(19)日本国特許庁(JP)

(12)特許公報(B2)

(11)特許番号
特許第7036665号
(P7036665)

(45)発行日　令和4年3月15日(2022.3.15)

(24)登録日　令和4年3月7日(2022.3.7)

(51)国際特許分類

		FI		
G06F	*16/178 (2019.01)*	G06F	16/178	
G06F	*16/182 (2019.01)*	G06F	16/182	
G06F	*16/23 (2019.01)*	G06F	16/23	
G06F	*3/06 (2006.01)*	G06F	3/06	304B

請求項の数　10　（全30頁）

(21)出願番号　特願2018-97972(P2018-97972)
(22)出願日　平成30年5月22日(2018.5.22)
(65)公開番号　特開2019-204216(P2019-204216A)
(43)公開日　令和1年11月28日(2019.11.28)
審査請求日　令和2年11月19日(2020.11.19)

(73)特許権者　000005108
株式会社日立製作所
東京都千代田区丸の内一丁目６番６号
(74)代理人　110001678
特許業務法人藤央特許事務所
(72)発明者　福地 開帆
東京都千代田区丸の内一丁目６番６号
株式会社日立製作所内
(72)発明者　根本 潤
東京都千代田区丸の内一丁目６番６号
株式会社日立製作所内
(72)発明者　早坂 光雄
東京都千代田区丸の内一丁目６番６号
株式会社日立製作所内
審査官　甲斐 哲雄

最終頁に続く

(54)【発明の名称】　データ管理方法およびデータ管理システム

(57)【特許請求の範囲】
【請求項１】
データを格納する記憶部と、プロセッサ上で稼働して前記データを処理する処理部とを各々有する複数のサーバを用い、
前記複数のサーバの記憶部に分散して格納した処理対象情報を、前記複数のサーバがトランザクション要求を受けて処理するデータ管理方法において、
前記サーバの記憶部は、
前記処理対象情報を有する第１のデータと、
前記トランザクション要求に基づいて前記処理対象情報を処理したトランザクション処理履歴情報を含む第２のデータと、
を格納しており、
前記第１のデータ及び第２のデータには、関連するトランザクションが異なる複数のデータが含まれており、
前記サーバには、第１のサーバと、複数の第２のサーバとが含まれており、
前記複数の第２のサーバは、前記第１のデータ及び第２のデータを同期させて保持しており、
前記トランザクション要求を受信した第１のサーバは、前記トランザクション要求にかかる第１のデータを少なくとも１つの前記第２のサーバの前記記憶部から読み出し、前記トランザクション要求を処理して処理結果を前記トランザクション要求の要求元に返信し、
トランザクション処理履歴情報を含む第２のデータと処理結果を反映した前記第１のデー

タとを自サーバに格納するとともに、当該処理を第２のサーバの前記第１のデータに反映させるための第２のデータを前記第２のサーバに送り、
前記第１のサーバは、
前記第１のデータを保持する第２のサーバをオンデマンドデータ取得元サーバとして設定し、
前記オンデマンドデータ取得元サーバは、前記第１のサーバをコミット同期サーバとして設定しており、前記第１のデータについての更新を反映させる第２のデータを前記第１のサーバに送信し、
前記第１のサーバは、前記第１のデータの読み出しでは、前記オンデマンドデータ取得元サーバにより同期されている第１のデータを自サーバの記憶部で検索し、前記自サーバの記憶部で取得できない場合にオンデマンドデータ取得元サーバの記憶部から前記第１のデータを取得する
データ管理方法。
【請求項２】
請求項１において、
前記第１のサーバは、新規に追加されたサーバであることを特徴とするデータ管理方法。
【請求項３】
請求項１において、
前記トランザクション要求にかかるクエリが、前記第１のデータを特定した第１のクエリを含む場合に、前記第１のデータを自サーバの記憶部で検索し、前記第１のクエリとは異なる第２のクエリを含む場合には、前記オンデマンドデータ取得元サーバの記憶部から前記第１のデータを取得する
データ管理方法。
【請求項４】
請求項１において、
前記第１のサーバは、前記第２のデータを保持する第２のサーバをバックグラウンドデータ取得サーバとして設定し、前記バックグラウンドデータ取得サーバから前記第２のデータを取得し、
前記オンデマンドデータ取得元サーバから取得する前記第２のデータのトランザクションは、前記自サーバの有する第１のデータにかかるトランザクションより新しく、
前記バックグラウンドデータ取得サーバから取得する前記第２のデータのトランザクションは、前記自サーバの有する第１のデータにかかるトランザクションより古い、
データ管理方法。
【請求項５】
請求項４において、
前記バックグラウンドデータ取得サーバから取得した第２のデータに基づいて、前記第１のデータを作成する
データ管理方法。
【請求項６】
請求項５において、
前記第２のデータには、ブロック番号が付されており、
前記自サーバは、前記ブロック番号に基づいて、受信した前記第２のデータを前記第１のデータへの反映要否を判断する
データ管理方法。
【請求項７】
請求項１において、
前記第２のデータには、前記第２のデータ同士の関係を示すハッシュ値が含まれており、
前記トランザクション要求の処理を開始する前に、前記第１のサーバは第２のサーバから前記第２のデータを取得しており、
前記第１のサーバは、前記処理の開始後に、前記処理のトランザクション処理履歴情報と

、前記第２のデータから作成したハッシュ値を作成して前記第２のサーバに送る
データ管理方法。
【請求項８】
請求項１において、
前記第２のデータには、ブロック番号が付されており、
前記第１のサーバは、保有する第２のデータのブロック番号を前記オンデマンドデータ取得元サーバに送信し、
前記オンデマンドデータ取得元サーバは、受信した前記ブロック番号に基づいて第２のデータを選択して前記第１のサーバに送信する
データ管理方法。
【請求項９】
請求項１において、
前記第１のサーバは、
前記第１のデータを保持する少なくとも１つの第２のサーバをオンデマンドデータ取得元サーバとして設定し、
複数の前記第２のサーバは、前記第１のサーバをコミット同期サーバとして設定しており、前記第１のデータについての更新を反映させる第２のデータを前記第１のサーバに送信する
データ管理方法。
【請求項１０】
データを格納する記憶部と、プロセッサ上で稼働して前記データを処理する処理部とを各々有する複数のサーバを用い、
前記複数のサーバの記憶部に分散して格納した処理対象情報を、前記複数のサーバがトランザクション要求を受けて処理するデータ管理システムにおいて、
前記サーバの記憶部は、
前記処理対象情報を有する第１のデータと、
前記トランザクション要求に基づいて前記処理対象情報を処理したトランザクション処理履歴情報を含む第２のデータと、
を格納しており、
前記第１のデータ及び第２のデータには、関連するトランザクションが異なる複数のデータが含まれており、
前記サーバには、第１のサーバと、複数の第２のサーバとが含まれており、
前記複数の第２のサーバは、前記第１のデータ及び第２のデータを同期させて保持しており、
前記トランザクション要求を受信した第１のサーバは、前記トランザクション要求にかかる第１のデータを少なくとも１つの前記第２のサーバの前記記憶部から読み出し、前記トランザクション要求を処理して処理結果を前記トランザクション要求の要求元に返信し、
トランザクション処理履歴情報を含む第２のデータと処理結果を反映した前記第１のデータとを自サーバに格納するとともに、当該処理を第２のサーバの前記第１のデータに反映させるための第２のデータを前記第２のサーバに送り、
前記第１のサーバは、
前記第１のデータを保持する第２のサーバをオンデマンドデータ取得元サーバとして設定し、
前記オンデマンドデータ取得元サーバは、前記第１のサーバをコミット同期サーバとして設定しており、前記第１のデータについての更新を反映させる第２のデータを前記第１のサーバに送信し、
前記第１のサーバは、前記第１のデータの読み出しでは、前記オンデマンドデータ取得元サーバにより同期されている第１のデータを自サーバの記憶部で検索し、前記自サーバの記憶部で取得できない場合にオンデマンドデータ取得元サーバの記憶部から前記第１のデータを取得する

データ管理システム。
【発明の詳細な説明】
【技術分野】
【０００１】
本発明は、ブロックチェーンシステムのデータ管理方法に関する。
【背景技術】
【０００２】
複数のサーバでネットワークを構成し、論理的に同一のデータを各サーバが保持しあうブロックチェーンシステムにおいて、新たなサーバをブロックチェーンネットワークに参加させ、トランザクション処理可能な状態とするには、新規サーバが事前にネットワーク上の全てのデータを取得することが必要となる。
【０００３】
しかし、ネットワーク上のデータが大容量である場合、全てのデータの取得が完了するまでに時間を要する。急な負荷上昇時のスケールアウト時には、新規サーバをトランザクション処理可能な状態で迅速に追加する必要がある。そのため、新規サーバがデータの取得に要する時間を短縮することが不可欠である。
【０００４】
分散コンピューティングシステムにおけるデータのリストア時間を短縮する方法としては、例えば、特許文献１が知られている。特許文献１では、新たなサーバが全データ（全ファイル）を予め取得するのではなく、アクセスがあったタイミングでオンデマンドに、ストレージシステムが既存サーバから該当データを取得する技術が開示されている。
【先行技術文献】
【特許文献】
【０００５】
【文献】特表２０１３－５３２３１４号公報
【発明の概要】
【発明が解決しようとする課題】
【０００６】
しかしながら、特許文献１に記載の技術は、ストレージ装置にてファイル単位でのオンデマンドデータ取得を行うため、ブロックチェーンシステムのように、各サーバがブロックチェーンシステムにとって論理的に同一のデータを保持するが、ファイル単位では異なるデータを保持する場合は利用することができない。
【０００７】
ブロックチェーンシステムにとって同一のデータであっても、ブロックチェーンシステムを構成する複数のサーバが異なるファイルシステムを利用する場合、ファイルレベルでは異なるデータの場合があるため、特許文献１に記載の技術をそのまま適用することはできない、という問題があった。
【０００８】
また、複数のサーバからファイルレベルでオンデマンドのデータ取得を行うと、前記特許文献１ではデータのコミットが完了しているか否かに関わらずリストアが進行するため、データの整合性が失われてしまう場合があった。
【０００９】
そこで本発明は、上記問題点に鑑みてなされたもので、ブロックチェーンシステムにおいて、新規サーバの追加を高速化し、可用性を向上させることを目的とする。
【課題を解決するための手段】
【００１０】
本発明は、データを格納する記憶部と、プロセッサ上で稼働して前記データを処理する処理部とを各々有する複数のサーバを用い、前記複数のサーバの記憶部に分散して格納した処理対象情報を、前記複数のサーバがトランザクション要求を受けて処理するデータ管理方法において、前記サーバの記憶部は、前記処理対象情報を有する第１のデータと、前記

トランザクション要求に基づいて前記処理対象情報を処理したトランザクション処理履歴情報を含む第２のデータと、を格納しており、前記第１のデータ及び第２のデータには、関連するトランザクションが異なる複数のデータが含まれており、前記サーバには、第１のサーバと、複数の第２のサーバとが含まれており、前記複数の第２のサーバは、前記第１のデータ及び第２のデータを同期させて保持しており、前記トランザクション要求を受信した第１のサーバは、前記トランザクション要求にかかる第１のデータを少なくとも１つの前記第２のサーバの前記記憶部から読み出し、前記トランザクション要求を処理して処理結果を前記トランザクション要求の要求元に返信し、トランザクション処理履歴情報を含む第２のデータと処理結果を反映した前記第１のデータとを自サーバに格納するとともに、当該処理を第２のサーバの前記第１のデータに反映させるための第２のデータを前記第２のサーバに送り、前記第１のサーバは、前記第１のデータを保持する第２のサーバをオンデマンドデータ取得元サーバとして設定し、前記オンデマンドデータ取得元サーバは、前記第１のサーバをコミット同期サーバとして設定しており、前記第１のデータについての更新を反映させる第２のデータを前記第１のサーバに送信し、前記第１のサーバは、前記第１のデータの読み出しでは、前記オンデマンドデータ取得元サーバにより同期されている第１のデータを自サーバの記憶部で検索し、前記自サーバの記憶部で取得できない場合にオンデマンドデータ取得元サーバの記憶部から前記第１のデータを取得する。

【発明の効果】

【００１１】

本発明によれば、複数のサーバが論理的に同一のデータを保持するブロックチェーンシステムにおいて、トランザクション処理可能な新規サーバの追加を高速化し、可用性を向上させることができる。

