論文 / 著書情報
Article / Book Information

題目(和文)	大規模データ処理基盤へのエージェント適用に関する研究
Title(English)	
著者(和文)	村田悠也
Author(English)	Yuya Murata
出典(和文)	学位:博士（工学）, 学位授与機関:東京工業大学, 報告番号:甲第10378号, 授与年月日:2016年12月31日, 学位の種別:課程博士, 審査員:寺野 隆雄,新田 克己,出口 弘,三宅 美博,小野 功
Citation(English)	Degree:, Conferring organization: Tokyo Institute of Technology, Report number:甲第10378号, Conferred date:2016/12/31, Degree Type:Course doctor, Examiner:,,,,
学位種別(和文)	博士論文
Type(English)	Doctoral Thesis

大規模データ処理基盤へのエージェント適用に関する研究

知能システム科学専攻

指導教員: 寺野隆雄

村田悠也

2016 年 12 月

目 次

第1章　序論

1.1　背景

システムから発生するデータや蓄積されるデータは年々増加し，その量や発生速度からビッグデータと呼ばれる [13]．特に，金融取引やモバイルアプリケーション，IOT，MVNO などの通信サービスで用いられるシステムでは短時間に継続的かつ大量にデータが発生する．このようなシステムは，ユーザーやデバイスの増加，サービスの拡張等により短時間に発生するデータが爆発的に増加する．例えば，自身の位置情報を利用したモバイルアプリケーションでは，GPS センサーの更新間隔から 1 秒間にユーザー数分のデータが発生する．さらに，利用者間での位置情報の共有やソーシャルサービスとの連携を考えると発生するデータ数はさらに増大する．アクティブユーザー数が 1 万人のとき，利用者間で位置情報を共有すると，1 万× 1 万の秒間 1 億件のデータが発生する．このアプリケーションはユーザーやサービスが 1 つ増えるごとにおよそ 2 万件のデータが毎秒追加され，データを蓄積してから処理しようとすると，ユーザーのアプリケーション利用時間が 1 時間のとき 3600 秒× 1 億件の 3600 億件のデータが蓄積されることになる．一般的なデータベースの秒間の実行可能クエリ数は 10 万件程度であることから，全データの処理に 100 時間かかる計算となる．また，1 レコードあたり 100 バイトとすると蓄積データサイズは 360TB と非常に大きい．上記のように短期間に継続的に発生するデータはストリームデータと呼ばれている．ストリームデータを集計するシステムでは，従来行われてきたデータ蓄積し処理するバッチ処理ではなく，データ到着と同時にリアルタイムに集計処理を行うことが望ましい．このように，ストリームデータをリアルタイムに集計する処理をリアルタイムデータ集計処理と呼び，その実現には厳しい処理性能課題の解決が必要となる．近年，高いデータ処理性能を持つエージェント技術をリアルタイムデータ集計処理に適用した高速マルチエージェントシステムが考案されているが，その設計や運用は熟練的技術を必要としたものとなっており，エージェントの導出法や実行中に発生する運用上の問題は明らかになっていない．

1.2 研究目的

本研究では，秒間数万〜数百万件のストリームデータをリアルタイムに集計処理するデータ処理基盤に，高いデータ処理性能を持つエージェント技術を適用するにあたり，処理性能課題を解決するための効果的なエージェント設計方法論と運用技術の確立をすることにある．本論文では，熟練的技能を必要としないエージェントシステムの運用およびその設計方法論とエージェント自身が動的に負荷分散を行う動的負荷分散機構を開発した．さらに，ストリームデータをリアルタイムに集計するシステムにおいて，従来データベース上で行われてきた複数の集計機能を必要とする複雑な集計システムへのエージェント適用を行った．ここで開発された運用技術，設計・運用方法論は実験やシミュレーションによりスケーラビリティ性能や問題への有効性について評価した．

1.3 論文構成

本論文構成について説明する．2章では，はじめに本研究の関連領域である大規模データ処理基盤に適用されている並列分散処理用のプログラミングモデルとデータ処理性能に優れたエージェント技術について解説する．次に，本研究で注目する大規模データのリアルタイムに集計する処理手法であるリアルタイムデータ集計処理について説明し，並列分散処理用のプログラミングモデルでの実現可能性について議論する．最後に，本研究で用いるエージェント技術について，モバイルエージェントや知的エージェントなどの他のエージェント技術と比較し，その違いを述べる．3章では，単純なストリームデータの集計システムであるセンサーログマイニングシステムを例に，エージェントシステムの運用問題を明らかにし，効果的な運用技術を開発する．4章では，センサーログマイニングより複雑な集計を必要とするランキングシステムについて，エージェント適用のための設計方法論を提案し，その効果を実験により検証する．特に，設計したエージェントシステムのスケーラビリティ性能やエージェントの分散配置問題について評価・考察を行う．5章で，本研究の成果をまとめ今後の課題を述べる．

第2章　関連する研究領域の手法

2.1　データ処理基盤のプログラミングモデル

この章では，本研究に関連する研究領域として，大規模データやストリームデータのデータ処理基盤を実現する並列分散処理用のプログラミングモデルについて説明する．

2.1.1　MapReduce

MapReduce は巨大なデータセットを分割し，クラスタマシン上に分散することで，高速なデータ処理を実現する並列分散処理用のプログラミングモデルである [11]．Hadoop[5] や Dryad[15] といった分散コンピューティング用のフレームワークで利用されている．MapReduce のデータはキーとバリューのペアで表現され，その処理は入力データのフィルタリングを行う Map フェーズと処理結果を集約する Reduce フェーズの 2 フェーズを段階的に適用することで実現する．データは，クラスタマシン上に分散され並列に処理される．これにより，クラスタマシンを増やすことで単一計算機上では不可能であった大規模データの短時間での集計処理を実行できる．一方，蓄積されたデータしか適用できないため，リアルタイムに発生するデータを処理することはできない．

2.1.2　Resilient Distributed Datasets(RDDs)

RDDs は，データの再利用を目的とした分散データセットである [26]．発生したデータに対して，任意のデータセットに分割し，それらのデータセットに対して並列分散処理を適用する．RDDs の処理では，分散されたデータセットに対して演算子 (Map, Filter, Reduce など) によるデータ変換を行い，新しいデータセットを生成する．データセットは書き換え不可のため，繰り返し利用することができる．さらにデータセットのサイズを小さくすることで，リアルタイムでのデータ処理が実行可能である．しかしながら，データの書き換えが必要な集計処理では RDDs のデータをデータベースを介して処理する必要があるためデータベースへのアクセスがボトルネックとなる．また，データセットを作成するにあたりデータ蓄積が必要となるため，データを 1 件ずつ取り扱うリアルタイム処理は実行できない．

2.1.3　ストリームプログラミングモデル

継続的に発生するストリームデータを高速なキャッシュメモリに蓄積し，予め設計した処理モデルを適用することにより高速なデータ処理を実現するプログラミングモデルである．CQL(Continuous Query Language)[8] や，CEP（Complex Event Processing）[20] といった技術により，時系列解析や複雑なデータ分析をリアルタイムに実行できる．既に，スケーラブルかつ高速なストリーム処理も提案され実現され利用されている．[7]．しかしながら，データ蓄積機能を持たないもしくは一時的な蓄積しか行わないためデータの更新を必要とする集計処理は実現できない．

2.1.4　Orleans

Orleans は Actor モデルを利用したプログラミングモデルである [9]．Actor モデル [1][14] は，Hewitt らにより提案された並行計算の数学的モデルで，Actor モデルの特性である並行性や局所性といった特徴からスケーラブルなシステムを実現可能である [2],[18]．Orleans のアプリケーションは全て一意の ID が割り振られた Actor によって構成されるためスケラービリティの高いシステムとなっている．また，Actor は仮想化されメッセージが送られてくるまで非実体状態となっている．この特性を利用し，クラウド環境上に実体を配置することで耐障害性を持ち永続的な動作を実現する．また動的にスケールアウト (Actor を増やし並列性を向上) することで割り当てられたリソース限界までスループットを向上させる機能を持っている．一方で，Orleans はクラウド上での動作を目的としたプログラミングモデルであるため Actor の実体がシステム上に存在しない．これにより，Actor の活性時に保持するデータやロジックの読み込みに通信が発生する．

2.1.5　エージェントプログラミングモデル (APM)

APM はエージェントと呼ばれる特殊なソフトウェアエンティティを利用したプログラミングモデルである [27]．エージェントプログラミングモデルを用いたエージェントアーキテクチャやエージェント間コミュニケーションの研究には，FIPA[4] や RETSINA[19] がある．エージェントは，メッセージハンドラや処理ロジック，データレコードを持ち，自身のデータを書き換えることでデータ更新を行う (図 1)．また，メモリ上で動作するインメモリデータストアでもあることから高速なデータアクセスを可能とする．エージェントの特性として反応性や非同期性を持ち，環境中に多数配置することで並列分散処理を実行できる (図 2)．

エージェントが持つ機能を以下に示す．また，それぞれの動作は図 1 にあるように，エージェントはメッセージ受信後，メッセージハンドラにより関連する処理ロジックが呼び出す，

呼び出された処理ロジックは，自身が保持するデータレコードを書き換えることで更新処理を行う．

- メッセージハンドラ
  メッセージと処理ロジックを関連付け，メッセージを受信すると対応する処理ロジックを呼び出す．メッセージは排他的に実行され，複数のメッセージが同時に実行されることは無い．

- データレコード
  エージェントが保持するデータで，各エージェントごとに独立に管理されている．そのため，他のエージェントは直接データを書き換えることができない．これにより，データの整合性の維持による排他制御を必要としない．

- 処理ロジック
  処理ロジックは，メッセージハンドラにより呼び出され実行される．処理はプログラミングされ，1 つのメッセージに対して複数の処理ロジックが実行されることもある．

エージェントは上記の機能や特性によりリアルタイム処理，高速なデータアクセス，複数処理の記述によるデータ集計機能を備えているため，リアルタイムでのデータ集計処理が可能となっている．しかしながら，他の手法に比べエージェントへ割り当てるリソース量が多いことやエージェントが多数配置される環境ではエージェントの分散配置問題を解く必要がある [30].

2.2 エージェントシステム

Wooldridge,Jennings[23] によるとエージェントは，自律性，永続性，社会性，反応性の 4 つの性質を持っているものと定義されている．表 1 は，エージェント技術とその性質についてまとめたものである．それぞれの性質は次のような状態を指す．自律性は，人間や他のエージェントが直接操作すること無く動作し，行動や内部状態によって制御されている．永続性は，エージェントが常に起動されている．社会性は，エージェント間で通信を行う．反応性は，環境を認識し環境の変化に反応して動作する．この表から APM により設計されるエージェントは，他のエージェント技術で定義されるエージェントと性質が異なり，自律性が存在しないことがわかる．これは APM のエージェントがメッセージパッシングに動作し，自律的にメッセージを送ったり，状態遷移が発生しないことを示している．そのため，厳密な意

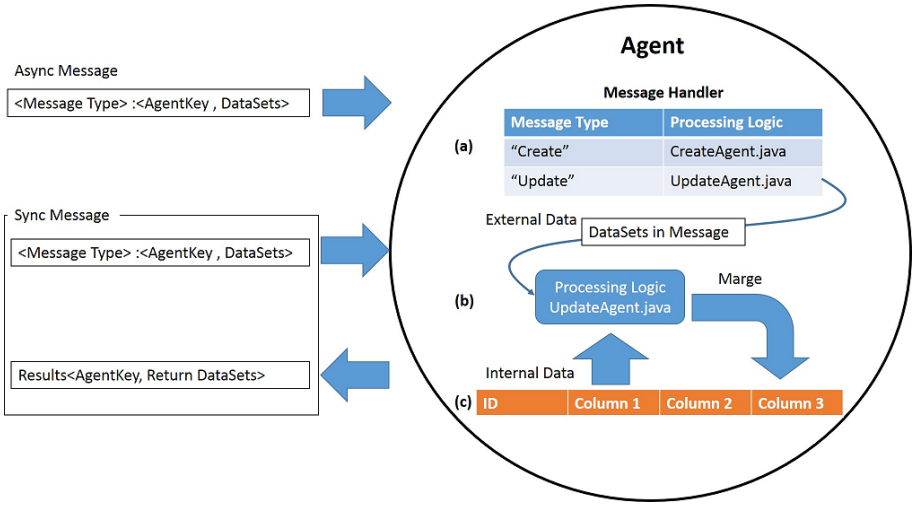

図 1: エージェントプログラミングモデル

味でのエージェントではなくアクターモデルの一種となっている．しかしながら，エージェントの特性である反応性を利用することでリアルタイム処理を実現し，非同期性によりスケラービリティの高いシステムを開発できる．また，エージェントの粒度の高さ (複数の機能をまとめられること) を利用することで図 3 の MapReduce で動作する複雑な集計システムを図 4 のようにシンプルに表現することができる．

2.3　リアルタイムデータ集計処理

リアルタイムデータ集計処理 (Realtime Data Aggregation, RDA) とはストリームデータをリアルタイムに集計する処理手法で，継続的に発生するデータを過去のデータへアクセスしながら繰り返し更新処理を適用することで，常に最新の結果を保つ処理手法である．トレーディングシステムやモバイルアプリケーション，通信サービスの従量課金システムといった，性能要件の厳しいシステムの登場に伴い開発された．RDA は高い処理性能が求められるため，その実現にはストリームデータのリアルタイム処理と高いデータアクセス性能が必要となる．RDA の処理要件を満たすデータ処理基盤は，並列性，リアルタイム性，データベースの 3 つの機能を有するプログラミングモデルにより設計されなければならない (表 2)．データベースの項目は一体型 (フレームワークに備え付けられている) か外部接続型 (単一のフレームワークでは実現不可能) の 2 つに分けられている．データアクセス数が秒間十万件を超える場合，従来利用されていた関係データベース (RDB) やキーバリューストアなどの分散データベース

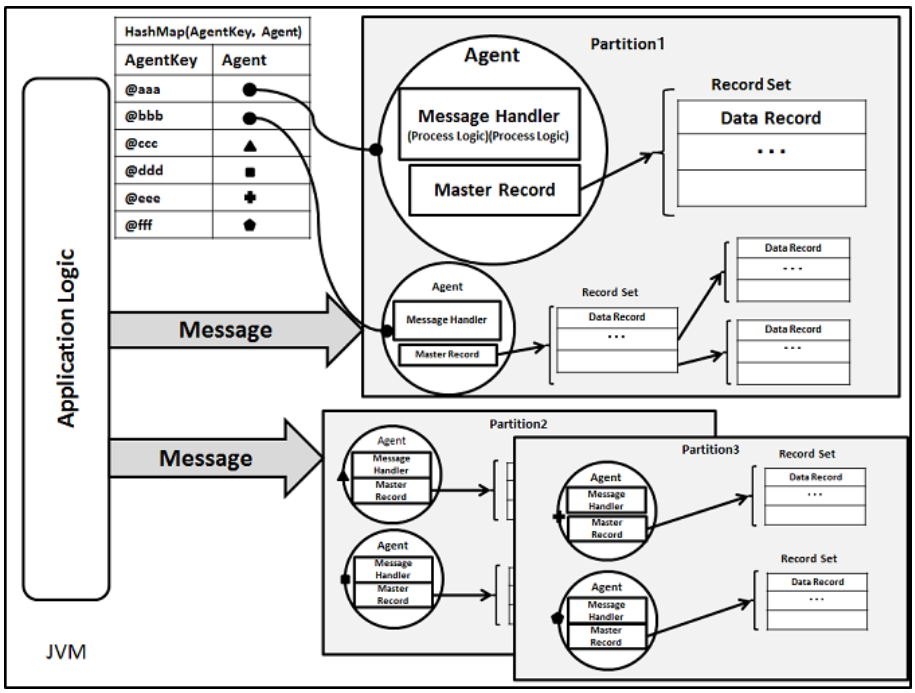

図 2: マルチエージェントシステム

を接続したシステムでは，RDA を実現することは不可能である．なぜなら，データベースが外部接続型の場合，データ整合性維持のための排他処理がデータベースもしくはデータセット (コレクション) のアクセス時に発生しシステム全体の性能が劣化するためである．また表中の△は，データ 1 件ごとのリアルタイム処理は実行できないが，データの蓄積が必要なく処理速度が数ミリ～数 100 ミリ秒以下と非常に高速なとき△としている．

表 2 を見ると APM と Orleans の特性に違いは無いが，システムへの適用目的が異なる．Orleans はクラウド上での実装を目的としているのに対し，APM は専用サーバーでの業務システム適用を目的としているため APM のほうがデータ処理性能は高い．よって，本研究では RDA の実現に APM を用いる．

表 1: エージェントと性質

エージェント技術	自律性	永続性	社会性	反応性
分散エージェント	○	○	○	○
モバイルエージェント	○	○	○	○
自律エージェント	○	○	○	○
知的エージェント	○	○	○	○
APM によるエージェント	×	○	○	○

表 2: RDA データ処理基盤の 3 機能とプログラミングモデルの対応

プログラミングモデル	API	並列性	リアルタイム性	データベース
MapReduce	Hadoop[5], Dryad[15]	○	×	一体型
RDDs	Spark[6]	○	△	外部接続
ストリームプログラミングモデル	STORM[3]	○	○	外部接続
Orleans	Orleans[9]	○	○	一体型
エージェントプログラミングモデル	AgentFramework[27]	○	○	一体型

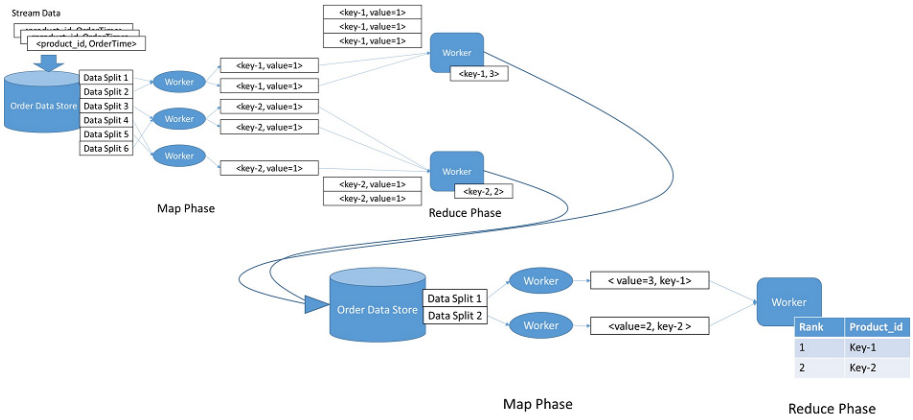

図 3: MapReduce でのランキング集計

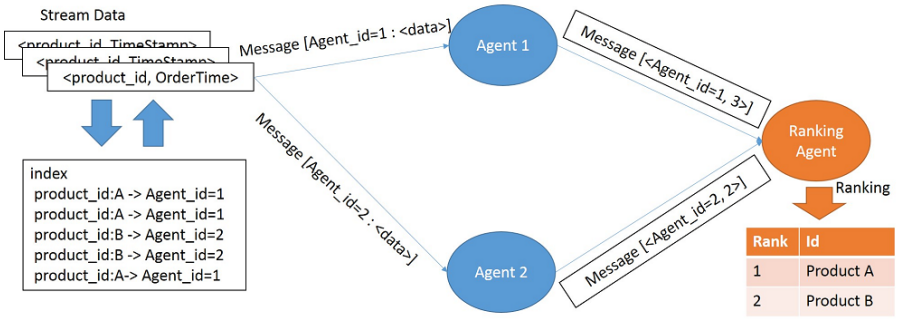

図 4: エージェントでのランキング集計

第3章　動的エージェント負荷分散機構

これまでに，RDA の特徴を有するトレーディングシステムやモバイルアプリケーションの基盤システムに対しエージェント適用が行われてきた [27]．上記のような複雑かつ性能が求められるシステムでは，エージェントの設計に熟練的技術が必要であり，またその運用もシステム開発者の経験に基づいて行われてきた．そのため，RDA を実現するエージェントシステムの設計問題や運用問題については十分に議論されいない．この章では，RDA システムへエージェントを適用する最初の一歩として，設計が容易な単純 RDA システムへのエージェント適用を目指す．単純 RDA システムは，集計目標が筆頭であることからエージェントの種類が単一である．これにより，エージェントの設計は要件定義により導出されたデータや機能をエージェントに組み込むだけでよい．また，エージェント間で通信が発生しないためエージェントを環境中に均一に配置することで分散配置問題を解決できる．よって，単純 RDA システムでは設計問題は存在しない．一方，環境中をエージェントが非同期に動作することから，システムの実行中に性能が劣化した場合，原因となるエージェントを特定することは困難であるため，エージェントシステムの運用問題が大きな課題となる．本章では，エージェントシステムの運用問題に注目し，その問題を明らかにした上で問題を解決する効果的な運用技術の開発を行う．

