(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号
特許第6613320号
(P6613320)
(45)発行日　令和1年11月27日(2019.11.27)　(24)登録日　令和1年11月8日(2019.11.8)

(51)Int.Cl.　F I
G06F 16/172 (2019.01)　G06F 16/172

請求項の数 11　(全 29 頁)

(21)出願番号　特願2017-558432(P2017-558432)
(86)(22)出願日　平成27年10月22日(2015.10.22)
(65)公表番号　特表2018-531439(P2018-531439A)
(43)公表日　平成30年10月25日(2018.10.25)
(86)国際出願番号　PCT/CN2015/092523
(87)国際公開番号　WO2017/066953
(87)国際公開日　平成29年4月27日(2017.4.27)
審査請求日　平成30年8月29日(2018.8.29)

(73)特許権者　502303739
オラクル・インターナショナル・コーポレイション
アメリカ合衆国カリフォルニア州９４０６５レッドウッド・シティー，オラクル・パークウェイ５００
(74)代理人　110001195
特許業務法人深見特許事務所
(72)発明者　シェン，シュガン
中華人民共和国、１００１９３　ベイジン、ハイディアン・ディストリクト、ヂョングアンツン・ソフトウェア・パーク、ビルディング・ナンバー・２４、オラクル・ビルディング

最終頁に続く

(54)【発明の名称】トランザクション処理環境において分散型キャッシングを提供するためのシステムおよび方法

(57)【特許請求の範囲】
【請求項１】
トランザクション処理環境に分散型キャッシングを提供するためのシステムであって、
１つ以上のマイクロプロセッサを含むコンピュータと、
前記コンピュータ上で実行されるとともに、キャッシング機能を提供する複数の層を含むトランザクション処理環境とを含み、前記複数の層は、
キャッシング要求を受取るためのバッファベースの分散型キャッシング（ＢＤＣ）層と、
キャッシング動作の複数のインプリメンテーションを含むインプリメンテーション層とを含み、各々のインプリメンテーションは異なるキャッシングプロバイダによるものであり、前記複数の層はさらに、
共通のキャッシング関連のインターフェイスを定義する共通の分散型キャッシング（ＣＤＣ）層を含み、
前記ＢＤＣ層から転送されたキャッシング要求を受取ると、前記ＣＤＣ層は、特定のキャッシングプロバイダによってキャッシング動作のインプリメンテーションをインプリメンテーション層からロードして、前記キャッシング要求に関連付けられたデータについてキャッシング動作を実行する、システム。
【請求項２】
１つ以上のキャッシングプロバイダを含む従属型サードパーティベンダー層をさらに含む、請求項１に記載のシステム。

【請求項３】
前記１つ以上のキャッシングプロバイダはCoherenceおよび／またはCloudStoreを含む、請求項２に記載のシステム。
【請求項４】
前記ＢＤＣ層は、前記システムにおいて使用される複数のデータタイプを登録し、各々の登録されたデータタイプは、そのデータタイプについての動作を定義するスイッチに関連付けられている、請求項１から３のいずれか一項に記載のシステム。
【請求項５】
各々の登録されたデータタイプについての定義された動作は、シリアライゼーション、デシリアライゼーション、符号化または復号化のうち１つ以上を含む、請求項４に記載のシステム。
【請求項６】
前記ＣＤＣ層は、前記共通のキャッシング関連のインターフェイスを提供するためのアプリケーションプログラミングインターフェイス（ＡＰＩ）を含む、請求項１から５のいずれか一項に記載のシステム。
【請求項７】
前記ＣＤＣ層は、前記特定のキャッシングプロバイダからキャッシング動作のインプリメンテーションをロードする際に使用される、キャッシングプロバイダスイッチおよび関連するＡＰＩを含む、請求項１から６のいずれか一項に記載のシステム。
【請求項８】
前記キャッシングプロバイダスイッチは複数のポインタを含み、前記複数のポインタの各々は、前記特定のキャッシングプロバイダによって提供されるキャッシング動作を指し示す、請求項７に記載のシステム。
【請求項９】
前記ＣＤＣ層は、前記ＢＤＣ層において定義される１つ以上のコールバック機能を呼出して、前記キャッシング要求に関連付けられた前記データのシリアライゼーションまたはデシリアライゼーションを実行する、請求項１から８のいずれか一項に記載のシステム。
【請求項１０】
トランザクション処理環境において分散型キャッシングを提供するための方法であって、１つ以上のマイクロプロセッサ上で実行されるとともに、キャッシング機能を提供する複数の層を含むトランザクション処理環境を設けるステップと、
バッファベースの分散型キャッシング（ＢＤＣ）層においてキャッシング要求を受取るステップと、
共通のキャッシング関連のインターフェイスを定義する共通の分散型キャッシング（ＣＤＣ）層において、前記ＢＤＣ層から転送される前記キャッシング要求を受取るステップと、
特定のキャッシングプロバイダによってキャッシング動作のインプリメンテーションを、前記ＣＤＣ層を介して、インプリメンテーション層からロードするステップとを含み、前記インプリメンテーション層はキャッシング動作の複数のインプリメンテーションを含み、各々のインプリメンテーションは異なるキャッシングプロバイダによるものであり、前記方法はさらに、
前記キャッシング要求に関連付けられたデータについて、ロードされたキャッシング動作を実行するステップを含む、方法。
【請求項１１】
トランザクション処理環境において分散型キャッシングを提供するための１セットの命令を格納するプログラムであって、前記命令は、１つ以上のプロセッサによって実行されると、前記１つ以上のプロセッサに以下のステップを実行させ、前記以下のステップは、
バッファベースの分散型キャッシング（ＢＤＣ）層においてキャッシング要求を受取るステップと、
共通の分散型キャッシング（ＣＤＣ）層において前記ＢＤＣ層から転送される前記キャ

ッシング要求を受取るステップとを含み、前記ＣＤＣ層は共通のキャッシング関連のインターフェイスを定義し、前記以下のステップはさらに、
特定のキャッシングプロバイダによるキャッシング動作のインプリメンテーションを、前記ＣＤＣ層を介して、インプリメンテーション層からロードするステップを含み、前記インプリメンテーション層は、キャッシング動作の複数のインプリメンテーションを含み、各々のインプリメンテーションは異なるキャッシングプロバイダによるものであり、前記以下のステップはさらに、
前記キャッシング要求に関連付けられたデータについて、ロードされたキャッシング動作を実行するステップを含む、プログラム。

【発明の詳細な説明】

【技術分野】

【０００１】

発明の分野：

本発明の実施形態は、概して、アプリケーションサーバおよびクラウド環境に関し、特に、トランザクション処理環境において分散型キャッシングを提供するためのシステムおよび方法に関する。

【背景技術】

【０００２】

背景：

Ｔｕｘｅｄｏ（登録商標）サーバ環境などのトランザクション処理環境においては、性能を向上させるためのユーザ関連データまたはアプリケーションデータをキャッシュする分散型キャッシングシステムは難易度の高いものになる可能性がある。なぜなら、クライアントプロセスとサーバプロセスとの間でデータを送信するのに複数のデータタイプを用いることができ、このため、さまざまな顧客が別々のキャッシングソリューションを選ぶ可能性があるからである。これらは、本発明の実施形態が対処するように意図される分野である。

【発明の概要】

【課題を解決するための手段】

【０００３】

概要：

一実施形態に従うと、トランザクション処理環境において分散型キャッシングを提供するためのシステムおよび方法がこの明細書中に記載される。最上層は、キャッシングシステムと対話するためにクライアントをキャッシュすることによってアプリケーションプログラミングインターフェイス（application programming interface：ＡＰＩ）をエクスポーズして使用できるようにすることができ、複数のバッファタイプと、各バッファタイプに対する１つ以上のコールバック機能とを登録することができる。共通のキャッシング層は共通のデータ構造をサポートすることができ、最上層からキャッシング要求を受取ったとき、そこにあるコールバック機能を用いて、キャッシング要求に関連付けられた特定のバッファタイプと共通のデータ構造との間で変換を行なうことができる。共通のキャッシング層は、キャッシング関連のＡＰＩを提供するために１組の共通のＡＰＩを定義することができるとともに、キーおよび値データのシリアライゼーション／デシリアライゼーションなどの複数のインプリメンテーションについての共通の挙動を定義することができる。プロバイダスイッチおよび関連するＡＰＩを用いて、インプリメンテーション層からプロバイダスイッチの特定のインプリメンテーションをロードすることができる。プロバイダスイッチは、特定のキャッシングプロバイダによって提供されるキャッシング動作を指し示すポインタを含み得る。構成ファイルは、データコーディングと、キャッシングシステムのためにどのキャッシングプロバイダを使用するべきかとを指定するのに用いることができる。

【０００４】

一実施形態に従うと、分散型キャッシングシステムによって使用されるデータシリアラ

イゼーションをサポートするためのシステムおよび方法がこの明細書中に記載される。データシリアライゼーションシステムは共通のデータ構造を含み得る。共通のデータ構造は、複数のデータタイプのシリアライズされたストリーム／バイトを格納するために用いることができる。データ構造は、キャッシュセッターのアーキテクチャを記述する情報を含むヘッダを含み得る。異なるアーキテクチャ上で使用するために、キャッシュされたデータを検索する場合、ヘッダ内のアーキテクチャ情報を用いて、データを、その異なるアーキテクチャ上で使用できるように変換することができる。キャッシュされたデータが同じアーキテクチャ上で検索される場合、キャッシュされたデータを変換することなく用いることができる。データ構造は、付加的に、メモリを効率的に使用できるようにするために可変長をもつボディ、後方互換性についてのバージョン情報、および、拡張用のオプション特徴のためのフィールドを含み得る。

【０００５】

一実施形態に従うと、分散型インメモリ・データグリッド（たとえば、Coherence）を、キャッシングプロバイダとしての分散型キャッシングシステムに統合するためのシステムおよび方法がこの明細書中に記載される。分散型キャッシングシステムにおけるプロキシサーバは、分散型インメモリ・データグリッドに対するクライアントとしての役割を果たし得るとともに、ＩＤＣクライアントから転送されたキャッシング要求を受取り得る。起動時に、プロキシサーバは、キャッシング・システム・キャッシュを定義する構成ファイルと、分散型インメモリ・データグリッドキャッシュにキャッシュするマップとをロードし、キャッシング・システム・キャッシュの名前を用いてサービスをアドバタイズすることができる。要求されたサービスを指定するキャッシング要求をキャッシング・クライアントから受取ると、プロキシサーバは、要求されたサービスに基づいてアクセスするために、分散型インメモリ・データグリッドにおける対応するキャッシュを決定することができる。