【図面の簡単な説明】

【００１２】

【図１】本発明の実施例１を示し、本発明を適用したブロックチェーンシステムの概要を示すブロック図である。

【図２】本発明の実施例１を示し、ブロックチェーンシステムの構成の一例を示すブロック図である。

【図３】本発明の実施例１を示し、ブロックチェーンプログラムの内部構成を示すブロック図である。

【図４】本発明の実施例１を示し、稼働モードフラグ情報の構成を示す図である。

【図５】本発明の実施例１を示し、データベースアクセスインタフェースの一例を示す図である。

【図６】本発明の実施例１を示し、ブロックチェーンプログラムのトランザクション処理の一例を示すフローチャートである。

【図７】本発明の実施例１を示し、ブロックデータの構成の一例を示す図である。

【図８】本発明の実施例１を示し、オンデマンドデータ取得処理の一例を示すフローチャートである。

【図９】本発明の実施例１を示し、ブロックデータ確定取得処理の一例を示すフローチャートである。

【図１０】本発明の実施例１を示し、バージョン判定処理の一例を示すフローチャートである。

【図１１】本発明の実施例１を示し、ブロックデータバックグラウンド取得処理の一例を示すフローチャートである。

【図１２】本発明の実施例１を示し、オンデマンドデータ取得の初期化処理の一例を示すフローチャートである。

【図１３】本発明の実施例１を示し、保持データ管理情報の構成を示す図である。

【図１４】本発明の実施例１を示し、オンデマンドデータ取得進捗管理情報の構成を示す図である。

【図１５】本発明の実施例２を示し、オンデマンドデータ取得の初期化処理の一例を示す

フローチャートの前半部である。
【図１６】本発明の実施例２を示し、オンデマンドデータ取得の初期化処理の一例を示すフローチャートの後半部である。
【図１７】本発明の実施例２を示し、稼働モードフラグ情報の構成を示す図である。
【図１８】本発明の実施例３を示し、ブロック確定処理の一例を示すフローチャートである。
【図１９】本発明の実施例３を示し、オンデマンドデータ取得の初期化処理の一例を示すフローチャートの前半部である。
【図２０】本発明の実施例３を示し、オンデマンドデータ取得の初期化処理の一例を示すフローチャートの後半部である。
【図２１】本発明の実施例３を示し、オンデマンドデータ取得処理の一例を示すフローチャートである。
【発明を実施するための形態】
【００１３】
以下、本発明の実施形態を添付図面に基づいて説明する。
【実施例１】
【００１４】
図１は、本発明の実施例１のブロックチェーンシステムの概要を示すブロック図である。図示の例では、既存のサーバ２２０－１とクライアント２００で構成されたブロックチェーンシステムに、新たなサーバ２２０－２を追加する例を示す。サーバ２２０－１とサーバ２２０－２は、同様に構成されて、ブロックチェーンプログラム３００がそれぞれ稼働する。ブロックチェーンプログラム３００は、スマートコントラクト３１０と、オンデマンドデータ取得モジュール３４０と、データベース２８５を含む。
【００１５】
まず、クライアント２００が、トランザクション要求を、新規追加したサーバ２２０－２上のブロックチェーンプログラム３００に発行する。次に、新規追加したサーバ２２０－２上のブロックチェーンプログラム３００は、スマートコントラクト３１０を実行し、ブロックチェーンシステムに参加してクライアント２００から要求された処理（例えば、契約や情報の更新）を実行する。オンデマンドデータ取得モジュール３４０は、スマートコントラクト３１０からデータベース２８５へのアクセスをフックする。
【００１６】
なお、図２では、オンデマンドデータ取得モジュール３４０がデータベース２８５へのアクセスをフックする例を示すが、後述の処理では、トランザクション処理モジュール３２０がデータベース２８５へのアクセスをフックして、オンデマンドデータ取得モジュール３４０に実行させる。
【００１７】
すなわち、スマートコントラクト３１０からデータベース２８５へのアクセスのフックは、トランザクション処理モジュール３２０またはオンデマンドデータ取得モジュール３４０のいずれか一方で実施すればよい。
【００１８】
オンデマンドデータ取得モジュール３４０は、上記フックしたアクセスについて、取得要求を受けたアクセス先データ（図中データ２）が自サーバ２２０－２上に存在するか否かを判定する。アクセスするデータが存在しない場合、新規のサーバ２２０－２上のオンデマンドデータ取得モジュール３４０は、スマートコントラクト３１０から取得要求を受けたデータの取得要求を、既存のサーバ２２０－１上のオンデマンドデータ取得モジュール３４０に送信する。
【００１９】
既存のサーバ２２０－１上のオンデマンドデータ取得モジュール３４０は、データの取得要求を受信すると、要求されたデータをデータベース２８５内のブロックチェーンデータ２９０から取得し、新規のサーバ２２０－２上のオンデマンドデータ取得モジュール３４

０にデータを渡す（オンデマンドデータ取得処理）。
【００２０】
新規のサーバ２２０－２上のオンデマンドデータ取得モジュール３４０は、受け取ったデータを必要に応じて自身のデータベース２８５のブロックチェーンデータ２９０に格納したのち、スマートコントラクト３１０にデータを渡す。
【００２１】
上記処理によって、新規に追加されたサーバ２２０－２は、オンデマンドデータ取得モジュール３４０を介して既存のサーバ２２０－１からスマートコントラクト３１０で利用するデータをオンデマンドで取得することができる。すなわち、追加されたサーバ２２０－２のオンデマンドデータ取得モジュール３４０は、既存のサーバ２２０－１のファイルシステムやデータベースの種類に関わらず、ブロックチェーンシステムのスマートコントラクト３１０で利用するデータを取得することが可能となる。
【００２２】
図２は、本発明の実施例１に係るブロックチェーンシステムの概略構成を示すブロック図である。ブロックチェーンシステムは、１つ以上のクライアント２００と、ネットワーク２１０を介してクライアント２００と接続された１つ以上のサーバ２２０－１～２２０－ｎと、管理端末２２５と、を含む計算機システムである。なお、以下ではサーバ２２０－１～２２０－ｎを個々に特定しない場合には、「－」以降を省略した符号「２２０」を使用する。他の構成要素の符号についても同様である。また、図示の例では、サーバ２２０－２を新規に追加したサーバとした例を示す。
【００２３】
クライアント２００は、１つ以上のサーバ２２０が提供するブロックチェーンサービスを利用するために使用する計算機である。クライアント２００では、ブロックチェーンサービスを利用するためのクライアントプログラムが稼働する。クライアントプログラムをサーバ２２０で稼働させることで、クライアント２００とサーバ２２０を兼用してもよい。また、クライアントプログラムを管理端末２２５で稼働させることで、クライアント２００と管理端末２２５とを兼用してもよい。
【００２４】
ネットワーク２１０は、クライアント２００と、サーバ２２０と、を相互に接続するネットワークである。ネットワーク２１０は、例えば、ＷＡＮ（Ｗｉｄｅ　Ａｒｅａ　Ｎｅｔｗｏｒｋ）、ＬＡＮ（Ｌｏｃａｌ　Ａｒｅａ　Ｎｅｔｗｏｒｋ）、インターネット、ＳＡＮ（Ｓｔｏｒａｇｅ　Ａｒｅａ　Ｎｅｔｗｏｒｋ）、公衆回線、または専用回線などである。
【００２５】
サーバ２２０は、クライアント２００に対してブロックチェーンサービスを提供する計算機である。サーバ２２０は、メモリ２７０に格納されたプログラムを実行するＣＰＵ２４０と、クライアント２００との通信に使用するネットワークインタフェース２５０と、ディスクドライブ（記憶部）２８０と、ディスクドライブ２８０への入出力を制御するディスクコントローラ２６０と、プログラムやデータを格納するメモリ２７０と、を搭載し、それらを内部的な通信路（例えば、バス２１５）によって接続している計算機である。なお、サーバ２２０は、仮想マシンで構成されてもよい。
【００２６】
サーバ２２０のメモリ２７０には、プログラムやデータが格納される。例えば、メモリ２７０には、ブロックチェーンプログラム３００と、データベース２８５を管理するデータベースプログラム２９５が格納される。本実施例１は、ブロックチェーンサービスを提供する主体が、例としてブロックチェーンプログラム３００であることを前提に説明する。
【００２７】
ブロックチェーンプログラム３００は、クライアント２００に対して、他のサーバ２２０におけるブロックチェーンプログラム３００と協調してスマートコントラクトをサービスし、クライアント２００から受信したトランザクション要求に基づいてスマートコントラクトを実行する。

【００２８】
ディスクコントローラ２６０は、メモリ２７０に格納された各種プログラムの入出力要求に基づいて、ディスクドライブ２８０のデータを例えばブロック単位で入出力する。
【００２９】
ディスクドライブ２８０は、メモリ２７０に格納された各種プログラムが読み書きするデータを格納するための記憶装置である。本実施例１において、ディスクドライブ２８０には、ブロックチェーンデータ２９０が格納される。ブロックチェーンデータ２９０は、データベースプログラム２９５を利用して、Ｋｅｙ－Ｖａｌｕｅ形式に構造化されたフォーマットでデータベース２８５に格納されてもよい。
【００３０】
管理端末２２５は、ブロックチェーンシステムの管理者が、ブロックチェーンプログラム３００の設定を変更するとき等に利用される。管理端末２２５は、サーバ２２０と同様のモジュール（ＣＰＵ２４０、ネットワークインタフェース２５０、ディスクコントローラ２６０、ディスクドライブ２８０、メモリ２７０、バス２１５）を含む計算機である。
【００３１】
管理端末２２５では管理用プログラムが実行され、ブロックチェーンシステムの管理者は管理用プログラムを使用することで、ブロックチェーンプログラム３００の設定を変更する。
【００３２】
図３は、ブロックチェーンプログラム３００の機能構成を示すブロック図である。ブロックチェーンプログラム３００は、スマートコントラクト（スマートコントラクト処理部）３１０と、トランザクション処理モジュール３２０と、ブロック受信モジュール３３０と、オンデマンドデータ取得モジュール（オンデマンドデータ取得部）３４０と、バックグラウンド取得モジュール３５０と、データベースアクセスモジュール３６０と、保持データ管理情報１３００と、オンデマンドデータ取得進捗管理情報３７０と、稼働モードフラグ情報４００と、を含む。
【００３３】
スマートコントラクト３１０は、サーバ２２０のＣＰＵ２４０によって実行される。スマートコントラクト３１０は、例えば、仮想通貨や証券といった金融資産の取引を処理するためのプログラムである。なお、スマートコントラクト３１０は複数種存在してもよい。
【００３４】
トランザクション処理モジュール３２０は、クライアント２００からのトランザクション要求を契機として、サーバ２２０のＣＰＵ２４０によって実行される。トランザクション処理モジュール３２０は、クライアント２００からトランザクション要求を受信し、当該トランザクション要求内容に基づいて対応するスマートコントラクト３１０を実行する。さらに、トランザクション処理モジュール３２０は、実行結果をサーバ２２０に分配し、確定した上で、クライアント２００にトランザクション処理結果を応答する。
【００３５】
ブロック受信モジュール３３０は、他のサーバ２２０のトランザクション処理モジュール３２０が送信するブロックデータを受信し、データベースにブロックデータと、当該ブロックデータに含まれるトランザクションの実行結果をコミットする。これによって、ブロックチェーンデータ２９０が更新される。
【００３６】
オンデマンドデータ取得モジュール３４０は、スマートコントラクト３１０のデータベースアクセスを契機としてサーバ２２０のＣＰＵ２４０にて実行される。オンデマンドデータ取得モジュール３４０は、スマートコントラクト３１０が必要とするデータ（Ｋｅｙ／Ｖａｌｕｅ）を、自サーバ２２０が保持しているか否かを判定し、保持していない場合は既存のサーバ２２０－１上のオンデマンドデータ取得モジュール３４０にデータの取得要求を送信し、既存のサーバ２２０－１から必要なデータを取得する。
【００３７】

バックグラウンド取得モジュール３５０は、タイマなどでバックグラウンド取得処理（図１１のＳ１１００）を所定の周期で実行する。バックグラウンド取得モジュール３５０は、サーバ２２０自身が保持していないブロックデータを、他のサーバ２２０から取得する。なお、バックグラウンドでのブロックデータの取得は、実施してもしなくてもよい。

【００３８】

バックグラウンドでのブロックデータの取得を実施することで、最終的に、新規のサーバ２２０－２は既存のサーバ２２０－１と同様の全データを保持することができる。ブロックデータ７００は、ブロックチェーンデータへの処理の履歴を格納しているので、これを用いて過去のブロックチェーンデータを作成することができるからである。一方で、全データが不要な場合は実施しなくてもよい。

【００３９】

また、バックグラウンド取得処理の実施の有無（稼働モードフラグ情報４００におけるバックグラウンド取得モードＣ４５０）や、タイマの間隔はブロックチェーンシステム管理者が管理端末２２５を利用して設定することができる。

【００４０】

データベースアクセスモジュール３６０は、スマートコントラクト３１０が、データベース２８５に読み書きする際に利用するものである。データベースアクセスモジュール３６０は、スマートコントラクト３１０がデータベースにアクセスするための統一的なインタフェースを提供する。そのため、スマートコントラクト３１０はデータベースの種別の違い等を意識せずにデータベース２８５へのアクセスが可能となる。

【００４１】

保持データ管理情報１３００は、オンデマンドデータ取得によりどのＫｅｙが取得済みであるかを管理するためのテーブルである。このテーブルは、オンデマンドデータ取得処理が実施されると、オンデマンドデータ取得モジュール３４０により更新される。

【００４２】

オンデマンドデータ取得進捗管理情報１４００は、オンデマンドデータ取得を開始したときの最新ブロックデータのブロック番号や、バックグラウンド取得モジュール３５０がどの番号のブロックデータまでを取得したか否かの管理を行うための情報で、メモリ２７０またはディスクドライブ２８０上に格納される。オンデマンドデータ取得進捗管理情報１４００は、バックグラウンド取得モジュール３５０のバックグラウンド取得処理の終了判定（図１１のステップＳ１１５０）で利用される。

