノードが取得する場合、クライアントノードから、データを取得する処理の指示を含んだプログラム情報が

キー情報とともに送信される。データベースノードでは、キー情報に基づいてデータを読み出して、クライアントノードへ送信する。このようにして、データベースノードにストアされているデータを、クライアントノードが取得する処理を実現できる。

【００３６】

上述した分散型データベースシステムにおいて、キー情報とデータのペアを、前記複数のデータベースノードのうち前記キー情報に基づいて特定されるデータベースノードにストアさせる手段が、前記キー情報とプログラム情報のペアを、前記クライアントノードから前記データベースノードへ送信することによって実現される場合、前記クライアントノードから送信されるプログラム情報は、ストアする処理の指示と、ストアすべきデータとを含み、前記データベースノードは、前記クライアントノードから送信されたキー情報とペアで、前記クライアントノードから送信されたプログラム情報に含まれるデータをストアする処理を行い、前記ストアの成功を示す通知を、前記処理を行った結果の情報として、前記クライアントノードへ送信するものであってもよい。

【００３７】

あるデータベースノードにデータをストアさせる場合、クライアントノードから、データをストアする処理の指示を含んだプログラム情報がキー情報とともに送信される。データベースノードでは、キー情報とペアでデータがストアされる。このストアの処理が完了すると、データベースノードからクライアントノードへ通知（ストアの成功を示す通知）が送信される。このようにして、データベースノードにデータをストアさせる処理を実現できる。

【００３８】

上述した分散型データベースシステムにおいて、前記クライアントノードから送信されたキー情報とデータのペアをストアしたデータベースノードは、前記複数のデータベースノードのうち自身の代替となり得るデータベースノードに、前記キー情報とデータのペアの複製をストアさせる手段をさらに備えてもよい。

【００３９】

キー情報とデータのペアが、あるデータベースノード（キー情報により特定されるデータベースノード）にストアされると、そのキー情報とデータのペアの複製が、他のデータベースノード（代替となり得るデータベースノード）にもストアされる。あるデータベースノードに障害が発生した場合には、他のデータベースノードにストアされている複製を用いるようにデータベースノードの担当範囲を変更することにより、容易にシステムを復旧することができる。

【００４０】

例えば、あるデータベースノードＡにキー情報とデータのペアがストアされており、その代替となり得るデータベースノードＢにキー情報とデータのペアの複製がストアされていたとする。データベースノードＡに障害が発生した場合、データベースノードＢの担当範囲に、データベースノードＡの担当範囲を含めるようにする。そうすると、データベースノードＢが、データベースノードＡの代わりをするようになる。このようにして、データベースノードの担当範囲を変更することにより、システムを復旧することができる。

【００４１】

または、あるデータベースノードに障害が発生した場合に、他のデータベースノードにストアされている複製を用いてデータベースノードの置き換え（追加・削除）をすることにより、容易にシステムを復旧することもできる。

【００４２】

例えば、あるデータベースノードＡにキー情報とデータのペアがストアされており、その代替となり得るデータベースノードＢにキー情報とデータのペアの複製がストアされていたとする。データベースノードＡに障害が発生した場合、新しいデータベースノードＸが追加される。データベースノードＸには、データベースノードＢからキー情報とデータのペアの複製がコピーされてストアされる。このデータベースノードＸは、データベースノードＡと同じキー情報によって特定されるようにされる。このようにして、新しいデー

タベースノードＸを古いデータベースノードＡに置き換えて、システムを復旧してもよい。

【００４３】

上述した分散型データベースシステムにおいて、前記データに対する処理は、前記キー情報とペアでストアされるデータの更新を伴うものであるかデータを読み出すだけであるかが区別可能であり、データを読み出すだけの処理である場合、前記クライアントノードが、前記キー情報とプログラム情報のペアを、前記複数のデータベースノードのうち前記キー情報に基づいて特定されるデータベースノードへ送信する手段は、前記キー情報により指示されるデータベースノード及び当該データベースノードの代替となり得るデータベースノードのうちから選択したデータベースノードへ送信するものであってもよい。

【００４４】

データを読み出すだけの処理は、キー情報により特定されるデータベースノードとその代替となり得るデータベースのうちから選択したデータベースノードにより実行可能である。すなわち、読み出しのみの処理を分散することが可能である。

【００４５】

例えば、あるデータベースノードＡにキー情報とデータのペアがストアされており、その代替となり得るデータベースノードＢにキー情報とデータのペアの複製がストアされていた場合、データベースノードＡとＢから選択されたデータベースノード（データベースノードＡとＢのいずれか一方または両方）から、データを読み出す処理を実行することができる。したがって、状況に応じて適切なデータベースノードからデータを読み出すことができる。例えば、データを読み出す処理を実行しようとしたときにデータベースノードＡの負荷が高かった場合には、代わりにデータベースノードＢからデータを読み出すことができる。別の例として、データベースノードＡとＢの負荷が低い場合には、両方へデータ読み出し要求を送って応答の早い方を使うことにより、データを読み出す処理の高速化が可能である。

【００４６】

上述した分散型データベースシステムにおいて、前記複数のデータベースノードの各々は、自身の担当する値の範囲を有し、前記キー情報に基づくデータベースノードの特定は、前記キー情報から算出されたハッシュ値を用いて、いずれのデータベースノードの担当する値の範囲に入るかを判断することにより行われてもよい。

【００４７】

キー情報に基づくデータベースノードの特定は、キー情報から算出されたハッシュ値を用いて行われる。この場合、キー情報から算出されたハッシュ値の一部（例えば、ハッシュ値の先頭の二文字など）を用いて、データベースノードを特定してもよい。例えば、キー情報から算出されたハッシュ値が「３９０２１６５０ａｅ９ｅ４３ｅｃｂ５３ｆ６６ｂｆ８６５ｄ９７３０１３５ｆ４３ｂ１」であったとする。このハッシュ値の先頭の二文字は「３９」である。一方、各データベースノードには、それぞれストアされるべきデータの範囲（担当範囲）を設定しておく。例えば、ノードＡの担当範囲が「０８～３４」、ノードＢの担当範囲が「３５～４８」・・・に設定されていたとする。このような場合、キー情報のハッシュ値の一部「３９」からノードＢを特定することができる。

【００４８】

上述した分散型データベースシステムにおいて、前記データベースノードには、前記データが、バージョン情報が付されたオブジェクトとしてストアされており、前記データに対する処理が、前記キー情報とペアでストアされるデータの更新を伴うものである場合、更新されたデータは、前記バージョン情報が更新されたオブジェクトとしてストアされるものであってもよい。

【００４９】

これにより、データベースノードにストアされるデータのバージョン管理が可能になる。例えば、あるデータベースノードＡにキー情報とデータのペアがストアされており、その代替となり得るデータベースノードＢにキー情報とデータのペアの複製がストアされて

いたとする。データベースノードＡのデータが更新され、その更新が未だデータベースノードＢに反映されない間、データベースノードＡのデータのバージョン情報は更新されるが、データベースノードＢのデータのバージョンは古いままである。したがって、このとき、例えば、データベースノードＡとＢの両方からデータを読み出す処理が実行されると、データベースノードＢのデータが古いバージョンのデータ（更新前のデータ）であることが分かる。

【００５０】

また別の例として、データの冗長化のために、あるキー情報とデータのペアが、二つのデータデースノード（ＤＢノードＡとＤＢノードＢ）に保存されており、一方のデータベースノード（ＤＢノードＡ）が障害により一時的にダウンし、その後に、そのデータベースノード（ＤＢノードＡ）が復旧してサービスを提供できるようになったとする。この場合、該当するキー情報とデータのペアを、再度、もう一方のデータベースノード（ＤＢノードＢ）からコピーする必要があるが、その際、二つのデータベースノード（ＤＢノードＡとＤＢノードＢ）の間で、該当のキー情報に対するバージョン番号を比較し、更新が必要な場合にだけ、データの転送（コピー）を行うようにすることができる。このようにして、データベースノード間でのデータ交換の効率化を図ることが可能になる。

【００５１】

上述した分散型データベースシステムにおいて、前記データベースノードは、あるクライアントノードから受信したプログラム情報に従って行っている処理が、前記データの更新を伴う処理である場合、該処理を行っている間に、別のクライアントノードから前記データの更新を指示するプログラム情報を受信したら、該別のクライアントノードに処理不能を通知するか、もしくは、該処理の終了を待って該別のクライアントノードから受け取った更新を指示するプログラムの処理を開始する手段をさらに備えてもよい。

【００５２】

例えば、データベースノードが、あるクライアントノードＡから受信したプログラム情報に従ってデータの更新を実行しているときに、別のクライアントノードＢからデータの更新を指示するプログラム情報を受信した場合には、その別のクライアントノードＢには、処理不能が通知される、あるいは、クライアントノードＡからのデータ更新の処理の終了を待って、クライアントノードＢからのデータ更新の処理を開始する。このようにして、複数のクライアントノードによるデータ更新のコンフリクトを制御することができる。

【００５３】

上述した分散型データベースシステムにおいて、前記クライアントノードは、前記プログラム情報を作成する手段をさらに備えてもよい。

【００５４】

クライアントノードの利用者は、自分でオリジナルのプログラムを作成でき、そのプログラムを用いて、データベースノードにストアされているデータを処理することができる。したがって、様々なデータ処理を実行させることが可能になる。

【００５５】

上述した分散型データベースシステムにおいて、前記データベースノードで実行されるプログラムは、オブジェクト指向型プログラミングにおけるオブジェクトであり、前記クライアントノードは、前記クライアントノードから送信されたプログラム情報に従って前記データベースノードが前記プログラムを実行するのに用いられるクラスの情報を作成する手段をさらに備えてもよい。

【００５６】

データベースノードで実行されるプログラムが、オブジェクト指向型プログラミングにおけるオブジェクトである場合に、クライアントノードの利用者は、自分でオリジナルのクラスを作成でき、そのクラスで定義されたオブジェクトを用いて、データベースノードにストアされているデータを処理することができる。したがって、様々なデータ処理を実行させることが可能になる。

【００５７】

データベースノードで実行されるプログラムが、オブジェクト指向型プログラミングにおけるオブジェクトである場合に、上記の分散型データベースシステムにおいて、前記クライアントノードから送信されるプログラム情報は、前記オブジェクトにより実行されるべきメソッドを示す情報と、該オブジェクトで用いられる変数に関する情報とを含み、前記クライアントノードは、メソッド呼び出しに応じて、呼び出されたメソッドを定義するクラスの情報を参照することにより、前記プログラム情報を作成するようにしてもよい。

【００５８】

これにより、クライアントノードの利用者は、ネットワークの向こう側に存在するデータベースノードにおいて処理が実行されるにもかかわらず、あたかも手元のクライアントノードにおいて処理が実行されるかのような感覚で、メソッド呼び出しを行うプログラムを開発することが可能になる。

【００５９】

上記の構成において、前記クライアントノードは、前記メソッド呼び出しの際に実行されるプロキシオブジェクト（擬似的なオブジェクトであり、例えばJava（登録商標）言語ではＰｒｏｘｙ技術で利用可能）に、キーの情報を含ませることにより、前記キー情報と前記プログラム情報のペアを作成するようにしてもよい。

【００６０】

これにより、プロキシオブジェクトが、クライアントノードにおけるメソッド呼び出しに基づいて、データベースノードにおいて実行されるべきメソッドを示す情報を生成するとともに、これをキー情報とペアにしてデータベースノードへ送信することが可能になり、クライアントノードの利用者は、キー情報とプログラム情報のペアを渡す先が、ネットワークの向こう側に存在するデータベースノードであることを意識しなくても、プログラムを開発することが可能になる。