3.1　はじめに

RDA は，ストリームデータをリアルタイムに集計するという特徴から，運用時に大きく分けて 2 つの問題を解決する必要がある．

1) 並列性の問題，分散環境上に構築された分散システムはシステムに到着するデータの負荷が偏るとスケーラビリティ性能が劣化し処理速度が遅くなる [22]．MapReduce を利用した Hadoop や RDDs による Spark では，データや処理の偏りが発生しない仕組みが提供されている [17]．一方，エージェントシステムでも計算機に到着するデータ数は均一に分散されている．しかしながら，データを集計するエージェントはデータの集約条件 (地域，年齢など) に基づいてデータを収集するため，システム利用者の属性に偏りが存在する場合，エージェントに到着するデータにも偏りが発生する．例えば，都道府県データを集計するエージェントシステムの場合，人口の多い県を集約条件に持つエージェントへは当然データが集中する．

2) データアクセス性能の問題，蓄積データが肥大化するとデータ整合性 (一貫性) のチェックや永続性を維持するためのレプリケーションコストが大きくなりデータのアクセス性能が劣化する．エージェントシステムでは，環境上に存在するエージェント数が極端に少ないとき，エージェント内で保持するデータレコード数が多くなる．そのため，データ参照時の性能劣化やレコード数と同じ数のメッセージが同時にエージェントに送信されることで処理速度が大幅に低下する．

これらの問題を解消するには，非同期に動作している多数のエージェントから，問題が発生している特定のエージェントを環境から探し，それぞれの問題に対応した負荷分散を行う必要がある．本研究では，単純 RDA システムであるセンサーログマイニングシステムをエージェントにより実装し，エージェントシステムの運用問題を明らかにし，解決のための効果的な運用技術を開発する．開発する運用技術は，システム開発者が環境中のエージェントの動向や負荷を考慮しなくても，適切な負荷の解消を行う技術となる．また，その運用技術適用による効果については実験により評価する．

3.2 単純 RDA システム

単純 RDA システムの例としてユーザーのモバイルアプリケーションから発生する GPS 等のセンサーログを分類し，集計するセンサーログマイニングシステムをエージェントにより実装する．

3.2.1 センサーログマイニングシステム概要

センサーログマイニングシステムは，ユーザーが使用するセンサー (スマートフォンの GPS センサやウェアラブル端末のセンサ) の毎秒のサービスアクセス数を利用ユーザの年齢ごとに集計するシステムである．システムの動作を図 5 に示す．システムに，入力されるデータがストリームデータでなく蓄積データの場合，図 5 は MapReduce の典型的なベンチマークであるログマイニングとなる．図 6 にログマイニングシステムの MapReduce による実装結果を載せる．ユーザーの年齢ごとに集計することで，アプリケーションの利用分析や利用が活発な年代のアクティブユーザーへリアルタイムにプロモーション活動を実施することができる．このシステムは，継続的に発生するセンサーログ (ストリームデータ) をリアルタイムに集計する RDA システムである．また，データの発生頻度やユーザー数の関係から膨大な量のデータが発生するためデータの蓄積は困難である．例えば，1 万人のユーザーが平均して 10 サービスを利用している場合，秒間 10 万件のデータが発生すると考えられる．

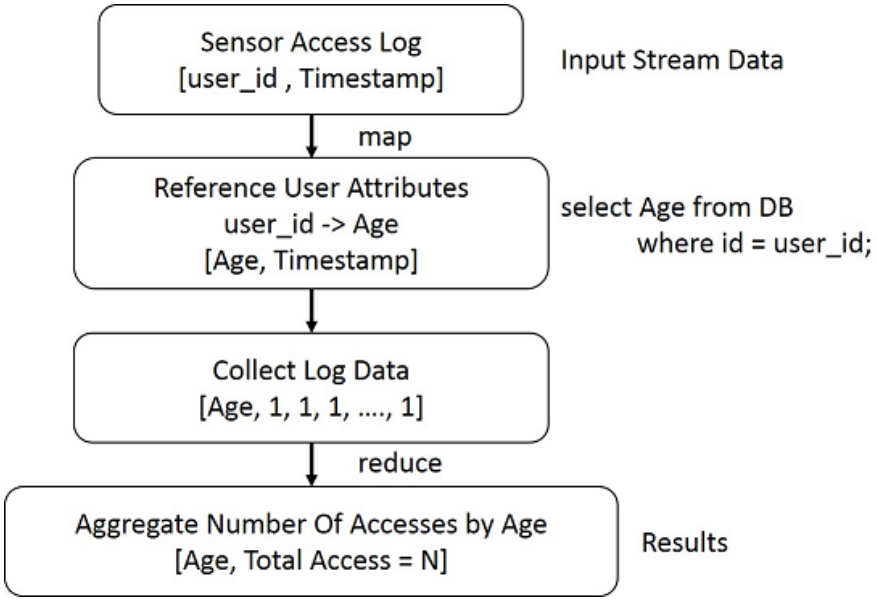

図 5: センサーログマイニングシステム

3.2.2　集計データ

センサーログマイニングシステムの入力データや集計データは次となる．

- 入力データ：　モバイルアプリケーションのログデータ　－　ユーザー ID とアクセス時のタイムスタンプからなる．
- 集計データ：　年齢別のアクセス回数　－　ユーザーの年齢ごとに発生したログデータをカウントした数．

3.3　エージェントシステムの設計

単純 RDA システムのエージェント適用は，集計目標がひとつ (年齢ごとのカウント) であるためエージェントタイプは 1 つのみとなる．なお，エージェントタイプとはエージェントに具体的なデータが挿入されていないエージェントの雛形のことである．具体的なデータを挿入することで動作するエージェントを導出できる．エージェントタイプが 1 つのとき，エージェント間での通信は発生しない．また，要件定義の処理やデータが全て単一のエージェントタイプに割り当てられ，システムシナリオ (システムの動きが記述された文章) から直接エージェントを導出できる．よって，エージェントシステムの設計が容易である．

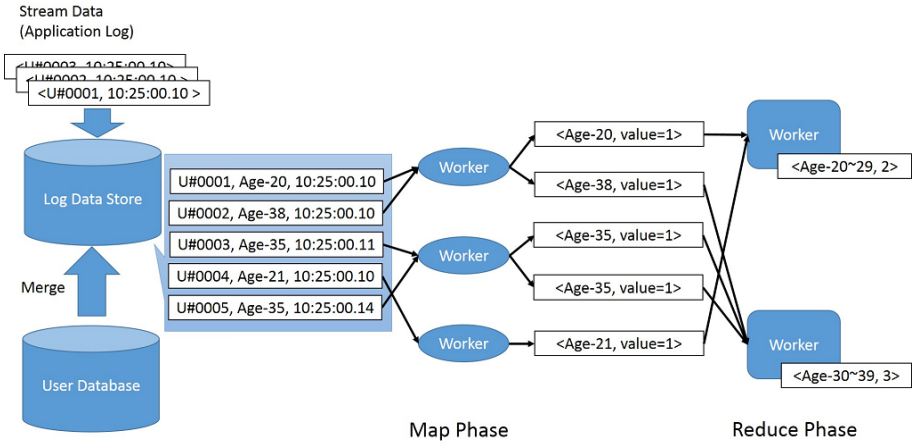

図 6: MapReduce によるログマイニングシステムの実装

3.3.1 エージェント設計

エージェントの抽出を行うには，対象となるシステム (センサーログマイニングシステム) に存在する役割を明確化する必要がある．この役割とは，システム内で実際にデータを扱う主体となる．単純 RDA システムであるセンサーログマイニングシステムでは，シナリオ中のデータを扱う処理が主体となるため，シナリオのデータと処理部分を抽出し，エージェントに割り当てればよい．また，集計項目はユーザー年齢のみであるため，エージェントタイプは 1 つとなる．図 5 からセンサーログマイニングシステムのシナリオは以下のように導出される．

- アプリケーションから発生したログデータをユーザー ID と紐付け，データに年齢を付加する．
- ユーザー年齢ごとにログデータを分類しまとめる．
- まとめたログデータをカウントする．
- ユーザー年齢ごとの集計結果を出力する．

シナリオ中に存在する処理は，ユーザー年齢の付加とログデータをカウントする処理の 2 つである．データの年齢ごとの分類はエージェントとの通信で自然に行われる．データレコードは，シナリオ中のデータの記述から作成することができ，図 7 のデータ形式となる．レコードの先頭列のエージェント ID はエージェントを識別する ID である．なお，後述する運用技

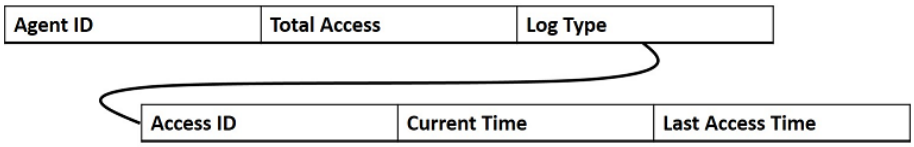

図 7: センサーログマイニングシステムのデータレコード

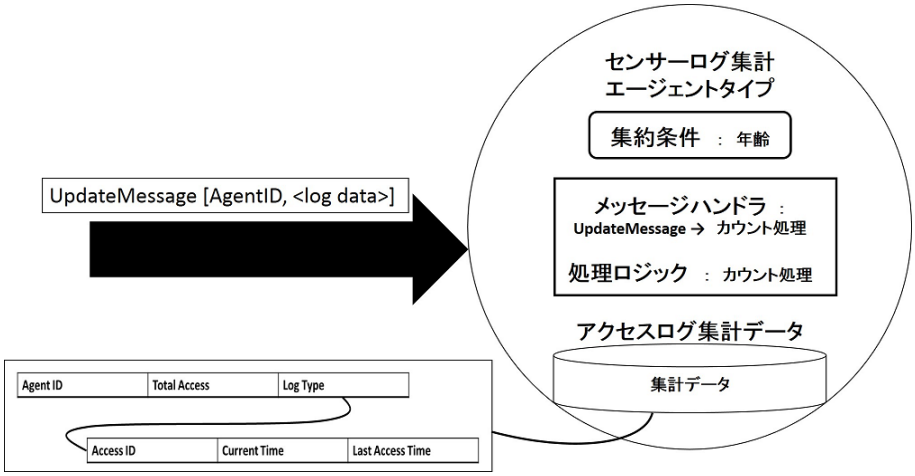

図 8: センサーログマイニングシステムのエージェントタイプ

術のためエージェント内のデータレコードにはタイムスタンプが付加される．次にエージェントのメッセージを受信するルール (集約条件) をシナリオから抽出する．集計項目からユーザー年齢が集約条件となる．最後に，設計した処理とデータレコード，集約条件をまとめエージェントタイプを導出する (図 8)．システム中にエージェントに送信されるメッセージの設計は，メッセージハンドラの設計により自動で設計される．メッセージハンドラは，シナリオ中のデータの更新処理から設計でき，このシステムではデータのカウント処理を呼び出すメッセージハンドラが設計される．このとき，同時にメッセージハンドラに対応するメッセージが設計される．なお，データ付加等のエージェント内で閉じる処理 (エージェントが持つデータのみで完了する処理) は，直近のメッセージハンドラにまとめられる．今回の場合では，ログデータカウントの処理をメッセージハンドラが呼び出すさいに，年齢の付加処理も一緒に呼び出される．また，エージェント生成時の初期化メッセージや削除時の削除メッセージ，データやエージェントに問い合わせるさおの読み取りメッセージなどはシナリオから導出されないが，エージェントシステムに標準機能として備わっている．

3.4 エージェントシステムの実装

エージェントフレームワークによりセンサーログマイニングを行うエージェントシステムを実装する．エージェントは，エージェント設計により導出されたデータレコード，メッセージ，メッセージハンドラをエージェント定義に記述することで生成できる．また，エージェントの具体的な処理 (メッセージハンドラにより呼び出される処理ロジック) は Java により記述する．

3.4.1 エージェント定義

設計されたエージェントを定義し，記述したものが次の XML である．エージェント定義では，保持するデータレコード，メッセージ，メッセージハンドラが記述される．

```
\footpage
<?xml version="1.0"?>
<agentdef package="rda" version="rda1.0">
  <entities>
    <!--Aggregate Agent Entity define-->
    <entity type="aggregate agent">
      <attribute name="AgentID" type="string"
                        primarykey="true" maxlength="32"/>
      <attribute name="Data" type="long"/>
      <attribute name="Log" type="log... "/>
    </entity>
    <entity type="log">
      <attribute name="AgentID" type="string"
                primarykey="true" relationto="AgentID" maxlength="32"/>
      <attribute name="AccessID" type="string"
                primarykey="true" maxlength... ="16"/>
    </entity>
  </entities>
  <messages>
    <!--message define-->
    <!--Agent Messages -->
    <message type="initAgent" class="rda.agent.message.InitMessage"/>
    <message type="readAgent" />
    <message type="updateAgent" class="rda.agent.message.UpdateMessage"/>
    <message type="deleteAgent" />
  </messages>
  <agents>
    <!--Message handler define-->
    <agent type="aggregate agent">
      <handler message="initAgent" class="rda.agent.handler.InitHandler"/>
```

```
      <handler message="readAgent" class="rda.agent.handler.ReadHandler"/>
      <handler message="updateAgent"
                class="rda.agent.handler.UpdateHandler"/>
      <handler message="deleteAgent"
                class="rda.agent.handler.DeleteHandler"/>
    </agent>
  </agents>
</agentdef>
```

3.4.2　データレコード

設計されたエージェントのデータレコード (図 7) を定義したものが，エージェント定義の entity である．AggregateAgent のエンティティのように，レコード内に他のエンティティをセットすることができる．これにより，単一のレコードに複数のレコードをひもづけることができる．ひもづけられたレコードは，はじめにエージェントのエンティティ(AggregateAgent) を呼び出し，関係データベースのように外部キーを指定することで，プログラム (処理ロジック) から読み込むことが可能.

3.4.3　メッセージ

エージェント定義の messages がシステム内でエージェントに送られるメッセージである．センサーログマイニングシステムで用いられるメッセージは，次の 4 つである..

- 初期化メッセージ　エージェントが生成時に送られるメッセージでエージェントのデータや状態を初期化する.
- 削除メッセージ　エージェントが削除されるときに送られるメッセージ.
- 更新メッセージ　エージェントが保持するデータレコードを更新するさいに送られるメッセージ.
- 読み取りメッセージ エージェントのデータを読むときに送られるメッセージ.

さらに，これらのメッセージは非同期メッセージ，同期メッセージとして送ることができ，エージェントからでのデータを取り出すさいは同期メッセージで送ることで受け取ることができる.

3.4.4 メッセージハンドラ

エージェント定義の agents がメッセージハンドラを定義している．メッセージと対応するハンドラ (呼び出される処理) からなる．なおメッセージハンドラは，データの整合性維持のため同時に複数のメッセージを処理することはできない．センサーログマイニングシステムでは UpdateMessage にカウントするデータを付加し送信することで，UpdateHandler がウント処理を行う処理ロジックが呼び出す．

3.4.5 処理ロジック

メッセージハンドラにより呼び出される処理．Java により記述され，エージェントが保持するデータレコードの操作を実行する．以下は，エージェントのデータを更新するさいの処理である．実際に使用されている処理ロジックについては付録を参照．

Algorithm 1 UpdateAgent

```
class UpdateHandler
        method onMessage(msg)
                //Update AgentData
                entity ⇐ Agent.getEntity()
                updateData ⇐ entity.getData() + msg.getData()
                entity.setData(updateData)

                //Insert Agent AccessLog
                logEntity ⇐ entity.createLog(msg.getTimeStamp())
```

3.5 エージェントシステムの運用

エージェントシステムは，エージェントの並列性と高いデータアクセス性能により高速なデータ処理を実現する．そのため，システム運用では並列性の維持とデータの肥大化の解消が重要となる．RDA の負荷はアプリケーションの利用環境に依存する．これは，アプリケーションを利用するユーザーの増加やサービスの拡張，通信・システムの負荷状況によりシステムへの時間あたりに到着するデータ量が増加・減少しするためである．センサーログマイニングシステムでは，センサーがアクセスするサービス数が 1 増加すると利用者の数だけ秒間のデータ発生数が増加する．また，リアルタイムにデータを集計するためシステム状況や通信状況の変化により，ある時間に発生するデータの到着に遅れが生じる場合がある．到着が遅れたデータは，次の時間に発生したデータと一緒に処理しなければならない．さらに，エー

ジェントはそれぞれ固有の集約条件を持つため，この条件の偏りにより負荷が偏る．例えば，20 代の利用者数が多いアプリケーションのログデータを集計した場合，20 代データを集約するエージェントに対して通信が集中することが予測される．

これにより，単にエージェントシステムを実装しただけではアプリケーションによるデータ負荷の変化に対応できずシステム性能が劣化し，RDA の性能要件を満たせなくなる．しかしながら，RDA を行うエージェントシステムは Hadoop や Spark といった他のデータ処理基盤の負荷分散手法が適用できない．これは，他のデータ処理基盤では処理を行うワーカー (ノード) とデータに関連性が存在しないのに対し，エージェントシステムではデータの ID から一意に処理を行うエージェントが決まる．つまり，Hadoop や Spark では同一 ID のデータであっても常に同じノードが処理するとは限らないのに対し，エージェントシステムは同一 ID を持つデータは必ず同じエージェントによって処理される．

そこで，上記の問題を従来エージェント技術に用いられた負荷分散手法を適用することで解決する．本節ではエージェントシステムに対してアプリケーションのデータ負荷に柔軟に対応する運用技術である動的エージェント負荷分散機構を開発する．

3.5.1　動的エージェント負荷分散

エージェントシステムでは，エージェントそれぞれに個別に負荷が発生する．ここでの負荷とはエージェントの処理が追いつかずに滞留したメッセージを指す．エージェントシステムの負荷分散は，エージェントへ個別に適用していく必要がある．また，エージェントは非同期であることからエージェントの外部に監視システムを置いて負荷分散することは難しい．これは，監視システムがエージェントにアクセスするさいに同期化してしまうためである．動的エージェント負荷分散は，エージェントが自身の負荷を測定し，自身に対して動的に負荷分散を適用する手法である．

上記の負荷分散手法をシステムに組み込んだものが動的エージェント負荷分散機構である．動的エージェント負荷分散機構は，次に説明する負荷検出とエージェント複製の 2 つの機能により実現される (図 9)．この機構により，アプリケーションの環境変化 (利用者状況の変化や通信・システムの状態変化に伴うによるデータ量の増減) に対して適応的に動作するエージェントシステムを運用可能となる．

負荷検出

エージェント負荷は，エージェントに備え付けられたメッセージキューから測定することができる．全てのエージェントは，送られてきたメッセージを一度メッセージキューに格納