【００４３】

スマートコントラクト３１０と、トランザクション処理モジュール３２０と、ブロック受信モジュール３３０と、オンデマンドデータ取得モジュール３４０と、バックグラウンド取得モジュール３５０と、データベースアクセスモジュール３６０の各機能部はブロックチェーンプログラム３００の要素としてメモリ２０２にロードされる。

【００４４】

ＣＰＵ２４０は、各機能部のプログラムに従って処理することによって、所定の機能を提供する機能部として稼働する。例えば、ＣＰＵ２４０は、トランザクション処理プログラムに従って処理することでトランザクション処理モジュール３２０として機能する。他のプログラムについても同様である。さらに、ＣＰＵ２４０は、各プログラムが実行する複数の処理のそれぞれの機能を提供する機能部としても稼働する。計算機および計算機システムは、これらの機能部を含む装置およびシステムである。

【００４５】

制御部１１０の各機能を実現するプログラム、テーブル等の情報は、ディスクドライブ２８０や不揮発性半導体メモリ、ＳＳＤ（Ｓｏｌｉｄ Ｓｔａｔｅ Ｄｒｉｖｅ）等の記憶デバイス、または、ＩＣカード、ＳＤカード、ＤＶＤ等の計算機読み取り可能な非一時的データ記憶媒体に格納することができる。

【００４６】

図４は、稼働モードフラグ情報４００の構成を示す図である。稼働モードフラグ情報４０

０は、オンデマンドデータ取得を実施するか否かや、オンデマンドデータの取得元や、コミット同期を実施するかどうかや、バックグラウンド取得を実施するかどうかや、バックグラウンド取得の取得元などを格納する。

【００４７】

稼働モードフラグ情報４００は、メモリ２７０またはディスクドライブ２８０上に格納される。稼働モードフラグ情報４００は、オンデマンドデータ取得モードＣ４１０と、オンデマンドデータ取得元Ｃ４２０と、コミット同期モードＣ４３０とコミット同期先Ｃ４４０と、バックグラウンド取得モードＣ４５０と、バックグラウンドデータ取得元Ｃ４６０と、から構成される。

【００４８】

オンデマンドデータ取得モードＣ４１０は、オンデマンドデータ取得モジュール３４０が、オンデマンドデータ取得処理を行うべきか否かを示し、ブロックチェーンシステム管理者が、管理端末２２５を利用して値を変更する。オンデマンドデータ取得モードＣ４１０は、オンデマンドデータ取得処理を実行する場合には「ｔｒｕｅ」が設定され、実行しない場合には「ｆａｌｓｅ」が設定される。

【００４９】

オンデマンドデータ取得元Ｃ４２０は、オンデマンドデータ取得モジュール３４０が、どのサーバ２２０からオンデマンドデータ取得処理を行うかを示す情報である。本実施例１では、サーバ２２０のＩＰアドレスとポート番号の組み合わせでデータを取得する計算機を特定する例を示す。

【００５０】

オンデマンドデータ取得モジュール３４０は、オンデマンドデータ取得モードＣ４１０と、オンデマンドデータ取得元Ｃ４２０とを利用し、オンデマンドデータ取得処理の実施の有無の判定と、オンデマンドデータ取得元の特定を行う。

【００５１】

コミット同期モードＣ４３０は、ブロックチェーンプログラム３００が、ブロックデータのコミット時に、他サーバ２２０にブロックデータを同期的に送信するか否かを指定するものである。ブロックデータを同期的に送信する場合には「ｔｒｕｅ」が設定され、送信しない場合には「ｆａｌｓｅ」が設定される。

【００５２】

コミット同期先Ｃ４４０は、ブロックチェーンプログラム３００のブロックデータコミット時に、どのサーバ２２０にブロックデータを同期的に送信するかを指定するものである。本実施例１では、サーバ２２０のＩＰアドレスとポート番号の組み合わせでブロックデータを送信する計算機を特定する。

【００５３】

バックグラウンド取得モードＣ４５０は、バックグラウンド取得モジュール３５０が、バックグラウンド取得処理を実施するかを指定する。バックグラウンド取得処理を実施する場合には「ｔｒｕｅ」が設定され、実施しない場合には「ｆａｌｓｅ」が設定される。

【００５４】

バックグラウンドデータ取得元Ｃ４６０は、バックグラウンド取得モジュール３５０が、どのサーバ２２０からブロックデータを取得するかを指定する。バックグラウンドデータ取得元Ｃ４６０には、ブロックデータを取得するサーバ２２０のＩＰアドレスとポート番号が格納される。

【００５５】

図５は、データベースアクセスインタフェース５００の一例を示す図である。データベースアクセスインタフェース５００は、データベースアクセスモジュール３６０で管理され、スマートコントラクト３１０がデータベース２８５へアクセスする際のインタフェースである。スマートコントラクト３１０は、データベースアクセスインタフェース５００を利用してデータベース２８５にデータの入出力を行う。

【００５６】

インタフェース名Ｃ５１０は各インタフェースの名前である。インタフェース種別Ｃ５２０は各インタフェースがどのような入力を受け付けるかによりインタフェースを分類するもので、例えば、単一のＫｅｙのみを入力に取るものは「単一Ｋｅｙクエリ」が設定され、２つのＫｅｙを入力に受け取り、それらが始点と終点を表す場合は「範囲Ｋｅｙクエリ」が設定され、入力に正規表現などの任意の文字列を受け付けるものは「リッチクエリ」が設定される。

【００５７】

入力Ｃ５３０は各インタフェースが入力に受け付ける値の種類が設定される。例えば、インタフェース名Ｃ５１０＝「Ｇｅｔ」は、入力に単一のＫｅｙを受け付ける。この他に、「Ｖａｌｕｅ」や「Ｓｔａｒｔｋｅｙ」および「Ｅｎｄｋｅｙ」や「ＱｕｅｒｙＳｔｒｉｎｇ」などが設定される。

【００５８】

出力Ｃ５４０は、各インタフェースの出力内容である。例えば、インタフェース名Ｃ５１０＝「ＧｅｔＲａｎｇｅ」は、複数のＫｅｙと、複数のＶａｌｕｅと、処理結果が成功か失敗かを示すＲｅｓｕｌｔとを出力する。

【００５９】

図６は、ブロックチェーンプログラム３００のトランザクション処理モジュール３２０が実行するトランザクション処理の詳細を説明するためのフローチャートの一例である。トランザクション処理は、ブロックチェーンプログラム３００がクライアント２００からのトランザクション要求を受け付けたことを契機に実行される。

【００６０】

まず、ブロックチェーンプログラム３００のトランザクション処理モジュール３２０は、クライアント２００からトランザクション要求を受信する（Ｓ６１０）。トランザクション要求には、スマートコントラクト３１０の実行時に指定するパラメータが含まれる。

【００６１】

次に、トランザクション処理モジュール３２０は、トランザクション要求で指定されたスマートコントラクト３１０を実行する（Ｓ６２０）。スマートコントラクト３１０は、トランザクション要求を実行するために必要なブロックチェーンデータ２９０を取得するアクセス（アクセス要求）をデータベース２８５へ発行する。

【００６２】

トランザクション処理モジュール３２０は、データベースアクセスモジュール３６０を利用するスマートコントラクト３１０からデータベース２８５へのアクセスを全てフックする（Ｓ６３０）。

【００６３】

上記フックしたデータベース２８５へのアクセスについて、トランザクション処理モジュール３２０は、オンデマンドデータ取得モジュール３４０を実行させて、既存のサーバ２２０からデータの取得または自身のサーバ２２０のデータベース２８５が保持するデータ（ブロックチェーンデータ２９０）の読み込みを実施させる（Ｓ８００）。

【００６４】

スマートコントラクト３１０は、オンデマンドデータ取得モジュール３４０が取得したデータでトランザクション要求に対応する処理を実行する。ここで、スマートコントラクト３１０の実行結果、すなわち、戻り値やブロックチェーンデータ２９０に反映するデータは、合意形成処理で使用するためメモリ２７０に退避しておく。

【００６５】

そして、トランザクション処理モジュール３２０は、合意形成処理を実施する（Ｓ６４０）。合意形成処理は、複数のサーバ２２０間で同一のトランザクションの実行結果をコミットするために行われる。例えば、トランザクション処理モジュール３２０は、Ｐｒａｃｔｉｃａｌ Ｂｙｚａｎｔｉｎｅ Ｆａｕｌｔ Ｔｏｌｅｒａｎｃｅプロトコルを利用し、複数のサーバ２２０が同一のトランザクションの実行結果を生成することを保証する。

【００６６】

次に、トランザクション処理モジュール３２０は、前記ステップＳ６４０で合意形成したトランザクションの実行結果を元に、トランザクションの実行結果を含むブロックデータ７００を生成し、他のサーバ２２０に生成したブロックデータを配信する（Ｓ６５０）。

【００６７】

そして、トランザクション処理モジュール３２０は、スマートコントラクト３１０の実行結果をブロックチェーンデータ２９０に書き込む（Ｓ９００）。

【００６８】

最後に、トランザクション処理モジュール３２０は、トランザクション要求に対する応答をクライアント２００に送信する（Ｓ６６０）。

【００６９】

以上の処理により、オンデマンドデータ取得モジュール３４０が、スマートコントラクト３１０からデータベース２８５へのアクセスを全てフックし、自サーバ２２０または他のサーバ２２０のデータベースから必要なデータを取得することで、ブロックチェーンシステムに参加するサーバ２２０のファイルシステムやデータベースの種類が異なる場合でも、確実にデータを取得することが可能となる。そして、オンデマンドデータ取得モジュール３４０は取得したデータをスマートコントラクト３１０に渡すことで、トランザクション処理を円滑に実施することができる。

【００７０】

図７は、トランザクション実行結果が格納されるブロックデータ７００の構成要素の一例を示した図である。

【００７１】

ブロックデータ７００は、１つまたは複数のトランザクション実行結果から生成され、ブロックチェーンプログラム３００がこのブロックデータをコミットする（Ｓ９００）ことで、トランザクションの実行結果がデータベース２８５に格納される。

【００７２】

ブロックデータ７００は、読み書き対象Ｋｅｙ　Ｃ７１０と、読み書き種別Ｃ７２０と、Ｖａｌｕｅ　Ｃ７３０と、ブロック番号Ｃ７４０と、トランザクション番号Ｃ７５０を１つのエントリに含み、さらにハッシュ値Ｃ７６０を含む。

【００７３】

読み書き対象ｋｅｙ　Ｃ７１０は、各トランザクションがデータベース２８５のどのＫｅｙに対して読み書きしたかを示す。読み書き種別Ｃ７２０は、読み書き対象ＫｅｙＣ　７１０のＫｅｙに対して、読み込みを行ったのか書き込みを行ったのかを示す。ＶａｌｕｅＣ７３０は、読み書き対象Ｋｅｙ　Ｃ７１０のＫｅｙについて読み込んだ値もしくは書き込んだ値を示す。

【００７４】

ブロック番号Ｃ７４０は、このトランザクションの実行結果が含まれるブロックの番号を示す。ブロック番号は一意な数値であり、例えば、新たなブロックデータ７００をブロックチェーンプログラム３００が生成するとき、前回のブロックデータ７００のブロック番号Ｃ７４０よりも１つ大きなブロック番号Ｃ７４０を採用する。なお、１つのブロックデータ７００において、このブロック番号Ｃ７４０は全て同一の値を保持する。トランザクション番号Ｃ７５０は、このブロックにおいて、トランザクションが何番目のものであるかを示す。

【００７５】

ハッシュ値Ｃ７６０には、当該ブロックデータの値と前回のブロックのハッシュ値から生成されたハッシュ値が格納される。新規サーバは、オンデマンドデータ取得の初期化時に、最新のブロックデータ７００を取得している。それを用いて、次のブロックデータのためのハッシュ値を生成することができる。