【００６１】

上述した分散型データベースシステムにおいて、前記データベースノードで実行されるプログラムは、オブジェクト指向型プログラミングにおけるオブジェクトであり、前記分散型データベースシステムは、前記クライアントノードから送信されたプログラム情報に従って前記データベースノードが前記プログラムを実行するのに用いられるクラスの情報を記憶しているストレージ又はサーバをさらに備え、前記複数のデータベースノードの各々は、前記クラスの情報を前記ストレージ又はサーバから取得する手段を備えてもよい。

【００６２】

データベースノードで実行されるプログラムが、オブジェクト指向型プログラミングにおけるオブジェクトである場合に、クラスの情報は、ストレージ又はサーバに記憶されており、各データベースノードは、これらのストレージ又はサーバから、クラスの情報を取得して利用することができる。このようにして、クラスの情報を、複数のデータベースノードで共有することができる。

【００６３】

上述した分散型データベースシステムにおいて、前記クライアントノードから送信されるキー情報とプログラム情報のペアを、前記データベースノードにおいて行うべき処理を一意に示す処理識別情報を伴うものとし、前記データベースノードは、前記処理識別情報に対応させて前記処理が実行済みであることを示す情報を記録する手段をさらに備え、前記クライアントノードから受信した前記キー情報と前記プログラム情報のペアが伴う処理識別情報が実行済みとして記録されていない場合に、当該処理を実行させるようにしてもよい。

【００６４】

これにより、例えば、クライアントノードからデータベースノードへ依頼した処理を行った結果の情報が、何らかの事情でクライアントノードに届かず、クライアントノードが再依頼する場合に、以前の依頼をデータベースノードが受信しておらず処理を行っていなかったのか、それとも処理後のデータベースノードからクライアントノードへの通信に障害があったのかを区別して、重複処理をすることなくリトライを扱うことが可能になる。

【００６５】
上述した分散型データベースシステムにおいて、前記クライアントノードから送信されるキー情報とプログラム情報のペアを、前記データベースノードにおいて行うべき処理に係るユーザを特定するためのユーザ情報を伴うものとし、前記ユーザ情報により特定されるユーザに対し、前記処理に関する通知を行う手段を備えるようにしてもよい。
【００６６】
これにより、例えば、データベースノードあるいは複数のデータベースノードを管理するサーバにおいて、ユーザに対する認証を行ってから依頼された処理に進むようにしたり、どのユーザがどのような処理を依頼したかを把握して、課金、ログの管理、レポート等のサービスを提供したりすることが可能になる。
【００６７】
上述した分散型データベースシステムの発明は、システム全体の方法の発明としても、汎用のコンピュータシステムを本システムとして動作させるためのプログラム（又はそのプログラムを記録した記録媒体）の発明としても、クライアントノード装置の発明としても、データベースノード装置の発明としても、汎用のコンピュータを本システムのクライアントノード又はデータベースノードとして動作させるためのプログラム（又はそのプログラムを記録した記録媒体）の発明としても、クライアントノードにおいて実行される方法の発明としても、データベースノードにおいて実行される方法の発明としても、勿論成立するものである。
【００６８】
また、本発明の原理は、上述した分散型データベースシステムのように、複数のデータベースノードの各々に、自身の担当するキー情報とペアをなすデータが分散してストアされる構成に適用することができるほか、共有ストレージをバックに有する複数のデータ処理ノードが、共有ストレージに保存されるデータの書き込みや読み出しを分散して行う構成に適用することもできる。
【００６９】
そのような本発明の原理に従う一つの例に係る分散型データ処理システムは、インデックス（データを一意に特定するための情報）に対応してデータを保存する機能を有する共有ストレージに対し分散して処理を行う分散型データ処理システムであって、前記分散型データ処理システムは、複数のデータ処理ノードと一つ以上のクライアントノードとを備え、前記クライアントノードは、データを保存するためのインデックスに基づいてキー情報を生成する手段と、前記キー情報と前記データのペアを、前記複数のデータ処理ノードのうち前記キー情報に基づいて特定されるデータ処理ノードへ、送信する手段とを備え、前記データ処理ノードは、前記共有ストレージに対し、前記クライアントノードから送信されたキー情報に基づいて求められるインデックスの識別情報に対応させて、前記クライアントノードから送信されたデータを書き出す手段を備える。
【００７０】
上記の構成によれば、共有ストレージにおいてインデックスに対応してデータが保存されるところ、そのインデックスに基づいてキー情報が生成され、このキー情報により特定されるデータ処理ノードが、共有ストレージへデータを書き込む処理を行うことになる。したがって、各々のインデックスのデータ領域に対するアクセスは、そのインデックスを担当するデータ処理ノードから生じるようになり、それ以外のデータ処理ノードからは生じないから、アクセスの競合が生じにくく、処理分散の効率を向上することが可能になる。
【００７１】
上記の分散型データ処理システムにおいて、前記クライアントノードは、前記共有ストレージに保存されたデータを検索するためのインデックスに基づいてキー情報を生成する手段と、前記キー情報と検索要求のペアを、前記複数のデータ処理ノードのうち前記キー情報に基づいて特定されるデータ処理ノードへ、送信する手段とをさらに備え、前記データ処理ノードは、前記クライアントノードから送信されたキー情報に基づいて求められる

インデックスの識別情報に対応して保存されているデータを、前記共有ストレージから読み出し、前記クライアントノードへ送信する手段をさらに備えるようにしてもよい。

【００７２】

これにより、共有ストレージからデータを読み出す処理についても、各々のインデックスのデータ領域に対するアクセスが、そのインデックスを担当するデータ処理ノードから生じるようになり、アクセスの競合を生じにくくして、処理分散の効率をさらに向上することが可能になる。また、キャッシュ（読み込んだデータを一時的に高速なメモリに保存しておくこと）によるアクセス性能の向上が可能になる。

【００７３】

上記の構成において、前記クライアントノードから送信される検索要求を、検索条件を含むプログラム情報とし、前記データ処理ノードは、前記プログラム情報に従ってプログラムを実行することにより、前記共有ストレージから読み出すべきデータを検索する処理を行うようにしてもよい。

【００７４】

上記の構成では、例えば、大量のログを長時間にわたって継続的に記録するような場合でも、各クライアントノードにおけるログの受付及び送信（分散化）と、各データ処理ノードにおけるインデックス処理及び共有ストレージへのアクセスとが、パイプラインで行え、クライアントノードやデータ処理ノードの台数を増やすことにより性能を上げるスケールアウトの実現が可能になる。さらに、そのようなログ記録からの検索を様々な検索条件に応じて行うことも可能になる。

【００７５】

上述した分散型データ処理システムの発明は、システム全体の方法の発明としても、汎用のコンピュータシステムを本システムとして動作させるためのプログラム（又はそのプログラムを記録した記録媒体）の発明としても、クライアントノード装置の発明としても、データ処理ノード装置の発明としても、汎用のコンピュータを本システムのクライアントノード又はデータ処理ノードとして動作させるためのプログラム（又はそのプログラムを記録した記録媒体）の発明としても、クライアントノードにおいて実行される方法の発明としても、データ処理ノードにおいて実行される方法の発明としても、勿論成立するものである。

【００７６】

また、上述した分散型データベースシステムのデータベースノードにおいて採用可能な各種構成は、上記の分散型データ処理システムのデータ処理ノードにおいても適宜採用できるものである。

【発明の効果】

【００７７】

本発明によれば、例えば、一つのキーに対応させて大容量のデータをストアすることができるとともに、一つの処理に必要なデータの全部が一つのデータベースノードに存在する可能性を高くして、通信回数を減らすことができる。また、本発明は、秘匿性の高いデータを扱うのにも適している。

【図面の簡単な説明】

【００７８】

【図１】本発明の実施の形態における分散型データベースシステムの一例（本システム）を説明するための図

【図２】本システムのクライアントノードの構成の一例を示すブロック図

【図３】本システムのデータベースノードの構成の一例を示すブロック図

【図４】本システムの概要（データの処理と分散）を説明するための図

【図５】本システムにおけるクラス情報の共有の一例を説明するための図

【図６】本システムの特徴（プログラミング環境）を説明するための図

【図７】本システムの特徴（データベース技術）を説明するための図

【図８】本システムの適用例（個人情報を扱うシステム）を説明する図

【図９】本システムの適用例（オンラインショップの購買履歴を扱うシステム）を説明する図
【図１０】本システムの適用例（商品在庫管理システム）を説明する図
【図１１】本システムの適用例（健康管理システム）を説明する図
【図１２】本システムの適用例（通信事業者のメールサービス）を説明する図
【図１３】本システムの適用例（通信事業者のメールサービス）を説明する図
【図１４】本システムの適用例（オンラインショップのサービスの一部）の詳細を説明する図
【図１５】図１４の適用例と同じサービスを別の仕組みで実現した適用例を説明するための図
【図１６】図１５の仕組みを採用したシステムの一例を説明するための図
【図１７】図１５の仕組みの利点の一例を説明するための図
【図１８】本システムの利点の一例を説明するための図
【図１９】本システムの利点の一例を説明するための図
【図２０】本発明の実施の形態における分散型データ処理システムの一例を説明するための図
【図２１】図２０のシステムの適用例（通信ログの処理）を説明する図
【発明を実施するための形態】
【００７９】
以下、本発明の実施の形態に係る分散型データベースシステムについて、例示のために、図面を用いて説明する。本実施の形態では、データストレージサービス等に用いられる分散型データベースシステムの場合を例示する。
【００８０】
本発明の実施の形態における分散型データベースシステムの一例（本システム）の構成を、図面を参照して説明する。図１は、本システムの構成を示す説明図である。図１に示すように、本システム１は、複数のデータが分散してストアされる複数のデータベースノード２と、本システム１の利用者が用いるクライアントノード３で構成されている。本システム１によって提供されるサービスは、複数のエンドユーザが用いるユーザ端末４から利用できるようになっている。なお、図１では、説明の便宜のため、データベースノード２やクライアントノード３やユーザ端末４の台数が少なく例示されているが、データベースノード２やクライアントノード３やユーザ端末４の台数がこれに限定されないのは勿論である。
【００８１】
図２は、本システムのクライアントノード３の構成を示すブロック図である。図２に示すように、クライアントノード３は、データ送受信部５と、データ処理部６と、プログラム作成部７を備えている。データ送受信部５は、データベースノード２との間でデータ通信を行う機能を備えている。データ処理部６は、データベースノード２と間で通信されたデータを処理する機能を備えている。プログラム作成部７は、本システム１で利用される種々のプログラムを作成する機能を備えている。
【００８２】
図３は、本システムのデータベースノード２の構成を示すブロック図である。図３に示すように、データベースノードは、データ送受信部８と、データ処理部９と、データ格納部１０と、クラスライブラリ格納部１１と、コンフリクト制御部１２と、バージョン管理部１３を備えている。データ送受信部８は、クライアントノード３との間でデータ通信を行う機能を備えている。データ処理部９は、クライアントノード３と間で通信されたデータを処理する機能を備えている。また、データ送受信部８は、他のデータベースノード２との間でデータ通信を行う機能も備えており、データ処理部９は、他のデータベースノード２と間で通信されたデータを処理する機能も備えている。
【００８３】
データ格納部１０には、クライアントノード３から受信したデータ（キー情報とデータ

のペアなど）が格納される。また、クラスライブラリ格納部１１には、本システムで利用するプログラムのクラス（オブジェクト指向型プログラミングにおけるオブジェクトが属するクラス）の情報が格納される。例えば、各クラスで実行されるメソッドの内容や、各クラスで用いられる変数の定義などが格納される。コンフリクト制御部１２は、データ格納部に格納されているデータを更新するときのコンフリクトを制御する機能を備えている。バージョン管理部１３は、データ格納部に格納されているデータのバージョンを管理する機能を備えている。