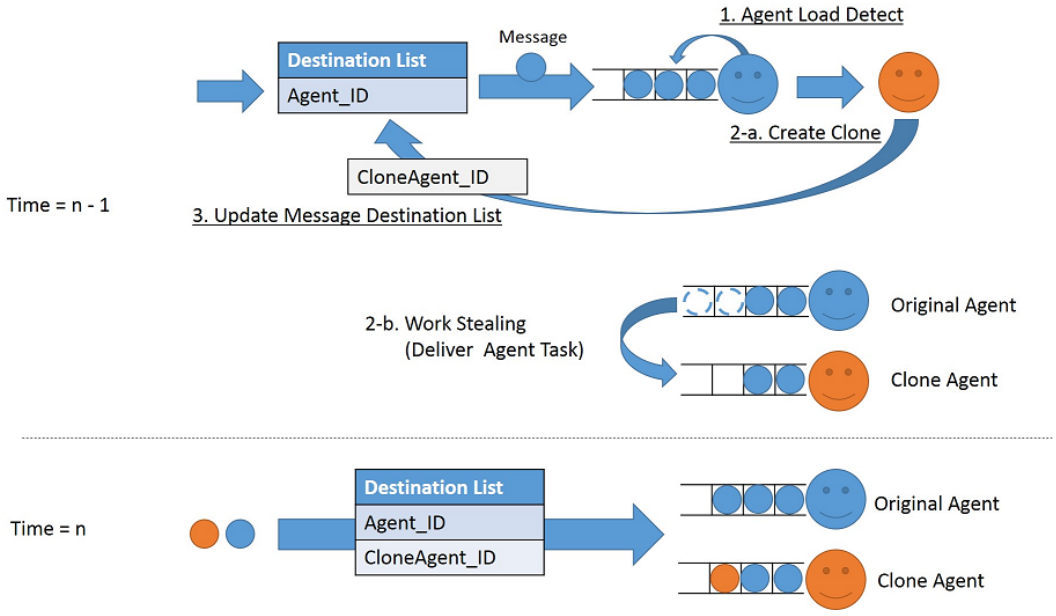

図 9: 動的エージェント負荷分散機構による負荷分散手順

し，キューからメッセージを取り出して処理を行う．メッセージキューは，エージェントが忙しいときメッセージを保持する．これにより，メッセージキューの長さからエージェントの負荷を測定することが可能となる．メッセージキューの長さから高負荷状態を検出するには，高負荷の判定するキューのしきい値を開発者が任意に設定する必要がある．メッセージキューのキュー長がしきい値より長くなったとき，エージェントは自身が負荷の高い状態であると検出する (図 9-1)．例えば，1 メッセージあたりの処理時間が 1ms で，すべての処理を 1 秒以内に行わなければならないといった性能要件が存在する場合，キュー長を 1000 とすることで，現在の負荷が 1 秒以内に処理が可能な負荷であるか判断できる，キュー長を超えたメッセージが滞留する場合，キューすべてのメッセージを処理しても 1 秒以上かかり性能要件が満たせないことがわかる．

エージェント複製

負荷検出後，高負荷のエージェントに対して複製による負荷分散を行う．エージェントの複製を利用した負荷分散手法に分散エージェントや知的エージェントで用いられてきたエージェントクローニング [21] や分解と合成 [16] と呼ばれる手法がある．この手法は，複製によるエージェントタスクの並列化や通信の分散，分解による肥大化したルール (状態) の解消，

合成によるルールの同期が行える．これにより，分散エージェントではエージェントの計算速度や応答速度の向上，知的エージェントではプランニング時の探索速度や分解された複数エージェントでの協調探索により探索性能の向上効果があることが報告されている．本システムでも並列性の維持やデータの肥大化の解消を目的としてエージェント技術の負荷分散手法を適用する．

RDA を行うエージェントシステムでは，他のエージェント技術と異なりエージェントはデータベースの機能を有している．そのため，エージェントクローニングや分解・合成を適用時にはデータの整合性を考慮する必要がある．負荷分散手法を適用した際に発生するデータの整合性の問題点として，複製や分解・合成時のデータの重複が考えられる．この問題を解決しなければ負荷分散手法を適用することはできないため，次の方法により解決を行う．

- タイムスタンプと同期ロジックによるデータ重複排除
  データ重複の問題は，エージェントの複製により複数のエージェント間で同一 ID のデータが更新されることで，データ整合性が保たれなくなる問題である．この問題により，処理結果の問い合わせ時に問い合わせ側が最新のデータが判別できなくなったり，誤った処理結果を受け取ることとなる．本システムでは，この問題の解決方法としてエージェント内で扱うデータに対しタイムスタンプを付加する方法 [12] と，データ問い合わせ側に同期用のロジックを組み込むことで解決した (図 10)．

上記手法を用い，エージェント技術の負荷分散手法を適用する．RDA を行うエージェントシステムにクローニングや分割といった負荷分散手法を適用すると次のようになる．

負荷が検出されたオリジナルエージェントは，自身と同じ集約条件を持ったクローンエージェントを複製する (図 9-2-a)．複製されたエージェントは，元のエージェントと同一集約条件をもつため，オリジナルエージェントとクローン間でメッセージが均等に分散される．このとき，キューに滞留したメッセージはオリジナルエージェントのメッセージキューからクローンエージェントのキューに半分渡される (図 9-2-b)．[10]．また，オリジナルエージェントは更新されなくなったデータを同期してから削除することでデータサイズを削減する．

次に，複製されたエージェントは自身の作成と同時にメッセージの宛て先リストを更新し，メッセージが受信できるようにする (図 9-3)．メッセージ宛て先リストは，表 3 のようにエージェント ID と宛先リストからなる．メッセージ送信者 (エージェントにメッセージを送るアプリケーションロジック) は，送信先のエージェント ID から宛て先リストを取得することにより，クローンエージェントの有無を確認できる．クローンエージェントが存在するとき，メッ

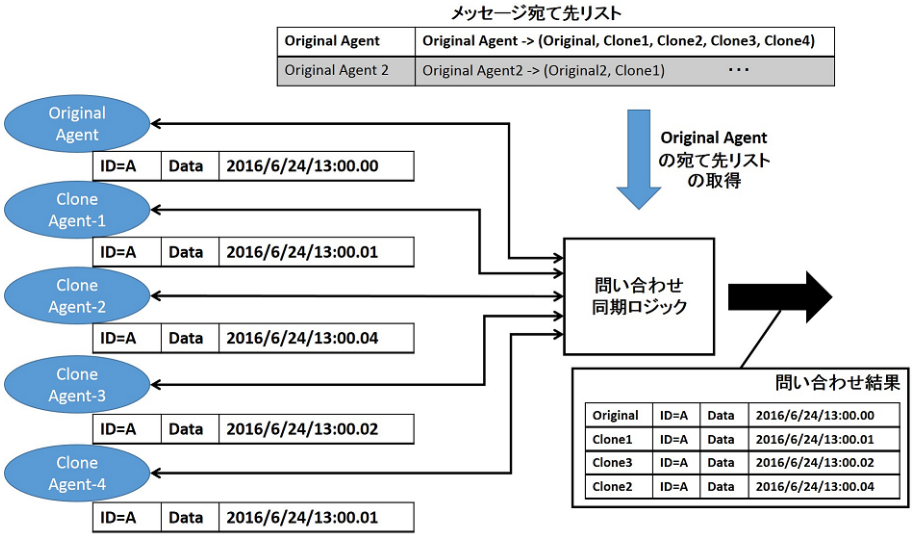

図 10: 問い合わせ時の同期ロジック

セージ送信者の ID から算出したハッシュ値を用い，宛先リストの内に存在するオリジナルエージェントかクローンエージェントのどちらかの ID を取得する．実際の操作を表 3 を利用し説明する．データ ID1 のデータを蒸したメッセージを Agent2 に送る場合，Agent2 をキーとして Agent2 と Agent2-1 のリストを取得する．次に，ID1 をハッシュ値に変換しリストの長さで割った余り (=1) をインデックスとしリストから Agent2-1 を取り出し，メッセージの宛先 ID に登録する．最後にメッセージを宛先 ID(Agent2-1) に送信する．この手順により，オリジナルエージェントとクローンエージェント間でメッセージの均等分散が可能となる．このようにメッセージを均等に分散できるクローンエージェントを増やすことで，局所的に Hadoop 等で用いられる負荷分散手法の適用が可能となる．

一方，エージェントが過剰に配置されている環境ではエージェントの切り替えによるコンテキストスイッチが原因で処理性能が劣化する．そのため，負荷減少時には作成されたクローンを消す必要がある．クローンエージェントの削除は，図 9-3 で更新したクローンエージェント ID を宛先リストから削除することで実現できる．削除されたクローンは，自身が保持するキューに格納されたデータを全て処理した段階で休眠する．休眠したエージェントのデータは，オリジナルエージェントの問い合わせ時に同期され，削除される．

表 3: エージェントのメッセージ宛て先リスト

Original AgentID	Destination List
AgentID = Agent_1	Agent_1 ->(Agent_1)
AgentID = Agent_2	Agent_2 ->(Agent_2, Agent_2-1 (Clone_1))
AgentID = Agent_3	Agent_3 ->(Agent_3, Agent_3-1 (Clone_1), Agent_3-2 (Clone_2))

3.6　性能評価

開発した動的エージェント負荷分散機構を RDA を行うエージェントシステムに組み込み評価する．評価方法として，ユーザー属性が均等な理想環境と属性が偏った環境で，動的エージェント負荷分散機構の有無によるシステムの処理性能と処理完了割合の推移を比較する．処理完了割合とは，エージェントが処理したデータ件数をその時間までに発生したデータの合計で割ったものである．つまり，性能要件を完全に満たしていた場合 100 %ととなる．

3.6.1　動的エージェント負荷分散の実験

エージェントを適用したセンサーログマイニングシステムの実験を行う．実験環境は表 4 の計算機で単一環境上での実験とした．単純 RDA システムでは，エージェント間の通信が存在しないためエージェント分散時の通信等による性能劣化が発生しない．つまり，計算機を並列に接続することで性能が線形に向上する環境であるため，システム全体の性能は接続された計算機で最も性能が低いものの性能と等しくなる．なお，エージェントの実行環境までのルーティングやパケットロス，帯域制限による性能劣化は考慮しない．実験システムの構成は，図 11 の構成で行った．エージェントシステムの設定としエージェント数を 10 とし，入力データの偏りの有無による動的エージェント負荷分散の有り無しによるスループットを比較した．なお，各パラメータはサーバーのリソースや，実験で最も性能の良いときのパラメータ値を設定し，結果は 100 回の実験結果の平均とした．また，性能要件はセンサーの更新間隔を 1 秒と想定し，キュー長は 1000 とした．図 12，13 はデータの偏りの有無による秒間 50 万件のデータを 100 秒間発生させたときの各エージェントの総入力データ数である．実験項目は，表 5 に従い条件を変えた 4 つの実験を行いその性能を比較した．

実験の結果，各条件での 100 秒間のスループットは図 14 となった．理想環境での実験 (a)

表 4: 実験環境

Resource	
CPU	AMD-FX8100 8Core 2.8GHz
Memory	12GB
OS	CentOS 6.3 64bit
Java	JDK 1.8.0 IBM Linux 64bit
JVM HeapSize	Xmx = 4096MB × 2

表 5: 実験条件

	データの偏り ’なし’	データの偏り ’あり’
従来システム	(a)	(b)
負荷分散機構	(c)	(d)

は発生データの 99 %のデータを処理し，一方，偏りのある実験 (b) ではおよそ 60 %のデータしか処理できなかった．このことから，特定エージェントへのデータ偏りにより 40 %の性能劣化が発生したことがわかる．この，データの偏りは動的負荷分散機構により解決できる．実験 (c) では，データの偏りが存在しても理想環境下の実験 (a) とのスループットの差は-0.001 %とほとんど性能が変わらない結果となり動的エージェント負荷分散機構によりエージェントの複製により性能が改善されたことが確認できる．実験 (d) では動的エージェント負荷分散機構が理想環境でも動作することを確認した．

データの偏りの有無による処理完了割合の性能比較

データの偏りによる性能の変化を詳細に分析する．実験 (a) と (b) を比較すると 40 %近い性能差が存在する．そこで，単位時間あたりの処理完了割合を比較し，その問題を検証する．比較の結果，データの偏りがある実験 (b) は非常に速い初期の段階でシステムの性能限界で

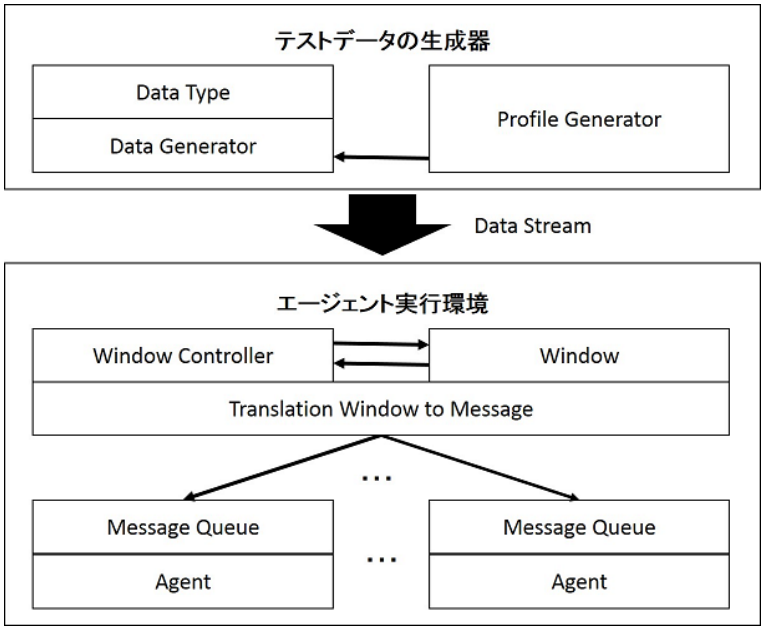

図 11: 実験システムの構成

ある 60 %に達しそれ以上のデータを処理できなくなっていた (図 15).

一方，エージェント負荷分散機構を組み込んだ実験 (c)(d) は，処理完了割合の曲線が理想環境での実験 (a) と一致している．このことから，性能要件 1 秒程度だとエージェント複製によるシステム性能への影響は確認できないほど小さいことがわかる．

なお，理想環境でも処理完了割合が 100 %になっていないのは，テストデータの生成に時間がかかっているためである．そのため，データ発生中に集計結果を問合せてしまうため 100 %とならない．

3.6.2　考察

4 つの実験から動的エージェント負荷分散は，システム実行中のデータの偏りによるエージェント負荷の解消に効果的であることが確認された．特にデータの偏りにより負荷が高くなったエージェントに対して，集中的に複製を作成することにより負荷を分散している (図 16)．複製数の最も多い AgentID=5 のエージェントの秒間のトランザクションとエージェント負荷 (キュー長) を負荷検出後から 30 秒間を確認すると，負荷が集中するとエージェントを複製し，メッセージを分散，データ処理の並列化を行っていることがわかる (図 18)．また複製を作成した直後は，オリジナルのエージェントのトランザクション数が非常に高い状態と

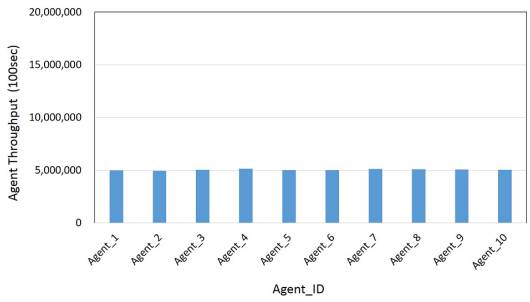

図 12: ユーザー属性に偏りがない場合のエージェントごとの総データ数

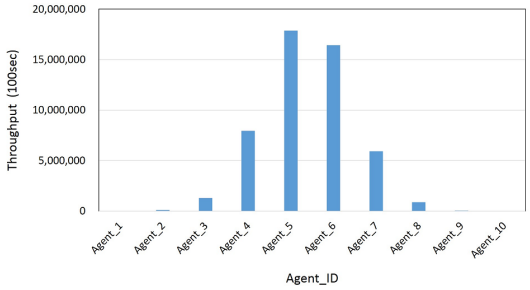

図 13: ユーザー属性に偏りがある場合のエージェントごとの総データ数

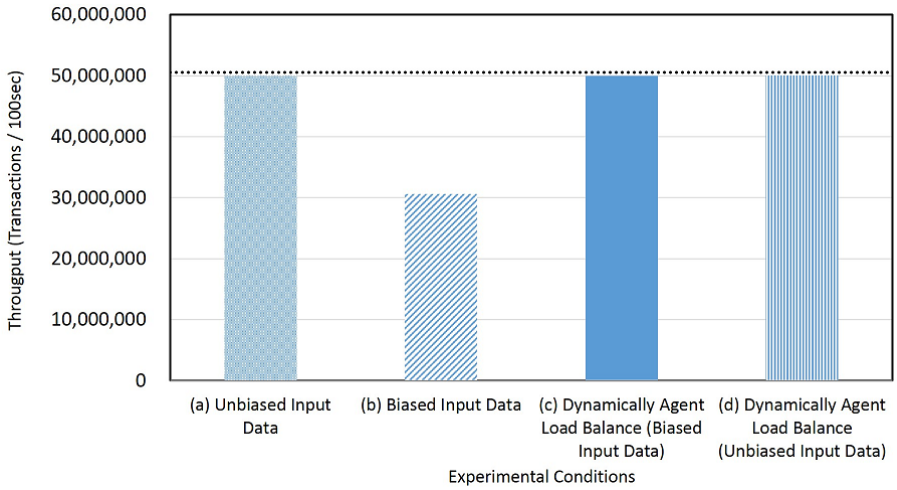

図 14: 100 秒間実行したときの全条件でのスループット

なっているが，時間が経つと同程度のトランザクション性能となっている．

次に，同じ構成でエージェントクローンの削除が行われるようにデータ発生間隔を変えたときのエージェント数の推移を確認する．図 17 をみると，負荷が高いときにエージェント数を増やし，負荷が落ち着くとエージェント数を減らしていることが確認できた．

一方，エージェント動的負荷分散では負荷検知後にエージェントの複製を行うことから，データ発生頻度よりも複製にかかる時間が大きいとき処理性能に影響する．本実験では 100 回の実験で，約 7000 回のエージェント複製が発生した．このときのエージェント複製にかかる時間は平均 4ms であった．よって，データの発生間隔が 4ms 以下のとき動的エージェント負荷分散は利用できない．