【００７６】

図８は、オンデマンドデータ取得モジュール３４０が実行する、オンデマンドデータ取得処理の詳細を説明するためのフローチャートの一例である。この処理は、図６のステップ

Ｓ８００で行われる。
【００７７】
スマートコントラクト３１０のデータベース２８５に対するアクセスをブロックチェーンプログラム３００上のトランザクション処理モジュール３２０がフックすると、トランザクション処理モジュール３２０はオンデマンドデータ取得モジュール３４０を利用してオンデマンドデータ取得処理を実施する。
【００７８】
なお、トランザクション処理モジュール３２０は、フックした情報に基づいてオンデマンドデータ取得モジュール３４０に、インタフェース名Ｃ５１０とその入力Ｃ５３０の情報を渡す。
【００７９】
まず、オンデマンドデータ取得モジュール３４０は、データベース２８５へのアクセス種別が読み込みであるか書き込みであるかを判定する（Ｓ８０１）。オンデマンドデータ取得モジュール３４０は、読み込みアクセスであった場合、ステップＳ８０２へ進み、書き込みであればステップＳ８０９に進む。
【００８０】
次に、ステップＳ８０２で、オンデマンドデータ取得モジュール３４０は、自身を実行中のブロックチェーンプログラム３００がオンデマンドデータ取得モードで稼働中であるか否かを判定する（Ｓ８０２）。オンデマンドデータ取得モジュール３４０は、ブロックチェーンプログラム３００がオンデマンドデータ取得モードで実行されている場合には、ステップＳ８０３に進み、そうでない場合にはステップＳ８０８へ進む。
【００８１】
ステップＳ８０８ではオンデマンドデータ取得モードでないので、オンデマンドデータ取得モジュール３４０は要求されたＫｅｙに対するＶａｌｕｅを、自身を実行中のサーバ２２０上のデータベース２８５から読み込む。
【００８２】
一方、ステップＳ８０３ではオンデマンドデータ取得モードなので、オンデマンドデータ取得モジュール３４０は、次に、データベース２８５へのアクセスに利用するインタフェース種別Ｃ５２０が単一Ｋｅｙクエリまたは範囲Ｋｅｙクエリであるかを判定する（Ｓ８０３）。
【００８３】
単一Ｋｅｙクエリまたは範囲Ｋｅｙクエリである場合、ステップＳ８０４に進んで、オンデマンドデータ取得モジュール３４０は、スマートコントラクト３１０がどのＫｅｙのＶａｌｕｅを必要としているのか判定可能である。
【００８４】
そのため、オンデマンドデータ取得モジュール３４０は、単一Ｋｅｙクエリまたは範囲Ｋｅｙクエリである場合、必要とするＫｅｙのＶａｌｕｅを自サーバ２２０のデータベース２８５が保持しているか否かを、保持データ管理情報１３００を参照して判定する（Ｓ８０４）。
【００８５】
オンデマンドデータ取得モジュール３４０は、自サーバ２２０のデータベース２８５にアクセス対象のデータが格納されていれば、ステップＳ８０８へ進んで、当該データをデータベース２８５から読み込みを行う。自サーバ２２０にアクセス対象のデータが存在しない場合、オンデマンドデータ取得モジュール３４０は、ステップＳ８０５に進んで、インタフェース名Ｃ５１０とその入力Ｃ５３０の情報を加えて、稼働モードフラグ情報４００のオンデマンドデータ取得元Ｃ４２０に記されたサーバ２２０に対して、オンデマンドデータ取得要求を送信する。
【００８６】
オンデマンドデータ取得要求を受信した他のサーバ２２０上のオンデマンドデータ取得モジュール３４０は、受け取ったインタフェース名Ｃ５１０とその入力Ｃ５３０の情報を元

にデータベース２８５へのアクセスを行い、アクセス対象のデータを取得してオンデマンドデータ取得要求の結果として送信する（Ｓ８０６）。

【００８７】

オンデマンドデータ取得モジュール３４０は、他のサーバ２２０からオンデマンドデータ取得要求の結果を受信すると（Ｓ８０７）、保持データ管理情報１３００を更新する（Ｓ８１２）。

【００８８】

上記ステップＳ８０３の判定にて、データベース２８５へのアクセスに利用するインタフェース種別Ｃ５２０が単一Ｋｅｙクエリまたは範囲Ｋｅｙクエリでない場合、オンデマンドデータ取得モジュール３４０は、ステップＳ８１０に進んで、既存のサーバ２２０にクエリの実行要求を送信する（Ｓ８１０）。

【００８９】

クエリ実行要求を受け取った既存のサーバ２２０は、当該クエリを実行し、クエリの結果を送信する。オンデマンドデータ取得モジュール３４０は、結果を受け取ると（Ｓ８１１）、ステップＳ８０７の処理は実施せずに、データ取得結果をスマートコントラクト３１０に返して保持データ管理情報１３００を更新する（Ｓ８１２）。上記ステップＳ８０７の処理を実施しないのは、単一Ｋｅｙクエリまたは範囲Ｋｅｙクエリでない場合、オンデマンドデータ取得モジュール３４０は、スマートコントラクト３１０がどのＫｅｙのＶａｌｕｅを必要としているのか判定できず、保持データ管理情報１３００を更新できないためである。

【００９０】

なお、ステップＳ８０１にて、書き込みアクセスであった場合は、書き込み処理（Ｓ８０９）と、保持データ管理情報１３００への書き込んだＫｅｙの登録を行い（Ｓ８０７）、書き込み完了の応答をスマートコントラクト３１０へ返す（Ｓ８１２）。

【００９１】

上記処理により、オンデマンドデータ取得モジュール３４０は、データベース２８５へのアクセスが読み込みで、自サーバ２２０にアクセス対象のデータがない場合には、他のサーバ２２０にデータ取得要求またはクエリ実行要求を送信することで、当該データを取得してスマートコントラクト３１０へ渡すことができる。

【００９２】

また、オンデマンドデータ取得モジュール３４０は、クエリの種別毎に処理を分けて既存のサーバ２２０－１に処理を依頼することで、いずれのＫｅｙを必要とするのか判断できないデータベース２８５に固有のクエリは全て既存のサーバ２２０－１に処理を依頼することで、正確な実行結果を得ることができる。

【００９３】

図９は、ブロックチェーンプログラム３００のトランザクション処理モジュール３２０がブロックデータ７００をデータベース２８５にコミットするときに実行するブロック確定処理の詳細を説明するためのフローチャートの一例である。この処理は、図６のステップＳ９００で行われる。

【００９４】

まず、ブロックチェーンプログラム３００は、稼働モードフラグ情報４００を参照してコミット同期モードＣ４３０が「ｔｒｕｅ」であるか否かを取得し、自サーバ２２０がコミット同期モードで稼働中か否かを判定する（Ｓ９１０）。

【００９５】

コミット同期モードで稼働中の場合、ブロックチェーンプログラム３００は、自身がコミット処理中のブロックデータ７００を、稼働モードフラグ情報４００のコミット同期先Ｃ４４０で指定されたサーバ２２０に送信し、応答を待つ（Ｓ９２０）。なお、ブロックデータ７００を受信したサーバ２２０は、ブロック確定処理（Ｓ９００）のステップＳ９３０以降を実施した後に、ブロック確定処理完了通知の応答を返す。これにより、既存のサーバ２２０－１は、ブロックチェーンデータ２９０の更新内容を、新規のサーバ２２０に

も同期的に反映させることができる。したがって、ブロックチェーンシステムはブロックチェーンデータ２９０の整合性を保証することができる。

【００９６】

次に、ブロックチェーンプログラム３００は、このブロック確定処理が、バックグラウンド取得処理（図１１のＳ１１００）によるものか否かを判定する（Ｓ９３０）。バックグラウンド取得処理によるものである場合は、ステップＳ９４０に進み、そうでない場合にはステップＳ９５０に進む。

【００９７】

ステップＳ９４０では、ブロックチェーンプログラム３００が、ブロックデータ７００の読み書き種別Ｃ７２０が書き込みである全てのトランザクションの実行結果について、ステップＳ１０００の処理を繰り返して実行する。

【００９８】

ステップＳ１０００では、ブロックチェーンプログラム３００が、書き込もうとする値がデータベース２８５に存在する値よりも古いものでないかを判定する。この処理については図１０で詳述する。なお、バックグラウンド取得によるブロックデータ受信処理は最新のブロックデータ７００ではなく、過去の古いブロックデータ７００を取得する。そのため、バックグラウンド取得により得たブロックデータ７００に含まれるトランザクションの実行結果において一部のＫｅｙは既に取得済みである場合があり、その際はステップＳ１０００にて、該当Ｋｅｙを書き込み対象外とする必要がある。

【００９９】

そして、書き込み対象外とならなかったＫｅｙについて、ブロックチェーンプログラム３００は、そのＶａｌｕｅ　Ｃ７３０とブロック番号Ｃ７４０とトランザクション番号Ｃ７５０とをデータベース２８５にコミットする（Ｓ９５０）。

【０１００】

次に、ブロックチェーンプログラム３００は、自身がオンデマンドデータ取得モード情報で稼働中かを、稼働モードフラグ情報４００を参照してオンデマンドデータ取得モードＣ４１０を参照して「ｔｒｕｅ」であるか否かに基づいて判定する（Ｓ９６０）。オンデマンドデータ取得モードで稼働中の場合はステップＳ９７０へ進み、そうでない場合には処理を終了する。

【０１０１】

ステップＳ９７０では、ブロックチェーンプログラム３００が、上記ステップＳ９５０にてデータベース２８５にコミットした全ての書き込み対象のＫｅｙについて、ステップＳ９８０の処理を繰り返して実行する。

【０１０２】

ステップＳ９８０では、ブロックチェーンプログラム３００が、保持データ管理情報１３００に各Ｋｅｙを追加して、該当Ｋｅｙを既に取得済みであるという情報を格納する（Ｓ９８０）。

【０１０３】

上記処理により、生成されたブロックデータ７００の確定処理と、データベース２８５のコミット処理が完了する。

【０１０４】

図１０は、ブロックチェーンプログラム３００が実行するブロックデータ確定処理における、書き込みデータのバージョン判定処理の詳細を説明するためのフローチャートの一例である。この処理は図９のステップＳ１０００で行われる。

【０１０５】

ブロックチェーンプログラム３００は、ブロックデータ７００のコミット時に、ブロックデータ７００中の、読み書き種別Ｃ７２０が「書き込み」の各トランザクションの実行結果において、データベース２８５に既に格納されている値と、書き込みデータのバージョンの比較を行う。

【０１０６】

まず、ブロックチェーンプログラム３００は、トランザクションの実行結果のＫｅｙ（Ｃ７１０）に対応するデータを、データベース２８５から取得する（Ｓ１００１）。

【０１０７】

次に、ブロックチェーンプログラム３００は、トランザクションの実行結果のブロックデータ７００からブロック番号Ｃ７４０とトランザクション番号Ｃ７５０を取得し、データベース２８５から取得したデータのブロック番号Ｃ７４０とトランザクション番号Ｃ７５０とを、比較する（Ｓ１００２）。

【０１０８】

なお、ブロックチェーンプログラム３００は、ブロックデータ７００中のトランザクションの実行結果をデータベース２８５に格納するとき、ブロック番号Ｃ７４０及びトランザクション番号Ｃ７５０を、Ｖａｌｕｅ　Ｃ７３０を格納するため、前記ステップＳ１００２の比較によるバージョンの判定処理が可能である。

【０１０９】

トランザクションの実行結果のほうがデータベース２８５中のデータより新しいデータである場合、更新が必要である応答を返し（Ｓ１００３）、データベース２８５中のデータより古いデータであれば更新不要を返す（Ｓ１００４）。

【０１１０】

上記処理によって、ブロックチェーンプログラム３００は、データベース２８５のデータに古いトランザクションの実行結果が上書きされるのを防ぐことができる。

【０１１１】

図１１は、ブロックチェーンプログラム３００のバックグラウンド取得モジュール３５０が実行するブロックデータバックグラウンド取得処理の詳細を説明するためのフローチャートの一例である。

【０１１２】

ブロックデータバックグラウンド取得処理は、タイマ等であらかじめ定められたスケジュールに基づいて実行される。まず、バックグラウンド取得モジュール３５０は、自身がバックグラウンド取得モードで稼働中か否かを、稼働モードフラグ情報４００のバックグラウンド取得モードＣ４５０を元に判定する（Ｓ１１１０）。

【０１１３】

バックグラウンド取得モードで稼働中の場合、ステップＳ１１２０へ進んで、バックグラウンド取得モジュール３５０は、稼働モードフラグ情報４００のバックグラウンドデータ取得元Ｃ４６０で指定されたいずれかの他サーバ２２０に、ブロックデータ７００の送信要求を、要求するブロックデータのブロック番号を加えて送信する（Ｓ１０２０）。

【０１１４】

なお、バックグラウンド取得モジュール３５０が要求するブロック番号は、図１４に示すオンデマンドデータ取得進捗管理情報１４００のバックグラウンド取得による取得済み最新ブロック番号Ｃ１４１０の値よりも「１」だけ大きなものを選択し、古いブロック番号から順番にバックグラウンドで取得する。

【０１１５】

なお、バックグラウンド取得処理による取得済み最新ブロック番号Ｃ１４１０の初期値を、オンデマンドデータ取得開始時ブロック番号Ｃ１４２０と同じ値とし、さらに毎回「１」だけ小さなものを選択することで、新しいブロック番号から順番に取得するようにしてもよい。

【０１１６】

バックグラウンド取得モジュール３５０から、ブロック送信要求を受け取ったサーバ２２０は、要求されたブロック番号のブロックデータ７００を返送する。次に、バックグラウンド取得モジュール３５０は、送信要求したブロックデータ７００を受信（Ｓ１１３０）した後、そのブロックデータ７００をブロック確定処理によってコミットする（Ｓ９００）。なお、ブロック確定処理は、上述の図９のフローチャートで行われる。

【０１１７】

次に、バックグラウンド取得モジュール３５０は、オンデマンドデータ取得進捗管理情報１４００のバックグラウンド取得による取得済み最新ブロック番号Ｃ１４１０の値を、ステップＳ９００でコミットしたブロック番号で更新する（Ｓ１１４０）。

【０１１８】

次に、バックグラウンド取得モジュール３５０は、オンデマンドデータ取得進捗管理情報１４００のバックグラウンド取得による取得済み最新ブロック番号Ｃ１４１０の値と、オンデマンドデータ取得開始時ブロック番号Ｃ１４２０の値とを比較する（Ｓ１１５０）。