【００８４】

コンフリクト制御は、複数のクライアントノードが、同一のキーに対して同時にデータ更新の処理を要求してきた場合に実行される。例えば、あるクライアントノードＡからデータの更新の要求を処理している間に、別のクライアントノードＢからデータの更新の要求があった場合には、その別のクライアントノードＢには、処理不能が通知されるか、あるいは、クライアントノードＡからのデータ更新の処理の終了を待って、クライアントノードＢからのデータ更新の処理が開始される。

【００８５】

次に、本システムの概要について説明する。本システムは、スケール型の大規模分散データベースシステムであり、データベースノードの台数は、例えば数百台以上である。図４は、本システムの概要を示した説明図である。図４では、説明の便宜上、８台のデータベースノード（ＤＢノードＡ～Ｈ）のみが例示されている。図４に示すように、各データベースノードには、それぞれＩＤ（例えば、１６進の数字）が割り当てられている。これらのデータベースノードは、割り当てられたＩＤの番号順に「リング状」になるように、仮想的に並べられている。つまり、最初のデータベースノード（例えば、ＤＢノードＡ）からＩＤの番号順に一巡すると、最後のデータベースノード（ＤＢノードＨ）の次は、再び最初のデータベースノード（ＤＢノードＡ）に戻るようになっている。

【００８６】

また、この割り当てられるＩＤの大小によって「向き」が決められており、あるデータベースノード（例えば、ＤＢノードＡ）よりＩＤの番号が次に大きいデータベースノード（ＤＢノードＢ）は、そのデータベースノード（ＤＢノードＡ）の「successor」と呼ばれる。また、あるデータベースノード（例えば、ＤＢノードＢ）よりＩＤの番号が次に小さいデータベースノード（ＤＢノードＡ）は、そのデータベースノード（ＤＢノードＢ）の「predecessor」と呼ばれる。

【００８７】

本システムでは、データベースノードの配置、範囲の割り当て、バックアップなどの仕組みに、ＤＨＴ（Distributed Hash Table）やConsistent Hashと呼ばれる技術と同様の技術を用いている。図４の例で説明すると、例えば、あるデータベースノード（例えば、ＤＢノードＢ）の担当範囲は、そのデータベースノードのＩＤ（ＤＢノードＢのＩＤ：３５）から次のデータベースノードのＩＤ（ＤＢノードＣのＩＤ：４９）までに設定される。そして、クライアントノードから送られたキーとデータのペアは、そのキーのハッシュ値を計算したときの最初の２文字が担当範囲に含まれるデータベースノードにストアされる。例えば、キー情報から算出されたハッシュ値が「３９０２１６５０ａｅ９ｅ４３ｅｃｂ５３ｆ６６ｂｆ８６５ｄ９７３０１３５ｆ４３ｂ１」であると、このハッシュ値の先頭の二文字は「３９」であり、ＤＢノードＢにストアされることになる。

【００８８】

また、本システムでは、データの多重度設定により、データの「複製」の数を指定することができ、同時に複数のデータベースノードに障害が発生しても、それに対応することができる。より具体的に説明すると、本システムでは、あるＩＤを持つデータベースノード（例えば、ＤＢノードＢ）が担当すべき情報を、そのデータベースノード（ＤＢノードＢ）が持つほかに、その情報のコピー（複製）を、そのデータベースノードのpredecessorとなるデータベースノード（ＤＢノードＡ）が持っている。このように二つのデータベースノードが情報を持つ状態は、多重度が「２」に設定されているともいえる。多重度は

、全データで共通である。多重度を大きくするには、predecessorの向きに、コピーを持つデータベースノードの数を増やせばよい。なお、データのコピーは、データベースノード間で行われる。つまり、あるデータベースノード（例えばＤＢノードＢ）でデータの保存、更新、削除が行われた場合、その情報は、そのデータベースノードのpredecessorとなるデータベースノード（ＤＢノードＡ）にも伝えられ、predecessorとなるデータベースノード（ＤＢノードＡ）でも、同様に、データの保存、更新、削除が行われる。ただし、データは同じものを複製するか、または差分の変換によってデータの複製を更新するだけであるため、複製されたデータのキーは変わらない。したがって、predecessorとなるデータベースノード（ＤＢノードＡ）が持っている「複製」は、本来、そのデータベースノード（ＤＢノードＡ）の担当範囲外のデータである。そのため、そのデータベースノード（ＤＢノードＡ）のデータには、そのキーに対応するプログラム情報が、クライアントノードから直接届くことがない。そのキーに対応するプログラム情報（job）は、本来のデータベースノード（ＤＢノードＢ）のみに届き、そのデータベースノード（ＤＢノードＡ）のデータの更新は、本来のデータベースノード（ＤＢノードＢ）から伝えられる情報のみによって行われる。

【００８９】

なお、この「複製」は、データの並列読み出しに利用することができる。そのため、本システムでは、プログラム情報（job）に「READ（読み込みのみ）」か「EDIT（更新を伴う）」かの区別ができるようにフラグが付けられている。「READ」のフラグが付けられた処理（読み込みのみの処理）については、多重度の数までpredecessorの方向に処理を分散することも可能である。なお、書込みの処理は「一貫性」を保つため、本来の担当範囲を受け持っているデータベースノードでしか行われない。

【００９０】

また、本システムでは、動的なデータベースノードの追加・削除が容易である。データベースノードを追加するときには、新しいＩＤを持ったデータベースノード（例えばＤＢノードＩ）を準備して、そのデータベースノードが担当しなければならない範囲のデータを、現在その範囲のデータを担当しているデータベースノード（例えばＤＢノードＨ）からコピーし、その後、新しいデータベースノード（ＤＢノードＩ）が「リング」に追加される。この場合、データの複製を終えてから、データベースノードを追加することにより、ダウンタイムを最小限にできる。

【００９１】

一方、データベースノードを削除するときには、削除対象のデータベースノード（例えばＤＢノードＨ）を削除する。それ以降は、そのデータベースノード（ＤＢノードＨ）が担当していた範囲のデータに対するプログラム情報（job）は、そのデータベースノードのpredecessorとなるデータベースノード（ＤＢノードＧ）に届くことになる。このデータベースノード（ＤＢノードＧ）は、削除対象のデータベースノード（ＤＢノードＨ）が担当していた範囲のデータの複製を持っているので、サービスの提供が継続される。この場合、predecessorとなるデータベースノードがデータの複製を持っていることを前提に、データベースノードを削除することにより、ダウンタイムを最小限にできる。

【００９２】

また、本システムは、ＫＶＳと同等もしくはそれ以上のスケールと拡張性を有しており、さらに「分散プログラミング」のインターフェースもサポートしている。本システムでは、データベースノードにデータが送られて、データの書込みなどが行われるだけでなく、データベースノードにプログラム情報が送られて、そのデータに対する「処理」が行われる。この場合、データは「オブジェクト」としてデータベースノードにストアされており、データベースノード上のオブジェクトは「Java（登録商標） Object」（Java（登録商標）で扱うことができる任意のオブジェクト）として表現できる。そして、データベースノード上で実行するプログラムも「Java（登録商標）のクラス情報」として記述できるため、通常のJava（登録商標）のプログラミングのような直感的で容易なツールによる「分散処理」のプログラム開発が可能になり、Java（登録商標） Class Libraryによる「分

散開発環境」を構築することができる。

【００９３】

本システムでは、本システムの利用者が指定したプログラムをデータベースノード上で実行できるようにするため、クライアントノードとデータベースノードで、プログラムの定義である「クラス情報」を共有することができる。共有の仕方には、３つのパターンがある。第１のパターンは、そのクラス情報を必要とするノードの各々に、クラス情報（オリジナルまたはそのコピー）を登録するパターンである。第２のパターンは、共有ストレージを使うパターンである。第３のパターンは、共有のウェブサーバを使うパターンである。

【００９４】

図５を参照して、クラス情報の共有について更に説明する。共有ストレージを使う場合には、利用者が、共有ファイルサーバにクラス情報を登録し、各データベースノードに、保存場所を教えてから、サービスを開始させる。そうすると、各データベースノードが、共有ファイルサーバからクラス情報を取得する。共有のウェブサーバを使う場合には、利用者が、共有ウェブサーバにクラス情報を登録し、各データベースノードに、ＵＲＬを教えてから、サービスを開始させる。そうすると、各データベースノードは、そのＵＲＬ（共有ウェブサーバ）からクラス情報をダウンロードする。

【００９５】

次に、図６と図７を参照して、本システムの特徴について説明する。図６に示すように、本システムでは、クライアントノード側で処理内容（job）を定義して、そのプログラム（job）をデータベースノードへ送り、データベースノード側で分散して処理を実行し、クライアントノードは、その結果だけを取得する。一方、従来のシステム（ＫＶＳ）では、クライアント側で処理を行うため、データベースサーバからデータを取得（get）し、その処理が終わったらデータをデータベースサーバへ送信（put）する。

【００９６】

したがって、本システムは、従来のシステム（ＫＶＳ）に比べて、データベースノードとの通信の回数が少なくて済み、データベースノードから送られるデータ量も小さく、しかも、データベースノードからデータが外に出ないため秘匿性の高いデータの取り扱いに適している、という特徴を有している。

【００９７】

図７には、本システムと従来のシステム（ＫＶＳ）との違いが概念的に示されている。なお、図７では、ノードやサーバで実行される処理が「略四角形」の図形で示されており、処理の対象となるデータが「円形」の図形で示されている。図７に示すように、従来のシステム（ＫＶＳ）では、処理そのものはクライアント側で実行される。そのため、従来のシステムでは、クライアントの処理能力に制限があり、大規模な処理を実行するのは困難である。それに対して、本システムでは、処理そのものをデータベースノードに分散させることができる（理論上、データベースノードの数だけ分散することができる）。したがって、本システムでは、クライアントノードの処理能力の制限を受けることなく（複数のデータベースノードの処理能力を利用して）、大規模な処理を実行することが可能である。

【００９８】

本システムは、データベースノードへの処理の実装が容易であり、処理の分散が可能である点で、従来のシステム（ＫＶＳ）より優れている。したがって、本システムは、扱うデータ量が多い（例えば、数千万件～数百億件）サービスで大規模なデータ処理を行うのに適している。例えば、携帯事業者などがメールを処理するためのシステムでは、１億人分のユーザのメール（ユーザ１人あたり数千件とすると、数千億件のメール）が、１万台のデータベースサーバに分散してストアされる。この場合、データベースサーバ１台あたり、１万人分のユーザのメール（ユーザ１人あたり数千件とすると、数千万件のメール）をストアし、処理するように、適度にデータ及びその処理が分散されることになる。また、本システムは、スケーラビリティ（拡張性）が高いので、データの増加率が高いサービ

スに適している。例えば、取り扱うデータ量が増えたときには、データベースノードの台数を追加するだけで対応可能である。また、本システムは、データがデータベースノードの外部に出ることがないので、秘匿性の高いデータを扱うサービスに適している。また、各データがいくつかの要素から構成される場合、単純なデータ取得だけでなく、データの構造に従った分析や処理が可能である。

【００９９】

以下、図面を参照しながら、本システムの具体的な適用例について説明する。図８には、個人情報を扱うシステムへの適用例が示されている。図８に示すように、本システムでは、キー情報（key）として識別番号（例えば、０１２３４５）を用い、そのキー情報とペアで各種のデータ（パスワード「ｈｏｇｅ」，年齢「２５」，氏名「山田太郎」，郵便番号「０１２－３４５６」，住所「東京都千代田区」など）がストアされている。

【０１００】

図８の「本システム」の「処理１」に示すように、本システムでは、「パスワードチェック」の処理を行うことができる。この場合、「ユーザ名とパスワード」が入力されると、「認証の成否」が出力として得られる。