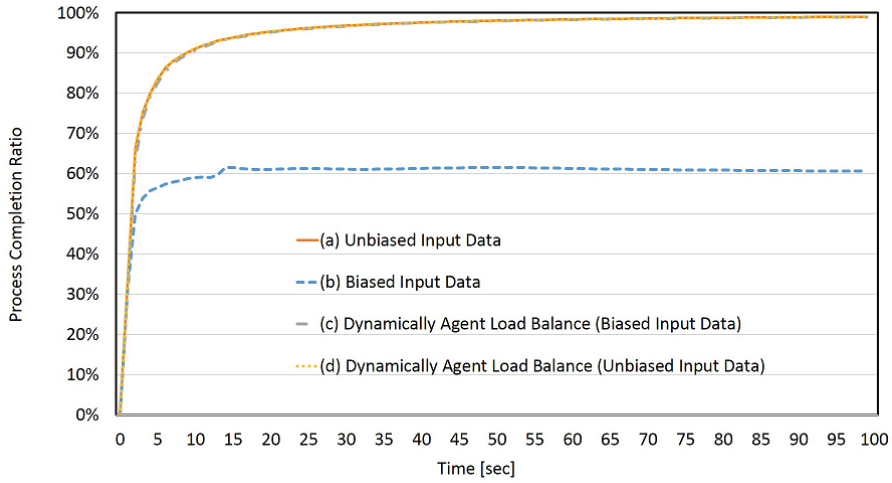

図 15: 実験 1：100 秒間実行したときの処理完了割合の推移

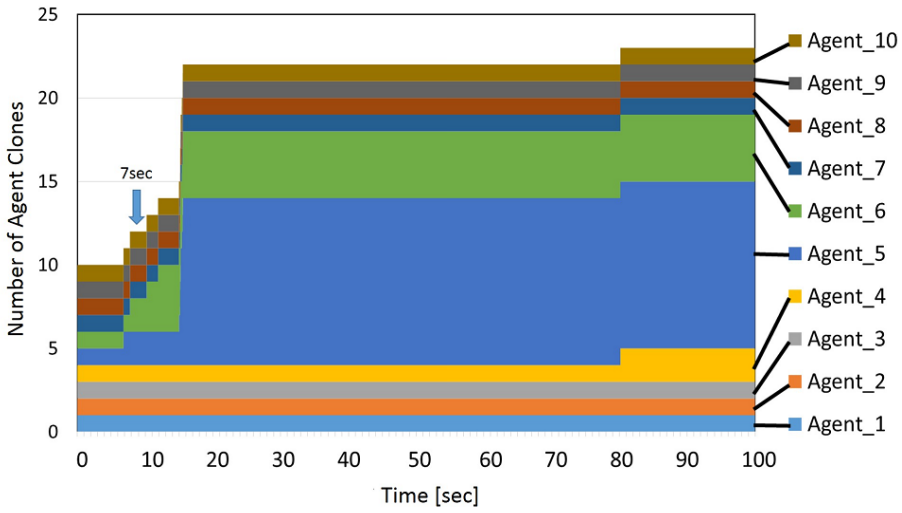

図 16: 動的エージェント負荷分散機構によるエージェント複製数の推移

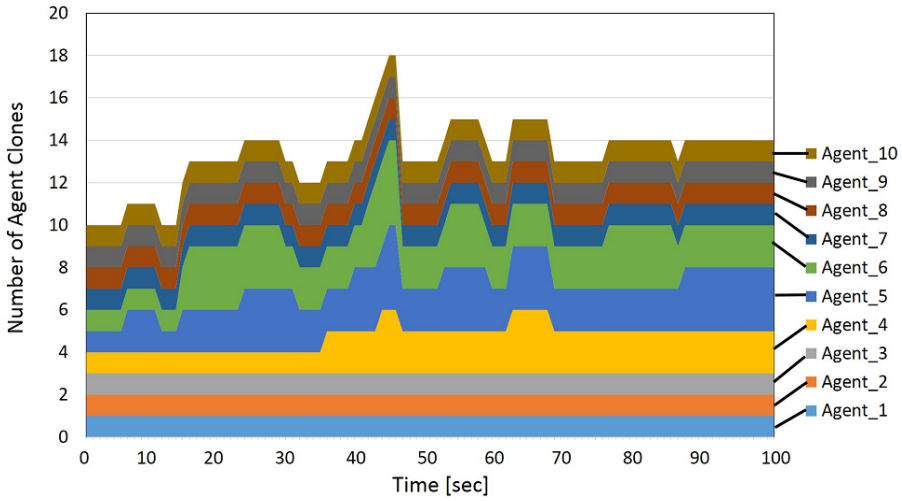

図 17: 動的エージェント負荷分散機構にクローンの削除機能を組み込んだときのエージェント複製数の推移

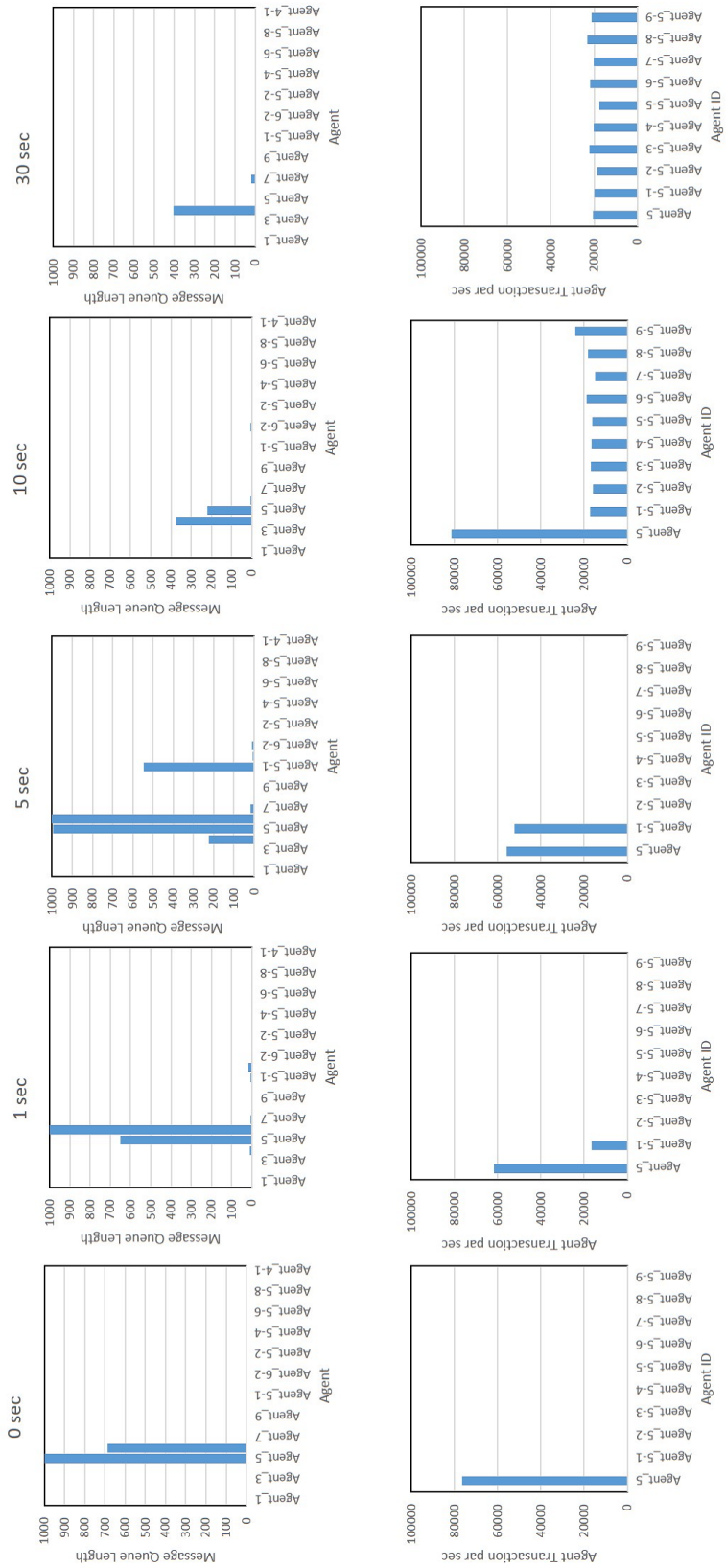

図 18: 動的エージェント負荷分散機構による負荷検出と負荷分散の時間的変化 ← 各エージェントのメッセージキューの長さ，→ エージェント ID=5 のオリジナルとクローンの秒間のトランザクション

3.7 まとめ

RDA を行うエージェントは，環境中を非同期に動作することから，システム実行中に性能が劣化した場合，原因を特定することが困難であった．本章では，上記の問題に対しエージェントが自身の負荷を動的に分散する動的エージェント負荷分散機構を開発することで解決した．動的エージェント負荷分散機構は，従来のエージェント技術で用いられてきた負荷分散手法を RDA 青行うエージェントシステムにも適用できるように変更したもので，エージェントの並列化による処理性能の向上や，データの肥大化解消によるアクセスの高速化を目的としている．本技術は，エージェント自身に負荷検出機能と複製機能を組み込むことで，エージェントが自身の負荷分散する．これにより，システム開発者が動作中の負荷を特定すること無くエージェントがアプリケーションから発生するデータに対して適応的に増えることで処理性能を維持していく仕組みとなっている．

第4章　複雑なRDAへのエージェント適用

トレーディングシステムやモバイルアプリケーション，通信の従量課金システムなどの一部は，発生するストリームデータをリアルタイムに集計するRDAシステムとなっている．これらのRDAシステムは複数の集計目標が存在する複雑なRDAシステムとなっていることが多い．本章では，集計目標が複数存在する複雑なRDAシステムへのエージェント技術の適用方法について示す．これまで，エージェントシステムはエージェントの設計に熟練的技術が必要とされてきた．これは，システム中から適切な役割を抽出しなければエージェントが導出できず，また，抽出されたエージェントは並列化可能な形となっていなければならないためである．また，エージェントが持つデータや通信を効率的に分散するように設計するには，分散配置問題とエージェント数を決定する運用問題を同時に解決しなければならない．前章でエージェントシステムに発生する負荷に合わせ適切なエージェント数とする運用技術を開発したため，運用問題については考慮する必要がなくなった．

4.1　はじめに

RDAシステムへのエージェント適用では，並列化可能なエージェントを設計し多数配置することで高い処理性能を実現する．パケットの従量課金システムへエージェントを適用した[29]は，分散環境でのエージェント分散配置問題の配置戦略について考察を行った．一方で，RDAを行うエージェントシステムの設計はシステム開発者の経験的設計に基づいたものとなっている．特に，集計目標が複数存在するRDAシステムは，複数のエージェントタイプが導出されるためエージェント間の通信を考慮して設計しなければならないため，単純RDAシステムと比較しエージェントの設計が困難となっている．複数のエージェントタイプが存在するシステムでは，システム中の役割の境界を見つけ適切にエージェントタイプを切り出す必要がある．また，分散配置問題についてもシステム要件と設計されたエージェントから配置方法を組み替えなければならない．本章では，経験的に行われてきた複雑RDAシステムのエージェント設計をエージェントの抽出を行う論理設計と，分散配置問題を解決する物理設計の2段階の設計手法により解決する．

4.2 複雑 RDA システム

複雑な RDA システムの例として，マラソンランナーの位置情報から走行距離を計算し，ユーザー属性 (地域や年齢) に基づいてリアルタイムにランキングするランキングシステムをエージェントにより実装する．上記のシステム要件から，ユーザー走行距離を集計しランキングするという段階的な集計が行われていることがわかる．なお，RDA システムの複雑さは，集計目標の数によって判断される．単一の集計目標の RDA システムは単純 RDA システムとなり，2 つ以上存在する場合は複雑 RDA システムとなる．段階的な集計を必要とする RDA システムは，単純な RDA システムと異なり，エージェントの設計が難しい．設計が困難な理由は主に二つある．

- システムシナリオからのエージェント抽出の問題
  単純な RDA システムでは，シナリオ中に存在するデータや処理は全て単一のエージェントタイプに割り当てられる．一方，複雑 RDA システムではシナリオ中に複数のエージェントタイプの処理，データが埋め込まれているためエージェントタイプ数に応じて適切にエージェントの機能を切り出す必要がある．山本 [28] によると，エージェントは業務的に意味のあるも，業務システム上の役割として定義されるが，これらを業務システム開発経験のない開発者がシナリオ中から見つけることは難しい．

- 分散環境へのエージェント配置の問題
  複数のエージェントが協調して動作するシステムをマルチエージェントシステムと呼ぶ．マルチエージェントシステムを分環境へ配置するには，エージェント間の通信頻度やエージェントのリソース量を考慮し配置しなければならない．RDA では，エージェント間の通信頻度と同時にエージェントへの集計結果の問い合わせやシステム実行中の負荷の変動を考慮しなければならないため，配置が非常に困難となっている．

単純 RDA システムでは，データの集計時にエージェント内のデータのみを参照し処理が実行されるが，ランキング集計では各エージェントが集計した結果を参照する必要がある．しかしながら，エージェントは他のエージェントのデータを直接読み書きすることはできない．そのため，エージェント間でデータを共有する仕組みが必要となる．最も単純なデータ共有方法は，単一のエージェントにデータを集める方法である．この方法はデータベースを保持したアプリケーションサーバーとほぼ同じ構成となってしまい，それ以上の拡張が不可能となってしまう．

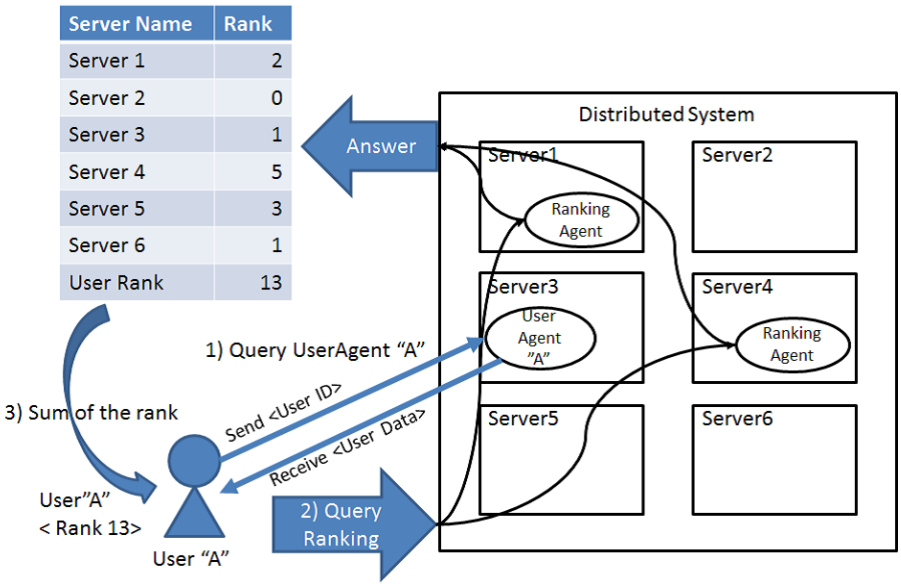

図 19: 問い合わせによるランキング集計

そこで，問い合わせにより各エージェントのデータを取得し集計する．図 19 は，問い合わせを利用したランキング集計法である．エージェントは，エージェント内で完了する処理と複数エージェントにまたがる処理で処理方法を変える必要がある．ここで，エージェント内で完了する処理とは，エージェント内に存在するデータ (メッセージにより受信したデータ，保持するデータレコード) にアクセスし実行される処理をさし，複数エージェントにまたがる処理とは，複数のエージェントに対してデータ参照が必要な処理をさす．なお，上記 2 つの処理をデータ構造で見た場合，前者はレコード形式のデータ構造を扱う処理に対し，後者はエージェント ID とデータからなる key-value 形式のデータ構造を扱う処理となる．そのため，前者の処理はデータアクセス範囲に制限がかかるが，データ処理に対する制限はない．一方，後者の処理は，データアクセス範囲に制限はないが，実行可能な処理は MapReduce 等で利用される分散アルゴリズムのみである．

ランキング処理は，計算コストが指数オーダであるためデータの増加により計算量が爆発的に増加する．Web のアクセス解析やユーザーの分析，EC や店舗での在庫管理システムなど多くの場面で利用されていることから，データ集計のための典型的な処理で性能要件の厳しい集計手法であることがわかる．よって，RDA システムでもランキング集計を実現することにより他のデータ集計処理も同様の方法で実現可能であると考える，

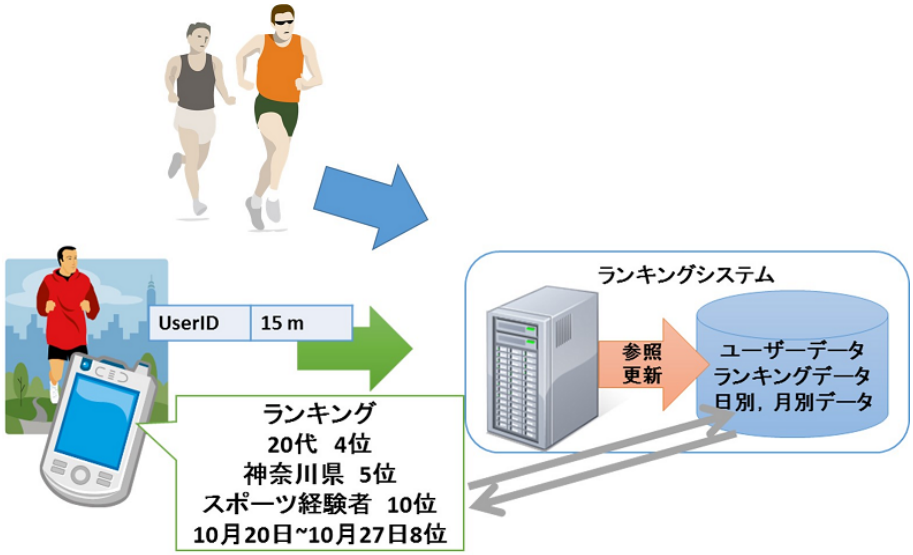

図 20: マラソンランナーのランキングシステムの概要

4.2.1 ランキングシステム概要

複雑な RDA システムであるマラソンのランキングシステムの概要を説明する (図 20)．マラソンのランキングシステムは，ランナーの持つスマートフォンから毎秒の走行距離を取得しランキングするシステムである．システムは取得したユーザーの走行距離とユーザー属性から年齢や地域といった複数のランキング表をリアルタイムに計算し，ユーザーは常に最新のランキング結果を問い合わせることができる．

4.2.2 集計データ

マラソンランナーのランキングシステムの入力データや集計データは次となる．

- 入力データ： アプリケーションから送信された走行データ ユーザー ID，秒間の走行距離，タイムスタンプ
- 集計データ 1： 各ユーザーの走行データ
  アプリケーション起動時からのユーザーの走行距離の合計．
- 集計データ 2: ユーザー属性ごとの各ユーザー走行距離
  年齢別，地域別のユーザーの走行データ．

4.3　エージェントシステムの設計

ランキングを行う RDA システムへエージェントを適用し，RDA の処理要件を満たしたエージェントシステムを設計する．はじめに，エージェントの種類を推定する．種類の推定にはデータキューブを用い，データキューブの軸の本数をエージェントの種類とする．次に，エージェントシステムの設計に，提案する 2 段階設計法 (エージェント設計，エージェント分散配置) にもとづき設計する [31]．複雑な RDA システムのエージェント適用では，システムシナリオ中から導出されるエージェントタイプを適切にシステム上の役割へ割り当てることで導出できる．また，複数の集計目標が存在することからエージェント間の通信が発生するためエージェント間通信を考慮した分散配置問題を解かなければならない [24],[25].

4.3.1　エージェントの種類の推定

エージェントの種類の推定にはデータキューブを用いる (図 21)．データキューブは，データを多次元的に管理するデータモデルで，OLAP(Online Analytical Processing) と呼ばれるリアルタイムでのデータ分析に用いられている．データキューブの特徴として，箱の中に存在するブロックの一つ一つは原子データと呼ばれる集計データの最小単位となっている．このブロックを組み換え，箱をスライスしたり組み合わせたりすることでデータの分析を可能とする．エージェントシステムでも，入力データを受け取るエージェントは，集計データの最小単位となっていなければならない．エージェントの並列性が最も高くなるようにエージェントの抽出が行われる．よって，データキューブの次元数はそのままエージェントの種類とできる．例えば，図 21 の底面をスライスしエージェントに置き換えた場合，図 22 のように直列に接続される．また，前面をスライスし，エージェントに置き換えた場合，図 23 となる．

今回実装するランキングシステムの場合，データキューブの前面をスライスしたものが集計目標となるため，エージェントの種類は 2 と推定できる．

4.3.2　エージェント設計

システム上の役割を明確化しエージェントを抽出するために，次の 4 ステップの手順によりエージェントを設計する (図 24).