【０００６】

一実施形態に従うと、トランザクション処理環境において分散型キャッシングシステムを用いて、サービスから戻された結果をキャッシュするためのシステムおよび方法がこの明細書中に記載される。構成ファイルは、サービスから戻された結果をキャッシュするのにどのキャッシュを用いるべきかと、キャッシュされた結果を識別する際に使用されるキーを如何に生成するかとを記述するエントリを含むキャッシングセクションを含み得る。サービスについての要求がクライアントから受取られると、トランザクション処理環境のアプリケーションサーバコアは、関連するキャッシュされたデータが、構成ファイルを用いて生成されたキーによって識別されるキャッシュに存在しているかどうかを判断することができる。存在している場合、アプリケーションサーバコアは、サービスを呼出す代わりに、キャッシュされたデータを直接戻すことができる。他の場合、アプリケーションサーバコアは、サービスを呼出し、生成されたキーを用いて、構成ファイルによって指定されたキャッシュにデータをキャッシュし、クライアントに結果を戻すことができる。

【図面の簡単な説明】

【０００７】

【図１】一実施形態に従った、トランザクション処理環境における分散型キャッシングを提供するためのシステムを示す図である。

【図２】一実施形態に従った、トランザクション処理環境において使用される分散型キャッシングを提供するためのシステムをさらに示す図である。

【図３】一実施形態に従った、キャッシングＡＰＩを示す詳細なクラス図である。

【図４】一実施形態に従った、トランザクション処理環境において使用される分散型キャッシングを提供するための方法を示す図である。

【図５】一実施形態に従った例示的な型付きバッファを示す図である。

【図６】一実施形態に従った、分散型キャッシングシステムによって使用されるデータシリアライゼーションをサポートするためのシステムを示す図である。

【図７】一実施形態に従った、分散型キャッシングシステムによって使用されるデータシ

リアライゼーションをサポートするための方法を示す図である。
【図８】一実施形態に従った、キャッシングプロバイダとしての分散型キャッシングシステムに分散型インメモリ・データグリッドを統合するためのシステムを示す図である。
【図９】一実施形態に従った、キャッシングプロバイダとしての分散型キャッシングシステムに分散型インメモリ・データグリッドを統合するためのシステムをさらに示す図である。
【図１０】一実施形態に従った、キャッシングプロバイダとしての分散型キャッシングシステムに分散型インメモリ・データグリッドを統合するための方法を示す図である。
【図１１】一実施形態に従った、トランザクション処理環境において分散型キャッシングシステムを用いて、サービスから戻された結果をキャッシュするためのシステムを示す図である。
【図１２】一実施形態に従った、トランザクション処理環境において分散型キャッシングシステムを用いて、サービスから戻された結果をキャッシュするための方法を示す図である。
【発明を実施するための形態】
【０００８】
詳細な説明：
トランザクション処理環境（たとえば、Ｔｕｘｅｄｏサーバ環境）においては、メッセージが１つのプロセスから別のプロセスに送信可能になる前にバッファが割当てられる必要がある。このような環境における複雑なアプリケーションは、さまざまなプロトコルを用いて複数のネットワークを介して通信する異種システムにインストールすることができる。そのため、異なるタイプのバッファが必要となり、各々のバッファタイプは、メッセージを初期化し、送信し、受信し、かつデータを符号化し、復号化するためのさまざまなルーチンを必要とする。
【０００９】
典型的には、ユーザ（たとえば、アプリケーション開発者）が性能向上のために分散型キャッシングシステムに型付きバッファをキャッシュする場合、ユーザは、特定の型付きバッファ用にカスタマイズされたコードを書込む必要がある。さらに、アプリケーションの開発が完成した後に異なるキャッシングプロバイダが用いられる場合、キャッシングプロバイダの変更に対応するために、コードを付加的に変更する必要がある。
【００１０】
分散型キャッシングシステム
一実施形態に従うと、トランザクション処理環境において分散型キャッシングを提供するためのシステムおよび方法がこの明細書中に記載される。最上層は、キャッシングシステムと対話させるためにクライアントをキャッシュすることによってアプリケーションプログラミングインターフェイス（ＡＰＩ）をエクスポーズして使用できるようにすることができ、複数のバッファタイプと、各バッファタイプに対する１つ以上のコールバック機能とを登録することができる。共通のキャッシング層は、共通のデータ構造をサポートすることができ、最上層からキャッシング要求を受取ると、そこにあるコールバック機能を用いて、キャッシング要求に関連付けられた特定のバッファタイプと共通のデータ構造との間で変換を行なうことができる。共通のキャッシング層は、キャッシング関連のＡＰＩを提供するために１組の共通のＡＰＩを定義し、キーおよび値データのシリアライゼーション／デシリアライゼーションなどの複数のインプリメンテーションについての共通の挙動を定義することができる。プロバイダスイッチおよび関連するＡＰＩを用いて、インプリメンテーション層からプロバイダスイッチの特定のインプリメンテーションをロードすることができる。プロバイダスイッチは、特定のキャッシングプロバイダによって提供されるキャッシング動作を指し示すポインタを含み得る。構成ファイルは、データコーディングと、キャッシングシステムのためにどのキャッシングプロバイダを使用するべきかとを指定するのに用いることができる。
【００１１】

階層化されたキャッシングシステムは、ユーザと下層のキャッシングプロバイダとの間を分離させることができ、これにより、システムが使いやすくなり、その拡張が容易になる。

【００１２】

たとえば、キャッシングシステムは、複数のキャッシングプロバイダとともに用いることができ、ユーザのアプリケーションを変更する必要なしに、複数のデータタイプをキャッシュすることができる。開発者がどんなバッファタイプをキャッシュする必要があるかどうかにかかわらず、かつ、そのキャッシングシステムのためにどのキャッシングプロバイダが構成されるべきであるかにかかわらず、ユーザは、同じセットのＡＰＩを用いてキャッシング動作を実行することができる。

【００１３】

一実施形態に従うと、分散型キャッシングシステムは、アクセス回数および更新回数を減らすことができ、各々がキーによって識別されている複数のタイプのバッファをキャッシュすることができ、複製アクセス、非複製アクセス、ローカルアクセスおよびリモートアクセスを提供することができる。

【００１４】

図１は、一実施形態に従った、トランザクション処理環境において使用される分散型キャッシングを提供するためのシステムを示す。

【００１５】

図１に示されるように、分散型キャッシングシステム１０５は、キャッシュにデータを格納する（１０６）かまたはキャッシュからデータを検索する（１０７）ために、キャッシング・クライアント（たとえば、Ｔｕｘｅｄｏクライアントまたはサーバ）１０１によって使用されるトランザクション処理システム（たとえば、Ｔｕｘｅｄｏサーバ環境）１００において、提供され得る。

【００１６】

図１においてさらに示されるように、分散型キャッシングシステムは、複数の層、たとえば、バッファベースの分散型キャッシング（buffer-based distributed caching：ＢＤＣ）層１０９、共通の分散型キャッシング（common distributed caching：ＣＤＣ）層１２５、分散型キャッシングのインプリメンテーション（implementation of distributed caching：ＩＤＣ）層１４１、および、従属型サードパーティベンダー（dependent third-party vendor：ＤＴＶ）層１４９を含み得る。構成ファイル１０３は、ＢＤＣ層、ＣＤＣ層およびＩＤＣ層の挙動を定義するために用いることができる。

【００１７】

一実施形態に従うと、ＢＣＤ層は、キャッシング・クライアントに対する最上層であり、複数の型付きバッファを処理することができ、型付きバッファをＣＤＣによってサポートされる共通の型付きバッファに変換した後、キャッシング・クライアントからＣＤＣ層にキャッシング要件を転送することができる。最上層の主要な機能は、さまざまな型付きバッファのシリアライゼーションおよびデシリアライゼーションである。

【００１８】

図１においてさらに示されるように、ＢＤＣ層は、キャッシングシステムと通信するためにキャッシング・クライアントによって使用される１組の外部ＡＰＩ１２１をエクスポーズすることができる。たとえば、外部ＡＰＩは、データをキャッシュするかまたはデータをキャッシュから検索するために、アプリケーションにおいて直接用いることができる。

【００１９】

以下の表１は、一実施形態に従った外部ＡＰＩの例示的なインターフェイスのリストを示す。

【００２０】

【表１】

tpgetcache(const char* name, long flags)	構成に従ってTuxedo キャッシュハンドルを入手
tpcacheput(TCACHE* tc, char* key, char* data, long len, long flags)	Tuxedo型バッファをキャッシュに 入れて、そのバッファをキーに関連付ける
tpcacheget(TCACHE* tc, char* key, char** odata, long* olen, long flags)	キャッシュからのキーに関連付けられた Tuxedo型バッファを入手
tpcacheremove(TCACHE* tc, char* key, long flags)	パラメータキーに関連付けられた エントリをキャッシュから削除
tpcachemremove(TCACHE* tc, char* keyarray[], int size, long flags)	パラメータキー配列に関連付けられた エントリをキャッシュから削除
tpcacheremoveall(TCACHE* tc, long flags)	キャッシュからすべてのエントリを削除

表1

【００２１】

表１に示されるように、一実施形態に従うと、外部ＡＰＩは、キャッシュに対するハンドルを取得し、キャッシュにバッファを入れ、バッファをキーに関連付け、バッファをキャッシュから削除して検索する際にアプリケーションが使用するべき方法を含み得る。

【００２２】

たとえば、ユーザは、Ｔｕｘｅｄｏキャッシュハンドルを得るために方法「tpgetcache」を用いることができる。方法のパラメータ「名」は、取得すべきキャッシュのタイプを指定することができる。方法についての戻された結果は、構造名ＴＣＡＣＨＥを指し示すポインタであり得る。「tpcacheput/tpcacheget」はデータをキャッシュに入れる／キャッシュから入手するために用いることができる。「tpcacheremove/tpcachemremove/tpcacheremoveall」は、キャッシュからアイテムを削除するために用いることができる。

【００２３】

一実施形態に従うと、ＢＤＣ層は、キャッシングシステムにおいてキャッシュすることができるデータタイプとして複数の型付きバッファ１３０を登録することができる。たとえば、複数の型付きバッファは、ストリング（一連の文字）、ｃａｒｒａｙ（文字配列）、ｐｔｒ（バッファを指すポインタ）、ＦＭＬ型付きバッファおよびＶＩＥＷ型バッファを含み得る。他のデータタイプも、それらのデータタイプがＢＤＣ層に登録されていれば、キャッシングのためにサポートされ得る。

【００２４】

図１に示されるように、各々の登録された型付きバッファは、型付きバッファを記述するメタデータ（たとえば、型付きバッファメタデータ１３２）と、符号化および暗号化ならびにシリアライゼーションおよびデシリアライゼーションを含むバッファ上での複数の動作を定義する１つ以上のコールバック機能１３１と、に関連付けることができる。