【０１１９】

取得済み最新ブロック番号Ｃ１４１０とオンデマンドデータ取得開始時ブロック番号Ｃ１４２０の値が同一であった場合、ステップＳ１１６０の処理へ進み、同一でない場合には処理を終了する。

【０１２０】

ステップＳ１１６０では、バックグラウンド取得モジュール３５０が、全てのバックグラウンド取得処理が完了したとみなし、以後はオンデマンドデータ取得処理のためのコミット同期処理（図９のステップＳ９２０）が不要であるため、バックグラウンド取得モジュール３５０は、コミット同期終了依頼を、稼働モードフラグ情報４００のオンデマンドデータ取得元Ｃ４２０で指定されたサーバ２２０に送信する（Ｓ１１６０）。

【０１２１】

なお、コミット同期終了依頼を受信したサーバ２２０上のブロックチェーンプログラム３００は、稼働モードフラグ情報４００のコミット同期モードＣ４３０を「ｆａｌｓｅ」に設定し、コミット同期の処理を以後実施しないようにする。

【０１２２】

次に、バックグラウンド取得処理も以後不要のため、バックグラウンド取得モジュール３５０は稼働モードフラグ情報４００のバックグラウンド取得モードＣ４５０を「ｆａｌｓｅ」に設定し、バックグラウンド取得モードでの稼働を終了する（Ｓ１１７０）。

【０１２３】

上記処理によって、ブロックチェーンシステムに追加されたサーバ２２０は、所定の周期毎にバックグラウンド取得処理を実行して、既存のサーバ２２０が既に処理したブロックデータ７００を蓄積することができる。

【０１２４】

図１２は、オンデマンドデータ取得初期化処理（Ｓ１２００）の一例を示すフローチャートである。ブロックチェーンプログラム３００が、ブロックチェーンシステム管理者などが管理端末２２５を利用してブロックチェーンプログラム３００の設定変更を実施したことを契機として、ブロックチェーンプログラム３００がオンデマンドデータ取得処理のための初期化処理を行う。

【０１２５】

まず、新規のサーバ２２０－２上のブロックチェーンプログラム３００は、稼働モードフラグ情報４００で所定の既存のサーバ２２０－１にコミット同期依頼を送信する（Ｓ１２０１）。

【０１２６】

既存のサーバ２２０－１上のブロックチェーンプログラム３００は、コミット同期依頼を受信すると（Ｓ１２０２）、稼働モードフラグ情報４００のコミット同期モードＣ４３０を「ｔｒｕｅ」に設定し、コミット同期モードフラグをセットする（Ｓ１２０３）。なお、このとき、既存のサーバ２２０－１では、稼働モードフラグ情報４００のコミット同期先Ｃ４４０に、コミット同期依頼を送信した新規のサーバ２２０－２の情報を格納する。以後、既存のサーバ２２０－１のブロックチェーンプログラム３００は、ブロックデータ７００のコミット時に、コミットするブロックデータ７００を、ステップＳ１２０１にてコミット同期依頼を送信したブロックチェーンプログラム３００にも同期的に送信する（図９のステップＳ９１０、Ｓ９２０）。

【０１２７】

次に、既存のサーバ２２０－１のブロックチェーンプログラム３００はコミット同期準備完了通知と、ブロックチェーンプログラム３００が保持する最新のブロックデータ７００とを送信する（Ｓ１２０４）。

【０１２８】

コミット同期依頼を送信した新規のサーバ２２０－２のブロックチェーンプログラム３００がコミット同期準備完了通知を受け付けると（Ｓ１２０５）、稼働モードフラグ情報４００のオンデマンドデータ取得モードＣ４１０を「ｔｒｕｅ」に設定し、オンデマンドデータ取得モードフラグをセットする（Ｓ１２０６）。なお、このとき、稼働モードフラグ情報４００のオンデマンドデータ取得元Ｃ４２０に、ステップＳ１２０１でコミット同期開始依頼を送信した既存のサーバ２２０－１の情報を格納する。

【０１２９】

次に、ステップＳ１２０４にて既存のサーバ２２０－１上のブロックチェーンプログラム３００が送信した最新のブロックデータ７００のブロックを、新規のサーバ２２０－２のブロックチェーンプログラム３００はブロック確定処理によりデータベース２８５にコミットする（Ｓ９００）。

【０１３０】

次に、新規のサーバ２２０－２では、ステップＳ１２０４にて既存のサーバ２２０－１上のブロックチェーンプログラム３００が送信した最新のブロックデータ７００のブロック番号を、オンデマンドデータ取得進捗管理情報１４００のオンデマンドデータ取得開始時ブロック番号Ｃ１４２０の値にセットする（Ｓ１２０８）。この値は、バックグラウンド取得処理の終了判定時（Ｓ１１５０）に利用される。

【０１３１】

さらに、上記のステップＳ９００とステップＳ１２０４の処理により、新規のサーバ２２０－２のブロックチェーンプログラム３００は、受け取ったブロックデータ７００と、このブロックデータ７００よりも古い全てのブロックデータ７００を仮想的に保持していることとなる。

【０１３２】

すなわち、新規のサーバ２２０－２では、オンデマンドデータ取得初期化処理時に、既存のサーバ２２０－１から受け取ったブロックデータ７００をコミットし、さらに、オンデマンドデータ取得開始時ブロック番号Ｃ１４２０を更新することで、このブロックデータ７００よりも古い全てのブロックデータ７００を仮想的に保持することができる。換言すれば、新規のサーバ２２０－２では、オンデマンドデータ取得開始時ブロック番号Ｃ１４２０を最古のブロック番号として扱うことができる。

【０１３３】

図１３は、オンデマンド取得処理により取得済みであるＫｅｙの一覧を管理し、オンデマンドデータ取得モジュール３４０がオンデマンドデータ取得処理の要否を判定するための保持データ管理情報１３００の構成例である。

【０１３４】

保持データ管理情報１３００は、ブロックチェーンプログラム３００がメモリ２７０上に保持するが、ディスクドライブ２８０上に格納してもよい。保持データ管理情報１３００のデータは、インタフェース種別Ｃ１３１０と、開始Ｋｅｙ　Ｃ１３２０と、終了Ｋｅｙ　Ｃ１３３０と、から構成される。

【０１３５】

インタフェース種別Ｃ１３１０は、このデータが、どのインタフェース種別における取得済みＫｅｙを示すものであるかを判断するためのものである。開始Ｋｅｙ　Ｃ１３２０は、前記インタフェース種別Ｃ１３１０が単一Ｋｅｙクエリである場合、取得済みのＫｅｙを、範囲Ｋｅｙクエリの場合、範囲の開始Ｋｅｙを示す。終了Ｋｅｙ　Ｃ１３３０は、前記インタフェース種別Ｃ１３１０が範囲Ｋｅｙクエリの場合、範囲の終了Ｋｅｙを示す。

【０１３６】

図１４は、ブロックチェーンプログラム３００上のオンデマンドデータ取得モジュール３

４０が保持する、オンデマンドデータ取得進捗管理情報１４００の構成例である。オンデマンドデータ取得進捗管理情報１４００は、バックグラウンド取得モジュール３５０がバックグラウンド取得の終了判定に利用するための管理情報である。

【０１３７】

オンデマンドデータ取得進捗管理情報１４００は、ブロックチェーンプログラム３００がメモリ２７０上に保持するが、ディスクドライブ２８０上に格納してもよい。オンデマンドデータ取得進捗管理情報１４００は、バックグラウンド取得による取得済み最新ブロック番号Ｃ１４１０と、オンデマンドデータ取得開始時ブロック番号Ｃ１４２０と、を含む。

【０１３８】

バックグラウンドによる取得済み最新ブロック番号Ｃ１４１０は、初期値は０であり、バックグラウンド取得モジュール３５０がブロックデータを受信した際に更新する。オンデマンドデータ取得開始時ブロック番号Ｃ１４２０は、オンデマンドデータ取得初期化処理（Ｓ１２００）において、ブロックチェーンプログラム３００が設定する。

【０１３９】

以上、本実施例１によれば、ブロックチェーンシステムにて新たに追加したサーバ２２０－２は、ブロックチェーンデータ２９０を、事前に全て取得するのではなく、ブロックチェーンプログラム３００内のオンデマンドデータ取得モジュール３４０により既存のサーバ２２０－１からオンデマンドに取得する。

【０１４０】

そのため、既存のブロックチェーンデータ２９０が大容量であっても、新たに追加したサーバ２２０－２上のブロックチェーンプログラム３００は、迅速にトランザクション処理を開始できる。なお、ブロックチェーンプログラム３００内にてオンデマンドデータ取得処理を実施するため、ブロックチェーンデータ２９０の整合性が損なわれることはない。

【０１４１】

加えて、既存のサーバ２２０－１が自身のブロックチェーンデータ２９０を更新する際には、新規のサーバ２２０－２にも更新する内容を同期的に反映させる。オンデマンドデータ取得モジュール３４０はスマートコントラクト３１０からデータベース２８５へのアクセスインタフェースが複数種類あることを考慮してオンデマンドデータ取得処理を行うため、トランザクションの実行結果の整合性が損なわれることはない。

【０１４２】

なお、上記実施例１では、ディスクドライブ２８０にブロックチェーンデータ２９０を保持するデータベース２８５を格納する例を示したが、これに限定されるものではなく、サーバ２２０に接続されたストレージ装置にデータベース２８５を格納するようにしてもよい。また、データベース２８５をＫｅｙ－Ｖａｌｕｅ形式で構成する例を示したが、これに限定されるものではなく、リレーショナル形式などを採用してもよい。

【実施例２】

【０１４３】

以下、図面を用いて本発明の実施例２を詳細に説明する。なお、以下の説明では実施例１との差分のみを示す。

【０１４４】

実施例１では、オンデマンドデータ取得元のサーバ２２０は固定の１つである必要があった。コミット同期処理（図９のステップＳ９２０）により、新規のサーバ２２０－２と既存のサーバ２２０－１は、オンデマンドデータ取得開始時ブロック番号Ｃ１４２０以降のブロックデータ７００を常に同期した状態で保持している。

【０１４５】

しかし、新規のサーバ２２０－２が、別の既存のサーバ２２０－ｎに対して再度コミット同期依頼を送信（図１２のステップＳ１２０１）するとき、実施例１の処理では、新規のサーバ２２０－２と既存のサーバ２２０－ｎがどのブロックデータ７００を保持しているのかを考慮していない。このため、コミット同期処理が正常に行われず、新規のサーバ２２０－２側でブロックデータ７００の欠損が生じる恐れがある。

【０１４６】
例えば、新規のサーバ２２０－２がブロック番号１０番までのブロックデータ７００を保持しており、既存のサーバ２２０－ｎがブロック番号１２番までのブロックデータを保持している状態で、オンデマンドデータ取得初期化処理Ｓ１２００（図１２）を実施すると、既存のサーバ２２０－ｎはブロック番号１３番からのブロックデータ７００を新規のサーバ２２０－２にコミット同期するため、新規のサーバ２２０－２はブロック番号１１番および１２番のブロックデータを受け取り損ねてしまう。
【０１４７】
そこで、本実施例２では、オンデマンドデータ取得初期化処理にて、新規のサーバ２２０－２と既存のサーバ２２０－１が所持するブロック番号を考慮したコミット同期開始処理を実施する。
【０１４８】
図１５および図１６は、オンデマンドデータ取得初期化処理（Ｓ１５００）を詳細に説明するためのフローチャートの一例である。本処理は前記実施例１の図１２と同様に実施されるが、ステップＳ１５０１からステップＳ１５０７の処理が前記実施例１とは相違する。
【０１４９】
まず、新規のサーバ２２０－２のブロックチェーンプログラム３００は、自身が保持するブロックデータ７００の最新ブロック番号の情報を加えて、既存のサーバ２２０－１のブロックチェーンプログラム３００に、コミット同期開始依頼を送信する（Ｓ１５０１）。
【０１５０】
既存のサーバ２２０－１のブロックチェーンプログラム３００が、コミット同期開始依頼を受信すると、新規のサーバ２２０－２が保持するブロックデータ７００の最新ブロック番号と、既存のサーバ２２０－１が保持するブロックデータの最新ブロック番号Ｃ１４１０とを比較する（Ｓ１５０２）。
【０１５１】
新規のサーバ２２０－２のほうがより新しいブロックデータ７００を保持している場合、既存のサーバ２２０－１上のブロックチェーンプログラム３００は、自身が保持していないが新規のサーバ２２０－２が保持している新しいブロックデータ７００の全てについて（Ｓ１５０３）、任意の他のサーバ２２０－ｎより取得し（Ｓ１５０４）、ブロック確定処理を実施する（Ｓ９００）。
【０１５２】
また、新規のサーバ２２０－２が保持するブロックデータ７００の最新ブロック番号と、既存のサーバ２２０－１が保持するブロックデータ７００の最新ブロック番号とを比較（Ｓ１５０５）する。
【０１５３】
既存のサーバ２２０－１のほうがより新しいブロックデータを保持している場合、既存のサーバ２２０－１上のブロックチェーンプログラム３００は、自身が保持しているが新規のサーバ２２０－２が保持していない新しいブロックデータ全てについて（Ｓ１５０６）、新規のサーバ２２０－２上のブロックチェーンプログラム３００にブロックデータ７００を送信する（Ｓ１５０７）。
【０１５４】
新規のサーバ２２０－２上のブロックチェーンプログラム３００は、ブロックデータを受信すると、そのブロックデータの確定処理をする（Ｓ９００）。その後の処理（Ｓ１２５０～Ｓ１２０８）は、実施例１と同様である。
【０１５５】
新規のサーバ２２０－２について、オンデマンドデータ取得元を変更する場合、ブロックチェーンシステムの管理者は、稼働モードフラグ情報４００のオンデマンドデータ取得元Ｃ４２０を別のサーバ２２０の情報に書き換え、図１５および図１６に示したオンデマンドデータ取得初期化処理Ｓ１５００を再度実行させる。
【０１５６】