要件定義

ランキングシステムの概要から，次のシナリオが導出される．シナリオは SVO(主語，動詞，目的語) により記述される．シナリオは，一般的なシステム開発と同様にシステム開発者とクライアント間で繰り返しヒアリングを行うことで導出できる．

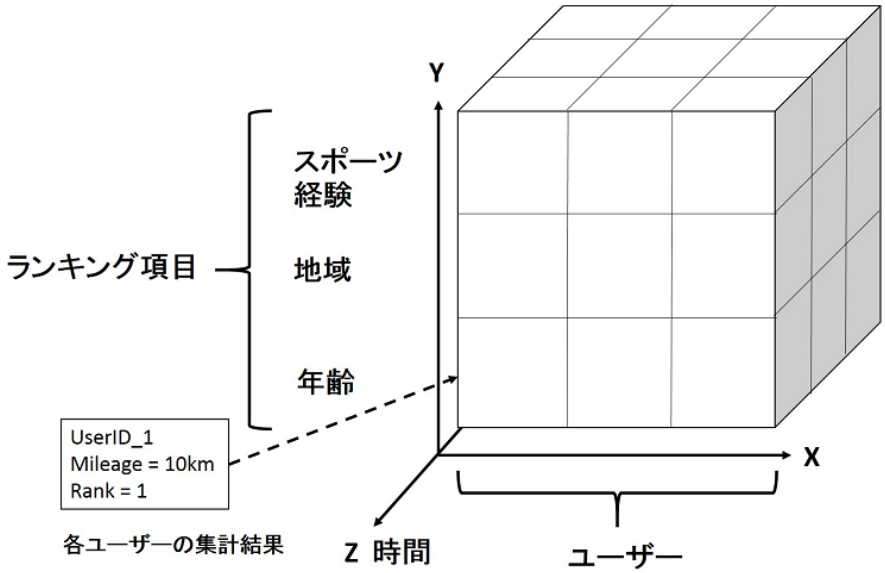

図 21: ランキングシステムのデータキューブ

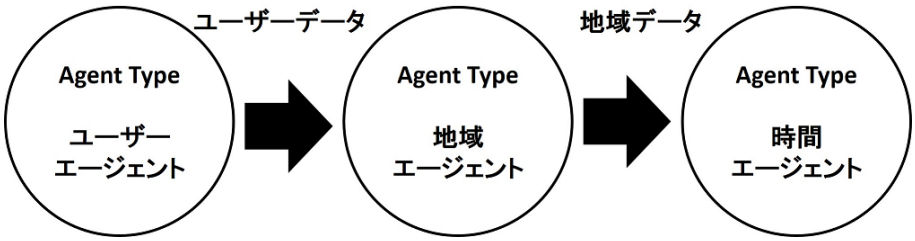

図 22: 直列に接続されたエージェントタイプ

- ユーザーはシステムに秒間の走行距離を送る.
- システムはユーザーの情報を参照する.
- システムは送られた位置データから距離を計算する.
- システムはユーザーの走行距離をランキングカテゴリ毎に集計する.
- ユーザーは，システムにランキング結果 (順位) を問い合わせる.

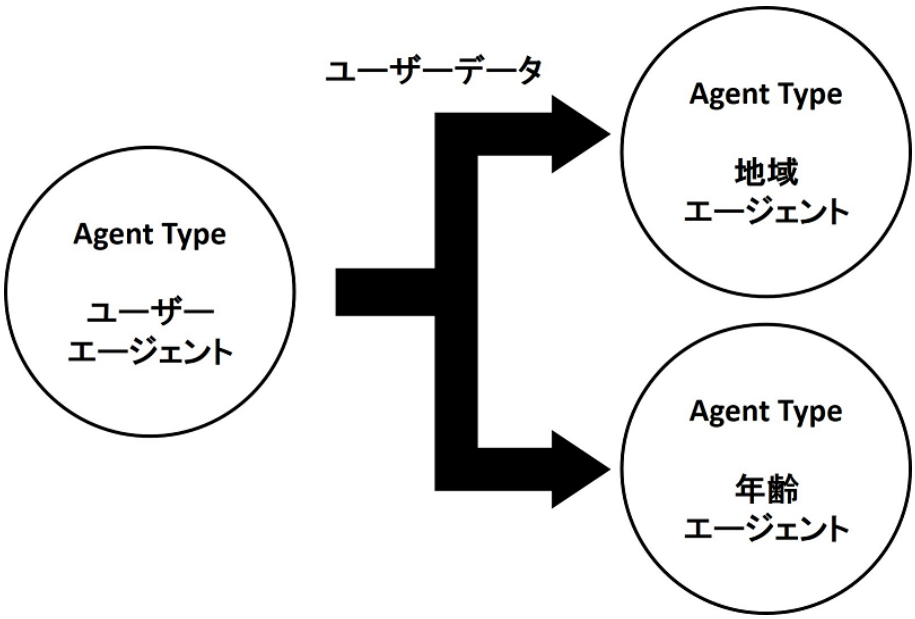

図 23: 並列に接続されたエージェントタイプ

データと処理の設計

データと処理の設計では，一般的なシステム設計で行われるプロセスフローとデータレコードを設計する．プロセスフロー設計は，各シナリオ中の動詞がプロセスとなり接続される (図 25 の矢印)．データレコードの設計では，ランキングシステムの集計データとシナリオの目的語から設計される (図 25 の上のレコード)．最後に，各ステップのプロセスをデータレコードと接続する．これにより，データレコードを基準としたプロセスの境界が明確し役割が抽出可能となる．

エージェントタイプ設計

エージェントタイプは，エージェントの種類の推定により 2 つできることが決定している．そのため，データと処理の設計から独立に管理されるデータを 2 つのエージェントタイプに割り当てることで設計できる．このとき，役割を表現する名前をエージェントタイプにつける (図 26)．

プロトタイピング

設計したエージェントタイプに，ランキングシステムに入力されるデータを割り当て実体化する (図 27)．具体的なデータを持ったエージェントはプログラミングすることで実装可能

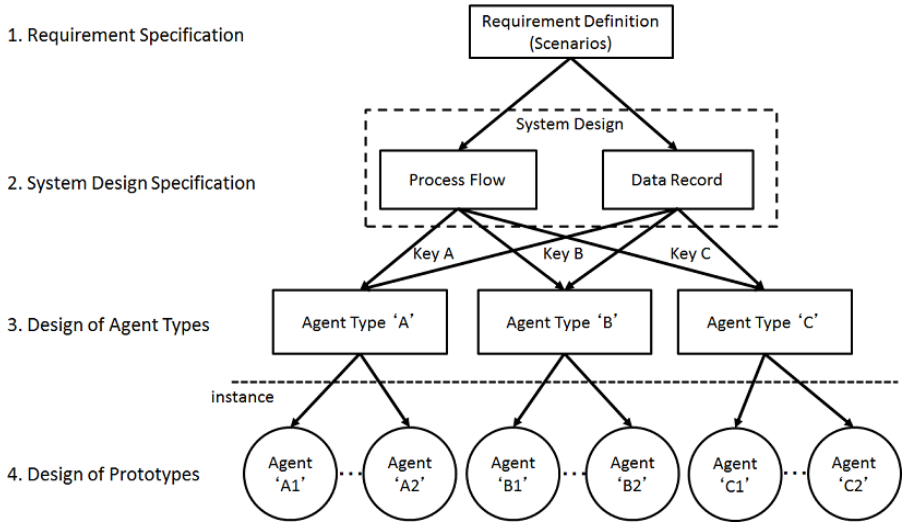

図 24: エージェント設計の 4 ステップ

である．このとき，入力されたデータに対してエージェントの処理やデータレコードに過不足がないか検証する．不足や余計なデータ・処理が存在する場合は，データと処理の設計を見直し再度エージェントタイプを設計することでエージェントシステムの完成度を上げていく．

4.3.3 エージェント分散配置

エージェントタイプが複数存在するシステムではエージェント間での通信が発生する．エージェント分散配置問題は，エージェント間の通信を考慮して複数のエージェントを分散環境に配置する問題である．設計の段階では，実行時のエージェントの通信量は推測できないためエージェントシステムで発生する 2 種の通信から 2 つの配置戦略を考案した [32](図 28).

エージェントタイプによる配置

エージェントタイプによる配置は，サーバーごとにエージェントタイプを分ける配置方法である (図 28 の上)．この配置方法は配置にかかるコストが低いため，再配置が容易となっている．また，エージェントが持つデータが全サーバーで重複しないため，問い合わせや同期にかかる処理コストが低い．しかしながら，エージェント間で発生する通信が必ずサーバ間をまたがる通信となるため内部の通信コストが最も高い配置となっている．

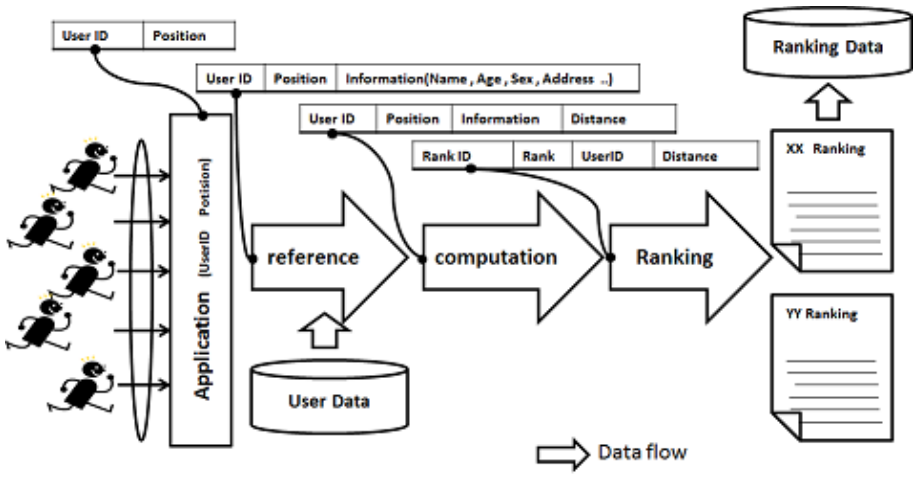

図 25: ステップ 2 : データと処理の設計

エージェントクローンによる配置

エージェントクローンによる配置では，全てのサーバに集約エージェント (エージェントから発生するメッセージを集約するエージェント) の複製を作成する配置方法である (図 28 の下)．この配置方法は，エージェント間の通信が必ず局所性を有しているため内部通信にかかる通信コストが発生しない．一方で，配置にかかるコストは高く，一度配置を行うと再配置時には全てのエージェントに影響する．また，エージェントの問い合わせや同期コストが高く，単一のエージェントの問い合わせであっても，そのエージェントのデータが他の環境にも存在するため問い合わせ用のメッセージを全環境に送る必要がある．配置されるエージェント数も多いため，サーバーリソースを最も使用する配置となっている．

4.4　エージェントシステムの実装

エージェント設計に基づいてエージェントを実装する．実装方法は，単純 RDA システムと同様でエージェントフレームワークのルールに則り，エージェント定義を記述し，エージェントの処理ロジックを Java でプログラミングする．なお，実際に設計された定義書やソースコードは付録を参照．

4.4.1　エージェント定義

エージェント設計により設計されたエージェント定義は付録を参照．また，エージェント定義の読み方や各機能についての記述は，3 章の単純な RDA システムの解説を参照．

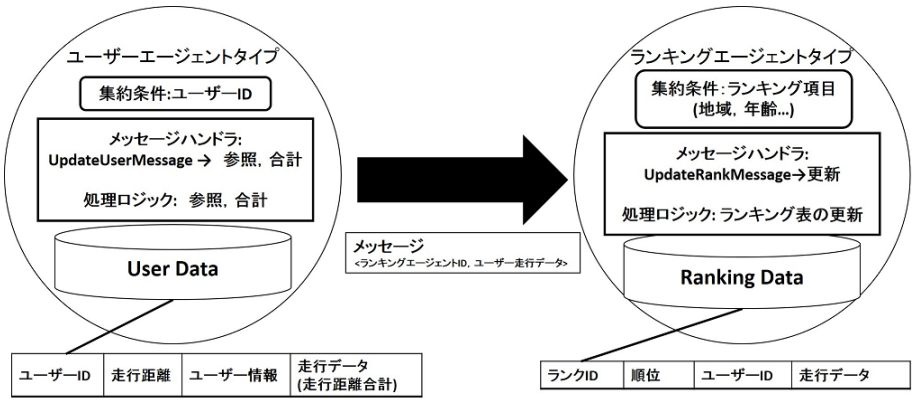

図 26: ステップ 3 : エージェントタイプ設計

4.5 エージェントシステムの運用

エージェントを適用した複雑な RDA システムの運用では，はじめにシステム要件に基づいてエージェントの配置戦略を選択する必要がある．エージェントの配置戦略は分散配置問題にあげた 2 つの配置戦略である．それぞれの配置戦略は，図 29 の関係にあり外部通信 (問い合わせ) と内部通信 (エージェント間の通信) のどちらを重視するかにより選択する．選択基準については，後述する性能評価により詳細に考察する．

4.6 性能評価

提案する設計手法により開発したランキングシステムの性能評価を行う．はじめに，スケラービリティ性能の確認として設計されたエージェントシステムがエージェントの増加により処理性能を向上するか実験する．次に，分散配置戦略の違いによるスケーラビリティ性能の変化を実験とシミュレーションにより評価し，同期による問い合わせコストによる配置戦略の選択方法について考察する．

4.6.1 エージェント分散配置の比較実験

エージェントシステムはエージェント数やサーバー数を増やすことで処理性能を向上するスケーラビリティの高いシステムである．本設計手法により構築したランキングシステムを，エージェントタイプによる配置とエージェントクローンによる配置の 2 配置戦略で配置したときの処理性能を比較する．比較結果を図 30 に示す．実験環境および構成は単純な RDA システムと同じ環境を 1 台から 6 台まで増やし比較実験した (表 4)．実験で使用したデータは，

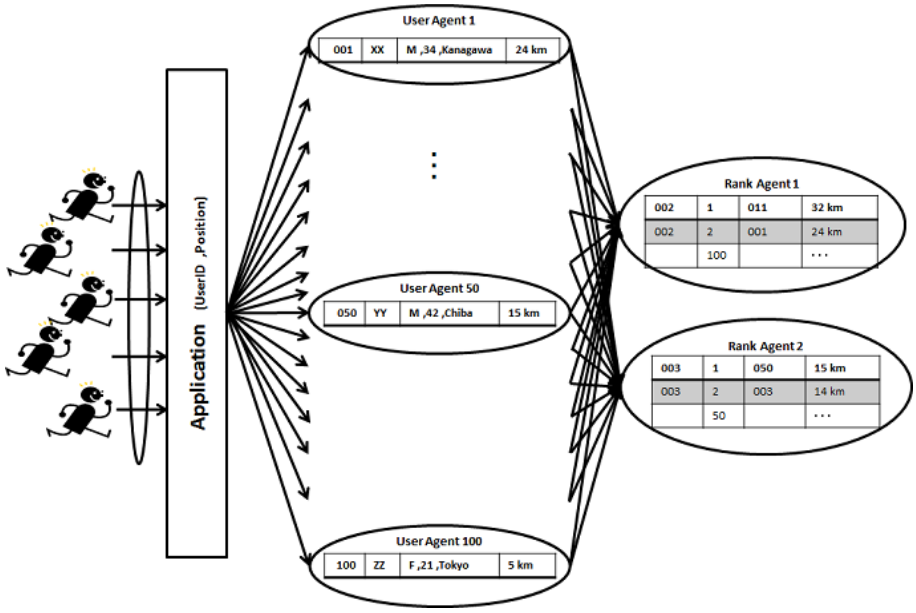

図 27: ステップ 4 : エージェントシステムのプロトタイプ

単純な RDA システムと同様に偏りが存在するデータを発生させ，秒間のトランザクション性能を測定した．単純な RDA システムと異なる点として，毎秒のデータ発生量は 2 倍の秒間 100 万件とし，初期エージェント数はユーザーエージェントが 1000 体，ランクエージェントが 10 体とした．

実験の結果，エージェントクローニングによる配置はサーバーの増加により性能がスケールに向上するのに対し，エージェントタイピングによる配置はサーバーの増加での性能向上幅は小さい．これは，サーバー間の通信がボトルネックとなっているためである．

拡大している部分は，旧システムと現システムの比較である．旧システムは，同様の設計を行っているがフロー制御等のないエージェントシステムの単純実装としている．

4.6.2　エージェント分散配置の比較シミュレーション

大規模分散時の 2 配置戦略における性能をシミュレーションにより比較する．大規模分散時の配置戦略は，サーバー間の通信コストが大きくなるため内部通信コストと外部通信コストのどちらに比重を置くかで性能が変化する．図 31 のように内部通信を考慮し配置を行った場合，エージェントクローニングが高い性能となる．一方，外部通信コストのみを考慮した場合，図 32 となる．次に，内部通信と外部通信の割合ごとの配置時のトランザクション性能を評価する (図 33)．なお，AgentCloning 100000:1 とはエージェントクローニングによる配

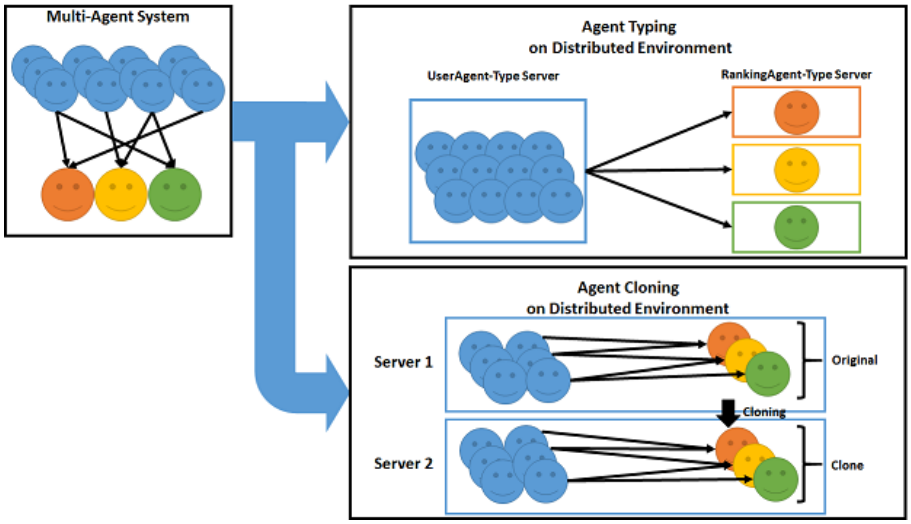

図 28: 2 つのエージェント分散配置戦略

置で 1ms に 1 回エージェントに問い合わせする場合の処理性能である．一方，AgentCloning 1:1 は 100000000 のユーザーが毎秒問い合わせすることを想定しているため，10ns に 1 回の問い合わせが発生していることになる．これらのことから，問い合わせ性能と稼働する計算機数から配置戦略を選択できる．

4.6.3 考察

複雑な RDA システムでは，エージェント間の通信が発生するため単純 RDA システムと異なり，エージェントの分散配置問題を考慮し運用する必要がある．しかしながら，システム実行前では問い合わせやエージェント間通信の頻度，データの偏りを予測することは不可能である．そのため分散配置問題を解決する 2 つの配置戦略は，内部通信を考慮するか外部通信を考慮するかの極端な配置方法となっている．よって，システム実行中に環境内のエージェント数の最適化と問い合わせの最適化を行っていく必要がある．これらは，エージェントの再配置の問題と呼ばれそれぞれの配置戦略ごとに方法が考えられる．

エージェントタイプによる配置

エージェントタイプによる配置時の再配置は，負荷の集中するエージェントの負荷分散と各エージェントタイプに割り当てられるサーバー数の変更を行うことである．エージェントタイプごとにサーバーが割り当てられるため，負荷の高いエージェントタイプに優先してサーバーを割り当てることで処理性能を向上できる．本実験では，エージェント負荷については

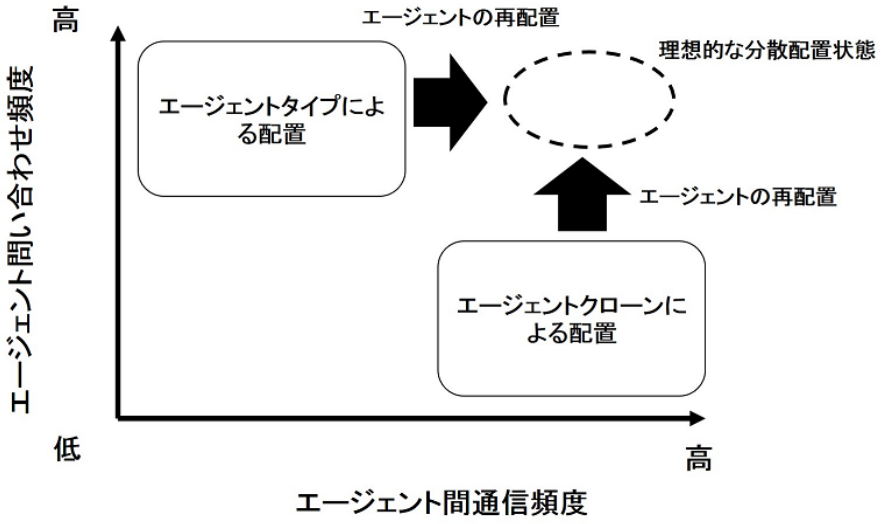

図 29: エージェント配置戦略と通信頻度の関係

動的エージェント負荷分散機構により動的にエージェントが増やされる．また，エージェントタイプに対するサーバー配分は，割り当て数の違いによる性能比較で最も性能が良い割り当てを行っている (図 34).