【０１０１】

ここで、本システムのjobの定義について、図８の「本システム」の「処理１」を例にして、詳しく説明する。図８の「処理１」の例において、第１行目は、プログラムの内容を記述するためのクラスの定義を示しており、この例では「Authenticate」というクラスが定義されている。第２行目は、このプログラムで利用する変数の定義を示しており、この例では、「pass」というString型の変数が定義されている。第３～５行目は、プログラムをオブジェクトとして生成するための方法（メソッド）を定義しており、この例では、文字列を引数とし、内部変数「pass」 に、該引数を保持するよう指示している。なお、このメソッドは、Java（登録商標）プログラミングで「constructor」と呼ばれているものである。第６～９行目は、データベースノードで実行されるメソッドを定義しており、この例では「Boolean applyTo」というメソッドが定義されている。また、「Object o」は、データベースノード上に保存されたデータ（オブジェクト）を引数として受け取ることを意味している。第７行目は、引数として受け取った（データベースノード上の）オブジェクトに対して行う処理の内容を記述しており、この例では「入力されたパスワードと保存されているパスワードを比較する」処理が示されている。ここでは、データベースノードに保持されたデータ（オブジェクト）を「Map（Java（登録商標）プログラムで定義されるクラスのひとつ）」として扱うものとする。また、このプログラムを利用するアプリケーションでは、データベースノードに保持されたデータ（オブジェクト）は「Map」というクラスに属するものとして扱い、その中で「password」というキーに対応する値として、ユーザのパスワード情報が保存されているものとする。このように、図８の「処理１」の例では、データベースノードに保存されているオブジェクト内の「password」 というキーに対応する値と、本システムの利用者がパスワードとして入力した内部変数の「pass」とを比較して、一致するかどうかを判断し、「Boolean」の真偽値（一致したか否かの値）を返す処理が行われる。

【０１０２】

また、図８の「本システム」の「処理２」に示すように、本システムでは、「所定の住所と年齢に該当するかの判断」の処理を行うことができる。この場合、「識別番号と住所と年齢の範囲」が入力されると、「該当するか否か」が出力として得られる。この例では、「GetCampain」というクラスが用いられ、「applyTo」というString型のメソッドが定義されている。そして、データベースノードに保存されているオブジェクトの「address（住所）」というキーに対応する値が「東京都」で始まるものであり、かつ、「age（年齢）」というキーに対応する値が２０以上であれば「選挙に行こう」という文字列を返す処理が行われる。

【０１０３】

図９には、オンラインショップの購買履歴を扱うシステムへの適用例が示されている。

図９に示すように、本システムでは、キー情報（key）として識別番号（例えば、０１２３４５）を用い、そのキー情報とペアで各種のデータ（パスワード「ｈｏｇｅ」，氏名「山田太郎」，購買履歴１「ビール」，購買履歴２「ウィスキー」など）がストアされている。

【０１０４】

図９の「本システム」の「処理１」でも、図８と同様の「パスワードチェック」の処理が行われる。ここでは、詳しい説明を省略するが、この例でも、「Authenticate」というクラスが用いられ、「ユーザ名とパスワード」が入力されると、「認証の成否」が出力として得られる。

【０１０５】

また、図９の「本システム」の「処理２」に示すように、本システムでは、「過去に酒類を買ったことがあるかの判断」の処理を行うことができる。この場合、「ユーザＩＤ」が入力され、過去に酒類の購買履歴があれば「酒類のキャンペーン」の情報が出力として返される。この例では、「GetCampain」というクラスが用いられ、「applyTo」というString型のメソッドが定義されている。そして、データベースノードに保存されているオブジェクトの「history」というキーに購買履歴に関する情報が配列で格納されており、該配列から、過去の購買履歴を検索し、購買履歴の中に「ビール」の文字列がある場合には、「ビールのキャンペーン」という文字列を返し、そうでなければ、「キャンペーンなし」という文字列を返す処理が行われる。このように、データベースノードに保存されるデータは、処理に応じた構造を持つデータをオブジェクトとして持つことができる（本例ではMapの中に、文字列や数値だけでなく、さらに配列を格納している）。

【０１０６】

図１０には、商品在庫管理システムへの適用例が示されている。図１０に示すように、本システムでは、キー情報（key）として商品ＩＤ（例えば、０１２３４５）を用い、そのキー情報とペアで各種のデータ（品名「チョコレート」，価格「１５０」，２０１０年４月の販売数「１２００」，２０１０年５月の販売数「１５００」など）がストアされている。なお、この例では、格納するデータを表現するための「SellInfo」という独自のクラスを定義している。このように、データベースノードに格納するデータは、オブジェクト指向型プログラミングで定義できる、任意の独自クラスのオブジェクトを持つ事もできる。

【０１０７】

図１０の「本システム」の「処理１」に示すように、本システムでは、「現在の在庫数」を求める処理を行うことができる。この場合、「商品ＩＤ」が入力されると、「在庫数」が出力として得られる。この例では、「GetRemains」というクラスが用いられ、「applyTo」というInteger型のメソッドが定義されている。そして、データベースノードに保存されているオブジェクトを、独自クラス「SellInfo」のオブジェクトとみなして、該オブジェクトの「remains（在庫数）」という変数に格納された値を返す処理が行われる。

【０１０８】

また、図１１の「本システム」の「処理２」に示すように、本システムでは、「月次の平均販売数の計算」の処理を行うことができる。この場合、「商品ＩＤ」が入力されると、「月次販売数の平均値」が出力として得られる。この例では、「GetSellAverage」というクラスが用いられ、「applyTo」というDouble型のメソッドが定義されている。そして、データベースノードに保存されているオブジェクトを、独自クラス「SellInfo」のオブジェクトと見なして、該オブジェクトの「sold（販売履歴）」に格納されている「sell-2010-01（２０１０年１月の販売数）」、・・・、「sell-2010-12（２０１０年１２月の販売数）」というキーに対応する値を合計し、その平均値を返す処理が行われる。

【０１０９】

図１１には、健康管理システムへの適用例が示されている。図１１に示すように、本システムでは、キー情報（key）として個人ＩＤ（例えば、０１２３４５）を用い、そのキー情報とペアで各種のデータ（２０１０年９月１日の取得カロリー「１６００」、２０１

０年９月２日の取得カロリー「１７５０」、体重「６５ｋｇ」など）がストアされている。

【０１１０】

図１１の「本システム」の「処理１」に示すように、本システムでは、「現在の体重を求める」処理を行うことができる。この場合、「ユーザＩＤ」が入力されると、「現在の体重」が出力として得られる。この例では、「GetWeight」というクラスが用いられ、「applyTo」というDouble型のメソッドが定義されている。そして、データベースノードに保存されているオブジェクトの「weight（体重）」というキーに対応する値を返す処理が行われる。

【０１１１】

また、図１１の「本システム」の「処理２」に示すように、本システムでは、「毎日の平均取得カロリーの計算」の処理を行うことができる。この場合、「ユーザＩＤ」が入力されると、「取得カロリーの平均値」が出力として得られる。この例では、「GetAverage」というクラスが用いられ、「applyTo」というDouble型のメソッドが定義されている。そして、データベースノードに保存されているオブジェクトの「start（開始日。この例では２０１０年９月１日）の取得カロリー」、「２０１０年９月２日の取得カロリー」、・・・というキーに対応する値を合計し、その平均値を返す処理が行われる。

【０１１２】

図１２と図１３には、通信事業者のメールサービスへの適用例が示されている。図１２は、従来のシステム（ＫＶＳ）ではデータが大きすぎるのに対して、本システムでは効率的に処理を行うことができる例である。図１２に示すように、本システムでは、キー情報（key）として個人ＩＤ（例えば、０３ａ９ｅ０）を用い、そのキー情報とペアで各種のデータ（最大保存メール数「９５０」、第１番目のメールの本文、第２番目のメールの本文など）がストアされている。

【０１１３】

図１２の「本システム」の「処理１」に示すように、本システムでは、「最新のメールを取得する」処理を行うことができる。この場合、「ユーザＩＤ」が入力されると、「最新メールの本文」が出力として得られる。この例では、「GetNewMail」というクラスが用いられ、「applyTo」というString型のメソッドが定義されている。そして、データベースノードに保存されているオブジェクトの「maxId（最新メール）」というキーに対応するメールの本文を返す処理が行われる。

【０１１４】

また、図１２の「本システム」の「処理２」に示すように、本システムでは、「保存メールの数を求める」処理を行うことができる。この場合、「ユーザＩＤ」が入力されると、「削除されていないメールの数」が出力として得られる。この例では、「GetCount」というクラスが用いられ、「applyTo」というInteger型のメソッドが定義されている。そして、データベースノードに保存されている「第１番目のメール」から「最新メール」までの全メールについて、本文が「null」でないメール（削除されていないメール）の数をカウントし、カウントした数（削除されていないメール数）を返す処理が行われる。

【０１１５】

図１３は、複数のキーにまたがる処理の例である。図１２と同様、本システムでは、キー情報（key）として個人ＩＤ（例えば、０３ａ９ｅ０）を用い、そのキー情報とペアで各種のデータ（最大保存メール数「９５０」、第１番目のメールの本文、第２番目のメールの本文など）がストアされている。

【０１１６】

図１３の「本システム」の「処理３」に示すように、本システムでは、複数の「ユーザＩＤ」にまたがる場合であっても、「保有メール数の平均値を算出する」処理を行うことができる。この場合、「ユーザＩＤ」の集合が入力されると、「保有メール数の平均値」が出力として得られる。この例では、「GetCount」というクラスが用いられ、「applyTo」というInteger型のメソッドが定義されている。そして、それぞれの「ユーザＩＤ」に

ついて、データベースノードに保存されている「第１番目のメール」から「最新メール」までの全メールについて、本文が「null」でないメール（削除されていないメール）の数をカウントし、カウントした数（削除されていないメール数）を返す処理が、各データベースノードにおいて行われる。その後、それぞれの「ユーザＩＤ」について得られた「メール数」を合計し、その平均値を求める処理が、クライアントノードにおいて行われる。

【０１１７】

以下には、本実施の形態に係る分散型データベースシステムを利用するアプリケーションの提供者が、プログラム開発やオペレーション（運用管理）をより簡易に行えるようにするためのフレームワークを含む本システムについて、図１４～１７を参照しながら説明する。

【０１１８】

本フレームワークにより、例えば、Ａｎｄｒｏｉｄ端末や、Java（登録商標）（Ａｐｐｌｅｔ、ＯＳＧＩ、他、任意のJava（登録商標）仮想マシンを含む）が稼働するコンピュータ等のユーザ端末から、クラウド（本システム）上のオブジェクトに簡単にアクセスし、クラウド上のデータオブジェクトをあたかもユーザ端末の手元にあるオブジェクトであるかのように扱えるようにするサービスを、容易に提供することが可能になる。

【０１１９】

このようなサービスを提供するためのアプリケーションプログラムは、既存技術では、クライアント（ユーザ端末）／サーバ（本システムではクライアントノード）／データベース（本システムではデータベースノード）の三層で開発することになるため、クライアントとサーバ側との間の通信に加え、サーバとデータベースとの間の通信をも意識して、アプリケーションプログラムを開発する必要がある。

【０１２０】

これに対し、本フレームワーク（例えばJava（登録商標）のようなオブジェクト指向型プログラミングに適用される）では、サーバとデータベースとの間の通信や、データが保存される場所、処理が行われる場所等を意識することなく、あたかもローカルに動くアプリケーションプログラムであるかのように開発することができる。

【０１２１】

また、既存技術では、サーバ側（サーバ／データベース）の管理運用に手間がかかるのに対し、本フレームワークでは、サーバ側の一切の運用管理（物理的なサーバの冗長化やサイジング（スケールの調整)等を含む）を事業者に任せることが可能になる。

【０１２２】

本フレームワークを含む本システムでは、クライアントノードにおいて、必要に応じて、キーに対応するデータベースノード上のオブジェクトと通信するための プロキシ（Proxy）オブジェクトを保持する。プロキシオブジェクト自体は、Java（登録商標）の技術として知られているものであるが、本フレームワークでは、プロキシオブジェクト(クライアント側オブジェクト)に、本システムにおけるキーの情報を保持するようにする。そして、後述するメソッド呼び出し時に、このプロキシオブジェクトを用いる。