そして、新規のサーバ２２０－２のブロックチェーンプログラム３００が変更前と異なるサーバ２２０－ｎに対してコミット同期開始以来処理を送信することで、異なるサーバ２２０－ｎからのコミット同期が開始され、オンデマンドデータ取得処理を行えるようになる。

【０１５７】

以上、本実施例２によれば、ブロックチェーンシステムにて新たに追加したサーバ２２０－２は、ブロックチェーンデータ２９０を、事前に全て取得するのではなく、ブロックチェーンプログラム３００内のオンデマンドデータ取得モジュール３４０により既存のサーバ２２０－１からオンデマンドに取得する。このとき、オンデマンドデータ取得元サーバ２２０－１を後から変更可能であるため、特定のサーバ２２０に負荷が集中し続けることはない。

【０１５８】

また、上記実施例２では、オンデマンドデータ取得元サーバを変更可能とするため、オンデマンドデータ取得処理時に、新規のサーバ２２０－２のブロックチェーンプログラム３００と、既存のサーバ２２０－１のブロックチェーンプログラム３００で、保持するブロックデータ７００のバージョン（ブロック番号）を比較し、両者のバージョンを先に合わせることで、新規のサーバ２２０－２が以前に別の既存のサーバ２２０－ｎからオンデマンドデータ取得処理を実施していたとしても、当該データを損なうことなく、新たな既存のサーバ２２０－１からオンデマンドデータ取得処理を実施することができる。

【実施例３】

【０１５９】

以下、図面を用いて本発明の実施例３を詳細に説明する。なお、以下の説明では、実施例２との差分のみを示す。

【０１６０】

本実施例３は前記実施例２とは異なり、オンデマンドデータ取得元のサーバ２２０を同時に複数設定可能とすることで、複数のサーバ２２０から並列的にオンデマンドデータ取得を行う。これにより、特定のサーバ２２０に負荷が偏ることなく、オンデマンドデータ取得処理を行うことが可能となる。

【０１６１】

図１７は、実施例３における、稼働モードフラグ情報１７００の構成図である。本実施例３では、前記実施例１の図４に示した稼働モードフラグ情報４００に代わって稼働モードフラグ情報１７００を使用する。

【０１６２】

稼働モードフラグ情報１７００は、実施例１、２における稼働モードフラグ情報４００（図４）とは、オンデマンドデータ取得元Ｃ１７１０と、コミット同期先Ｃ１７２０が異なる。

【０１６３】

本実施例３の稼働モードフラグ情報１７００のオンデマンドデータ取得元Ｃ１７１０およびコミット同期先Ｃ１７２０は、複数のサーバ２２０の情報を設定可能である。ブロックチェーンシステムの管理者などが、管理端末２２５を利用してオンデマンドデータ取得元Ｃ１７１０に所定の個数のサーバ２２０の情報を設定する。

【０１６４】

図１８は、実施例３における、ブロックチェーンプログラム３００が実行するブロックデータ確定処理の詳細を説明するためのフローチャートの一例である。なお、前記実施例１、２におけるブロックデータ確定処理（図９）とは、ステップＳ１８１０およびステップＳ１８２０が異なる。

【０１６５】

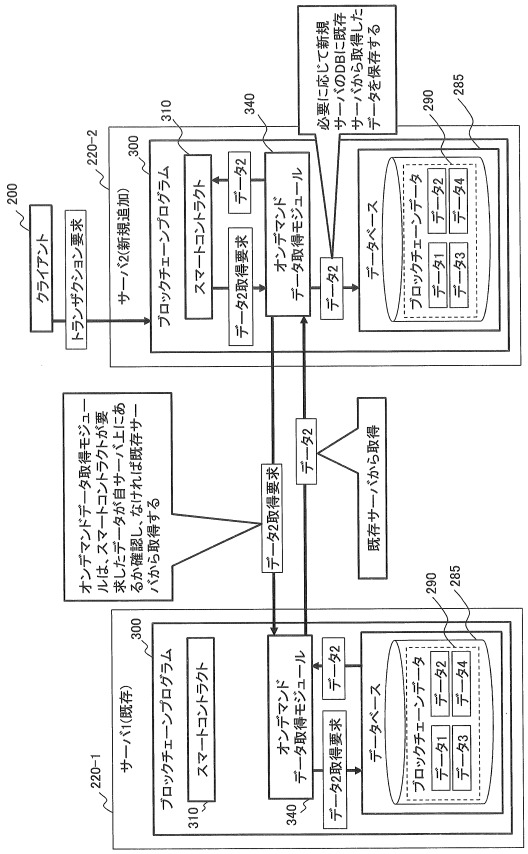
まず、ブロックチェーンプログラム３００は、稼働モードフラグ情報４００のコミット同期モードＣ４３０が「ｔｒｕｅ」であるか否かを取得し、自身がコミット同期モードで稼働中かを判定する（Ｓ９１０）。

【０１６６】
自身がコミット同期モードで稼働中の場合、ブロックチェーンプログラム３００は、稼働モードフラグ情報１７００のコミット同期先Ｃ１７２０で指定された全てのサーバ２２０について（Ｓ１８１０）、自身がコミット処理中のブロックデータ７００を送信し、応答を待つ（Ｓ１８２０）。
【０１６７】
なお、ブロックデータ７００を受信したサーバ２２０は、ブロック確定処理のステップＳ９３０以降を実施したのち、ブロック確定処理完了通知の応答を送信元のサーバ２２０に返す。なお、ブロックチェーンプログラム３００は、コミット同期先Ｃ１７２０の全てのサーバ２２０についてステップＳ１８２０の処理が完了するとステップＳ９３０に進む。
【０１６８】
ステップＳ９３０以降は前記実施例１と同様であり、ブロックデータ７００の確定処理と、データベース２８５のコミット処理が完了する。
【０１６９】
図１９、図２０は、実施例３における、オンデマンドデータ取得初期化処理を詳細に説明するためのフローチャートの一例である。本処理の流れは実施例２の図１５と同様であるが、ステップＳ１９０１およびステップＳ２００１が異なる。
【０１７０】
まず、新規のサーバ２２０－２のブロックチェーンプログラム３００は、自身が保持するブロックデータ７００の最新ブロック番号の値および稼働モードフラグ情報１７００のオンデマンドデータ取得元Ｃ１７１０の値を加えて、稼働モードフラグ情報１７００のオンデマンドデータ取得元Ｃ１７１０で指定された全ての既存のサーバ２２０のブロックチェーンプログラム３００に、コミット同期開始依頼を送信する（Ｓ１９０１）。
【０１７１】
その後の処理は前記実施例２の図１５と同様であるが、既存のサーバ２２０－１のブロックチェーンプログラム３００は、ステップＳ２００１にて、コミット同期モードフラグをセットするだけではなく、ステップＳ１９０１で新規のサーバ２２０－２が送信した稼働モードフラグ情報１７００のオンデマンドデータ取得元Ｃ１７１０の値を、稼働モードフラグ情報１７００のコミット同期先Ｃ１７２０に書き込む。
【０１７２】
これにより、ブロック確定処理におけるコミット同期処理にて、これら複数のサーバ２２０の全てに対して同期的にブロックデータ確定依頼が送信され、コミット処理を行うサーバ２２０と、稼働モードフラグ情報１７００のコミット同期先Ｃ１７２０に設定されたサーバ２２０とで、ブロックチェーンデータ２９０が同期された状態となる。その後の処理は、実施例２における図１５と同様である。
【０１７３】
図２１は、実施例３における、オンデマンドデータ処理を詳細に説明するためのフローの一例である。処理の流れは実施例１の図８と同様であるが、ステップＳ２１０１からステップＳ２１０３が前記実施例１と相違する。
【０１７４】
ステップＳ２１０１にて、オンデマンドデータ取得モジュール３４０は、既存のサーバ２２０－１からオンデマンドデータ取得処理を行う。そのとき、オンデマンドデータ取得モジュール３４０は、例えば、稼働モードフラグ情報１７００のオンデマンドデータ取得元Ｃ１７１０のサーバ情報の中から、ランダムまたはラウンドロビンで１つのサーバ２２０を選択し、選択されたサーバ２２０にデータ取得要求を送信する。
【０１７５】
あるいは、データベース２８５へのアクセスに利用するインタフェース種別Ｃ５２０が範囲Ｋｅｙクエリのとき、オンデマンドデータ取得モジュール３４０は、範囲Ｋｅｙクエリで取得する範囲を等分し、稼働モードフラグ情報１７００のオンデマンドデータ取得元Ｃ１７１０で指定されたサーバ２２０の全てに対してデータ取得要求を送信してもよい。

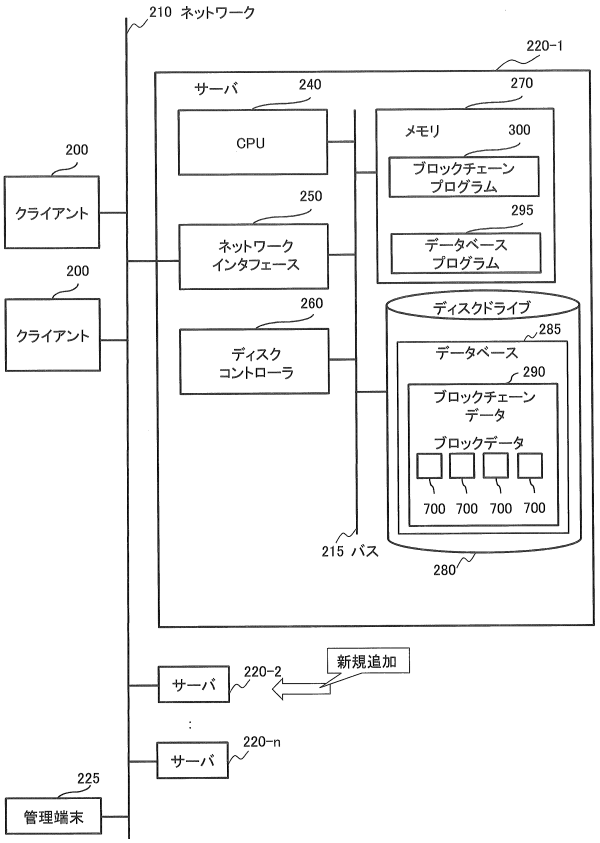
【０１７６】
そして、オンデマンドデータ取得モジュール３４０は、ステップＳ２１０２にて、全てのサーバ２２０からの応答を待ち、ブロックデータ７００の取得結果を統合する。ステップＳ２１０３においても、オンデマンドデータ取得モジュール３４０は、例えば、稼働モードフラグ情報１７００のオンデマンドデータ取得元Ｃ１７１０のサーバ情報の中から、ランダムまたはラウンドロビンで１つのサーバ２２０つを選択し、選択されたサーバ２２０にデータ取得要求を送信する。
【０１７７】
以上、本実施例３によれば、ブロックチェーンシステムにて新たに追加したサーバ２２０－２は、ブロックチェーンデータ２９０を、事前に全て取得するのではなく、ブロックチェーンプログラム３００内のオンデマンドデータ取得モジュール３４０により既存のサーバ２２０からオンデマンドに取得する。このとき、オンデマンドデータ取得元のサーバ２２０を複数設定可能であるため、特定のサーバ２２０に負荷が集中することはない。
【０１７８】
以上のように、本実施例３では、オンデマンドデータ取得元Ｃ１７１０に複数のサーバ２２０の情報を登録しておくことで、オンデマンドデータ取得時に、複数のサーバから並列的にブロックデータ７００の取得処理を実行したり、複数のサーバ２２０をラウンドロビン（連続的）で切り替えてブロックデータ７００の取得処理を実行することが可能となる。
【０１７９】
＜まとめ＞
なお、本発明は上記した実施例に限定されるものではなく、様々な変形例が含まれる。例えば、上記した実施例は本発明を分かりやすく説明するために詳細に記載したものであり、必ずしも説明した全ての構成を備えるものに限定されるものではない。また、ある実施例の構成の一部を他の実施例の構成に置き換えることが可能であり、また、ある実施例の構成に他の実施例の構成を加えることも可能である。また、各実施例の構成の一部について、他の構成の追加、削除、又は置換のいずれもが、単独で、又は組み合わせても適用可能である。
【０１８０】
また、上記の各構成、機能、処理部、および処理手段等は、それらの一部又は全部を、例えば集積回路で設計する等によりハードウェアで実現してもよい。また、上記の各構成、および機能等は、プロセッサがそれぞれの機能を実現するプログラムを解釈し、実行することによりソフトウェアで実現してもよい。各機能を実現するプログラム、テーブル、ファイル等の情報は、メモリや、ハードディスク、ＳＳＤ（Ｓｏｌｉｄ　Ｓｔａｔｅ　Ｄｒｉｖｅ）等の記録装置、または、ＩＣカード、ＳＤカード、ＤＶＤ等の記録媒体に置くことができる。
【０１８１】
また、制御線や情報線は説明上必要と考えられるものを示しており、製品上必ずしも全ての制御線や情報線を示しているとは限らない。実際には殆ど全ての構成が相互に接続されていると考えてもよい。
【符号の説明】
【０１８２】
２００　クライアント
２２０－１～２２０－ｎ　サーバ
２９０　ブロックチェーンデータ
３００　ブロックチェーンプログラム
３１０　スマートコントラクト
３２０　トランザクション処理モジュール
３４０　オンデマンドデータ取得モジュール
７００　ブロックデータ
１４００　オンデマンドデータ取得進捗管理情報