エージェントクローンによる配置

エージェントクローンによる配置時の再配置は，メッセージが送られないクローンを削除することである．必要のないクローンを削除することにより，問い合わせ時の同期回数を減らし性能を向上できる．本実験では動的エージェント負荷分散機構により余分なクローンの削除が行われているため再配置後の性能が向上が含まれた結果となっている．

4.7　まとめ

複雑な RDA システムであるランキングシステムに対しエージェントを適用する設計方法論を提案した．ランキング処理は，従来行われてきたデータ集計の典型的な処理となっているためランキングシステムの設計方法を確立することで，他の複雑な RDA システムへの適用も可能となる．エージェントシステムの設計方法論は，エージェントの導出を行うエージェント設計と分散環境上へのエージェントの配置を行うエージェント分散配置の 2 段階の設計により行う．エージェント設計は，要件定義からプロトタイピングまでの設計手順を繰り返し，実行することで，抽象的なエージェントの設計を具体化していく設計手法である．エー

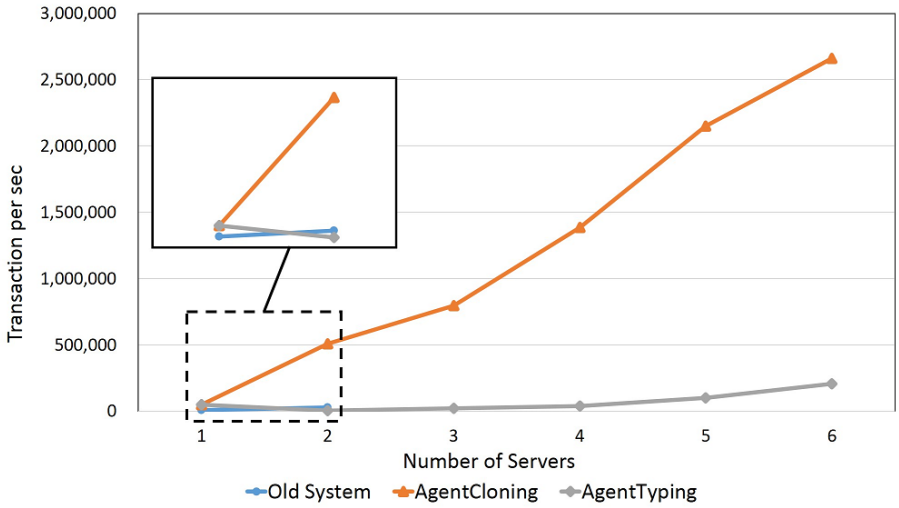

図 30: 2 段階設計法を適用したランキングシステムの 2 配置戦略での処理性能比較

ジェント分散配置は，内外の通信頻度を考慮し，エージェントタイプによる配置とエージェントクローンによる配置の 2 つの配置戦略を提案した．本章では，2 つの配置戦略がエージェントシステムの運用上重要な問題であることを実験とシミュレーションにより示し，その選択方法について考察した．結果，ほとんどの場合エージェントクローニングによる配置が有効であることを示した．一方，エージェントタイピングによる配置は，サーバー数が多く問い合わせ頻度が非常に高いときに効果的な配置方法であることを示した．

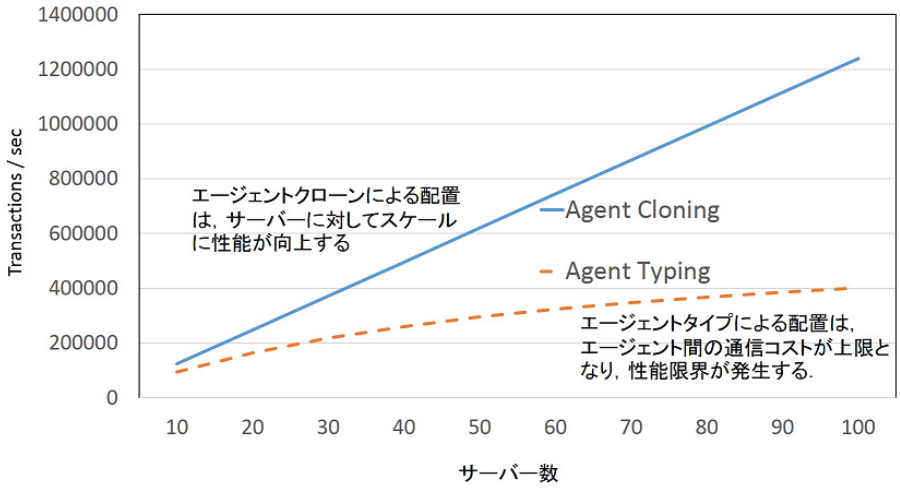

図 31: 内部通信のみを考慮したときのエージェント分散配置の性能比較

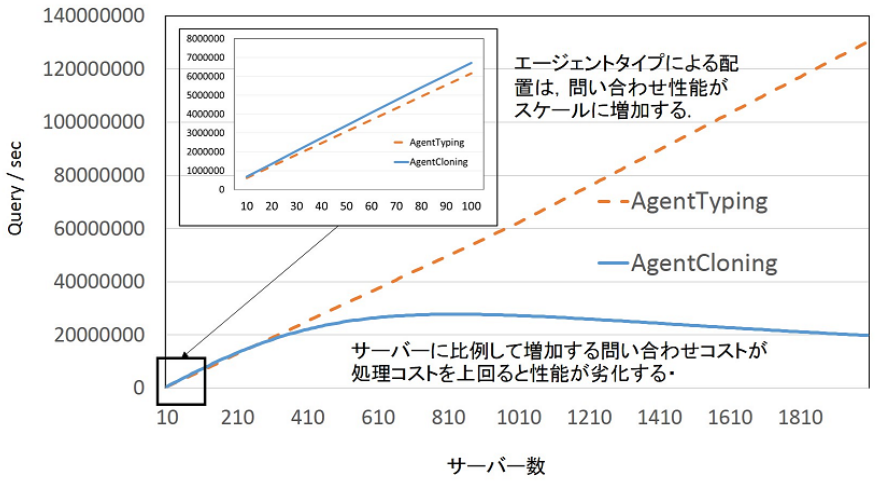

図 32: 外部通信のみを考慮したときのエージェント分散配置の性能比較

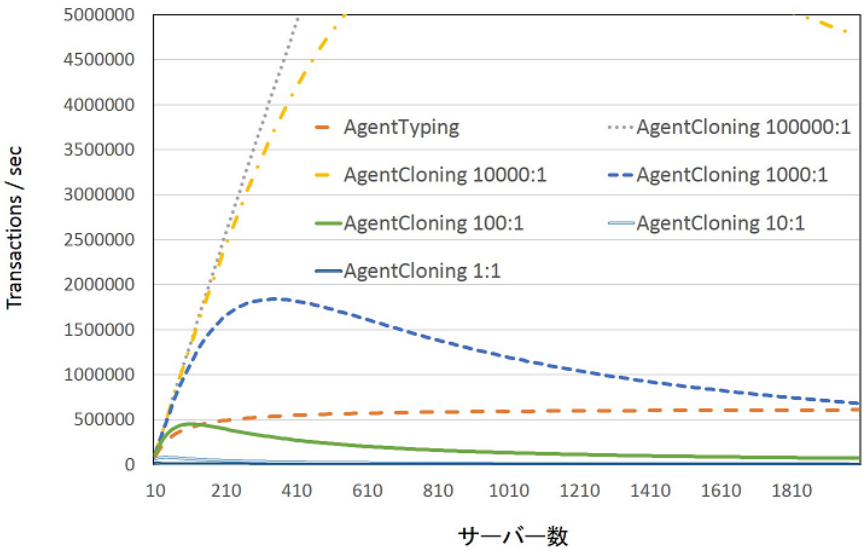

図 33: 内部通信と外部通信の割合を考慮したときのエージェント分散配置の性能比較

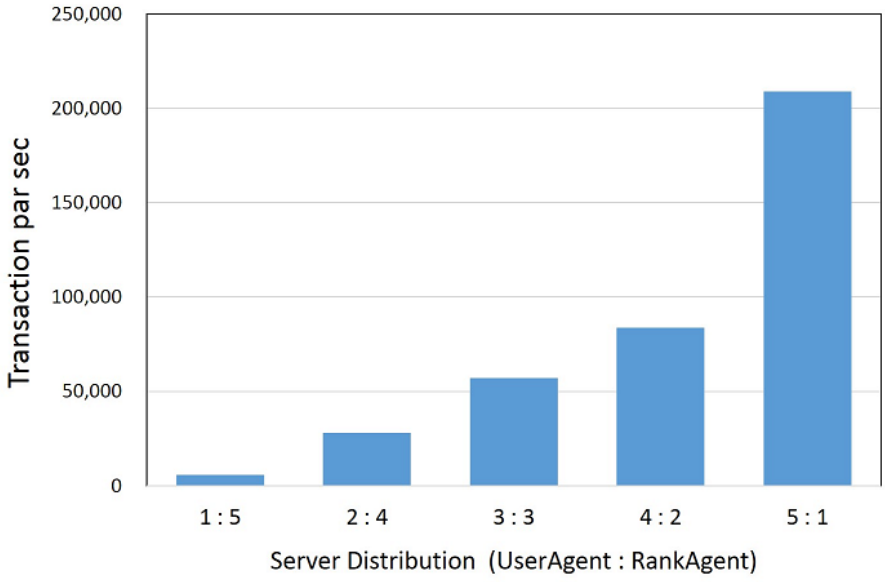

図 34: エージェントタイプによる配置のサーバ配分時の性能比較 (6 サーバー)

第5章　結論

本研究の成果として，単純なRDAシステムによりエージェントシステムの運用問題を明らかにし，解決する運用技術を開発した．また，複雑なRDAシステムでは，データキューブによるエージェント数の推定と2段階設計方法を提案することで，RDAシステムのエージェントの設計方法と効果的な運用技術が確立した．エージェントの種類推定と2段階設計法は，開発するシステムのデータキューブとシナリオから分散可能なエージェントの抽出，分散環境上への配置を行う設計法となっている．また，運用技術は予め開発者が任意にエージェント負荷の閾値を設定することにより，アプリケーションから時間的なデータ発生状況の変化にあわせて動的にエージェント数の制御を行う技術となっている．これにより，従来必要とされていた経験的設計や熟練した運用技術を必要とせずに，RDAシステムの開発が可能となった．

開発した運用技術である動的エージェント負荷分散機構は，従来のエージェント技術で用いられてきた負荷分散手法をRDAでも実行できるようにしたもので，エージェントシステムのスケーラビリティ維持する機能を持つ．この手法は，エージェント自身に負荷検出する機能と複製を作成する機能を備えることでメッセージパッシングにエージェントが負荷分散する仕組みとなっている．システム開発者は，システム実行中のエージェント負荷を考慮しなくてもエージェント自身がアプリケーションから発生するデータに対して適応的に増えることで処理性能を維持する．

エージェントシステムの設計で問題となるのは役割の抽出によるエージェントの導出と通信が発生するエージェントの分散環境上での配置である．2段階設計法は，上記の問題を解決したエージェント設計，エージェント分散配置の2段階からなる設計方法である．1段階目のエージェント設計は，要件定義からプロトタイピングまでの設計手順を繰り返し，実行することでシステム上の役割を抽出しエージェントの導出を行う設計手法である．2段階目のエージェント分散配置は，内外の通信頻度を考慮し，エージェントタイプによる配置とエージェントクローンによる配置の2つの配置戦略を選択する設計である．2つの配置戦略は，サーバー数や内外の通信頻度により選択を変えなければならない．この配置戦略は，実験やシミュレーションからほとんどの場合エージェントクローニングによる配置が最も効果的であることが確認された．一方，問い合わせ通信 (外部通信) 頻度が非常に高く，サーバー数が多い環

境ではエージェントタイプによる配置が有効である．

本研究では，はじめに単純なRDAシステムでエージェントシステムの運用の問題を抽出し解決するための運用技術を開発し，次に，集計システムとして典型的かつ処理性能を必要とするランキング集計を行う複雑なRDAシステムの設計方法論を開発した．これにより，スケーラビリティが高いRDAシステムをエージェントにより開発・実装することが可能となった．また，それぞれの運用技術・設計方法論は機械的に適用できるため，RDAシステムの開発に経験的・熟練的な技能は必要が無い開発方法論である．

5.1 今後の課題

今後の課題として，エージェントタイプによる配置での再配置問題をあげる．エージェントタイプでは，実験を繰り返すことで最適なサーバー配分を決定した，しかしながら，この最適なサーバー配分はエージェントタイプ内の処理コストや内外の通信コストのバランスによって変動する．例えば，ランクエージェントが保持するデータレコード数が多いとき，問い合わせ時のランキング処理時間もレコード数に比例して長くなると考えられる．このとき，サーバー内で同時に実行可能なスレッド数以上のエージェントが配置されているとき，エージェントの処理待ちが発生してしまう．エージェントは一度に1つのメッセージしか処理できないため，この処理待ち時間中に来たユーザーエージェントからのメッセージも処理されずにメッセージキューに蓄積されることになる．これにより，ランクエージェントが複製され，さらにエージェントが増えるという悪循環が発生してまう．上記の問題は，ランクエージェント側にサーバーを割り当てることで解決できる．しかしながら，ランクエージェントはユーザーエージェントと異なり，多くのデータレコードを保持しているためエージェント移動コストが高い．今後，エージェントの環境移動について移動コスト問題を考慮したサーバー配分方法を研究する必要がある．

謝辞

本論文を作成するにあたり，多くの方々にご助力を頂きましたことを心より感謝しております.

指導教官である寺野隆雄先生には修士，博士課程と長期にわたり研究指導していただき，研究の考え方や研究分野についての貴重なご指摘を多く頂きましたことを深く感謝いたします.連携教授でありました山本学先生には，技術的なご助言および議論を通して様々な知識，示唆を頂きました．また，先生の深いご見識による処理手法や設計上の性能問題へのご指摘により本システムを完成させることができました．心より感謝いたします．日々の研究生活において研究の進め方や考え方のご指導を頂きました吉川厚先生に感謝いたします．論文審査を通し，研究への貴重なご指摘や論文をまとめるためのご助言を頂きました新田克己先生，小野功先生，出口弘先生，三宅美博先生，野田五十樹先生に心より感謝いたします．寺野研究室の皆様には，ゼミや議論で研究へのご助言や示唆を頂きましたことを感謝いたします，

参考文献

[1] Actor foundry. `http://osl.cs.uiuc.edu/af/`. Accessed: 2016-6-24.

[2] Akka documentation. `http://akka.io/docs/`. Accessed: 2016-5-30.

[3] Apache storm documentation. `http://storm.apache.org/releases/`. Accessed: 2016-6-24.

[4] The foundation for intelligent physical agents. `http://www.fipa.org/`. Accessed: 2016-8-12.

[5] Hadoop documentation. `http://hadoop.apache.org/docs/`. Accessed: 2016-6-24.

[6] Spark documentation. `http://spark.apache.org/docs/`. Accessed: 2016-5-30.

[7] Lisa Amini, Navendu Jain, Anshul Sehgal, Jeremy Silber, and Olivier Verscheure. Adaptive control of extreme-scale stream processing systems. In *Distributed Computing Systems, 2006. ICDCS 2006. 26th IEEE International Conference on*, pp. 71–71. IEEE, 2006.

[8] Arvind Arasu, Shivnath Babu, and Jennifer Widom. The cql continuous query language: Semantic foundations and query execution. *The VLDB Journal*, Vol. 15, No. 2, pp. 121–142, June 2006.

[9] P Bernstein, Sergey Bykov, Alan Geller, Gabriel Kliot, and Jorgen Thelin. Orleans: Distributed virtual actors for programmability and scalability. Technical report, MSR Technical Report (MSR-TR-2014-41, 24). http://aka. ms/Ykyqft, 2014.

[10] Robert D Blumofe and Charles E Leiserson. Scheduling multithreaded computations by work stealing. *Journal of the ACM (JACM)*, Vol. 46, No. 5, pp. 720–748, 1999.

[11] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. *Commun. ACM*, Vol. 51, No. 1, pp. 107–113, January 2008.

[12] J Gray, et al. Transaction processing: concepts and techniques (excerpt), passage. *Transaction Processing: Concepts and Techniques*, pp. 373–445, 1993.

[13] BIG DATA WORKING GROUP. ビッグデータの分類. Technical report, Cloud Security Alliance. `http://www.cloudsecurityalliance.jp/bigdata_wg.html`, 2014.

[14] Carl Hewitt. Actor model of computation: scalable robust information systems. *arXiv preprint arXiv:1008.1459*, 2010.

[15] Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. In *ACM SIGOPS Operating Systems Review*, Vol. 41, pp. 59–72. ACM, 2007.

[16] Toru Ishida. *Parallel, distributed and multiagent production systems*, Vol. 878. Springer Heidelberg, 1994.

[17] David Karger, Eric Lehman, Tom Leighton, Rina Panigrahy, Matthew Levine, and Daniel Lewin. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pp. 654–663. ACM, 1997.

[18] Rajesh K Karmani, Amin Shali, and Gul Agha. Actor frameworks for the jvm platform: a comparative analysis. In *Proceedings of the 7th International Conference on Principles and Practice of Programming in Java*, pp. 11–20. ACM, 2009.

[19] Advanced Agent-Robotics Technology Lab. Retsina agent architecture. Technical report, Carnegie Mellon University. `https://www.cs.cmu.edu/~softagents/retsina_agent_arch.html`. Accessed: 2016-8-12.

[20] David C Luckham and Brian Frasca. Complex event processing in distributed systems. *Computer Systems Laboratory Technical Report CSL-TR-98-754. Stanford University, Stanford*, Vol. 28, , 1998.

[21] O. Shehory, K. Sycara, P. Chalasani, and S. Jha. Agent cloning: an approach to agent mobility and resource allocation. *IEEE Communications Magazine*, Vol. 36, No. 7, pp. 58, 63–67, Jul 1998.

[22] Andrew S Tanenbaum and Maarten Van Steen. *Distributed systems*. Prentice-Hall, 2007.

[23] Michael Wooldridge and Nicholas R Jennings. Intelligent agents: Theory and practice. *The knowledge engineering review*, Vol. 10, No. 02, pp. 115–152, 1995.

[24] Yuya Murata, Gaku Yamamoto and Takao Terano. Computing your marathon ranking as a testbed for agent-based mobile systems. In *Proc. Fifth International Workshop on Collaborative Agents – Research & Development 2014(CARE2014), AAMAS2014 Workshop W27*, 2014.

[25] Yuya Murata, Gaku Yamamoto and Takao Terano. Improving the performance of mobile application systems through agent deployment strategies in a distributed environment. In *5th World Congress on Social Simulation*, 2014.

[26] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J Franklin, Scott Shenker, and Ion Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pp. 2–2. USENIX Association, 2012.

[27] 山本学. エージェントプログラミングモデルの高速トランザクション処理システムに対する効果の考察. 合同エージェントワークショップ&シンポジウム 2008(JAWS2008), 2008.

[28] 山本学. エージェントプログラミングモデルに基づくデータキャッシュの性能評価. 合同エージェントワークショップ&シンポジウム 2010(JAWS2010), 2010.

[29] 山本学. 高速マルチエージェントシステムによるリアルタイムデータ集計処理の考察. 合同エージェントワークショップ&シンポジウム 2011(JAWS2011), 2011.