【０１２３】

具体的には、クライアントノードにおいて、上記のプロキシオブジェクト対するJava（登録商標）のメソッド呼び出し（method call）をフック（hook）し、自動的に「リモートプロシージャーコール」に変換することにより、データベースノード上のオブジェクトに対するメソッド呼び出しを実行する。

【０１２４】

プロキシによるフックを行うと、Java（登録商標）のメソッド呼び出し時に、呼び出したメソッドの本来のメソッド名と引数に関する情報とが取得できる。これを利用して、取得した情報をもとに、該当するメソッドを呼び出すジョブ（job）を自動生成する仕組みを、本システムに含ませておく。

【０１２５】

本フレームワークにより自動生成されたジョブは、呼び出されるメソッド名と引数に関

する情報を含むものであり、本システムによって、上記キーの値により特定されるデータベースノードへ配信され、分散実行される。

【０１２６】

図１６は、上述した本フレームワーク（自動ジョブ生成技術と呼ぶことがある）を含む本システムを説明するための図である。

【０１２７】

クライアントノードは、プロキシオブジェクトを保持しており、プロキシオブジェクトには、メソッドが呼び出された時に実行されるフック用のメソッドが含まれており、メソッドが呼び出された時に、本来のメソッド名と引数に関する情報（例えば引数のクラス及び値)が渡される。これにより、フック用のメソッドの実行結果が、本来のメソッドの実行結果になる。

【０１２８】

このプロキシオブジェクトに、データベースノード上のオブジェクトを特定するキー情報が保持される。そして、上記フック用のメソッドが呼び出された時に、ジョブが自動生成され、キーのハッシュ値等に基づいて特定されるデータベースノードへ配信される。

【０１２９】

データベースノードへ配信されるのは、キー情報とジョブであり、ジョブには、メソッド名と、引数に関する情報（引数のクラスと値)と、上記メソッドを実行して結果を返すという処理内容とが含まれる。

【０１３０】

このジョブを受け取ったデータベースノードは、キーに対応する対象オブジェクトをデータベースからロードし、その対象オブジェクト内のデータに対して、ジョブにより規定される処理を実行し、実行結果をクライアントノードへ返す。このとき、処理によってデータに変更があった場合、データベースノードは、変更後の対象オブジェクトをデータベースにストアする。

【０１３１】

以上に説明した自動ジョブ生成技術を採用した本システムの具体的な適用例として、図１５の買い物カートの例を説明する。本例では、オンラインショッピングにおいて買い物カートに入れられたみかんの値段と数から全体の金額を計算する。

【０１３２】

まず、図１５と同じ適用例を、本フレームワークを含まない本システムにより実現する場合について、図１４を用いて説明しておく。図１４では、図８～１３に示したのと同様の手法で、みかんの値段と数をセットする処理のためのジョブ（「OrangeSet」）と、みかんの合計金額を計算する処理のためのジョブ（「OrangeTotal」）とが、定義されている。

【０１３３】

図８～１３では、定義されたジョブをキーにより特定されるデータベースノードへ配信して処理を実行させるためのプログラムの記述を省略していたが、図１４には、そのプログラムが「Application例１」で記述されている。

【０１３４】

「Application例１」におけるプログラムは、まず、「jobcast.store」という本システムへの処理の登録を行うためのメソッドを使って、処理の対象としたいキーと実行したい処理（ここでは「orange」）とを登録する。そして、みかんの値段と数をセットする処理を行うために、対応するジョブ（「OrangeSet」）を生成し、「jobcast.execute」という本システムにおけるジョブの配信を行うためのメソッドを使って、生成したジョブをキーにより特定されるデータベースノードへ配信し、処理を実行させる。さらに、みかんの合計金額を計算する処理を行うために、対応するジョブ（「OrangeTotal」）を生成し、「jobcast.execute」という本システムにおけるジョブの配信を行うためのメソッドを使って、生成したジョブをキーにより特定されるデータベースノードへ配信し、処理を実行させる。

【０１３５】
つまり、図８～１４に例示したスキームでは、アプリケーションプログラムの開発者は、各々の処理のためのジョブを定義するプログラムと、そのジョブを生成、配信、実行するためのプログラムとを、本システムに特有の方式で記述することになる。
【０１３６】
これに対し、図１５に例示するスキームでは、手元にあるコンピュータ単体で処理を行う（リモートで分散して処理を行うのではない）場合と、ほぼ同様の方式で本システムを利用するアプリケーションプログラムを記述することができる。これが、自動ジョブ生成技術を採用した本システムの利点であり、図１７に具体的に例示されている。
【０１３７】
図１７は、Java（登録商標）のプログラミングの例であり、単体での処理の場合、その処理（ここでは「Orange」）のインターフェースを定義した上で、「OrangeImpl」として処理内容を定義する。そして、定義された処理を実行するための「Application例２」が、「Orange」オブジェクトを生成し、「OrangeImpl」に定義された各メソッドを呼び出すように、記述される。
【０１３８】
この単体での処理のためのプログラムと、図１５の自動ジョブ生成技術を使った分散処理のためのプログラムとを比較すると、インターフェースの定義と処理内容の定義は共通であり、定義された処理を実行するための「Application例２」におけるメソッド呼び出しも共通であり、「Application例２」におけるオブジェクト生成を、プロキシを用いながらキーを設定するものに変えるだけでよいことが分かる。したがって、開発者は、本システムを利用したアプリケーションプログラムを、従前に慣れ親しんだ方式で記述することが可能になる。
【０１３９】
ここで、図１５に戻り、本システムで自動生成を実現するための仕組みについて、より詳細に説明する。
【０１４０】
Java（登録商標）では、プロキシという技術を使うことによって、あるインターフェース（本例では「Orange」）で定義されるメソッド（*1） を持つ仮想的なオブジェクト（*2）を生成して、上記(*1) のメソッドが呼ばれたときに、メソッド名と引数（メソッド呼び出しの際のパラメータ）の情報を受け取る仮想的なメソッドを定義することができる。すなわち、プロキシで仮想的なオブジェクトを作っておくことで、それ以降、メソッド呼び出しの際に、当該メソッド呼び出し時のメソッド名と引数の情報を、『特定のメソッド』（*3）に渡すことができる。
【０１４１】
さらに、本フレームワークでは、上記（*2）のオブジェクトにキー情報を持たせておいて、上記（*3）の『特定のメソッド』において、呼び出された本来のメソッド名（*4）とメソッドが呼び出された際の引数の情報（*5）とからジョブ（*6）を生成し、キーで特定されるデータベースノードへジョブを送信し、その結果（送信先のデータベースノードにおいてキーで特定されるデータオブジェクトに対してジョブを実行することにより得られた結果）を受け取る、ということを行う。すなわち、上記（*3）の『特定のメソッド』というものが、ジョブの生成と、ジョブの配信と、結果の受信とを自動的に行うことになる。
【０１４２】
上記（*3） の『特定のメソッド』が生成するジョブは、メソッド呼び出しに応じて実行すべき処理内容が特定されたオブジェクトであるが、この生成を実現するために、図１５に「jobの定義例」として記載されているように、一般化したジョブのクラス（「class GenericJob」）が予め定義されている。
【０１４３】
個々に生成される上記（*6）のジョブは、一般化したジョブのクラスの一つのオブジェ

クト（インスタンス）であり、上記（*4）のメソッド名と、上記（*5）の引数の情報とを持つ。そして、ジョブの実行時には、これらメソッド名と引数の情報から、データベースノード上に保存されている対象オブジェクトに定義されている本来のメソッド情報を探してきて、対象オブジェクトのメソッドを呼び出すことで、本来のメソッドの実行がデータベースノード上で行われることになる。

【０１４４】

上述した仕組みにより、分散処理を考えない場合と同様に「obj.set0range(10, 100)」というメソッド呼び出しを行うと、事前に「getProxy」で生成されているプロキシオブジェクトの、『特定のメソッド』が呼び出され、本来呼び出されたメソッド名（「set0range」）と、メソッドが呼び出されたときの引数情報（10(int), 100(int)）とをもとにjobインスタンスが生成される。このjobの中身は、メソッド名である set0rangeと、引数のタイプ情報であるint, intという情報と、引数の値である10, 100という情報を持つ。

【０１４５】

そして、上記『特定のメソッド』は、プロキシオブジェクトを生成したときに渡したキー情報をもとに、対象となるデータベースノードを選択し、そのデータベースノードへ上記のjob インスタンスを送付して、そのデータベースノードがjob を実行した結果を、ネットワークを介して受け取る。受け取った結果が、最終的に、上記 「obj.set0range(10, 100)」の実行結果として扱われる。

【０１４６】

なお、データベースノードがjob を実行する際には、job オブジェクトの applyTo が実行されるため、その中では、メソッド名（set0range） 及び引数の情報（int, int型の10, 100という情報）から、対象のオブジェクトの該当メソッドを探し出して実行する。ここで、保存されているのは0rangeImplクラスのオブジェクトであるので、0rangeImplのset0range(10, 100)が呼び出され、その結果が返ることになる。

【０１４７】

「obj. getTotal()」というメソッド呼び出しについても、同様に、事前に「getProxy」で生成されているプロキシオブジェクトの、『特定のメソッド』が呼び出され、本来呼び出されたメソッド名（「getTotal」）をもとにjobインスタンスが生成される。このjobの中身は、メソッド名である getTotalという情報を持つ。

【０１４８】

そして、上記『特定のメソッド』は、プロキシオブジェクトを生成したときに渡したキー情報をもとに、対象となるデータベースノードを選択し、そのデータベースノードへ上記のjob インスタンスを送付して、そのデータベースノードがjob を実行した結果を、ネットワークを介して受け取る。受け取った結果が、最終的に、上記 「obj. getTotal()」の実行結果として扱われる。

【０１４９】

なお、データベースノードがjob を実行する際には、job オブジェクトの applyTo が実行されるため、その中では、メソッド名（getTotal）から、対象のオブジェクトの該当メソッドを探し出して実行する。ここで、保存されているのは0rangeImplクラスのオブジェクトであるので、0rangeImplのgetTotal()が呼び出され、その結果が返ることになる。

【０１５０】

図１５の仕組みにより自動生成されるジョブは、図１４で生成される各処理のジョブと機能的には同等のものになる。つまり、図１５の「obj.set0range(10, 100)」に対応して生成されるジョブは、「class GenericJob」の一つのオブジェクト(インスタンス)であり、メソッド名がorangeSet、パラメータ情報が（int, int）型の（10, 100）であるジョブになるため、図１４の「0rangeSet」クラスのオブジェクト（job1 = new 0rangeSet(10, 100)）とほぼ同等である。また、図１５の「obj. getTotal()」に対応して生成されるジョブは、「class GenericJob」の一つのオブジェクト(インスタンス)であり、メソッド名がgetTotal、パラメータ情報が無しであるジョブになるため、図１４の「0rangeToral」クラスのオブジェクト（job2 = new 0rangeTotal()）とほぼ同等である。