【００２５】

一実施形態に従うと、ＢＤＣ層は、各々の登録されたデータタイプ毎に、メタデータおよびコールバック機能を提供するスイッチ（データ構造）を含み得る。

【００２６】

以下のリスト１は、一実施形態に従った例示的なスイッチを示す。

【００２７】

【数１】

```
struct tmtype_sw_t {
        char type[TMTYPELEN];          /* type of buffer */
        char subtype[TMSTYPELEN];  /* sub-type of buffer */
        long dfltsize;                 /* default size of buffer */
        /* buffer initialization function pointer */
        …
        long (_TMDLLENTRY *presend) _((char _TM_FAR *, long, long));
        /* post-send buffer manipulation func pointer */
        void (_TMDLLENTRY *postsend) _((char _TM_FAR *, long, long));
        /* post-receive buffer manipulation func pointer*/
        long (_TMDLLENTRY *postrecv) _((char _TM_FAR *, long, long));
        /* encode/decode function pointer */
        long (_TMDLLENTRY *encdec) _((int, char _TM_FAR *, long, char _TM_FAR *,
long));
        /* routing function pointer */
        …
};
```

リスト1

【００２８】
リスト１に示されるように、スイッチは、バッファのタイプ、サブタイプおよびサイズ、シリアライゼーション動作「presend」および「presend2」、デシリアライゼーション動作「postrecv」、ならびに符号化／暗号化動作「encdec」を記述するメタデータを含み得る。

【００２９】
一実施形態に従うと、ＣＤＣ層は、キャッシング関連のＡＰＩを提供するために１組の共通のＡＰＩ（たとえば、キャッシングＡＰＩ１２７）を含み得るとともに、名前／キー値ペアを含むデータのシリアライゼーション／デシリアライゼーションなどの複数のインプリメンテーションについての共通の挙動を定義し得る。ＣＤＣ層は、シリアライゼーション／デシリアライゼーションのインプリメンテーションのためのコールバック登録をサポートすることによって、複数のタイプのデータをシリアライズ／デシリアライズすることができる。

【００３０】
一実施形態に従うと、共通のデータ構造（たとえば、共通の型付きバッファ）１３３は、キャッシュされるべき型付きバッファのシリアライズされたバイトを格納するために、ＣＤＣ層によってサポートすることができる。ＢＤＣ層に登録される登録済みの型付きバッファまたは他のデータタイプの各々は、シリアライズされ、共通のデータ構造に格納され得る。

【００３１】
一実施形態に従うと、ＣＤＣ層は実際のキャッシング動作を実行せず、必要なインターフェイスを定義する。これらのインターフェイスは、キャッシング動作の特定のインプリメンテーションに依拠しない。構成に基づいて、これらのインターフェイスの特定のインプリメンテーションは、ＩＤＣ層から動的にロードすることができる。

【００３２】
一実施形態に従うと、キャッシングプロバイダスイッチ１４３および特定の方法はＣＤＣ層に提供されて、プロバイダスイッチの特定のインプリメンテーションを検索するのに使用され得る。プロバイダスイッチは、キャッシングプロバイダによって提供されるキャッシング動作を指し示す複数のポインタを含み得る。プロバイダスイッチの各インプリメ

ンテーションのロードは、ＢＤＣ層からのキャッシング要求に応じて、キャッシングＡＰＩによって開始され得る（１２８）。以下のリスト２は、一実施形態に従った例示的なプロバイダスイッチを示す。

【００３３】

【数２】

```
#ifndef __TMTDCInnerCache_type
#define __TMTDCInnerCache_type
/**
 * @brief structure of the inner cache offered by the implementation layer
 */
typedef struct _TMTDCInnerCache TMTDCInnerCache;
#endif

/**
 * @brief interface/switch definitions for all cache related operations.
 * The implementation of the switch may not depend on Tuxedo. Higher version
 *may offer more operations.
 */
typedef struct{
      int version; /**< version, must be 1*/
      int flags;  /**< flags, reserved, must be 0.*/

      /**< get internal cache handle*/
      TMTDCInnerCache* (*getcache)(TMProperties* prop, const char* cname);

      /**< check if a given feature is supported or not*/
      int (*issupported)(const char* feature);

      /**< put a key/value pair into the cache*/
      int (*put)(TMTDCInnerCache* ic, const TMTDCCacheKey* key, const
      TMTDCCacheValue* value, int flags);

      /**< get a value of a key from the cache*/
      int (*get)(TMTDCInnerCache* ic, const TMTDCCacheKey* key,
      TMTDCCacheValue* value, int flags);

      /**< remove a key from the cache*/
      int (*remove)(TMTDCInnerCache* ic, const TMTDCCacheKey* key, int flags);

      /**< remove a set of keys from the cache*/
      int (*removekeys)(TMTDCInnerCache* ic, const TMTDCCacheKey* keyarray, int
      size, int flags);

      /**< remove all keys from the cache*/
      int (*removeall)(TMTDCInnerCache* ic, int flags);

      /**< destroy internal cache handle*/
      int (*freecache)(TMTDCInnerCache*
ic);                                        }TMTDCCachingProviderSwitch;

/**< get provider switch*/
typedef TMTDCCachingProviderSwitch* (*TMTDCGetCachingProviderSW)(TMProperties*
prop);
```

リスト2

【００３４】
リスト２に示されるように、プロバイダスイッチは、「入れる（put）」、「入手する（get）」および「削除する（remove）」などのキャッシング動作を示すポインタを含み得る。方法「TMTDCGetCachingProviderSW」は、構成ファイルによって定義されるように、キャッシングインターフェイスの特定のインプリメンテーションを検索するために用いられる特定の方法である。
【００３５】
再び図１を参照すると、ＩＤＣ層は、実際のキャッシング動作を提供することができ、各キャッシングプロバイダ毎にクライアントおよびサーバを含み得る。上述のとおり、各々のクライアントは、プロバイダスイッチのインプリメンテーションであってもよく、ＣＤＣ層によって使用されるべく提供することができる。クライアントは、１つ以上のダイナミックライブラリによって提供することができ、このため、クライアントの各々を動的にＣＤＣ層にロードすることができる。サーバは、実際のキャッシングサーバまたはプロバイダとしてサードパーティアプリケーションを用いることができる。各々のキャッシングプロバイダは、それ自体のＩＤＣインプリメンテーションを有することができ、これにより、ユーザが、アプリケーションコード変更のない構成によってＩＤＣインプリメンテーションを変更することが可能となる。
【００３６】
一実施形態に従うと、ＤＴＶ層は、複数のサードパーティキャッシングプロバイダ（たとえば、CoherenceおよびCloudStore）を含み得る。
【００３７】
一実施形態に従うと、ＢＤＣ層およびＣＤＣ層は、トランザクション処理環境に対するクライアントアプリケーションとそこにあるサーバ（たとえば、ＴｕｘｅｄｏクライアントおよびＴｕｘｅｄｏサーバ）との両方に存在し得るとともに、ＩＤＣ層およびＤＴＶ層はサーバ上にのみ存在し得る。
【００３８】
一実施形態に従うと、構成ファイルは単純な行指向型フォーマットのファイルであり得る。構成ファイルにおけるプロパティは、行の観点から処理することができるため、新しく導入されたプロパティがファイルのローディングプロセスに影響を与えることはない。表２は、構成ファイルにおけるプロパティのサンプルリストを示す。
【００３９】

【表２】

プロパティセット	記述
provider.[xxx]	キャッシングプロバイダによって用いられるプロパティ
options.[xxx]	TDCオプションによって用いられるプロパティ
cache.[xxx]	キャッシュによって用いられるプロパティ
coh.[xxx]	オラクルCoherenceに基づいたキャッシングプロバイダ インプリメンテーションによって用いられるプロパティ
configfile	値によって指し示されるプロパティファイルからの インポートプロパティを示すURL。ローカルファイルの ために、値はfile:///pathとなるだろう。プロトコルが 指定されない場合、URLはローカルファイルとして得られる。 プロパティが定義されていない場合、クライアントは、 tdc[cachename]によって指定されるtuxedoサービスに 関連付けられた関連構成を得るだろう
options.encoding	1は、すべてのキャッシングデータを符号化する必要が あることを示す。異なるデータ表現を有するマシンに 位置する、ユーザ用のキャッシングを提供するプロパティ。 これはデフォルトでは0に設定される。 その値はtdc.options.encoding.[cachename]によって無効に することができる
provider.default	デフォルトプロバイダ名が構成ファイルにおいて 指定されていれば、そのデフォルトプロバイダ名を示す
cache	使用されるべき[cachename]を示す
provider.library. providername]	プロバイダ[providername]によって用いられる インプリメンテーションライブラリ名を示す。 ライブラリ名が[xxx]であれば、ライブラリはunixでは libxxx.soとなるだろう。
cache.provider. cachename]	キャッシュ[cachename]によって用いられるプロバイダidを示す。 対応するプロパティ「tdc.provider.library.「providername]」が 指定される必要がある。
cache.options.enco ding.[cachename]	1は、このキャッシュにおけるすべてのキャッシングデータを 符号化する必要があることを示す。このプロパティが 設定されていなければ、tdc.options.encodingの値が 用いられる。
coh.cache.name.[ca chename]	Tuxedoキャッシュ[cachename]のための Coherenceクラスタにおける使用済みのキャッシュ名を示す。 オラクルCoherenceに基づいたTuxedoキャッシュが 用いられる場合、必要となる。

表2

【００４０】

一実施形態に従うと、階層化されたキャッシングシステムは、ユーザと下層にあるキャッシングプロバイダとの間を分離し得るとともに、ユーザがシステム上でキャッシング動作を実行するために同じセットのインターフェイスを用いることを可能にし得る。

【００４１】

具体例として、ＢＤＣ層が型付きバッファをキャッシュするようにとのキャッシング要

求をキャッシング・クライアントから受取ると、ＢＤＣ層は、ＣＤＣ層にキャッシング要求を転送する（１２３）ことができ、これにより、型付きバッファをシリアライズするためにＢＤＣ層における対応するコールバック機能を呼出す（１３７）ことができ、シリアライズされたバイトを共通のデータ構造に格納する（１３８）ことができる。ＣＤＣ層は、共通セットのインターフェイスを用いて、構成ファイルによって指定される特定のキャッシングプロバイダからキャッシング動作にアクセスして、キャッシング動作を実行することができる。

【００４２】

図２は、一実施形態に従った、トランザクション処理環境において使用される分散型キャッシングを提供するためのシステムをさらに示す。

【００４３】

図２に示されるように、ＣＤＣ層におけるキャッシングＡＰＩはさらに、共通のキャッシングインターフェイスおよび他の共通の挙動、たとえばシリアライゼーションおよびデシリアライゼーションなど、を定義する複数のオブジェクトを含み得る。ユーザにエクスポーズされる外部ＡＰＩとは異なり、上述のとおり、キャッシングＡＰＩは、キャッシングシステムによって使用されるべく提供される。