[30] 山本学. エージェントプログラミングモデルの業務システムへの適用からの考察. コンピュータ ソフトウェア, Vol. 31, No. 3, pp. 3 109–3 119, 2014.

[31] 村田悠也, 山本学, 寺野隆雄. リアルタイムデータ集計を行う高速マルチエージェントシステムにおけるエージェント分散配置の性能評価. 合同エージェントワークショップ&シンポジウム 2012(JAWS2012), 2012.

[32] 村田悠也, 山本学, 寺野隆雄. リアルタイムデータ集計アプリケーションにおける効果的なエージェント分散配置. 合同エージェントワークショップ&シンポジウム 2014(JAWS2014), 2014.

付 録A　　RDAシステム関連資料

- センサーログマイニングシステム　エージェント定義
- センサーログマイニングシステム　ソースコード
  - テストデータ生成
  - エージェント
  - エージェントハンドラ
  - メッセージキュー
  - ウィンドウ制御
  - 動的エージェント負荷分散　のプログラムを添付
- ランキングシステム　エージェント定義
- ランキングシステム　ソースコード
  - エージェント (２タイプ)
  - エージェントハンドラ (２タイプ)
  - エージェント間通信
  - 動的エージェント負荷分散・　エージェント分散配置　のプログラムを添付

A.1 センサーログマイニングシステム

```
01: <?xml version="1.0"?>
02: <agentdef package="rda" version="rda1.0">
03:     <entities>
04:         <!--AggregateAgent Entity define-->
05:         <entity type="aggregateagent">
06:             <attribute name="AgentID" type="string" primarykey="true" maxlength="32"/>
07:             <attribute name="Condition" type="aggregatecondition" singleentity="true"/>
08:             <attribute name="Data" type="long" />
09:             <attribute name="ConnectionCount" type="long" />
10:             <attribute name="MessageLatency" type="long" />
11:             <attribute name="MessageQueueLength" type="long" />
12:             <attribute name="Log" type="log" />
13:         </entity>
14:         <entity type="aggregatecondition">
15:             <attribute name="AgentID" type="string" primarykey="true" relationto="AgentID" maxlength="32
   : "/>
16:             <attribute name="Conditions" type="string" maxlength="32"/>
17:             <attribute name="LastAccessTime" type="timestamp" />
18:         </entity>
19:         <entity type="log">
20:             <attribute name="AgentID" type="string" primarykey="true" relationto="AgentID" maxlength="32
   : "/>
21:             <attribute name="AccessID" type="string" primarykey="true" maxlength="16"/>
22:             <attribute name="CurrentTime" type="long"/>
23:             <attribute name="LastAccessTime" type="timestamp" />
24:         </entity>
25:     </entities>
26:     <messages>
27:         <!--system define-->
28:         <!--AggregateAgent Messages -->
29:         <message type="initAgent" class="rda.agent.message.InitMessage"/>
30:         <message type="readAgent" />
31:         <message type="updateAgent" class="rda.agent.message.UpdateMessage"/>
32:         <message type="readLogAgent"/>
33:
34:         <!--agent system define-->
35:         <message type="dispose" />
36:         <message type="putmessage" class="rda.agent.queue.PutMessage"/>
37:     </messages>
38:     <agents>
39:         <!--system define-->
40:         <agent type="aggregateagent">
41:             <handler message="initAgent" class="rda.agent.handler.InitHandler"/>
42:             <handler message="readAgent" class="rda.agent.handler.ReadHandler"/>
43:             <handler message="updateAgent" class="rda.agent.handler.UpdateHandler"/>
44:             <handler message="readLogAgent" class="rda.agent.handler.ReadLogHandler"/>
45:
46:             <handler message="dispose" class="rda.agent.disposer.handler.DisposeHandler"/>
47:             <handler message="putmessage" class="rda.agent.queue.handler.PutMessageHandler"/>
48:         </agent>
49:     </agents>
50: </agentdef>
```

```java
package rda.data.type;

import rda.data.Data;
import rda.data.DataType;
import rda.data.test.TestData;

public class FlatData implements DataType{
    private final String name;
    private final Data data;

    private static Long count;

    private Long term, period;
    private Long volume;

    public FlatData(Long time, Long period, int volume, int numberOfUser, int valueOfUser, int datamode, long seed) {
        this.name = "FlatType";
        this.data = new Data();

        this.term = time;
        this.period = period;
        this.volume = (time+1)*volume/2;

        //initialise
        data.init(numberOfUser, valueOfUser, datamode, seed);

        count = -1L;
    }

    @Override
    public TestData nextData(Long time) {
        if((time == 0) && (count == -1)) count = volume.longValue()-1;

        count++;
        TestData test = (TestData) data.getData();

        if(count > volume.longValue()) {
            test = null;
            count = -1L;
        }

        return test;
    }

    @Override
    public String getName() {
        return this.name;
    }

    @Override
    public String toString(){
        Long n = term * 1000 / period;
        Long result = n * volume;

        return name + " DataN_" + result;
    }

    @Override
    public String toString(Long time) {
        if(time == -1L) return toString();
        return String.valueOf(volume);
    }
}
```

```
 1: package rda;
 2:
 3: import com.ibm.agent.exa.AgentException;
 4: import com.ibm.agent.exa.impl.HPAEntity;
 5: import com.ibm.agent.exa.TxID;
 6: import com.ibm.agent.exa.entity.EntitySet;
 7: import com.ibm.agent.exa.entity.EntityKey;
 8: import com.ibm.agent.exa.entity.Entity;
 9: import java.util.Iterator;
10:
11: /**
12:  * Generated code for aggregateagent.
13:  *
14:  * <p>entity type="aggregateagent tablename="aggregateagent <br>
15:  * attribute name="AgentID" type="STRING" primarykey="true" <br>
16:  * attribute name="Condition" type="aggregatecondition" <br>
17:  * attribute name="Data" type="LONG" <br>
18:  * attribute name="ConnectionCount" type="LONG" <br>
19:  * attribute name="MessageLatency" type="LONG" <br>
20:  * attribute name="MessageQueueLength" type="LONG" <br>
21:  * attribute name="Log" type="log" <br>
22:  **/
23: public class Aggregateagent extends HPAEntity {
24:     /**
25:     * Primary key size
26:     **/
27:     public static final int PKINDEXSIZE = 1;
28:
29:     /**
30:     * Primary key index of AgentID
31:     **/
32:     public static final int PKINDEX_AGENTID = 0;
33:
34:     /**
35:     * Column index of AgentID
36:     **/
37:     public static final int AGENTID = 0;
38:
39:     /**
40:     * Column index of Condition
41:     **/
42:     public static final int CONDITION = 1;
43:
44:     /**
45:     * Column index of Data
46:     **/
47:     public static final int DATA = 2;
48:
49:     /**
50:     * Column index of ConnectionCount
51:     **/
52:     public static final int CONNECTIONCOUNT = 3;
53:
54:     /**
55:     * Column index of MessageLatency
56:     **/
57:     public static final int MESSAGELATENCY = 4;
58:
59:     /**
60:     * Column index of MessageQueueLength
61:     **/
62:     public static final int MESSAGEQUEUELENGTH = 5;
63:
64:     /**
65:     * Column index of Log
66:     **/
67:     public static final int LOG = 6;
68:
69:     /**
70:      * This constructor is used by the runtime.
71:      * An application should not create an instance with this constructor
72:     **/
73:     public Aggregateagent() {
74:         super();
75:     }
76:
77:     /**
78:      * Get the version string
79:     **/
80:     public String getVersion() {
81:         return "rda1.0";
82:     }
83:
84:     /**
85:      * Get a value of AgentID.
86:      * The setter method of AgentID is not generated because this attribute is
 a primarykey.
87:      * @return AgentID
88:      **/
89:     public final String getAgentID(TxID tx) {
90:         // generated code
91:         return getString(tx,0);
92:     }
93:
94:     /**
95:      * Get a set of Condition.
96:      * Entity type of this entity set is aggregatecondition.
97:      * A returned entity set has a single entity.
98:      * The setter method of Condition is not generated because this attribute
is a EntitySet.
99:      * @return an entity set containing Aggregatecondition
100:      * @throws AgentException
101:      **/
102:     public final EntitySet getConditionSet(TxID tx) throws AgentException {
103:         // generated code
104:         return getEntitySet(tx,1);
105:     }
106:
107:     /**
108:      * Get a value of Condition.
109:      * @param tx a transaction context
110:      * @return Aggregatecondition
111:      * @throws AgentException
112:      **/
113:     public final Aggregatecondition getCondition(TxID tx) throws AgentExceptio
n {
114:         // generated code
115:         EntitySet es = getEntitySet(tx,1);
116:         if (es == null) return null;
117:         return (Aggregatecondition)es.getSingleEntity();
118:     }
119:
120:     /**
121:      * Create a value of Aggregatecondition.
```

```
   122:      * @param tx a transaction context
   123:      * @return Aggregatecondition
   124:      **/
   125:     public final Aggregatecondition createCondition(TxID tx) throws AgentExcep
tion {
   126:         // generated code
   127:         EntitySet es = getEntitySet(tx,1);
   128:         Aggregatecondition entity = (Aggregatecondition)es.createEntity(tx,new
 Object[1]);
   129:         return entity;
   130:     }
   131:
   132:     /**
   133:      * @return Data
   134:      **/
   135:     public final long getData(TxID tx) {
   136:         // generated code
   137:         return getLong(tx,2);
   138:     }
   139:
   140:     /**
   141:      * Set a value to Data.
   142:      * @param tx a transaction context
   143:      * @param value a value to be set to Data
   144:      **/
   145:     public final void  setData(TxID tx, long value) throws AgentException {
   146:         // generated code
   147:         setLong(tx,2,value);
   148:     }
   149:
   150:     /**
   151:      * @return ConnectionCount
   152:      **/
   153:     public final long getConnectionCount(TxID tx) {
   154:         // generated code
   155:         return getLong(tx,3);
   156:     }
   157:
   158:     /**
   159:      * Set a value to ConnectionCount.
   160:      * @param tx a transaction context
   161:      * @param value a value to be set to ConnectionCount
   162:      **/
   163:     public final void  setConnectionCount(TxID tx, long value) throws AgentExc
eption {
   164:         // generated code
   165:         setLong(tx,3,value);
   166:     }
   167:
   168:     /**
   169:      * @return MessageLatency
   170:      **/
   171:     public final long getMessageLatency(TxID tx) {
   172:         // generated code
   173:         return getLong(tx,4);
   174:     }
   175:
   176:     /**
   177:      * Set a value to MessageLatency.
   178:      * @param tx a transaction context
   179:      * @param value a value to be set to MessageLatency
   180:      **/
   181:     public final void  setMessageLatency(TxID tx, long value) throws AgentExce
ption {
   182:         // generated code
   183:         setLong(tx,4,value);
   184:     }
   185:
   186:     /**
   187:      * @return MessageQueueLength
   188:      **/
   189:     public final long getMessageQueueLength(TxID tx) {
   190:         // generated code
   191:         return getLong(tx,5);
   192:     }
   193:
   194:     /**
   195:      * Set a value to MessageQueueLength.
   196:      * @param tx a transaction context
   197:      * @param value a value to be set to MessageQueueLength
   198:      **/
   199:     public final void  setMessageQueueLength(TxID tx, long value) throws Agent
Exception {
   200:         // generated code
   201:         setLong(tx,5,value);
   202:     }
   203:
   204:     /**
   205:      * Get a set of Log.
   206:      * Entity type of this entity set is log.
   207:      * The setter method of Log is not generated because this attribute is a E
ntitySet.
   208:      * @return an entity set containing Log
   209:      * @throws AgentException
   210:      **/
   211:     public final EntitySet getLog(TxID tx) throws AgentException {
   212:         // generated code
   213:         return getEntitySet(tx,6);
   214:     }
   215:
   216:     /**
   217:      * Get a value of Log.
   218:      * @param tx a transaction context
   219:      * @param AccessID
   220:      * @return Log
   221:      * @throws AgentException
   222:      **/
   223:     public final Log getLog(TxID tx,String AccessID) throws AgentException {
   224:         // generated code
   225:         EntitySet es = getEntitySet(tx,6);
   226:         Entity parent = es.getParent();
   227:         Object[] primaryKeys = new Object[]{parent.getObject(tx,0),AccessID};
   228:         EntityKey ek = new EntityKey("log", primaryKeys);
   229:         Log entity = (Log)es.getEntity(ek);
   230:         return entity;
   231:     }
   232:
   233:     /**
   234:      * Create a value of Log.
   235:      * @param tx a transaction context
   236:      * @param AccessID
   237:      * @return Log
   238:      **/
   239:     public final Log createLog(TxID tx,String AccessID) throws AgentException
```

```
{
  240:         // generated code
  241:         if (AccessID.length() > 16) {
  242:             throw new AgentException("AccessID > maxlength(16)");
  243:         }
  244:         EntitySet es = getEntitySet(tx,6);
  245:         Object[] primaryKeys = new Object[]{null,AccessID};
  246:         Log entity = (Log)es.createEntity(tx,primaryKeys);
  247:         return entity;
  248:     }
  249:
  250:     /**
  251:      * Get an iterator of Log.
  252:      * @param tx a transaction context
  253:      **/
  254:     public final Iterator<Entity> getLogIterator(TxID tx) throws AgentExceptio
n {
  255:         // generated code
  256:         EntitySet es = getEntitySet(tx,6);
  257:         return es.iterator(tx);
  258:     }
  259:
  260: }
```

```
    1: package rda;
    2:
    3: import com.ibm.agent.exa.AgentException;
    4: import com.ibm.agent.exa.impl.HPAEntity;
    5: import com.ibm.agent.exa.TxID;
    6:
    7: /**
    8:  * Generated code for aggregatecondition.
    9:  *
   10:  * <p>entity type="aggregatecondition tablename="aggregatecondition <br>
   11:  * attribute name="AgentID" type="STRING" primarykey="true" relationto="AgentI
D" <br>
   12:  * attribute name="Conditions" type="STRING" <br>
   13:  * attribute name="LastAccessTime" type="TIMESTAMP" <br>
   14: **/
   15: public class Aggregatecondition extends HPAEntity {
   16:     /**
   17:     * Primary key size
   18:     **/
   19:     public static final int PKINDEXSIZE = 1;
   20:
   21:     /**
   22:     * Primary key index of AgentID
   23:     **/
   24:     public static final int PKINDEX_AGENTID = 0;
   25:
   26:     /**
   27:     * Column index of AgentID
   28:     **/
   29:     public static final int AGENTID = 0;
   30:
   31:     /**
   32:     * Column index of Conditions
   33:     **/
   34:     public static final int CONDITIONS = 1;
   35:
   36:     /**
   37:     * Column index of LastAccessTime
   38:     **/
   39:     public static final int LASTACCESSTIME = 2;
   40:
   41:     /**
   42:      * This constructor is used by the runtime.
   43:      * An application should not create an instance with this constructor
   44:     **/
   45:     public Aggregatecondition() {
   46:         super();
   47:     }
   48:
   49:     /**
   50:      * Get the version string
   51:     **/
   52:     public String getVersion() {
   53:         return "rda1.0";
   54:     }
   55:
   56:     /**
   57:      * Get a value of AgentID.
   58:      * The setter method of AgentID is not generated because this attribute is
 a primarykey.
   59:      * @return AgentID
   60:      **/
   61:     public final String getAgentID(TxID tx) {
   62:         // generated code
   63:         return getString(tx,0);
   64:     }
   65:
   66:     /**
   67:      * @return Conditions
   68:      **/
   69:     public final String getConditions(TxID tx) {
   70:         // generated code
   71:         return getString(tx,1);
   72:     }
   73:
   74:     /**
   75:      * Set a value to Conditions.
   76:      * @param tx a transaction context
   77:      * @param value a value to be set to Conditions
   78:      **/
   79:     public final void  setConditions(TxID tx, String value) throws AgentExcept
ion {
   80:         // generated code
   81:         if (value != null && value.length() > 32) {
   82:             throw new AgentException("Conditions > maxlength(32)");
   83:         }
   84:         setString(tx,1,value);
   85:     }
   86:
   87:     /**
   88:      * @return LastAccessTime
   89:      **/
   90:     public final java.sql.Timestamp getLastAccessTime(TxID tx) {
   91:         // generated code
   92:         return getTimestamp(tx,2);
   93:     }
   94:
   95:     /**
   96:      * Set a value to LastAccessTime.
   97:      * @param tx a transaction context
   98:      * @param value a value to be set to LastAccessTime
   99:      **/
  100:     public final void  setLastAccessTime(TxID tx, java.sql.Timestamp value) th
rows AgentException {
  101:         // generated code
  102:         setTimestamp(tx,2,value);
  103:     }
  104:
  105: }
```

```
    1: package rda;
    2:
    3: import com.ibm.agent.exa.AgentException;
    4: import com.ibm.agent.exa.impl.HPAEntity;
    5: import com.ibm.agent.exa.TxID;
    6:
    7: /**
    8:  * Generated code for log.
    9:  *
   10:  * <p>entity type="log tablename="log <br>
   11:  * attribute name="AgentID" type="STRING" primarykey="true" relationto="AgentI
D" <br>
   12:  * attribute name="AccessID" type="STRING" primarykey="true" <br>
   13:  * attribute name="CurrentTime" type="LONG" <br>
   14:  * attribute name="LastAccessTime" type="TIMESTAMP" <br>
   15: **/
   16: public class Log extends HPAEntity {
   17:     /**
   18:     * Primary key size
   19:     **/
   20:     public static final int PKINDEXSIZE = 2;
   21:
   22:     /**
   23:     * Primary key index of AgentID
   24:     **/
   25:     public static final int PKINDEX_AGENTID = 0;
   26:
   27:     /**
   28:     * Primary key index of AccessID
   29:     **/
   30:     public static final int PKINDEX_ACCESSID = 1;
   31:
   32:     /**
   33:     * Column index of AgentID
   34:     **/
   35:     public static final int AGENTID = 0;
   36:
   37:     /**
   38:     * Column index of AccessID
   39:     **/
   40:     public static final int ACCESSID = 1;
   41:
   42:     /**
   43:     * Column index of CurrentTime
   44:     **/
   45:     public static final int CURRENTTIME = 2;
   46:
   47:     /**
   48:     * Column index of LastAccessTime
   49:     **/
   50:     public static final int LASTACCESSTIME = 3;
   51:
   52:     /**
   53:      * This constructor is used by the runtime.
   54:      * An application should not create an instance with this constructor
   55:     **/
   56:     public Log() {
   57:         super();
   58:     }
   59:
   60:     /**
   61:      * Get the version string
   62:     **/
   63:     public String getVersion() {
   64:         return "rda1.0";
   65:     }
   66:
   67:     /**
   68:      * Get a value of AgentID.
   69:      * The setter method of AgentID is not generated because this attribute is
 a primarykey.
   70:      * @return AgentID
   71:      **/
   72:     public final String getAgentID(TxID tx) {
   73:         // generated code
   74:         return getString(tx,0);
   75:     }
   76:
   77:     /**
   78:      * Get a value of AccessID.
   79:      * The setter method of AccessID is not generated because this attribute i
s a primarykey.
   80:      * @return AccessID
   81:      **/
   82:     public final String getAccessID(TxID tx) {
   83:         // generated code
   84:         return getString(tx,1);
   85:     }
   86:
   87:     /**
   88:      * @return CurrentTime
   89:      **/
   90:     public final long getCurrentTime(TxID tx) {
   91:         // generated code
   92:         return getLong(tx,2);
   93:     }
   94:
   95:     /**
   96:      * Set a value to CurrentTime.
   97:      * @param tx a transaction context
   98:      * @param value a value to be set to CurrentTime
   99:      **/
  100:     public final void  setCurrentTime(TxID tx, long value) throws AgentExcepti
on {
  101:         // generated code
  102:         setLong(tx,2,value);
  103:     }
  104:
  105:     /**
  106:      * @return LastAccessTime
  107:      **/
  108:     public final java.sql.Timestamp getLastAccessTime(TxID tx) {
  109:         // generated code
  110:         return getTimestamp(tx,3);
  111:     }
  112:
  113:     /**
  114:      * Set a value to LastAccessTime.
  115:      * @param tx a transaction context