【０１５１】
以上、図１４の適用例に対して、図１５の実現手法が可能であることを説明したが、図８～１３の適用例に対しても、同様に、自動ジョブ生成技術を用いた実現手法を採ることが可能なことは、勿論である。このように自動ジョブ生成技術を用いれば、本システムを利用する大規模な並列処理プログラムの開発が可能となり、そのように開発したプログラムを実行することにより、自動的にデータの処理や保存がクラウド上で分散して行われることになる。
【０１５２】
上述したジョブ自動生成を行う本システムにおいて、データベースノード上のデータを「読み込む」処理と「変更を伴う」処理とを区別するために、以下のような拡張機能を付加してもよい。
【０１５３】
ここで、「変更を伴う」処理とは、値に変更があったら保存しなおすことが必要な処理であり、複数同時に実行できないように、サーバ側で排他ロックを行う。「読み込み」処理は、値の変更を意識しないため、保存しなおしはスキップされる処理であり、複数同時に実行できるため、サーバ側では共有ロックを行う。
【０１５４】
具体的には、例えば、Java（登録商標）のアノテーション（annotation）と呼ばれる技術で、「読み込み」か「変更を伴う」をマーク付けしておく。マークは、メソッド毎に定義することが可能である。そして、ジョブを自動生成する際に、マークの有無で実際に生成するジョブを区別し、例えば、デフォルト（マークがない場合）は「変更を伴う」処理として実行すればよい。
【０１５５】
以上に詳述した本システムにおいて、各ジョブにオペレーションＩＤを付与することにより、以下のように処理の確実性を担保することが可能になる。なお、オペレーションＩＤを付与するジョブは、自動生成されるジョブに限らず、本システムで使われるジョブのすべてに付与することが可能である。
【０１５６】
まず、クライアントノードにおいて、各ジョブに一意なオペレーションＩＤ（例えばＵＵＩＤを利用） を付与してから、データベースノードへ配信する。そして、データベースノードにおいて、あるジョブによる処理が「実行済み」の場合には、そのジョブに付与されているオペレーションＩＤに対応させて結果を記録しておく。
【０１５７】
これにより、ネットワークもしくはデータベースノードで障害が発生し、処理の実行結果がクライアントノードに返ってこない場合に、クライアントノードからリトライをすると、ジョブを受信したデータベースノードは、受信されたオペレーションＩＤと記録とを照らし合わせることにより、そのジョブが以前に受信されて「実行済み」のもの（サーバ側でジョブを実行した後、結果がクライアントに届くまでの間に障害があったもの）なのか、そうでない（初めて実行する）ものなのかを判断することができる。
【０１５８】
そして、データベースノードは、「実行済み」であると判断すれば、そのオペレーションＩＤに対応させて記録してあった結果を、クライアントノードに返し、「実行済み」でないと判断すれば、改めて処理を実行して、その結果をクライアントノードに返す。
【０１５９】
図１８と図１９には、本システムの利点がまとめられている。図１８に示すように、本システムは、１回のデータ転送量が小さく、複数要素にまたがる処理でも、通信回数が少ない。また、処理の結果だけを返すため、秘匿性のあるデータの取り扱いに適している。さらに、図１９に示すように、本システムでは、単一の項目へのアクセスをするときの通信量が少なく、複数の項目へアクセスするときの通信量も少ない。そして、バックエンド処理（データベースノード側での処理）のため、秘匿性のあるデータがネットワークに流

出することがない。なお、図１８と図１９における「従来」のシステムは、ＫＶＳで細かくデータを分割して保存するようにしたシステムであり、「参考例」のシステムは、ＫＶＳで大きなデータを単位として保存するようにしたシステムである。

【０１６０】

以上には、本発明の実施の形態に係るシステムとして、分散型データベースシステムを例にとって詳述したが、本発明の原理に従う別のシステムとして、図２０に例示するような分散型データ処理システム（これも「本システム」と呼ばれる）を構成することも可能である。

【０１６１】

例えば、大量のログを、長時間にわたって継続的に、高性能かつスケーラブルに処理、保存したい場合に、分散型データベースシステムの各データベースノードにデータをストアするのでもよいが、それとは別のバックエンドのストレージにデータを保存するようにし、そのストレージに対する処理を分散して行う分散型データ処理システムを構築することも効果的である。

【０１６２】

図２０の分散型データ処理システムは、デバイスからログのレコード情報を入力するクライアントノードと、並列分散的にレコードの保存／検索の処理を行う複数のデータ処理ノードとから構成される。

【０１６３】

図２０におけるクライアントノードは、レコードを受付部として機能するのに加えて、分散型データベースシステムのクライアントノードに対応する機能を有する。具体的には、受け付けたレコード情報からキーを生成し、キーにより特定されるデータ処理ノードへジョブを配信することにより、処理を分散化する。

【０１６４】

図２０における各データ処理ノードは、データの保存先が各ノードではなくバックエンドに設けられた共有ストレージである点が異なるが、分散型データベースシステムのデータベースノードに対応する機能を有し、クライアントノードから送信されてきたジョブを実行することにより、レコードを処理する。

【０１６５】

上記のデータ処理ノードは、ログのレコード情報にインテックス処理（indexing）を行い、インデックスに応じて、ストレージに書き込む。例えば、クライアントノードから受信したレコードを、その都度、インデックスに対応するデータ領域に追加書き込みしていくことにより、ログが蓄積される。よって、バッチ処理ではなくリアルタイムで、継続的に（２４時間３６５日の稼働で）ログの処理が可能である。

【０１６６】

このような本システムは、通信事業者の通信ログや、企業のネット利用ログや、機械のセンサーログ等の処理、保存を行う応用の他、商店・販売店・オンラインショップの販売／購買記録等、大量の記録を行う応用に、適している。

【０１６７】

当該応用においては、デバイスが多数存在して、各々のデバイスにおいてレコードが任意のタイミングで生成され、生成されたレコードを各デバイスがネットワーク経由でクライアントノード（デバイスに対し、フロントエンドサーバとして働く）へ入力する。多数のデバイス中に、異なる種類のレコードを生成するデバイスが混在しても構わない。

【０１６８】

クライアントノードは、すべて同じ、ステートレスな処理を行うため、多数のクライアントノードを並列で動作させることができ、各デバイスは、どのクライアントノードへレコードを送信してもよい。

【０１６９】

データ処理ノードは、図１～１９を参照して説明したデータベースノードと同様に、キーをもとに複数のデータ処理ノードで処理を分散するものであるため、スケールアウト（

台数を増やすことで全体性能が向上する）を実現可能である。

【０１７０】

以上のように、図２０のシステムにおいては、ログの生成（デバイス）、ログの受付（クライアントノード）、インデックス処理（データ処理ノード）、書き込み（ストレージ）の各段階が直列に接続され、続々とパイプライン的に処理を進めることができ、かつ、各段階が多数の装置の並列分散処理で実行されるため、 大量のログを高速に処理する高性能化が可能である。

【０１７１】

図２０におけるストレージは、共有ストレージであればよいが、本発明者が提案した特願２０１０－２３７８２８のストレージ技術を用いれば、ストレージの書き込み／読み出しにおいてもスケールアウトと高性能化を実現することが可能になる。つまり、追加書き込み（蓄積）を行うデータ領域を、上記先行出願のストレージ上に設けるようにすれば、ストレージ容量もスケーラブルに拡張可能となる。

【０１７２】

図２１には、図２０のシステムの適用例として、通信ログの処理を示す。例えば、デバイスとしての通信機器が、「どのアドレスから、どのURLに対してアクセスがあった」というレコード情報を生成し、任意に選択したクライアントノードへ送信する。

【０１７３】

クライアントノードは、デバイスから受け取ったレコード情報に基づいて、例えば「送信元アドレス」をキーにし、後述するジョブを生成して、キーに応じて特定されるデータ処理ノードへ送信する。

【０１７４】

データ処理ノードは、クライアントノードから受け取ったジョブを実行する。例えば、キーである「送信元アドレス」をインデックス（データベースでいう検索キーに相当するものとしてもよい） として扱い、キー毎にレコード情報を追加書き込み（蓄積）する。

【０１７５】

例えば、レコード（ログの内容）が、先頭から「通信時刻（８バイト）；送信元アドレス（４バイト）；送信先アドレス（４バイト）；送信元ポート（２バイト）；送信先ポート（２バイト）；アクセスするＵＲＬ（文字列）」のように並んでおり、インデックスとすべき要素が送信元アドレスの場合、クライアントノードは、レコードの先頭から８バイト目から４バイト分をキーとする。そして、クライアントノードは、キーのハッシュ値を計算し、計算値に従って、ジョブの配信先となるデータ処理ノードを特定する。

【０１７６】

上記のキーとジョブを受信したデータ処理ノードは、インデックスとすべき要素（ここでは送信元アドレス）であるキーに対応するデータ領域に、受け取ったジョブに含まれているレコード情報を追加書き込み（蓄積）する。例えば、受け取ったキーに基づいて、出力先のファイル名を決定し、レコード情報を共有ストレージ内に保存されるファイルとして出力する。ファイル名は、例えば、キーと受信した日時とから「key-date-time.log」のように定めればよい。

【０１７７】

以上では、レコード情報の保存について説明したが、クライアントノードからデータ処理ノードへ、レコード情報を検索するためのジョブを送信することにより、共有ストレージからの読み出しの処理も分散化することが可能である。この場合のジョブは、キーに対応するデータ領域からレコード情報を読み出して、レコード一覧を作成し、このレコード一覧を結果としてクライアントノードへ返すものである。

【０１７８】

図２１に例示したジョブでは、検索条件としてインデックス（ここでは送信元アドレス）だけが指定されており、データ処理ノードは、インデックスの示すデータ領域から読み出されたレコード情報をすべて結果に含めて返すことになるが、クライアントノードから送信するジョブに詳細な検索条件（例えば、時刻、送信先アドレス、ポート番号、それら

の任意の組合せ等）を設定することにより、データ処理ノードが、インデックスの示すデータ領域から読みだされたレコード情報のうち検索条件に合致するものを抽出して、抽出したレコードの一覧をクライアントノードへ返すことも可能である。

【０１７９】

本システムでは、インデックスとすべき要素をキーとしてジョブが配信されるため、アクセス対象となるデータ領域によって、処理を行うデータノードが分散化されることになり、複数のデータ処理ノードから同じデータ領域がアクセスされる競合が生じる可能性を低減することができる。

【０１８０】

さらに、本システムでは、そのインデックスについて保存（ストレージへの書き込みを含む）の処理を行う同じデータ処理ノードが、検索（ストレージからの読み出しを含む）の処理も行うことになる。

【０１８１】

ここで、一つのインデックスについて保存や検索の処理を行うデータ処理ノードが、複数存在して、その都度選択されるようにしてもよい。そのために、例えば、送信元アドレス（インデックスとすべき要素）と時刻情報（その他の要素）とを組み合わせたものをキーとして、データ処理ノードの特定を行ってもよい。この場合、データ処理ノードでは、キー情報全体をインデックスとするのではなく、キー情報からインデックスとなる部分を抜き出してインデックスとすることになる。

【０１８２】

なお、図１～１９を参照して説明した分散型データベースシステムにおける様々な仕組みや工夫は、図２０～２１を参照して説明した分散型データ処理システムにも適用することが可能である。

【０１８３】

以上、本発明の実施形態について説明したが、上述の実施形態を本発明の範囲内で当業者が種々に変形、応用して実施できることは勿論である。

【産業上の利用可能性】

【０１８４】

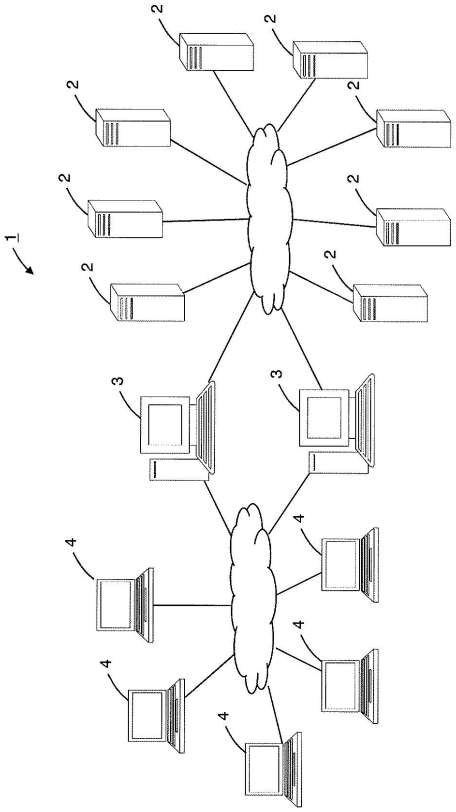
本発明の原理に従う分散型データベースシステムあるいは分散型データ処理システムによれば、例えば、様々なデータストレージサービスを提供することができ、有用である。