【図面】

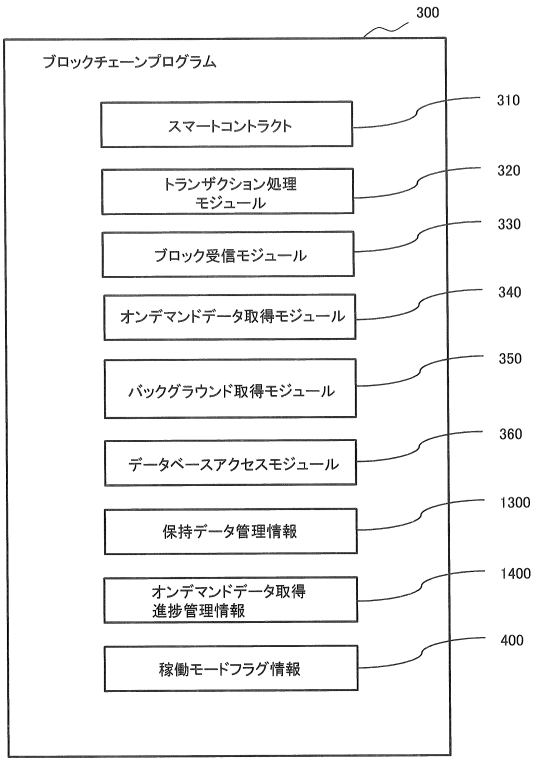
【図１】


クライアント
200
トランザクション要求
サーバ2(新規追加)
220-2
300
ブロックチェーンプログラム
310
スマートコントラクト
データ2
データ2取得要求
オンデマンド
データ取得モジュール
340
データ2
データベース
290
285
ブロックチェーンデータ
データ1
データ3
データ2
データ4
必要に応じて新規
サーバのDBに既存
サーバから取得した
データを保存する
オンデマンドデータ取得モジュー
ルは、スマートコントラクトが要
求したデータが自サーバ上にあ
るか確認し、なければ既存サー
バから取得する
データ2取得要求
データ2
既存サーバから取得
サーバ1(既存)
220-1
データ2取得要求

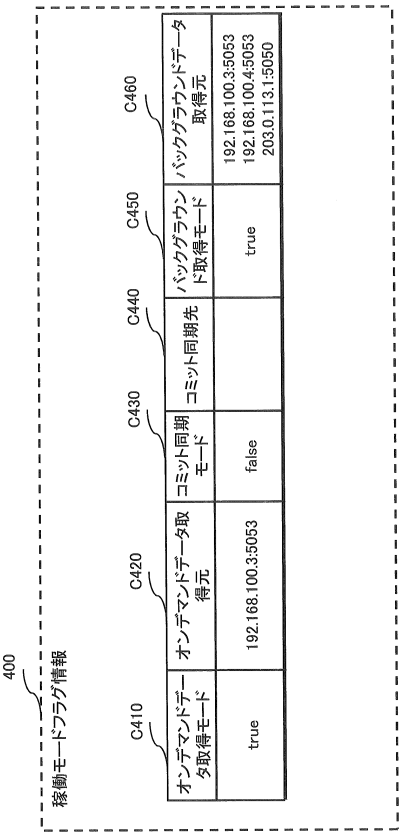
【図２】


210 ネットワーク
220-1
サーバ
240
CPU
メモリ
270
300
ブロックチェーン
プログラム
295
データベース
プログラム
250
ネットワーク
インタフェース
260
ディスク
コントローラ
ディスクドライブ
285
データベース
290
ブロックチェーン
データ
ブロックデータ
700 700 700 700
215 バス
280
200
クライアント
200
クライアント
サーバ
220-2
新規追加
サーバ
220-n
225
管理端末

【図３】


300
ブロックチェーンプログラム
スマートコントラクト
310
トランザクション処理
モジュール
320
ブロック受信モジュール
330
オンデマンドデータ取得モジュール
340
バックグラウンド取得モジュール
350
データベースアクセスモジュール
360
保持データ管理情報
1300
オンデマンドデータ取得
進捗管理情報
1400
稼働モードフラグ情報
400

【図４】


400
稼働モードフラグ情報
C410
オンデマンドデータ取得モード
true
C420
オンデマンドデータ取得元
192.168.100.3:5053
C430
コミット同期モード
false
C440
コミット同期先
C450
バックグラウンド取得モード
true
C460
バックグラウンドデータ取得元
192.168.100.3:5053
192.168.100.4:5053
203.0.113.1:5050

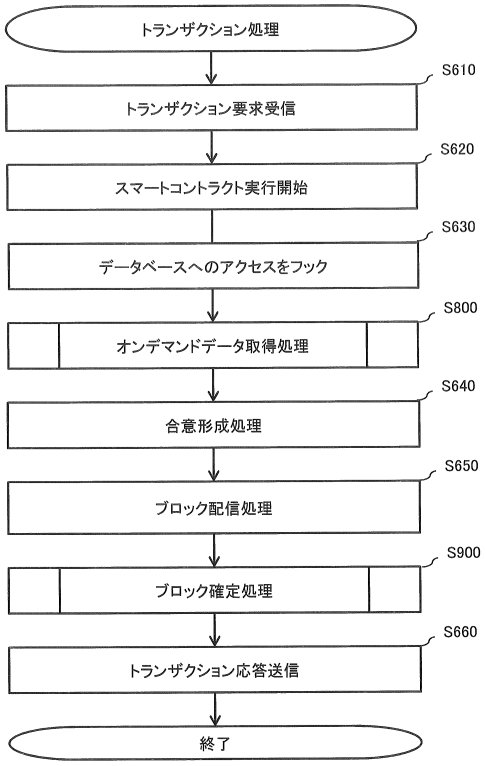
【図５】

500

データベースアクセスインタフェース

C510 インタフェース名	C520 インタフェース種別	C530 入力	C540 出力
Get	単一Keyクエリ	Key	Value, Result
Put	単一Keyクエリ	Key, Value	Result
GetRange	範囲Keyクエリ	StartKey, EndKey	Keys, Values, Result
Query	リッチクエリ	QueryString	Keys, Values, Result

【図６】



【図７】

700

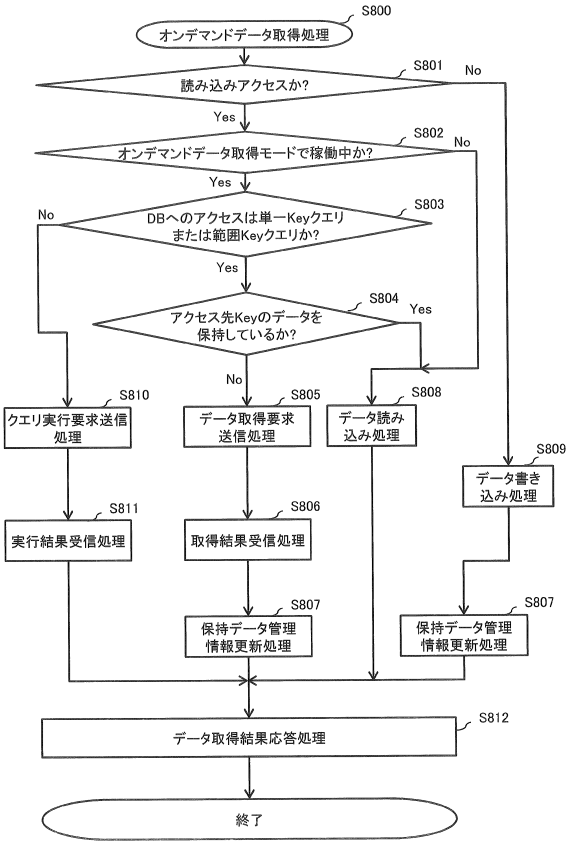
ブロックデータ

C710 読み書き対象Key	C720 読み書き種別	C730 Value	C740 ブロック番号	C750 トランザクション番号
A	読み	a	18	0
F	読み	f	18	1
G	書き	g	18	1
H	書き	h	18	2
...	...	...	...	...