【００４４】

一実施形態に従うと、キャッシングＡＰＩは、外部ＡＰＩのユーザによって取得されるキャッシュを表わし得るキャッシュオブジェクト２２３を含み得る。このキャッシュは、キャッシュ関連の動作をすべて提供することができ、外部ＡＰＩからキャッシング動作を受取ることができる。キャッシュは、たとえば、構成ファイルの「tdc.cache」プロパティによって指定されるキャッシュ名によって識別され得る。

【００４５】

図２に示されるように、キャッシュオブジェクトは、キャッシュキーオブジェクト２２５およびキャッシュ値オブジェクト２２７に関連付けることができる。キャッシュキーオブジェクトは、キャッシュのためのキーを生成および検索するための方法を含み得るとともに、キャッシュ値オブジェクトは、シリアライゼーションインターフェイスを含み得る。

【００４６】

一実施形態に従うと、キャッシングプロバイダ２２９は、構成ファイルにおいて、プロパティcache.providerによって指定されるように、プロバイダ名によって識別される１組のキャッシング動作を定義することができる。プロセスレベルコンテキスト（ＴＵＸＰ）において維持されるグローバルなキャッシュプロバイダコンテナ２３０は、すべてのキャッシングプロバイダを暗黙的に維持することができる。キャッシュマネージャ２２４は、トランザクション処理環境においてキャッシュを管理するために用いることができる。キャッシュマネージャは、ユーザが直接キャッシュを作成するべき場合に暗黙的に作成することができるものであるが、単一のキャッシングプロバイダに関連付けることができる。キャッシュマネージャが作成されたとき、キャッシングプロバイダが存在していなければ、関連付けられたキャッシングプロバイダを内部に作成することができる。グローバルなキャッシュマネージャコンテナ２２８は、スレッド内で暗黙的に作成されたマネージャを管理するためにスレッドレベル・コンテキスト（ＴＵＸＴ）において維持することができる。

【００４７】

図３は、一実施形態に従ったキャッシングＡＰＩの詳細なクラス図を示す。

図４は、一実施形態に従った、トランザクション処理環境において使用される分散型キャッシングを提供するための方法を示す。

【００４８】

図４に示されるように、ステップ４１１において、分散型キャッシングシステムは、トランザクション処理環境において提供することができる。分散型キャッシングシステムは、キャッシング・クライアントからキャッシング要件を受取るためのＢＤＣ層と、共通セ

ットのキャッシングインターフェイスを提供するためのＣＤＣ層と、共通セットのキャッシングインターフェイスのインプリメンテーションを提供するためのＩＤＣ層と、実際のキャッシングプロバイダを提供するためのＤＴＶ層とを含み得る。

【００４９】

ステップ４１３において、分散型キャッシングシステムのＢＤＣ層は、キャッシング・クライアントからキャッシング要求を受取ることができる。キャッシング要求は、ＢＤＣ層によってエクスポーズされるＡＰＩを用いて開始することができる。

【００５０】

ステップ４１５において、ＢＤＣ層はＣＤＣ層にキャッシング要求を転送することができる。キャッシング要求に関連付けられた型付きバッファは、ＢＤＣ層において定義されるコールバック機能を用いて、共通のデータ構造に変換することができる。

【００５１】

ステップ４１７において、ＣＤＣ層は、インプリメンテーション層から共通セットのキャッシングインターフェイスの特定のインプリメンテーションをロードし、特定のインプリメンテーションを用いてキャッシング動作を実行することができる。

【００５２】

シリアライゼーションサポート

上述のとおり、ＣＤＣ層は、型付きバッファ上でシリアライゼーション動作／デシリアライゼーション動作を呼出すために、コールバック機能のための登録機能を定義することができる。

【００５３】

一実施形態に従うと、型付きバッファまたは別のデータタイプは、登録機能を用いて、シリアライゼーション・ハンドラ／デシリアライゼーション・ハンドラに登録される必要がある。登録機能の例は以下のとおりであり得る。

【００５４】

【数３】

```
int _tmtdc_regtype(_TCADEF, int type, TMTDCCacheSerializeCB cb1,
TMTDCCacheDeserializeCB cb2, int flags)
```

【００５５】

上述の登録機能においては、シリアライゼーションコールバック機能（cb1）およびデシリアライゼーション機能（cb2）が登録される。これらのコールバック機能は、上述のとおり、ＢＤＣ層において定義することができ、以下のリスト３に例示することができる。

【００５６】

【数４】

```
/**
 * @brief serialize callback function for a user-defined type
 * The function converts a user-defined data buffer into a serialized
 * stream. It needs to be registered via _tmtdc_regtype with a given type
 * before the type can be used.
 * @param iptr [in] pointer pointed to user-defined buffer
 * @param ilen [in] the length of the user-defined buffer
 * @param optr [out] the buffer to store the output data
 * @param osize [in] indicates the size of the output buffer as the input or
 *  the real size of the output data as the output
 * @param arg [in] the additional parameter used by the callback
 * @param flag [in|out] as input, the possible value can be<p/>
 * TDC_SERIALIZER_ENCODE indicates to do encoding<p/>
 * as output, the possible value may be<p/>
 * TDC_SERIALIZER_USEIPTR indicates the serialized data is in the buffer
 * referenced by iptr<p/>
 * TDC_SERIALIZER_USEOPTR indicates that the serialized data is in the buffer
 *referenced by optr.
 * @retval positive the length of the output data
 * @retval -1 errors
 * @retval negative  indicates a larger buffer is needed and the negative
 *absolute value is the desired buffer size
 *
 * @see _tmtdc_regtype
 */
typedef int (* TMTDCCacheSerializeCB)(char* iptr,  int ilen, char* optr, int
osize, void* arg, int* flag);

/**
 * @brief deserizlize callback function for a user-defined type
 * The function converts a stream into a user-defined data buffer. It needs
 * be registered via _tmtdc_regtype with a given type before the type
 * can be used.
 * @param iptr [in] pointer pointed to user-defined buffer
 * @param ilen [in] the length of the user-defined buffer
 * @param optr [out] the buffer to store the output data
 * @param osize [in] indicates the size of the output buffer
 * @param arg [in] the additional parameter used by the callback
 * @param flag [in] the possible value can be<p/>
 * TDC_SERIALIZER_ENCODE indicates to do decoding<p/>
 * @retval positive the length of the output data
 * @retval -1 errors
 * @retval negative  indicates a larger buffer is needed and the negative
 * absolute value is the desired buffer size

 * @see _tmtdc_regtype
 */
typedef int (*TMTDCCacheDeserializeCB)( char* iptr,  int ilen, char* optr,
int osize, void* arg, int flag);
```

リスト3

【００５７】

リスト３に示されるように、登録された型付きバッファまたは別のユーザ定義型データタイプはシリアライズされたストリームに変換され、キャッシュに格納され得る。その後

、シリアライズされたストリームは、キャッシュから検索された後、変換されてユーザ定義型データバッファに戻され得る。

【００５８】

分散型キャッシングシステムにおいて使用される型付きバッファを含むさまざまなデータタイプをサポートするために、これらのデータタイプのシリアライズされたストリームを格納するための共通のデータ構造を提供することができる。

【００５９】

一実施形態に従うと、分散型キャッシングシステムによって使用されるデータシリアライゼーションをサポートするためのシステムおよび方法がこの明細書中に記載される。データシリアライゼーションシステムは、複数のデータタイプのシリアライズされたストリーム／バイトを格納するために用いることができる共通のデータ構造を含み得る。データ構造は、キャッシュセッターのアーキテクチャを記述する情報を含むヘッダを含み得る。異なるアーキテクチャ上で使用するために、キャッシュされたデータが検索される場合、ヘッダ内のアーキテクチャ情報を用いて、この異なるアーキテクチャ上で使用できるようにデータを変換することができる。キャッシュされたデータが同じアーキテクチャ上で検索される場合、キャッシュされたデータは、変換されずに用いることができる。データ構造は、付加的に、メモリを効率的に使用できるようにするために可変長をもつボディ、後方互換性についてのバージョン情報、および、拡張用のオプション特徴のためのフィールドを含み得る。

【００６０】

図５は、一実施形態に従った例示的な型付きバッファを示す。

図５に示されるように、型付きバッファ（たとえば、Ｔｕｘｅｄｏ型バッファ）５１１は、フレキシブル・メッセージ・ヘッダ（flexible message header：ＦＭＬ）５１３、およびユーザ型付きコンテナモジュール（user typed container module：ＴＣＭ）５１７を含み得る。ＴＣＭはさらに、型付きコンテナヘッダー（typed container header：ＴＣＨ）５１９および型付きコンテナボディ（typed container body：ＴＣＢ）５２１を含み得る。ＴＣＢはＴ＿ＢＵＦＦＥＲ５２３およびユーザデータ５２９を含み得る。Ｔ＿ＢＵＦＦＥＲは、型付きバッファのタイプ５２５およびサブタイプ５２７を格納するために用いることができる。型付きバッファは、他のバッファに格納することができるいくつかの非ユーザＴＣＭを有し得る。

【００６１】

図６は、一実施形態に従った、分散型キャッシングシステムによって使用されるデータシリアライゼーションをサポートするためのシステムを示す。

【００６２】

図６に示されるように、データ構造はＣＤＣ層によってサポートされる共通の型付きバッファ１３３であってもよい。データ構造はヘッダ６１１およびボディ６１３を含み得る。ヘッダは、４バイトで位置合わせされる必要があり、複数のフィールド、たとえば、マジックフィールド６１５、メジャーバージョンフィールド６１７、マイナーバージョンフィールド６１８、長さフィールド６１９、vdataフィールド６２２、exlenフィールド６２３、タイプフィールド６２５、およびフラグフィールド６２１を含み得る。

【００６３】

一実施形態に従うと、ボディは、ボディのサイズを示すための長さのフィールド６２９と、ソース型付きバッファのタイプ６３１およびサブタイプ６３３を格納するために用いられるＴ＿Ｂｕｆｆｅｒのフィールド６２６と、ユーザデータフィールド６３５とを含み得る。

【００６４】

一実施形態に従うと、「マジック」のフィールドは、ヘッダを区別するために用いられるcharであり得るとともに、長さのフィールドは、ヘッダのサイズを示し得る。フラグのフィールドは、メジャーバージョンのフィールドにおける状態を示し得るかまたはオプション特徴を制御し得る。メジャーバージョンのフィールドは、データ構造における構造の

変更を示すことができる。たとえば、新しいフィールド／メンバが追加もしくは削除されるか、または、フィールドの意味が変更される。マイナーバージョンのフィールドは、メジャーバージョン内の変更を示すために用いることができる。たとえば、新しいビット・フラグがフラグのフィールドに投入される。