【符号の説明】

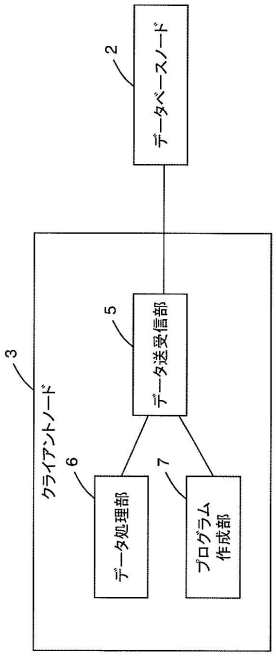
【０１８５】

１　分散型データベースシステム（本システム）
２　データベースノード
３　クライアントノード
４　ユーザ端末
５　データ送受信部
６　データ処理部
７　プログラム作成部
８　データ送受信部
９　データ処理部
１０　データ格納部
１１　クラスライブラリ格納部
１２　コンフリクト制御部
１３　バージョン管理部

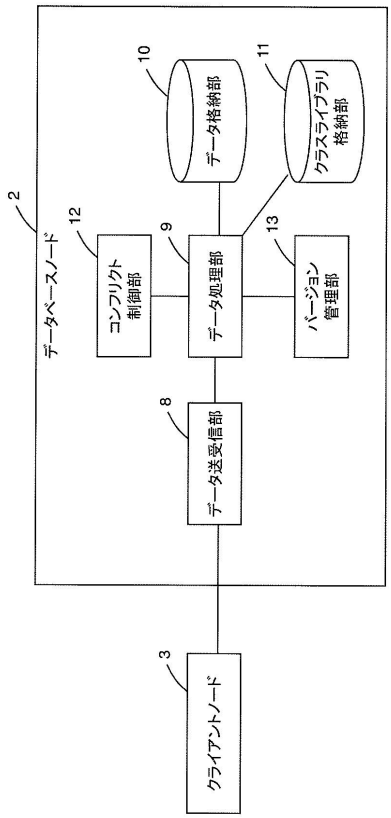
【図１】


1
2
3
4

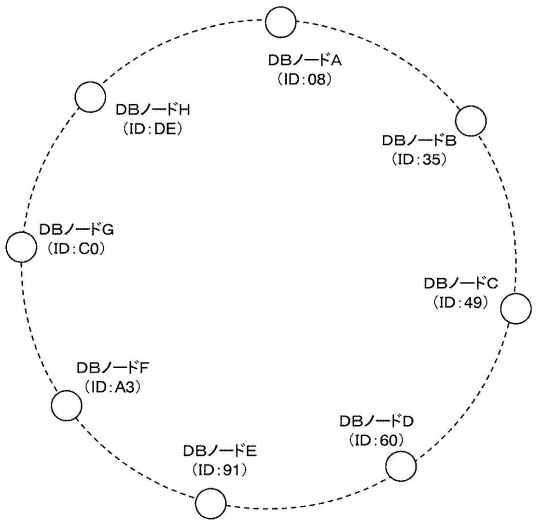
【図２】


データベースノード
2
データ送受信部
5
クライアントノード
3
データ処理部
6
プログラム作成部
7

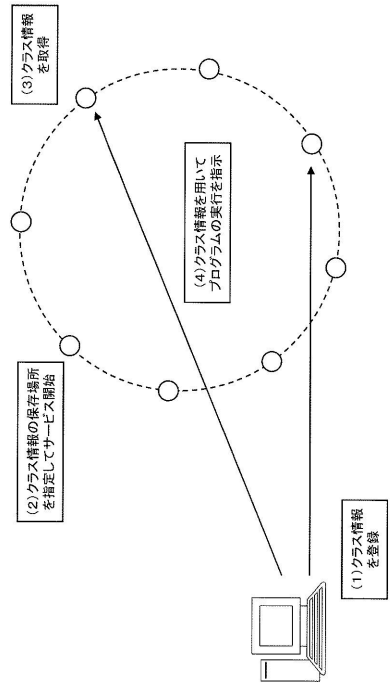
【図３】


データベースノード
2
データ格納部
10
クラスライブラリ格納部
11
コンフリクト制御部
12
データ処理部
9
バージョン管理部
13
データ送受信部
8
クライアントノード
3

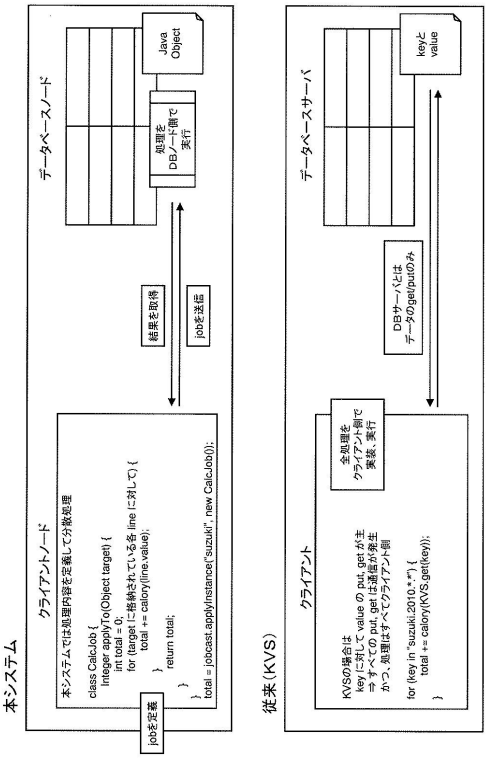
【図４】


DBノードA
(ID:08)
DBノードB
(ID:35)
DBノードC
(ID:49)
DBノードD
(ID:60)
DBノードE
(ID:91)
DBノードF
(ID:A3)
DBノードG
(ID:C0)
DBノードH
(ID:DE)

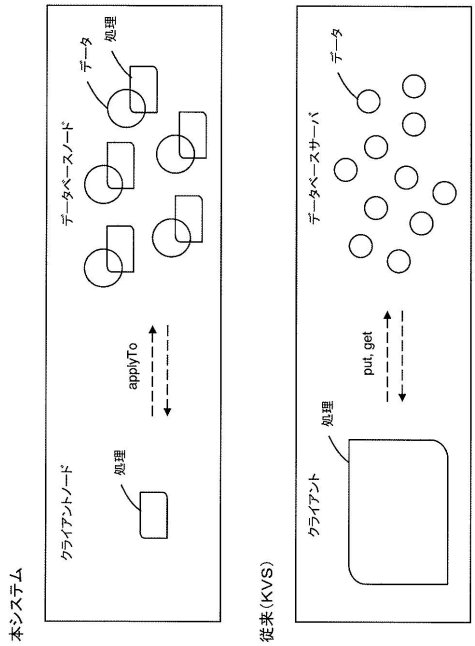
【図５】



【図６】



【図７】



【図８】

個人情報を扱うシステム

	処理 1「パスワードチェック」 概要：ユーザがデータにアクセスする際の認証 入力：ユーザ＋ユーザが入力したパスワード 出力：accepted or denied	処理 2「住所＋年齢に該当するかどうかの判断」 概要：都内で20歳以上は選挙の呼びかけ、など 入力：識別番号＋住所＋年齢の範囲 出力：true or false

本システム
KEY = 識別番号、VALUE = [各要素の値の列]
012345 = [
password = hoge,
age = 25,
name = 山田太郎,
zip = 012-3456,
address = 東京都千代田区
]
※各情報は Map オブジェクト内に格納する
	job の定義例
	class Authenticate {
	String pass;
	Authenticate(String in) {
	pass = in;
	}
	Boolean applyTo(Object o) {
	return pass.equals(((Map)o).get("password"));
	}
	}
		job の定義例
		class GetCampain {
		String applyTo(Objecto o) {
		if (((Map)o).get("address").startsWith("東京都")
		&& ((Map)o).get("age") >= 20) {
		return "選挙に行こう";
		}
		return "";
		}
		}

【図９】

オンラインショップの購買履歴を扱うシステム

	処理 1.「パスワードチェック」 概要：ユーザがデータにアクセスする際の認証 入力：ユーザ＋ユーザが入力したパスワード 出力：accepted or denied	処理 2.「過去に『商品』を買ったことがある」 概要：履歴からキャンペーン情報を提供 入力：ユーザID 出力：true or false

本システム
KEY＝ユーザID、VALUE＝{ 各要素に値のテーブル }
012345 = {
password = hoge,
name = 山田太郎,
history = [ビール、ウィスキー、 ...]
}
※各情報は Map オブジェクト内に格納する
	Job の定義例
	class Authenticate {
	String pass;
	Authenticate(String in) {
	pass = in;
	}
	Boolean applyTo(Object o) {
	return pass.equals(((Map)o).get("password"));
	}
	}
		Job の定義例
		class GetCampain {
		String applyTo(Objecto o) {
		String[] array = ((Map)o).get("history");
		for (i = 0; i < array.length; i++) {
		if (array[i].equals("ビール")) {
		return "ビールのキャンペーン";
		}
		}
		return "キャンペーンなし";
		}
		}

	history は Map 内にさらに配列情報として格納する	

【図１０】

商品在庫管理システム

	処理 1.「現在の在庫数」 概要：指定の商品の在庫数を確認する 入力：商品ID 出力：在庫数	処理 2.「月次の平均販売数」 概要：指定の商品の毎月の売れ行きを確認する 入力：商品ID 出力：月次販売数の平均値

本システム
KEY＝商品ID、VALUE＝{ 各要素に値のテーブル }
012345 = {
item = チョコレート,
price = 150,
sell-2010-04 = 1200,
sell-2010-05 = 1500, ...
remains = 200
}
ただし、上記は以下の独自クラスに格納する。
class SellInfo {
String item; // Item 名
int price; // 値段
int remains; // 在庫数
Map sold; // 販売数のテーブル
}
	Job の定義例
	class GetRemains {
	Integer applyTo(Objecto o) {
	return ((SellInfo)o).remains;
	}
	}
		Job の定義例
		class GetSellAverage {
		Double applyTo(Objecto o) {
		for (t = 1; t <= 12; t++) {
		v = ((SellInfo)o).sold.get("sell-2010-" + t);
		total += v; count++;
		}
		}
		}

【図１１】

健康管理システム

	処理 1.「現在の体重」 概要：ユーザの現在体重を取得 入力：ユーザID 出力：現在体重	処理 2.「毎日の平均取得カロリー」 概要：ユーザの取得カロリーの平均を計算 入力：ユーザID 出力：取得カロリーの平均値

本システム
KEY＝個人ID、VALUE＝要素＝値のテーブル
012345 = {
2010-09-01 = 1600
2010-09-02 = 1750
weight = 65Kg
}
※各情報は Map オブジェクト内に格納する
	Job の定義例
	class GetWeight {
	Double applyTo(Objecto o) {
	return ((Map)o).get("weight");
	}
	}
		Job の定義例
		class GetAverage {
		Double applyTo(Objecto o) {
		t = ((Map)o).get("start");
		while ((v = ((Map)o).get(t)) != null) {
		total += v; count++; t++
		}
		}
		}