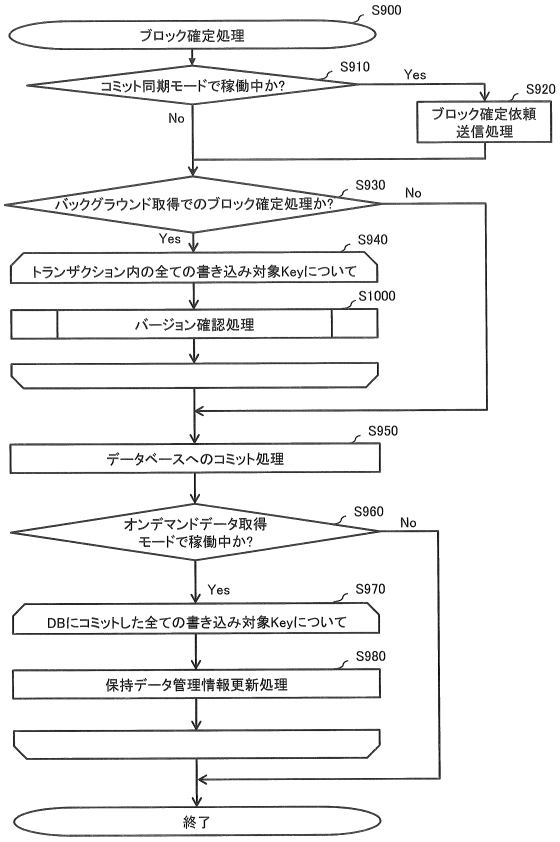
C760

ハッシュ値

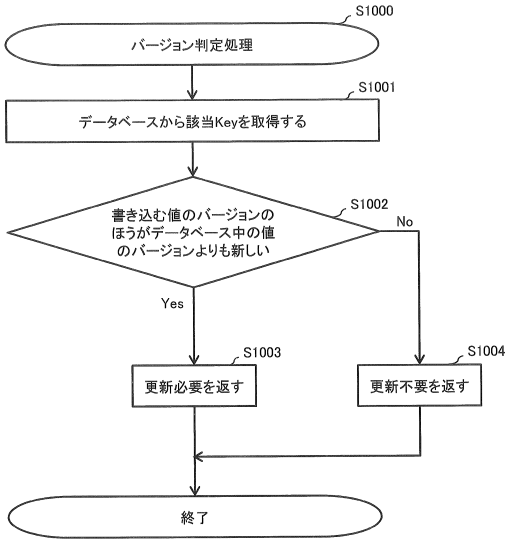
【図８】



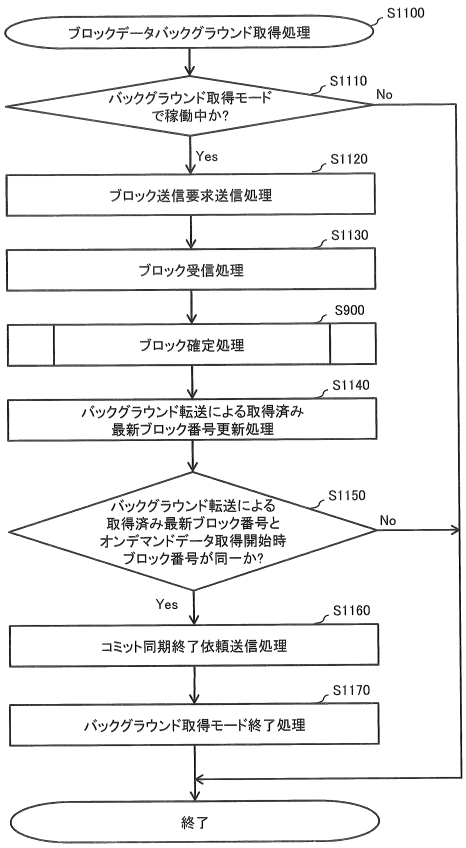
【図９】



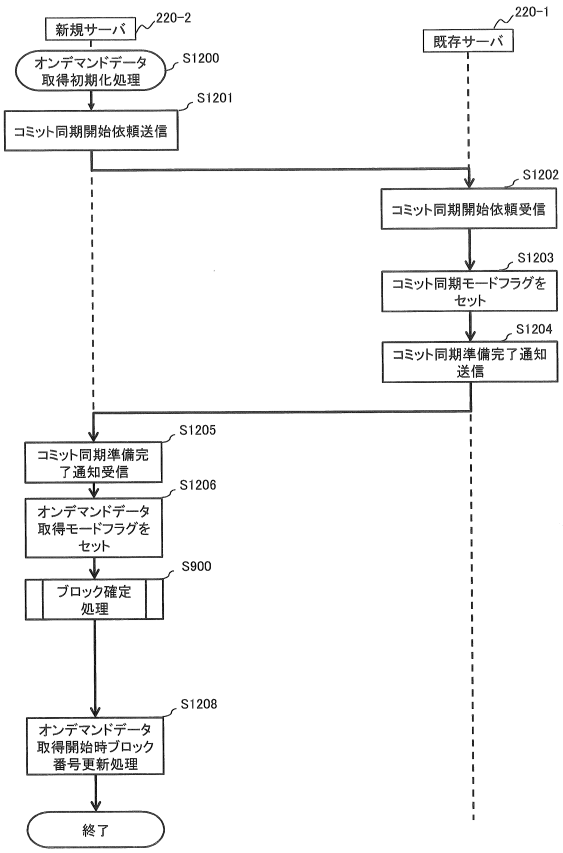
【図１０】



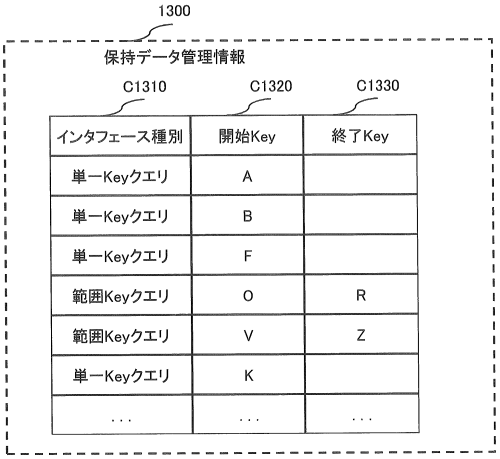
【図１１】



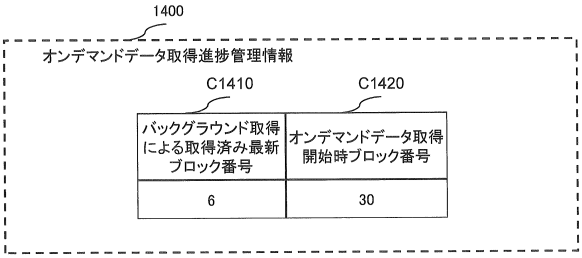
【図１２】



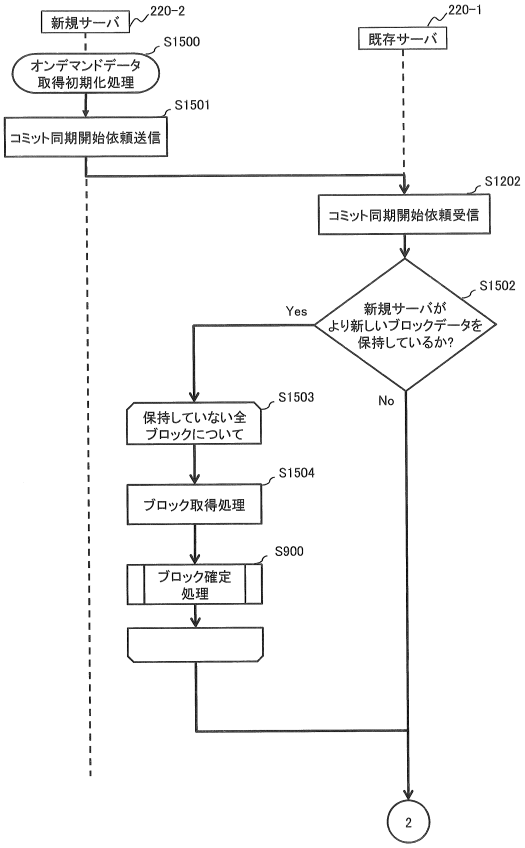
【図１３】



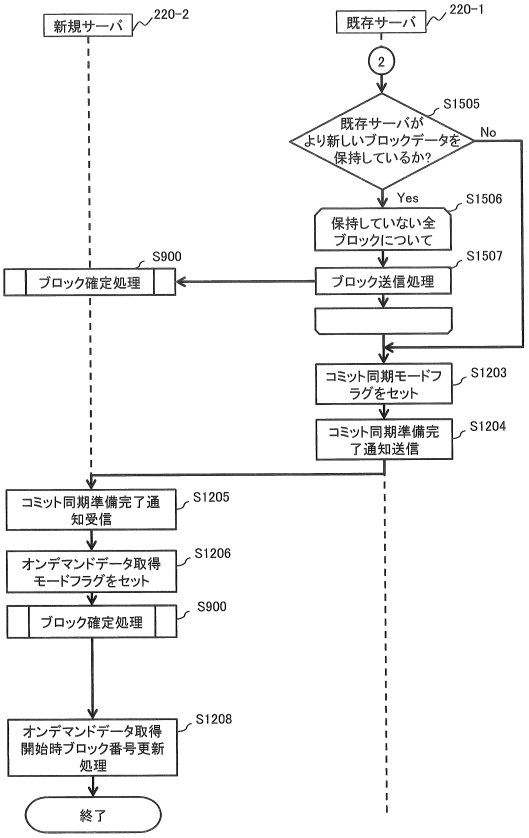
【図１４】



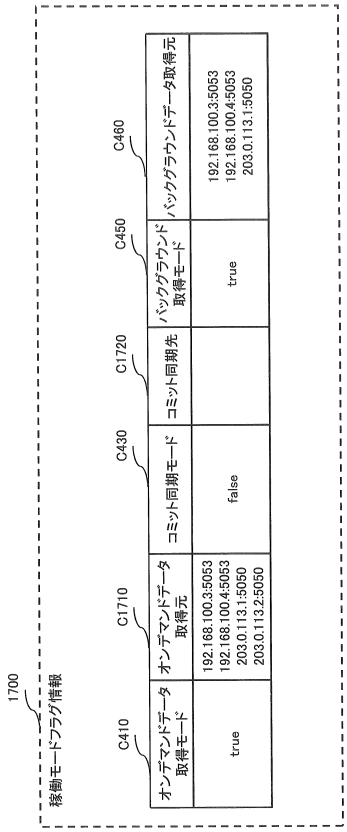
【図１５】



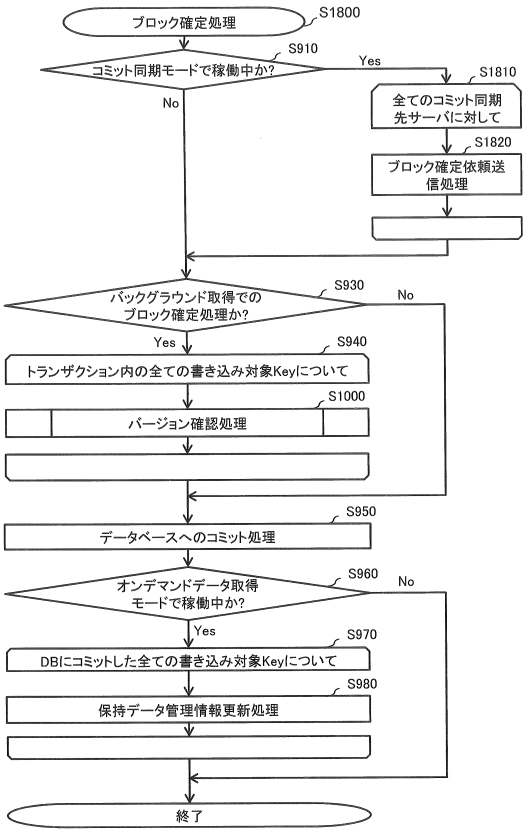
【図１６】



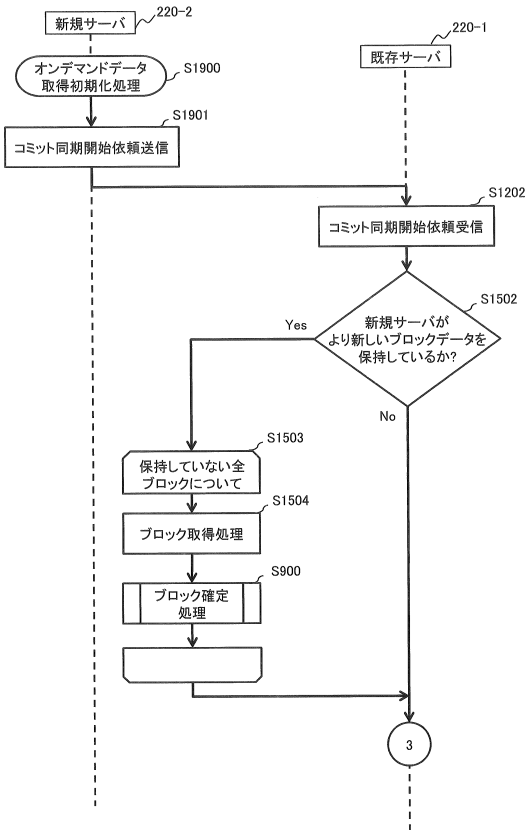
【図１７】



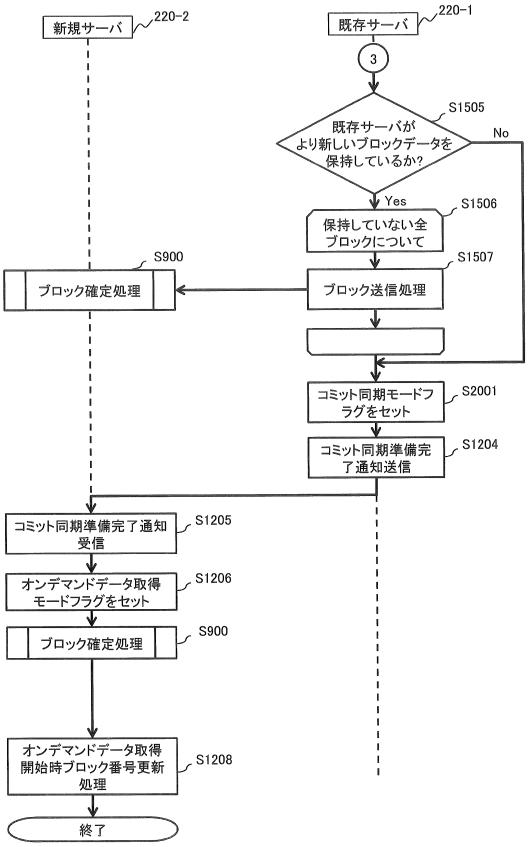
【図１８】



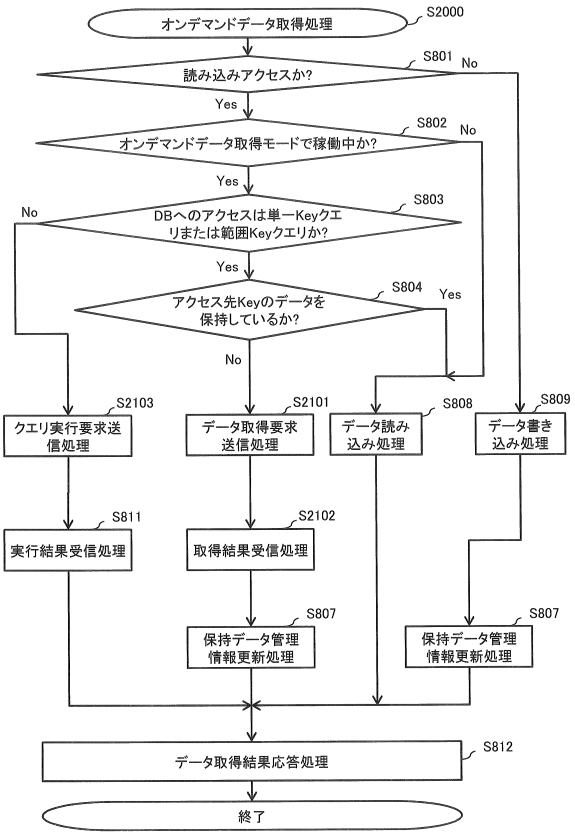
【図１９】



【図２０】



【図２１】


オンデマンドデータ取得処理
S2000
読み込みアクセスか?
S801
No
Yes
オンデマンドデータ取得モードで稼働中か?
S802
No
Yes
DBへのアクセスは単一Keyクエリまたは範囲Keyクエリか?
S803
No
Yes
アクセス先Keyのデータを保持しているか?
S804
Yes
No
クエリ実行要求送信処理
S2103
データ取得要求送信処理
S2101
データ読み込み処理
S808
データ書き込み処理
S809
実行結果受信処理
S811
取得結果受信処理
S2102
保持データ管理情報更新処理
S807
保持データ管理情報更新処理
S807
データ取得結果応答処理
S812
終了

フロントページの続き

(56)参考文献　国際公開第２００７／０６３９４４（WO，Ａ１）
特開２０１５－０１４９２９（ＪＰ，Ａ）
特開２０１３－０６５１０４（ＪＰ，Ａ）
特開２０１７－０９１１４９（ＪＰ，Ａ）

(58)調査した分野　(Int.Cl.，ＤＢ名)
Ｇ０６Ｆ　１６／１０－１６／２９
Ｇ０６Ｆ　３／０６