【００６５】

一実施形態に従うと、exlenのフィールドは、一般的にはヘッダに含まれないであろういずれの余分なヘッダデータ（たとえば、オプションの特徴）をも記述し得る。「exlen」がゼロでなければ、プレースホルダ「vdata」から始まる最初の「exlen」*４バイトがヘッダデータの一部になり得る。余分なヘッダデータは、さまざまなフィールドを有し得る。各々のフィールドは、４バイトで位置合わせされたアドレスから始まり得るとともに、そのタイプを示すunsigned shortとしてfirst shortを用い、フィールドのデータの長さ（バイト）を示すunsigned shortとしてsecond shortを用い得るとともに、シリアライズされ得る。

【００６６】

一実施形態に従うと、vdataのフィールドは、余分なヘッダデータおよびボディを含む可変データを示すためのプレースホルダであり得る。ヘッダのうち最初の４バイトは、他の用途のために変更される可能性はなく、ヘッダにおけるメンバは、ビッグエンディアンを用いることができる（vdataは含まれていない）。ヘッダのサイズは可変データを含んでおらず、そのため、長さのフィールドに影響を与えることはない。

【００６７】

一実施形態に従うと、メジャーバージョン、長さおよびフラグのそれぞれのフィールドは、メジャーバージョン間でのプロトコル互換性をサポートするために用いることができる。加えて、フラグのフィールドは、符号化されていないデータ、符号化されたデータおよび自己記述型データのうちの１つであり得るデータの状態を示すことができる。

【００６８】

一実施形態に従うと、データが符号化されていない場合、シリアライゼーション中にどの動作も実行されず、ボディは、型付きバッファにおける元のユーザＴＣＭと同じであり得る。キャッシュセッター（たとえば、キャッシュにデータを格納するアプリケーション）と同じアーキテクチャ（たとえば、同じ文字符号化または同じエンディアン）のマシンに存在するキャッシュゲッター（たとえば、キャッシュされたデータを検索するアプリケーション）は、キャッシュされたデータを検索して用いることができる。

【００６９】

一実施形態に従うと、データが符号化されると、型付きバッファにおける元のユーザＴＣＭが特定の符号化機構を用いてシリアライズされるため、キャッシュゲッターがデータをデシリアライズするために適切な対応機構を用いていれば、如何なるプラットフォーム上のキャッシュゲッターでもデータを正確に取得することができるようになる。

【００７０】

一実施形態に従うと、この明細書中に記載されるデータ構造は「自己記述型」モードまたは状態を提供することができ、これにより、データ構造のボディを、型付きバッファにおいて、たとえば、「符号化されていない」状態で、元のユーザＴＣＭと同じ状態にすることが可能になる。さらに、キャッシュセッターの元のアーキテクチャについての追加情報は、データ構造のヘッダに含めることができる。異なるアーキテクチャに位置するキャッシュゲッターがキャッシュされたデータを入手する場合、追加情報を用いて、異なるアーキテクチャにおいて使用できるようにデータを変換することができる。キャッシュゲッターがキャッシュセッターと同じアーキテクチャに位置する場合、データは、変換されずに直接用いられ得る。

【００７１】

一実施形態に従うと、データ構造は依然として「符号化された」モードをサポートすることができ、このため、Ｔｕｘｅｄｏ符号化アルゴリズムをサポートする他のプロダクトとデータを共有することができるようになる。シナリオの例として、Ｔｕｘｅｄｏアプリ

ケーションがキャッシュセッターとして機能するとともに、ＪＡＴＭＩパッケージを用いるＷｅｂｌｏｇｉｃアプリケーションがキャッシュゲッターとして機能する例を挙げることができる。

【００７２】

一実施形態に従うと、アーキテクチャは、以下のうち１つ以上によって異種となり得る。すなわち、１）異なるエンディアン（バイトオーダー、リトルエンディアンまたはビッグエンディアン）；２）異なる文字集合（ASCII、EBCDICなど）；および、３）異なるサイズを有するタイプ（たとえば、長さが４バイトまたは８バイトであり得る）。

【００７３】

一実施形態に従うと、「自己記述」状態が用いられる場合、構造上の相違についての情報を、データ構造のヘッダにおいて指定することができる。異種環境においては、「自己記述」モードは、符号化モードと比べて２つの利点を有する。すなわち、１）キャッシュセッターについて、「符号化されていない（unencoded）」場合と同じ性能；および、２）ゲッターがキャッシュゲッターと同じアーキテクチャに位置する場合、キャッシュゲッターについて、「符号化されていない（unencoded）」場合と同じ性能を有する。

【００７４】

具体例として、キャッシュセッターはデータ構造を用いて、シリアライズされたバイトをそれら自体のフォーマット（たとえば、ビッグエンディアンまたはリトルエンディアン）でキャッシュに格納することができ、かつ、データがどの「エンディアン」にあるかを指定する追加情報を格納することができる。キャッシュゲッターは、データを検索すると、追加情報を用いて、データをローカルフォーマットに変換することができる。

【００７５】

一実施形態に従うと、異種環境においては、ライブラリは各々のマシンに提供することができ、このため、受取るマシンは、キャッシュセッターのマシンのアーキテクチャにかかわらず、ライブラリを用いて、キャッシュされたデータを変換することができる。

【００７６】

そのため、データ構造は、キャッシングシステムの性能を向上させることができ、Ｔｕｘｅｄｏ型付きバッファを含むさまざまなデータタイプのシリアライズされたバイトを格納することができる。

【００７７】

一実施形態に従うと、型付きバッファのユーザＴＣＭをシリアライズするために２つのステップを実行することができる。第１のステップは表示することであり、第２のステップは符号化することである。上述のとおり、シリアライゼーションおよびデシリアライゼーションのための機能はＢＤＣ層におけるコールバック機能において実現される。

【００７８】

一実施形態に従うと、型付きバッファのユーザＴＣＭをシリアライズすることができるものの、不必要なデータを減らすために型付きバッファのユーザデータおよびＴ－バッファだけがキャッシュされる必要があり、型付きバッファの残りはキャッシュされる必要がない。

【００７９】

リスト４は、一実施形態に従った型付きバッファの例示的なヘッダを示す。

【００８０】

【数５】

```
typedef
 struct{
       char magic;                 /**< magic char, 0x0D (tuxedo13c)*/
       unsigned char majorversion; /**< major version, 1-511*/
       unsigned char minorversion; /**< minor version, 0-511*/
       unsigned char len;          /**< integers(4 bytes) used by the header*/
       TM32I flag;                 /**< flags, 0x01 - body encoding */
       unsigned short exlen;    /**< integers(4 bytes) used by the header extension/
       unsigned short type;       /**< data type*/
       TM32I vdatalen;             /**< vdata length */
       char vdata[4];               /**< place holder for body*/
}TMTDCCacheDataBuffer;
```

リスト4

【００８１】

図７は、一実施形態に従った、分散型キャッシングシステムによって使用されるデータシリアライゼーションをサポートするための方法を示す。

【００８２】

図７に示されるように、ステップ７１１において、分散型キャッシングシステムがトランザクション処理環境において提供され得る。

【００８３】

ステップ７１３において、複数のデータタイプのシリアライズされたストリーム／バイトを格納するために用いられるデータ構造が提供され得る。データ構造は、キャッシュ設定アプリケーションを実行するシステムアーキテクチャについての情報を含むヘッダを含む。

【００８４】

ステップ７１５において、異なるシステムアーキテクチャ上で実行されるキャッシュ入手アプリケーションはキャッシュされたデータを検索し、ソースシステムアーキテクチャについての情報を用いて、異なるアーキテクチャ上で使用できるようにデータを変換する。

【００８５】

Coherenceとの統合

一実施形態に従うと、キャッシングプロバイダ（たとえば、CoherenceまたはCloudStore）は構成によってキャッシングプロバイダとして用いることができる。

【００８６】

一実施形態に従うと、分散型インメモリ・データグリッド（たとえば、Coherence）を、キャッシングプロバイダとしての分散型キャッシングシステムに統合するためのシステムおよび方法がこの明細書中に記載される。分散型キャッシングシステムにおけるプロキシサーバは分散型インメモリ・データグリッドに対するクライアントとして機能し得るとともに、ＩＤＣクライアントから転送されたキャッシング要求を受取り得る。始動時に、プロキシサーバは、キャッシング・システム・キャッシュを定義する構成ファイルと、分散型インメモリ・データグリッドキャッシュにキャッシュするマップとをロードすることができ、キャッシング・システム・キャッシュの名前を用いて、サービスをアドバタイズすることができる。要求されたサービスを指定するキャッシング要求をキャッシング・クライアントから受取ると、プロキシサーバは、要求されたサービスに基づいてアクセスするために、分散型インメモリ・データグリッドにおける対応するキャッシュを決定することができる。

【００８７】

一実施形態に従うと、キャッシュされるべきデータはシリアライズされてから、プロキシサーバに転送することができ、このプロキシサーバは、シリアライズされたデータを対応するインメモリ・データグリッドキャッシュに格納することができる。キャッシュされたデータが、シリアライズされたバイトとして対応するインメモリ・データグリッドキャッシュから検索されて、キャッシング・クライアントに送り返されると、キャッシング・クライアントは、データを元のデータにデシリアライズすることができる。

【００８８】

図８は、一実施形態に従った、キャッシングプロバイダとしての分散型キャッシングシステムに分散型インメモリ・データグリッドを統合するためのシステムを示す。

【００８９】

図８に示されるように、複数のコンピュータノード（たとえば、ノードＡ８０５およびノードＢ８０７）は、トランザクション処理環境（たとえば、Ｔｕｘｅｄｏサーバ環境）におけるクラスタまたはドメインにおいて機能するように構成することができる。各々のコンピュータノードはアプリケーションサーバ（たとえば、アプリケーションサーバＡ８１１およびアプリケーションサーバＢ８１３）を含み得る。

【００９０】

図８にさらに示されるように、分散型インメモリ・データグリッドクラスタ（たとえば、Coherence）８３１は、複数のコンピュータノード上でサポートされ得る。分散型インメモリ・データグリッドは、複数のコンピュータノードにわたって分散された複数のメンバを含み得る。たとえば、CoherenceメンバＡ８１９およびCoherenceメンバＣ８２３は、コンピュータノードＡ上に存在し得るとともに、CoherenceメンバＢ８２１およびCoherenceメンバＤ８２５はコンピュータノードＢ上に存在し得る。

【００９１】

一実施形態に従うと、１つ以上のプロキシサーバは、ロードバランシングおよび性能向上のために各々のコンピュータノード上に設けることができる。たとえば、Coherenceのためのプロキシサーバ８１５および８１７は、それぞれ、コンピュータノードＡおよびコンピュータノードＢ上に設けられる。