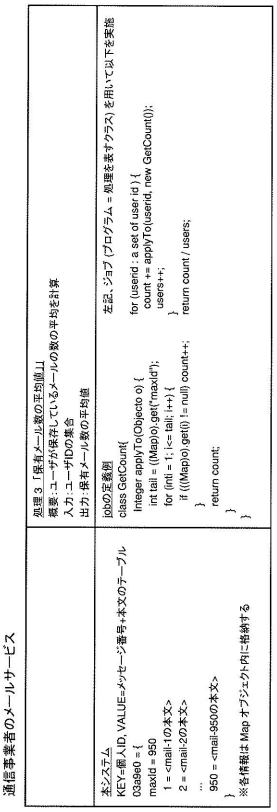
【図１２】

通信事業者のメールサービス

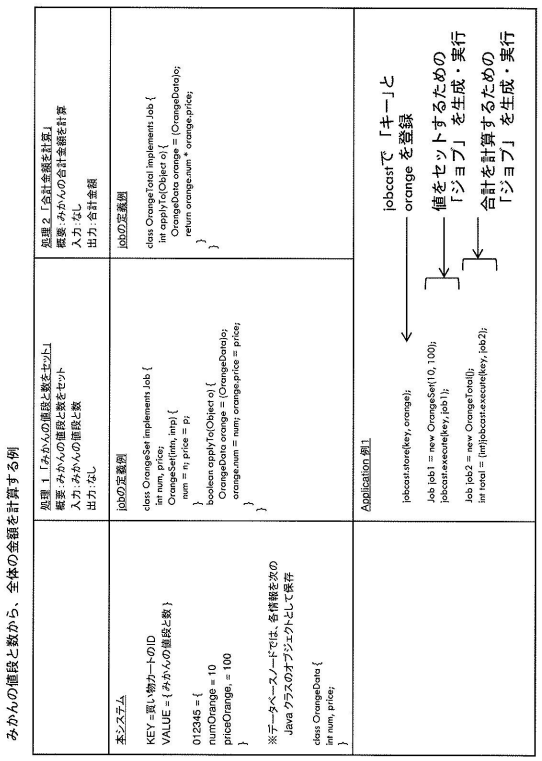
	処理 1.「最新のメールを取得」 概要：メールボックスから最新メールを取得 入力：ユーザID 出力：最新メールの本文	処理 2.「残存メールの数」 概要：削除されていないメールの数をカウント 入力：ユーザID 出力：削除されていないメールの数

本システム
KEY＝個人ID、VALUE＝メッセージ番号＋本文のテーブル
03a9e0 = {
maxId = 950
1 = <mail-1の本文>
2 = <mail-2の本文>
...
950 = <mail-950の本文>
}
※各情報は Map オブジェクト内に格納する
	Job の定義例
	class GetNewMail {
	String applyTo(Objecto o) {
	return ((Map)o).get(o.get("maxId"));
	}
	}
		Job の定義例
		class GetCount{
		Integer applyTo(Objecto o) {
		tail = ((Map)o).get("maxId");
		for (int i = 1; i<= tail; i++) {
		if (((Map)o).get(i) != null) count++;
		}
		return count;
		}
		}

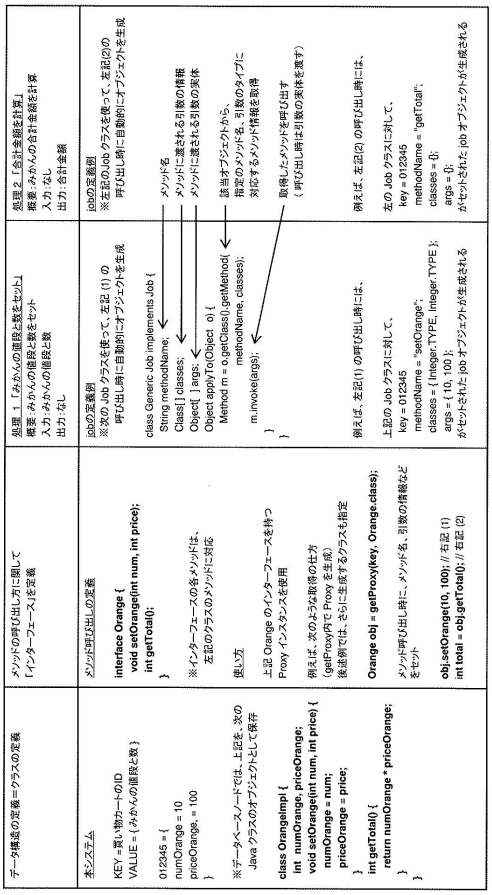
【図１３】



【図１４】

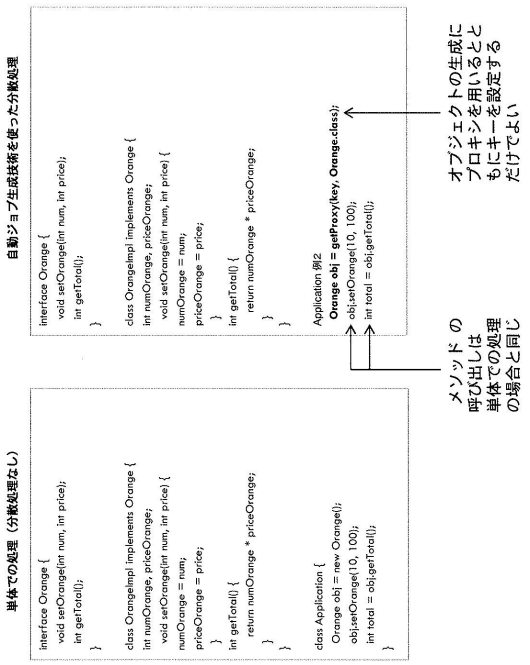


【図１５】



【図１６】

【図１７】



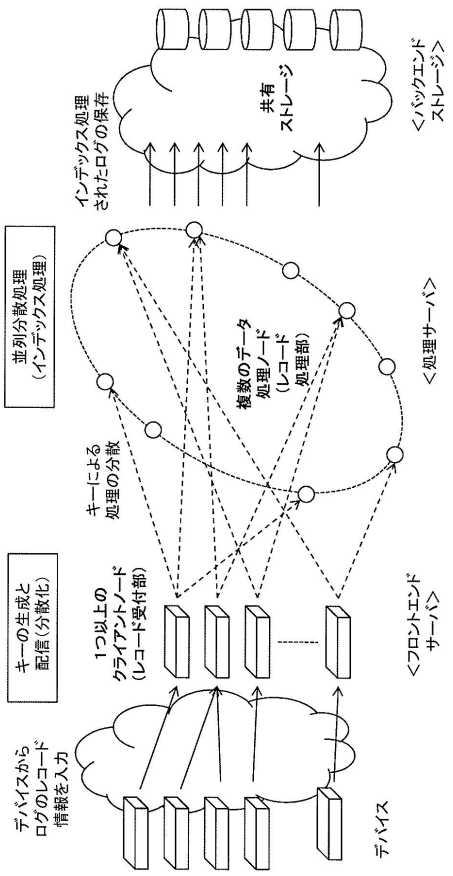
【図１８】

	1回のデータ転送量	複数要素にまたがる処理	秘匿性のあるデータ
本システム	◎（転送量＝小）	◎（通信回数＝少）	◎（結果だけ返す）
従来	◎（転送量＝小）	△（通信回数＝多）	×
参考例	△（転送量＝大）	○（通信回数＝少）	×

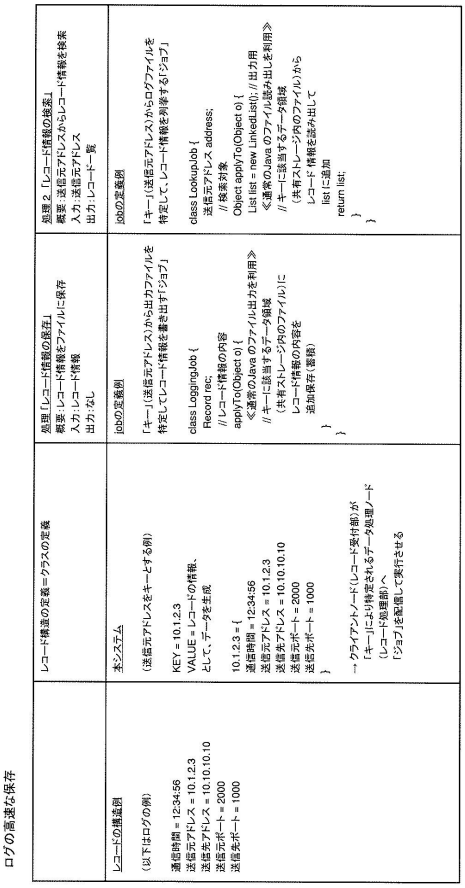
【図１９】

	単一の項目へのアクセス	複数の項目へのアクセス	データの秘匿性
本システム	◎ 通信量：少ない	◎ 通信回数：少ない	◎ バックエンド処理のため 秘匿性のあるデータを ネットワークに流す必要なし
従来	◎ 通信量：少ない	△ 通信回数：多い	× フロントエンド処理のため、 秘匿性のあるデータを ネットワークに流す必要あり
参考例	△ 通信量：多い	○ 通信回数：少ない	

【図２０】



【図２１】


ログの高速な保存
レコード構造の定義＝クラスの定義
処理「レコード情報の保存」
概要：レコード情報をファイルに保存
入力：レコード情報
出力：なし
処理 2「レコード情報の検索」
概要：送信元アドレスからレコード情報を検索
入力：送信元アドレス
出力：レコード一覧
レコードの構造例
（以下はログの例）
通信時間 = 12:34:56
送信元アドレス = 10.1.2.3
送信先アドレス = 10.10.10.10
送信元ポート = 2000
送信先ポート = 1000
本システム
（送信元アドレスをキーとする例）
KEY = 10.1.2.3
VALUE = レコードの情報.
として、データを生成.
10.1.2.3 = {
通信時間 = 12:34:56
送信元アドレス = 10.1.2.3
送信先アドレス = 10.10.10.10
送信元ポート = 2000
送信先ポート = 1000
}
→ クライアントノード（レコード受付部）が
「キー」により特定される データ処理ノード
（レコード処理部）へ
「ジョブ」を配信して実行させる
jobの定義例
「キー」（送信元アドレス）から出力ファイルを
特定して レコード情報を書き出す「ジョブ」
class LoggingJob {
Record rec;
//レコード情報の内容
applyTo(Object o) {
≪過去のJava のファイル出力を利用≫
//キーに該当するデータ領域
（共有ストレージ内のファイル）に
レコード情報の内容を
追加保存（蓄積）
}
}
jobの定義例
「キー」（送信元アドレス）からログファイルを
特定して、レコード情報を列挙する「ジョブ」
class LookupJob {
送信元アドレス address;
// 検索対象
Object applyTo(Object o) {
List list = new LinkedList(); // 出力用
≪過去のJava のファイル読み出しを利用≫
//キーに該当するデータ領域
（共有ストレージ内のファイル）から
レコード情報を読み出して
list に追加
return list;
}
}

フロントページの続き

(56)参考文献 高槻芳, 特選フリーソフト ＲＯＭＡ Ｒｕｂｙで動くキーバリュー型データストア, 日経Ｌｉｎｕｘ, 日本, 日経ＢＰ社, ２０１０年 4月 8日, 第１２巻, 第5号, ｐｐ. ９２－９４
まつもとゆきひろ, まつもとゆきひろ 技術を斬る, 日経Ｌｉｎｕｘ, 日本, 日経ＢＰ社, ２００９年１０月 8日, 第１１巻, 第１１号, ｐｐ. １５０－１５６
西澤無我, ＲＯＭＡとそのプラグイン機構, ＳｏｆｔｗａｒｅＤｅｓｉｇｎ, 日本, （株）技術評論社, ２０１０年 2月１８日, Ｎｏ. ２３２, ｐｐ. ３６－４１
前橋孝広, 大規模データ処理のための分散システムの実装とその応用, 電子情報通信学会論文誌, 日本, 社団法人電子情報通信学会, ２０１０年 7月 1日, 第Ｊ９３－Ｄ巻, 第7号, ｐｐ. １０７２－１０８１

(58)調査した分野(Int.Cl., ＤＢ名)
Ｇ０６Ｆ １２／００
Ｇ０６Ｆ １７／３０