【００９２】

一実施形態に従うと、各々のプロキシサーバは、Ｔｕｘｅｄｏ　Ｊａｖａ（登録商標）サーバによって実現することができ、分散型キャッシングシステムにおいてサーバとして機能し得る。キャッシング・クライアント（たとえば、Ｔｕｘｅｄｏクライアントまたはサーバ）からのキャッシング要求は、トランザクション手順を呼出すことによって各々のプロキシサーバに転送され得る。各々のプロキシは、その後、１つ以上のキャッシング要求を分散型インメモリ・データグリッドに転送して、対応する応答を受取ることができる。

【００９３】

一実施形態に従うと、各々のプロキシサーバは、分散型インメモリ・データグリッドに対するＪａｖａ（登録商標）クライアントとして、直接、機能し得る。構成ファイルは、分散型インメモリ・データグリッドに如何にアクセスするかを指定することができ、たとえば、１つ以上の分散されたキャッシュまたは複製されたキャッシュにアクセスするべきかどうかを指定することができる。読取りのアクセスが高く、および書込みのアクセスが低い場合、複製されたキャッシュをアクセス用に構成することができる。

【００９４】

図９は、一実施形態に従った、キャッシングプロバイダとしての分散型キャッシングシステムに分散型インメモリ・データグリッドを統合するためのシステムをさらに示す。

【００９５】

図９に示されるように、プロキシサーバ９３０は、Coherenceのためのキャッシングプロバイダスイッチインプリメンテーション９４３からトランザクション手順の呼出し９２９を受取るために、ＣＤＣ層に存在し得る。トランザクション手順の呼出しは、キャッシング・クライアントから分散型キャッシングシステムへのキャッシング要求を受取った後

、キャッシングプロバイダ・スイッチ・インプリメンテーションをホストしているサーバによって生成することができる。トランザクション手順の呼出しは、共通の型付きバッファから（９２８）のシリアライズされたバイトを含み得る。

【００９６】

一実施形態に従うと、Coherenceのためのキャッシングプロバイダ・インプリメンテーションは、ＩＤＣ層からＣＤＣ層に提供されるＩＤＣクライアントであり得る。Coherence９３２はデフォルトのキャッシングプロバイダとなるように構成することができる。このため、プロバイダスイッチのインプリメンテーションと、プロバイダスイッチインプリメンテーションをロードするのに用いられる方法とが、Ｔｕｘｅｄｏダイナミックライブラリlibtuxおよびlibwsに統合され得る。この場合、前者はＴｕｘｅｄｏサーバおよび固有のクライアントによって使用され得るとともに、後者はＴｕｘｅｄｏ／ＷＳクライアントのために使用され得る。

【００９７】

一実施形態に従うと、Coherenceに基づいたプロバイダスイッチ・インプリメンテーションまたはＩＤＣクライアントは、tpcallによってプロキシサーバにキャッシング要求を転送することができる。不必要なコピーを減じるようにとの要求のタイプに基づいて、さまざまなバッファタイプを用いることができる。表３は、キャッシング要求のタイプに基づいて用いることができる型付きバッファのサンプルリストを示す。

【００９８】

【表３】

要求タイプ/コマンド	ユーザTCM	MetaTCM (FML32のタイプ)
Getcache	STRING(空)	コマンドを格納
Put	CARRAY (キャッシュされるべきデータを格納)	コマンドおよびキーを格納
Get	STRING (キーを格納)	コマンドを格納
Remove	STRING (キーを格納)	コマンドを格納
Mremove	STRING (空)	コマンドおよびキー配列を格納
Removeall	STRING (空)	コマンドを格納

表3

【００９９】

一実施形態に従うと、プロキシサーバは、始動されると、構成ファイルから構成プロパティをロードすることができる。この場合、構成プロパティが定義することができる提供されたＴｕｘｅｄｏキャッシュは、分散型キャッシングシステムにおける、構成された論理キャッシュまたは物理キャッシュであり得る。プロキシサーバは、Ｔｕｘｅｄｏキャッシュ名を用いて、サービスをアドバタイズすることができる。構成ファイルからのプロパティのリストの例を以下のリスト５に示すことができる。

【０１００】

【数６】

```
#global option encoding setting
#options.encoding=no
#* configurations for Tuxedo cache "tc"
#* option encoding setting
#cache.options.encoding.tc=no
#* physical cache used in Oracle Coherence
coh.cache.name.tc=tux_distributed
#* configurations for Tuxedo cache "tc2"
#* option encoding setting
#cache.options.encoding.tc2=no
#* physical cache used in Oracle Coherence
#coh.cache.name.tc2=tux2_distributed
```

リスト5

【０１０１】

一実施形態に従うと、上述のプロパティであれば、プロキシサーバは、「tc」という名のＴｕｘｅｄｏサービスをアドバタイズすることができる。ＩＤＣクライアントは、tpcallによってプロキシサーバがアドバタイズしたサービス「tc」に対し、キャッシュ「tc」についての要求を転送することができる。リスト３に示されるように、プロパティは、分散型キャッシングシステムにおいて構成されたキャッシュについての、マップされたCoherenceキャッシュを指定することができる。キャッシング・クライアントがコマンド「get cache」によってキャッシュを取得する必要がある場合、プロキシサーバは、このキャッシュのために必要なすべてのプロパティ（たとえば、プロパティoptions.encoding）をキャッシング・クライアントに送り返すことができる。

【０１０２】

一実施形態に従うと、要求されたサービスの名前が、構成ファイルによってCoherenceキャッシュにマッピングされたＴｕｘｅｄｏキャッシュの名前と同じであり得る場合、プロキシサーバは、要求されたサービスに基づいてCoherenceキャッシュ名を決定することができる。

【０１０３】

一実施形態に従うと、プロキシサーバはCoherenceのクライアントとして機能することができる。プロキシサーバは、プロキシサーバによって用いられるCoherence構成ファイルによって定義されるように、固有のクライアント（Coherenceクラスタのメンバ）またはリモートクライアントであり得る。

【０１０４】

一実施形態に従うと、Coherenceキャッシュは、アプリケーションの要件によって決定されるように、分散型キャッシュ、ローカルキャッシュまたは他の任意のタイプのキャッシュであり得る。

【０１０５】

一実施形態に従うと、キャッシュにデータを格納するために、たとえば「入れる（put）」というコマンドがアプリケーションにおいて呼出されると、データはシリアライズされてからプロキシサーバに転送され得る。プロキシサーバがバイト［］のタイプであり得るシリアライズされたデータを受取ると、プロキシサーバはシリアライズされたデータをCoherenceに格納することができる。

【０１０６】

同様に、データをキャッシュから検索するために、たとえば「入手する（get）」というコマンドがアプリケーションにおいて呼出されると、キャッシュされたデータは、バイト［］のタイプを備えたCoherenceから検索されて、さらにキャッシング・クライアント

に送り返され、このキャッシング・クライアントが、データをその元のフォーマットにデシリアライズすることができる。

【０１０７】

図１０は、一実施形態に従った、キャッシングプロバイダとしての分散型キャッシングシステムに分散型インメモリ・データグリッドを統合するための方法を示す。

【０１０８】

図１０に示されるように、ステップ１０１１において、プロキシサーバは、トランザクション処理環境において分散型キャッシングシステムに提供することができる。プロキシサーバは、キャッシング・システム・キャッシュを定義する構成ファイルと、分散型インメモリ・データグリッドキャッシュにキャッシュするマップとをロードし、キャッシング・システム・キャッシュの名前を用いてサービスをアドバタイズすることができる。

【０１０９】

ステップ１０１３において、プロキシサーバは、キャッシング要求を受取ることができる。キャッシング要求は、分散型キャッシングシステムのＩＤＣクライアントからのトランザクション手順の呼出しの際に転送される。

【０１１０】

ステップ１０１５において、プロキシサーバは、トランザクション手順の呼出しの際に、要求されたサービスに基づいてアクセスするために、分散型インメモリ・データグリッドにおける対応するキャッシュを決定することができる。

【０１１１】

サービスキャッシング

上述の分散型キャッシングシステムは以下のキャッシング特徴を提供することができる。すなわち、アクセス回数および更新回数が極めて少なく、キーによって識別されたＴｕｘｅｄｏバッファをキャッシュすることができ、複製アクセス、非複製アクセス、ローカルアクセスおよびリモートアクセスを含む調整可能な品質サービスを提供する、というキャッシング特徴を提供することができる。

【０１１２】

一実施形態に従うと、トランザクション処理環境において分散型キャッシングシステムを用いて、サービスから戻された結果をキャッシュするためのシステムおよび方法がこの明細書中に記載される。構成ファイルは、サービスから戻された結果をキャッシュするのにどのキャッシュを用いるべきかと、キャッシュされた結果を識別する際に使用されるキーを如何に生成するかとを記述するエントリを含むキャッシングセクションを含み得る。サービスについての要求がクライアントから受取られると、トランザクション処理環境のアプリケーションサーバコアは、関連するキャッシュされたデータが、構成ファイルを用いて生成されたキーによって識別されるキャッシュに存在しているかどうかを判断することができる。存在している場合、アプリケーションサーバコアは、サービスを呼出す代わりに、キャッシュされたデータを直接戻すことができる。他の場合、アプリケーションサーバコアは、サービスを呼出し、生成されたキーを用いて、構成ファイルによって指定されるキャッシュにデータをキャッシュし、クライアントに結果を戻すことができる。

【０１１３】

一実施形態に従うと、キャッシュされるべきサービスについて戻された結果は、サービス要求における入力キーワードに基づいた、データベースからの検索結果であり得る。

【０１１４】

サービス（たとえば、Ｔｕｘｅｄｏサービス）が、或る期間内で特定の要求に応じて、同じ結果を戻すのに比較的長い時間を費やし得る場合、サービスキャッシング特徴により、システム性能を著しく向上させることができる。加えて、このキャッシング特徴は、ユーザが、既存のコードを変更することなく、サービスから戻された結果をキャッシュすることを可能にする。

【０１１５】

一実施形態に従うと、構成ファイルは、キャッシュされた結果を識別するために用いら

れるべきキーを如何に生成するかを指定することができる。このような識別用途のキーは、単純な連続ストリング（simple solid string）、サービス名から構成されるストリング、要求データから構成されるストリング、ならびに、サービス名および要求データから構成されるストリング、のうちの１つであり得る。

【０１１６】

一実施形態に従うと、要求データがキーを生成するために用いられるべき場合、要求データ全体またはデータの一部を、要求データを含んでいる型付きバッファに従ってキーの一部として用いることができる。ユーザは以下を用いることができる。

【０１１７】

１）開始インジケータ／終了インジケータによって識別されるＳＴＲＩＧＮ／ＣＡＲＲＡＹバッファの一部；

２）ＶＩＥＷ／ＶＩＥＷ３２バッファのうちの１つのフィールドもしくはいくつかのフィールド；

３）ＦＭＬ／ＦＭＬ３２バッファのうちの１つのフィールドもしくはいくつかのフィールド；または、

４）ＸＭＬバッファのうちの１つのフィールドもしくはいくつかのフィールド。

【０１１８】

一実施形態に従うと、トランザクション処理環境においては、ＦＭＬ型バッファなどの型付きバッファは、複数セットの名前値ペアを含み得る。型付きバッファは、クライアントからサーバに要求を送信し、サーバからクライアントに結果を戻すために用いることができる。キャッシュされたサービス結果は、特定のキャッシング要求についての外部からのデータを含み得る複数セットの名前値ペアを含み得る。たとえば、キャッシュされた名前値ペアのうちのいくつかは特定の要求のために必要とされないかもしれないが、但し、これらの名前値ペアは後続のキャッシング要求のために必要とされるかもしれない。

【０１１９】

そのため、一実施形態に従うと、要求データから生成されたキーを用いることにより、クライアントアプリケーションが、必要なデータを正確に示し、それらのデータだけを検索することが可能となり、これにより、アクセス時間を減らし、性能を増強させることができる。

【０１２０】

図１１は、一実施形態に従った、トランザクション処理環境において分散型キャッシングシステムを用いて、サービスから戻された結果をキャッシュするためのシステムを示す。

【０１２１】

図１１に示されるように、分散型キャッシングシステムは、分散されたかまたは複製された複数のキャッシュ（たとえば、キャッシュＡ１１２１およびキャッシュＢ１１２３）を含み得るキャッシングプロバイダ１１１９としてCoherenceを用いるように構成することができる。

【０１２２】

さらに図示されるように、トランザクション処理環境のための構成ファイル１１０４は、分散型キャッシングシステムを如何に用いるかを記述するエントリを含み得るサービスキャッシングセクション１１０７を含み得る。

【０１２３】

たとえば、キャッシングセクションは、サービスＡのためのキャッシュＡ１１１０およびサービスＢのためのキャッシュＢ１１１１によって例示されるように、特定のサービスから戻されたものをキャッシュするためにどのキャッシュを使用すべきかを記述することができる。

【０１２４】

一実施形態に従うと、セクションにおける追加のエントリは、サービスＡのためのキーを生成するための方法１１１３、およびサービスＢのためのキーを生成するための方法１

１１５によって示されるように、特定のサービスからのキャッシュ済み結果を識別する際に使用すべきキーを如何に生成するかを記述することができる。上述のとおり、キー生成方法は、要求データを含むバッファのタイプに基づいて、キーを生成する際に、要求におけるデータのうちどのフィールドを使用すべきかを定義することができる。

【０１２５】

図１１を参照して、アプリケーションサーバコア（たとえば、Ｔｕｘｅｄｏコア）１１１４は、戻された結果をサービス１１０２からクライアントアプリケーション（たとえば、Ｔｕｘｅｄｏクライアント）１１０１に渡すために提供することができる。アプリケーションサーバコアは、複数のサービス（たとえば、サービスＡ１１０５およびサービスＢ１１０９）を実行することができるアプリケーションサーバＡ（たとえば、Ｔｕｘｅｄｏサーバ）１１１７に関連付けることができる。アプリケーションサーバコアはまた、結果についての特定のサービスを呼出すべきか、またはキャッシュにキャッシュされた結果を用いるべきかどうかを判断する際に使用されるサービスキャッシング論理１１２８にも関連付けることができる。

【０１２６】

具体例として、サービスＡ１１２５についての要求がアプリケーションサーバによって受取られると、それに関連付けられたアプリケーションサーバコアは、サービスＡのために構成されたキャッシュをチェックして、構成ファイルにおけるサービスについての関連するエントリに従って生成されたキー１１２７によって識別されるような、関連するキャッシュデータがキャッシュに存在するかどうかを判断することができる。存在する場合、アプリケーションサーバは、サービスＡを呼出す代わりに、直接、キャッシュされたデータを戻すことができる。存在しない場合、アプリケーションサーバコアはサービスＡを呼出し（１１０８）、クライアントアプリケーションに結果を戻すことができる。クライアントアプリケーションに結果を転送し返す前に、アプリケーションサーバコアは、生成されたキーを用いるデータを、サービスＡによって使用されるように構成されたキャッシュにキャッシュすることができる。

【０１２７】

図１２は、一実施形態に従った、トランザクション処理環境において分散型キャッシングシステムを用いて、サービスから戻された結果をキャッシュするための方法を示す。

【０１２８】

図１２に示されるように、ステップ１２１１において、分散型キャッシングシステムは、特定のサービスのために戻された結果をキャッシュするためにどのキャッシュを使用すべきかと、キャッシュされた結果を識別する際に使用されるキーを如何に生成すべきかとを記述しているエントリを備えた構成ファイルを含み得るトランザクション処理環境において提供することができる。

【０１２９】

ステップ１２１３において、アプリケーションサーバに関連付けられたアプリケーションサーバコアは、サービスについての要求が、サービスをホストしているアプリケーションサーバに対してクライアントアプリケーションから送信されていることを検出することができ、関連するキャッシュされたデータが構成ファイルを用いて生成されたキーによって識別されるキャッシュに存在しているかどうかを判断することができる。

【０１３０】

テップ１２１５において、関連するキャッシュされたデータがキャッシュに存在している場合、アプリケーションサーバコアはサービスを呼出す代わりに、キャッシュされたデータを直接戻すことができる。他の場合、アプリケーションサーバコアは、サービスを呼出し、構成ファイルによって指定されたキャッシュに、生成されたキーを用いるデータをキャッシュし、クライアントに結果を戻すことができる。

【０１３１】

この発明は、この開示の教示に従ってプログラミングされた１つ以上のプロセッサ、メモリおよび／またはコンピュータ読取り可能記憶媒体を含む、１つ以上の従来の汎用また

は特化型デジタルコンピュータ、コンピューティング装置、マシン、またはマイクロプロセッサを使用して都合よく実現されてもよい。ソフトウェア技術の当業者には明らかであるように、この開示の教示に基づいて、適切なソフトウェアコーディングが、熟練したプログラマによって容易に準備され得る。

【０１３２】

実施形態によっては、本発明は、本発明のプロセスのうちいずれかを実行するためにコンピュータをプログラムするのに使用できる命令が格納された非一時的な記憶媒体または（１つもしくは複数の）コンピュータ読取り可能な媒体であるコンピュータプログラムプロダクトを含む。この記憶媒体は、フロッピー（登録商標）ディスク、光ディスク、ＤＶＤ、ＣＤ－ＲＯＭ、マイクロドライブ、および光磁気ディスクを含む、任意の種類のディスク、ＲＯＭ、ＲＡＭ、ＥＰＲＯＭ、ＥＥＰＲＯＭ、ＤＲＡＭ、ＶＲＡＭ、フラッシュメモリデバイス、磁気もしくは光カード、ナノシステム（分子メモリＩＣを含む）、または、命令および／もしくはデータを格納するのに適した任意の種類の媒体もしくはデバイスを含み得るものの、これらに限定されない。

【０１３３】

本発明のこれまでの記載は例示および説明を目的として提供されている。すべてを網羅するかまたは本発明を開示された形態そのものに限定することは意図されていない。当業者には数多くの変更および変形が明らかであろう。実施形態は、本発明の原理およびその実際の応用を最もうまく説明することによって他の当業者がさまざまな実施形態および意図している特定の用途に適したさまざまな変形を理解できるようにするために、選択され説明されている。本発明の範囲は添付の特許請求の範囲およびそれらの同等例によって規定されるものと意図されている。

【図１】

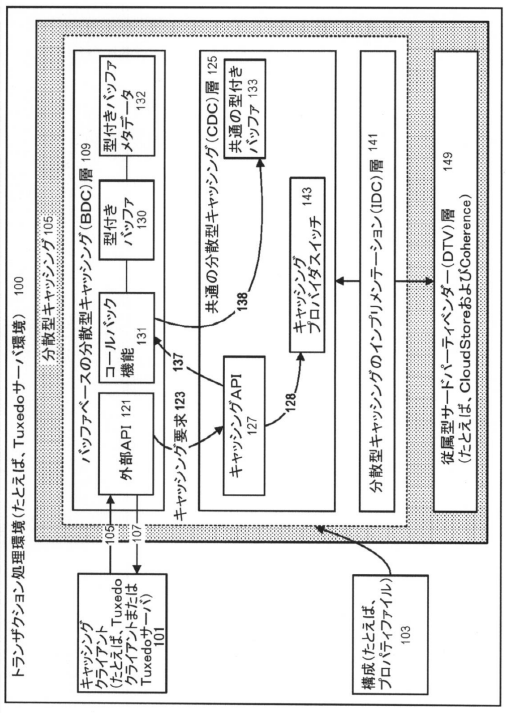

FIGURE 1

【図２】

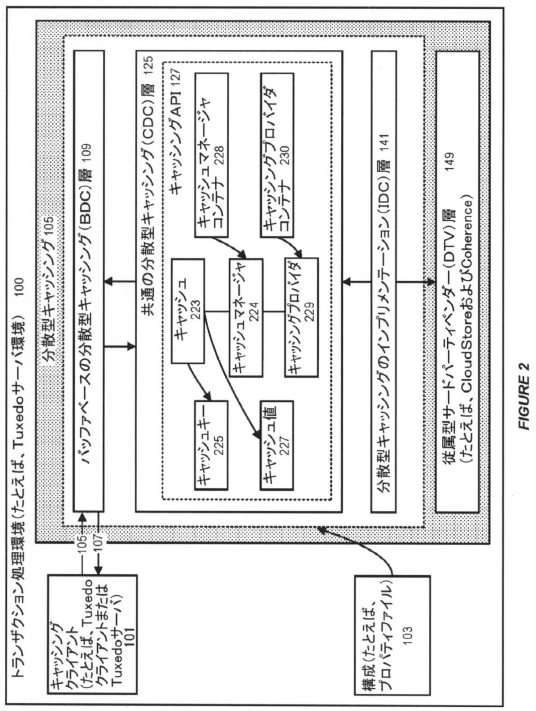

FIGURE 2

【図３】

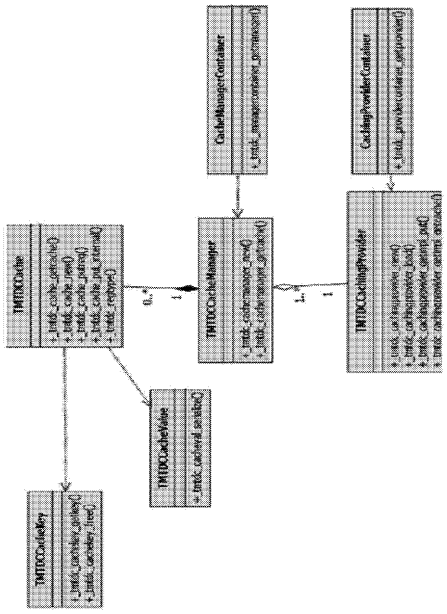

FIGURE 3

【図４】

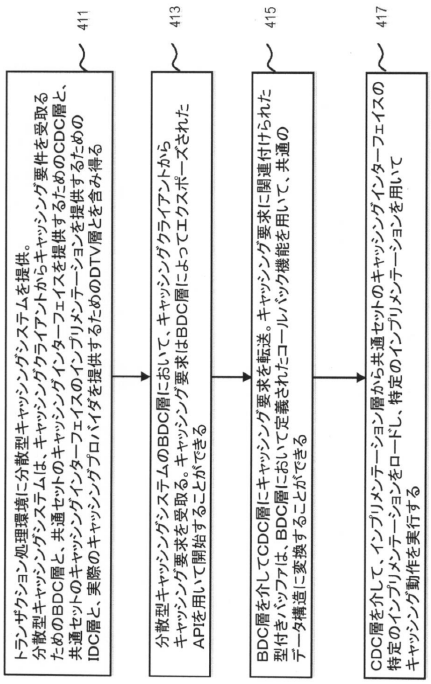

FIGURE 4

【図５】

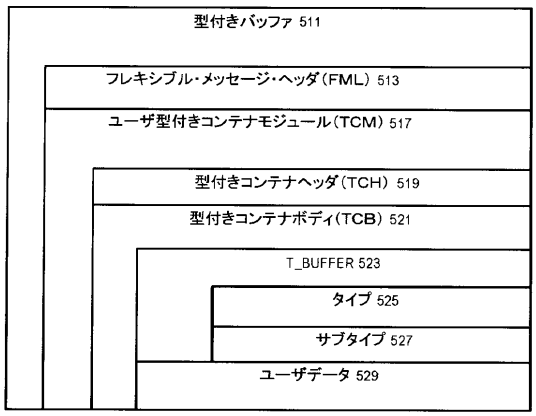

FIGURE 5

【図６】

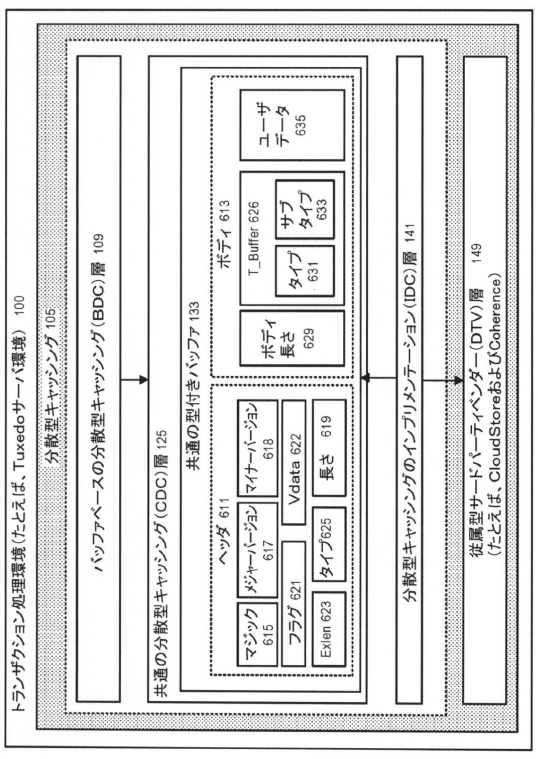

FIGURE 6

【図７】

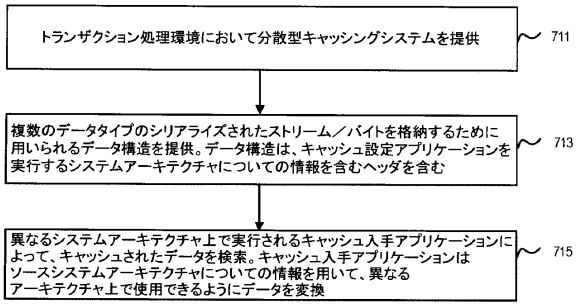

FIGURE 7

【図８】

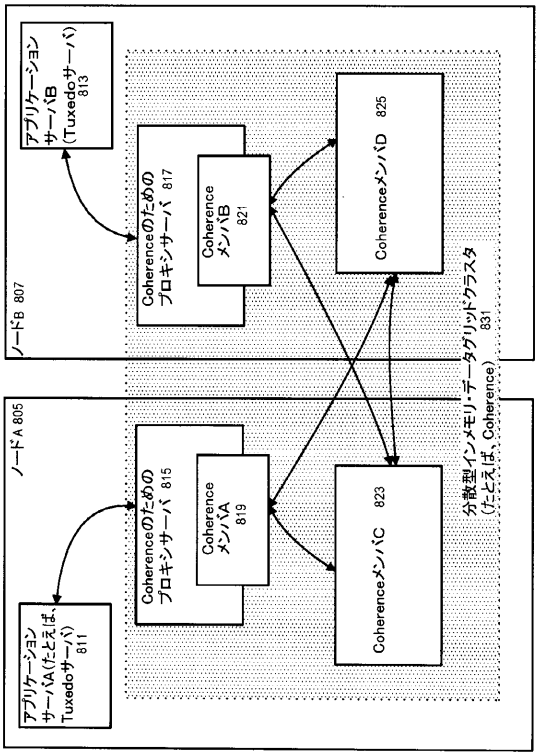

FIGURE 8

【図９】

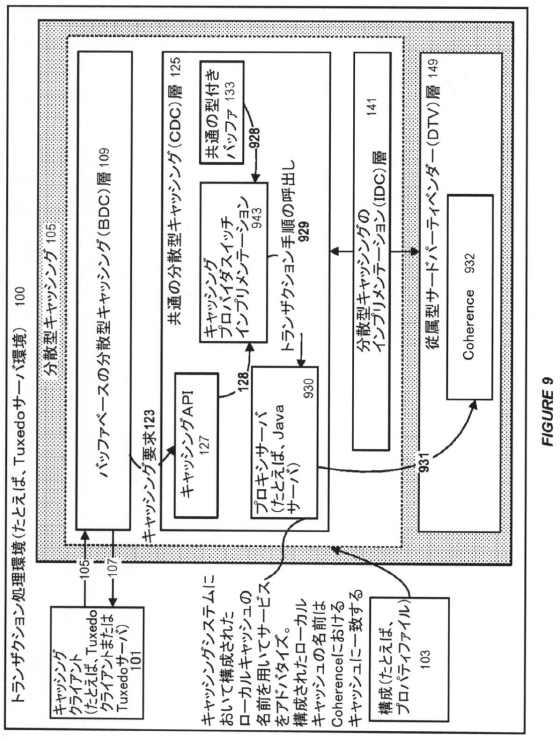

FIGURE 9

【図１０】

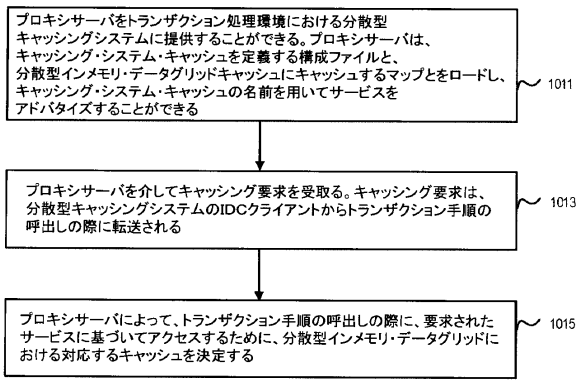

FIGURE 10

【図１１】

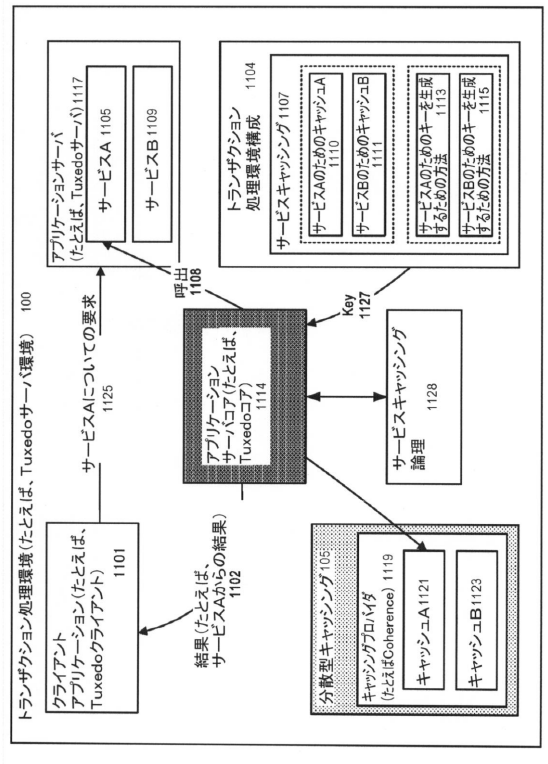

FIGURE 11

【図１２】

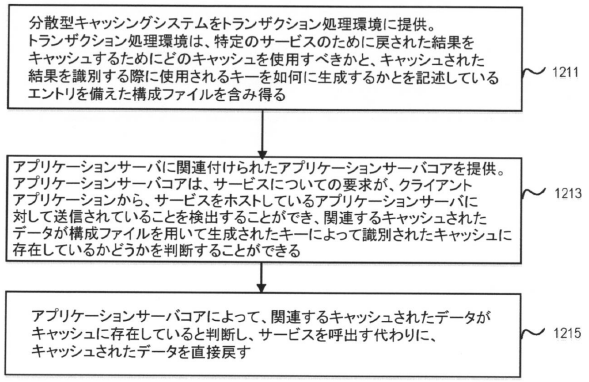

FIGURE 12

フロントページの続き

(72)発明者　リトル，トッド
アメリカ合衆国、６００６７－６６７５　イリノイ州、パラタイン、ウエスト・イリノイ・アベニュ、１１５５
(72)発明者　ジン，ジム・ヨンシュン
中華人民共和国、１００１９３　ベイジン、ハイディアン・ディストリクト、ヂョングアンツン・ソフトウェア・パーク、ビルディング・ナンバー・２４、オラクル・ビルディング
(72)発明者　ホウ，ジェシー
中華人民共和国、１００００００　ベイジン、ハイディアン・ディストリクト、シサンチ・ストリート、フェイリボーティング、ナンバー・１

審査官　鹿野　博嗣

(56)参考文献　米国特許出願公開第２０１２／００５４４４０（ＵＳ，Ａ１）
特表２０１４－５２９１１１（ＪＰ，Ａ）

(58)調査した分野(Int.Cl.，ＤＢ名)
Ｇ０６Ｆ　１６／１７２