  116:      * @param value a value to be set to LastAccessTime
  117:      **/
  118:     public final void  setLastAccessTime(TxID tx, java.sql.Timestamp value) th
rows AgentException {
  119:         // generated code
```

```
120:         setTimestamp(tx,3,value);
121:     }
122:
123: }
```

```
    1: package rda.agent.handler;
    2:
    3: import java.sql.Timestamp;
    4:
    5: import rda.Log;
    6: import rda.agent.message.InitMessage;
    7:
    8: import com.ibm.agent.exa.Message;
    9: import com.ibm.agent.exa.MessageHandler;
   10: import com.ibm.agent.exa.TxID;
   11: import rda.Aggregateagent;
   12: import rda.Aggregatecondition;
   13:
   14: public class InitHandler extends MessageHandler {
   15:     @Override
   16:     public Object onMessage(Message msg) throws Exception {
   17:         try {
   18:             InitMessage initMsg = (InitMessage)msg;
   19:
   20:             Aggregateagent agent = (Aggregateagent)getEntity();
   21:
   22:             TxID tx = getTx();
   23:             Aggregatecondition cond = agent.createCondition(tx);
   24:
   25:             cond.setConditions(tx, initMsg.condition);
   26:
   27:             Long time = System.currentTimeMillis();
   28:             Timestamp registerTime = new Timestamp(time);
   29:             cond.setLastAccessTime(tx, registerTime);
   30:
   31:             agent.setData(tx, 0);
   32:
   33:             agent.setConnectionCount(tx, 0);
   34:
   35:             agent.setMessageLatency(tx, 0);
   36:             agent.setMessageQueueLength(tx, 0);
   37:
   38:             Log log = agent.createLog(tx, "init");
   39:
   40:             log.setLastAccessTime(tx, registerTime);
   41:             log.setCurrentTime(tx, time);
   42:
   43:             //System.out.println("InitHandler of Agent:" + getAgentKey() + " w
as initialized");
   44:
   45:             return "hello from agent:" + getAgentKey();
   46:         } catch(Exception e) {
   47:             throw e;
   48:         }
   49:     }
   50: }
```

```java
package rda.agent.handler;

import java.sql.Timestamp;

import rda.Log;

import com.ibm.agent.exa.Message;
import com.ibm.agent.exa.MessageHandler;
import com.ibm.agent.exa.TxID;
import java.util.List;
import rda.Aggregateagent;
import rda.agent.queue.MessageObject;
import rda.agent.message.UpdateMessage;

public class UpdateHandler extends MessageHandler{

    @Override
    public Object onMessage(Message msg) throws Exception {
        UpdateMessage updateMsg = (UpdateMessage) msg;
        MessageObject msgObj = (MessageObject) updateMsg.messageData;

        Aggregateagent agent = (Aggregateagent)getEntity();

        TxID tx = getTx();
        long updateData = 0;
        for(Object data : (List)msgObj.data){
            updateData =  updateData + (int)data;
        }

        agent.setData(tx, agent.getData(tx)+updateData);

        //Agent Status
        //Connection
        agent.setConnectionCount(tx, agent.getConnectionCount(tx) + 1);
        //Message Latency
        agent.setMessageLatency(tx, msgObj.latency());
        //Agent Message Queue
        //agent.setMessageQueueLength(tx, msgObj.getLength());

        // Update Log Records
        Log log = agent.getLog(tx, "update");
        if(log == null)
            log = agent.createLog(tx, "update");

        // Update LastAccessTime
        Long time = System.currentTimeMillis();
        Timestamp updateTime = new Timestamp(time);
        log.setLastAccessTime(tx, updateTime);
        log.setCurrentTime(tx, time);

        //Long message = avgLatency;

        return 0L;
    }
}
```

```
 1: package rda.agent.handler;
 2:
 3: import com.ibm.agent.exa.Message;
 4: import com.ibm.agent.exa.MessageHandler;
 5: import com.ibm.agent.exa.TxID;
 6: import rda.Aggregateagent;
 7: import rda.agent.reader.AgentInfo;
 8:
 9: public class ReadHandler extends MessageHandler {
10:
11:     @Override
12:     public Object onMessage(Message arg0) throws Exception {
13:         Aggregateagent agent = (Aggregateagent) getEntity();
14:
15:         TxID tx = getTx();
16:
17:         AgentInfo info = new AgentInfo(
18:                 /*AgentID*/agent.getAgentID(tx),
19:                 /*data*/ agent.getData(tx),
20:                 /*count */ agent.getConnectionCount(tx)
21:         );
22:
23:         return info;
24:     }
25:
26: }
```

```java
package rda.agent.handler;

import java.util.Iterator;

import rda.Log;
import com.ibm.agent.exa.Message;
import com.ibm.agent.exa.MessageHandler;
import com.ibm.agent.exa.TxID;
import com.ibm.agent.exa.entity.Entity;
import rda.Aggregateagent;
import rda.agent.logger.LogInfo;

public class ReadLogHandler extends MessageHandler{

    @Override
    public Object onMessage(Message msg) throws Exception {
        Aggregateagent agent = (Aggregateagent)getEntity();

        TxID tx = getTx();

        StringBuilder accessIDList = new StringBuilder();
        StringBuilder accessTimeList = new StringBuilder();
        Iterator<Entity> it = agent.getLogIterator(tx);
        while(it.hasNext()){
            Log log = (Log) it.next();

            accessIDList.append(log.getAccessID(tx));
            accessIDList.append(",");


            accessTimeList.append(log.getLastAccessTime(tx));
            accessTimeList.append(",");
        }

        LogInfo info = new LogInfo(
            /*AgentID*/      agent.getAgentID(tx),
            /*ConnectCount*/ agent.getConnectionCount(tx),
            /*AccessLog*/        accessIDList.toString(),
            /*AccessTime*/       accessTimeList.toString());

        return info;
    }
}
```

```
    1: package rda.agent.queue.handler;
    2:
    3: import com.ibm.agent.exa.Message;
    4: import com.ibm.agent.exa.MessageHandler;
    5: import com.ibm.agent.exa.TxID;
    6: import rda.Aggregateagent;
    7: import rda.agent.queue.MessageQueue;
    8: import rda.agent.queue.PutMessage;
    9: import rda.manager.AgentMessageQueueManager;
   10:
   11: public class PutMessageHandler extends MessageHandler{
   12:     private MessageQueue mq;
   13:
   14:     @Override
   15:     public Object onMessage(Message msg) throws Exception {
   16:         AgentMessageQueueManager manager = AgentMessageQueueManager.getInstanc
e();
   17:
   18:         PutMessage putMsg = (PutMessage)msg;
   19:
   20:         Aggregateagent agent = (Aggregateagent)getEntity();
   21:         TxID tx = getTx();
   22:         String id = agent.getAgentID(tx);
   23:
   24:         //Get MessageQueue
   25:         //MessageQueue mq = (MessageQueue)manager.getMQMap().get(id);
   26:
   27:         //System.out.println(" >> Agent Put MessageQueue Check!! "+ id +" - "+
putMsg.toString());
   28:
   29:         //Put Message
   30:         /*if(mq == null){
   31:             mq = new MessageQueue(id, 1000, 100L, 10L);
   32:             mq.setAgentType(new UpdateAgent(id));
   33:             manager.register(mq);
   34:         }
   35:
   36:         agent.setMessageQueueLength(tx, mq.getQueueLenght());
   37:
   38:         mq.put(putMsg.msgPack);
   39:         */
   40:         return 0L;//mq.getID() + " : success put messages = "+mq.getQueueLengh
t();
   41:     }
   42: }
```

```
    1: package rda.agent.queue;
    2:
    3: import java.util.concurrent.BlockingQueue;
    4: import java.util.concurrent.LinkedBlockingDeque;
    5: import java.util.concurrent.TimeUnit;
    6: import rda.agent.template.AgentType;
    7: import rda.clone.AgentCloning;
    8: import rda.manager.AgentManager;
    9: import rda.manager.IDManager;
   10:
   11: public class MessageQueue extends MessageQueueProcess{
   12:     private AgentManager manager;
   13:     private BlockingQueue<Object> queue;
   14:     private String name;
   15:     private AgentType agent;
   16:     private Integer size;
   17:     private long getwait, putwait;
   18:
   19:     public MessageQueue(AgentManager manager, String name, Integer size, Long
queuewait, Long agentwait){
   20:         this.manager = manager;
   21:
   22:         this.name = name;
   23:         this.size = size;
   24:         this.getwait = agentwait;
   25:         this.putwait = queuewait;
   26:         this.queue = new LinkedBlockingDeque<>(size+1);
   27:
   28:         //Message Queue Length @RECORDS
   29:         QueueObserver observe = new QueueObserver(name, queue);
   30:         register(observe);
   31:     }
   32:
   33:     private void register(QueueObserver observe){
   34:         manager.add(observe);
   35:     }
   36:
   37:     @Override
   38:     public Object get(){
   39:         Object obj = null;
   40:         try {
   41:             obj = queue.poll(getwait, TimeUnit.MILLISECONDS);
   42:         } catch (InterruptedException ex) {
   43:         }
   44:
   45:         return obj;
   46:     }
   47:
   48:     @Override
   49:     public void put(Object msgpack) throws MessageQueueEvent{
   50:         boolean success = false;
   51:         try {
   52:             success = queue.offer(msgpack, putwait, TimeUnit.MILLISECONDS);
   53:         } catch (InterruptedException ex) {
   54:         }
   55:
   56:         if(!success) eventClone();
   57:
   58:         if(isClone() && ((queue.size() + orgQueue.size()) < size/2)) eventDele
te();
   59:     }
   60:
   61:     //Load Balancer Cloning updgrade
   62:     public void eventClone()  throws MessageQueueEvent{
   63:         IDManager id = manager.getIDManager();
   64:         String cloneID = AgentCloning.cloning(manager, name, queue);
   65:         MessageQueueEvent.printState("cloning", id.getOrigID(name), cloneID, m
anager.getNumAgents());
   66:
   67:         throw new MessageQueueEvent(name, id.getOrigID(name), cloneID);
   68:     }
   69:
   70:     //Load Balancer Cloning degrade
   71:     public void eventDelete() {
   72:         IDManager id = manager.getIDManager();
   73:         MessageQueueEvent.printState("delete", name, id.getOrigID(name), manag
er.getNumAgents());
   74:         String deleteID = AgentCloning.delete(manager, name);
   75:     }
   76:
   77:     //Only AgnetClone
   78:     private LinkedBlockingDeque orgQueue;
   79:     public void setOriginalQueue(Object originalState){
   80:         this.orgQueue =  (LinkedBlockingDeque) originalState;
   81:
   82:         this.checkClone = true;
   83:
   84:         //Work Stealing
   85:         Object obj;
   86:         int i = size / 2;
   87:         while((obj = orgQueue.pollFirst()) != null){
   88:             i--;
   89:             if(i <= 0) break;
   90:             try {
   91:                 put(obj);
   92:             } catch (MessageQueueEvent mqev) {
   93:                 mqev.printEvent();
   94:             }
   95:         }
   96:     }
   97:
   98:     private Boolean checkClone = false;
   99:     private Boolean isClone(){
  100:         return checkClone;
  101:     }
  102:
  103:     //MessageQueue Process Overrides
  104:     @Override
  105:     public Boolean getRunnable() {
  106:         return manager.getState();
  107:     }
  108:
  109:     @Override
  110:     public void setAgentType(AgentType type) {
  111:         this.agent = type;
  112:     }
  113:
  114:     @Override
  115:     public AgentType getAgentType() {
  116:         return this.agent;
  117:     }
  118:
  119:     public String getID(){
  120:         return name;
```

```
121:     }
122:
123:     public Integer getQueueLenght(){
124:         return queue.size();
125:     }
126: }
```

```
    1: package rda.agent.queue;
    2:
    3: import rda.agent.template.AgentType;
    4:
    5: public abstract class MessageQueueProcess extends Thread{
    6:     abstract public Boolean getRunnable();
    7:
    8:     abstract public Object get();
    9:     abstract public void put(Object message) throws MessageQueueEvent;
   10:
   11:     abstract public void setAgentType(AgentType type);
   12:     abstract public AgentType getAgentType();
   13:
   14:     @Override
   15:     public void run(){
   16:         AgentType agent = getAgentType();
   17:         System.out.println(">AgentID::"+agent.getID());
   18:         System.out.println(">> Start--MessageQueue of "+agent.getID()+" run me
ssage processing");
   19:         while(getRunnable()){
   20:             Object msgpack = get();
   21:
   22:             if(msgpack != null)
   23:                 agent.sendMessage(msgpack);
   24:
   25:         }
   26:         System.out.println(">> Stop--MessageQueue of "+agent.getID()+" shutdow
n message processing");
   27:     }
   28: }
```

```java
package rda.agent.queue;

import rda.agent.template.MessageTemplate;

public class MessageObject extends MessageTemplate{
    public String id;
    public Object data;

    public MessageObject(String destID, Object data) {
        super();
        this.id = destID;
        this.data = data;
    }

    @Override
    public String toString() {
        return id + ": datasize=" + data;
    }
}
```

```
    1: package rda.agent.queue;
    2:
    3: import rda.log.AgentLogPrint;
    4:
    5: public class MessageQueueEvent extends Exception{
    6:     private String name, clonename, original;
    7:
    8:     public MessageQueueEvent(String name, String originalID, String clonename)
 {
    9:         this.name = name;
   10:         this.clonename = clonename;
   11:         this.original = originalID;
   12:     }
   13:
   14:     public void printEvent(){
   15:         //if(AgentMessageQueueManager.getInstance().getAutoMode() == 1){
   16:             //System.out.println(">MQEvents:"+name+"-msg="+msgpack.toString())
;
   17:             AgentLogPrint.printAgentLoad(original, name, clonename);
   18:         //}
   19:     }
   20:
   21:     public static void printState(String state, String originalID, String clon
eID, Integer numAgents){
   22:         AgentLogPrint.printAgentState(state, originalID, cloneID, numAgents);
   23:     }
   24: }
```

```
 1: package rda.window;
 2:
 3: import java.util.List;
 4: import java.util.concurrent.CopyOnWriteArrayList;
 5: import rda.data.test.DataTemplate;
 6:
 7: public class Window{
 8:    private String destID;
 9:     private Integer size;
10:     private WindowController manager;
11:     private List win = new CopyOnWriteArrayList();
12:
13:     public Window(WindowController manager, String id, Integer limit) {
14:         this.destID = id;
15:         this.size = limit;
16:         this.manager = manager;
17:     }
18:
19:     public String getDestID(){
20:         return destID;
21:     }
22:
23:     public void setDestID(String id){
24:         destID = id;
25:     }
26:
27:     public void pack(Object obj){
28:         DataTemplate data = (DataTemplate) obj;
29:         win.add(data.getData());
30:
31:         if(win.size() >= size) manager.addExecutable(this);
32:     }
33:
34:     public List unpack(){
35:         return this.win;
36:     }
37:
38:     public Integer getSize(){
39:         return this.win.size();
40:     }
41: }
```

```java
package rda.window;

import java.util.Collection;
import java.util.Map;
import java.util.Queue;
import java.util.concurrent.ConcurrentHashMap;
import java.util.concurrent.ConcurrentLinkedQueue;
import rda.data.test.DataTemplate;
import rda.manager.AgentManager;

public class WindowController{
    private WindowAliveThread aliveThread;
    private Map<String, Window> windowMap = new ConcurrentHashMap<>();
    private Queue executableQueue = new ConcurrentLinkedQueue<>();
    private Integer size;

    public WindowController(int limit, Long aliveTime, int poolsize) {
        this.size = limit;

        //Alive Thread
        this.aliveThread = new WindowAliveThread(this, aliveTime);
        this.aliveThread.start();
    }

    public void pack(Object data){
        String destID = ((DataTemplate)data).toID;

        if(windowMap.get(destID) == null){
            Window window = new Window(this, destID, size);
            windowMap.put(destID, window);
        }

        try{
            windowMap.get(destID).pack(data);
        }catch(NullPointerException e){
            Window window = new Window(this, destID, size);
            windowMap.put(destID, window);
            windowMap.get(destID).pack(data);
        }
    }

    //Hash
    public void pack(AgentManager manager, Object data){
        String destID = "";
        //Window ID Settings
        if(((DataTemplate)data).toID.equals(((DataTemplate)data).id))
            destID = ((DataTemplate)data).toID;
        else destID = ((DataTemplate)data).toID+","+((DataTemplate)data).id;

        if(windowMap.get(destID) == null){
            Window window = new Window(this, destID, size);
            windowMap.put(destID, window);
        }

        try{
            windowMap.get(destID).pack(data);
        }catch(NullPointerException e){
            Window window = new Window(this, destID, size);
            windowMap.put(destID, window);
            windowMap.get(destID).pack(data);
        }
    }

    public Collection getWindows(){
        return windowMap.values();
    }

    public void addExecutable(Window window){
        executableQueue.add(window);
        windowMap.remove(window.getDestID());
        //System.out.println("Window Controller Size = "+executableQueue.size(
));
    }

    public void returnExecutable(Window window){
        if(executableQueue.contains(window))
            remove();
        else
            executableQueue.add(window);
        //System.out.println("Window Controller Size = "+executableQueue.size(
));
    }

    public Window get(){
        return (Window)executableQueue.peek();
    }

    public void remove(){
        executableQueue.poll();
    }

    public boolean check(String id){
        return windowMap.get(id) != null;
    }

    public void close(){
        aliveThread.stop();
    }
}
```

```
    1: package rda.window;
    2:
    3: import java.util.Collection;
    4: import java.util.concurrent.Executors;
    5: import java.util.concurrent.ScheduledExecutorService;
    6: import java.util.concurrent.TimeUnit;
    7:
    8: public class WindowAliveThread implements Runnable{
    9:     private static final String name = "Window Alive Thread";
   10:     private final ScheduledExecutorService schedule = Executors.newSingleThrea
dScheduledExecutor();
   11:
   12:     private WindowController ctrl;
   13:     private Long aliveTime;
   14:     public WindowAliveThread(WindowController ctrl, Long aliveTime) {
   15:         this.ctrl = ctrl;
   16:         this.aliveTime = aliveTime;
   17:     }
   18:
   19:     public void start(){
   20:         System.out.println("> "+name + " : Start !");
   21:         schedule.scheduleAtFixedRate(this, 0, aliveTime, TimeUnit.MILLISECONDS
);
   22:     }
   23:
   24:     public void stop(){
   25:         try {
   26:             schedule.shutdown();
   27:             if(!schedule.awaitTermination(0, TimeUnit.SECONDS))
   28:                 schedule.shutdownNow();
   29:         } catch (InterruptedException ex) {
   30:             schedule.shutdownNow();
   31:         }
   32:
   33:         System.out.println("> " + name +" : Stop !");
   34:     }
   35:
   36:     @Override
   37:     public void run() {
   38:         try{
   39:             Collection collect = ctrl.getWindows();
   40:             if(collect == null) return;
   41:
   42:             for(Window win : (Collection<Window>)collect){
   43:                 ctrl.addExecutable(win);
   44:             }
   45:         }catch(Exception e){
   46:             e.printStackTrace();
   47:         }
   48:     }
   49: }
```

```java
package rda.clone;

import rda.manager.AgentManager;
import rda.manager.IDManager;

public class AgentCloning {
    public static String cloning(AgentManager manager, String sourceID, Object originalState){
        if(!manager.getAutoMode()) return "";

        System.out.println(">> Start Agent Cloning");

        IDManager id = manager.getIDManager();
        String originalID = id.getOrigID(sourceID);

        //Clone
        String cloneID = manager.createCloneAgent(originalID, originalState);
        id.regID(originalID, cloneID);

        System.out.println(">> Agent Cloning New Copy From "+ originalID);

        return cloneID;
    }

    public static String delete(AgentManager manager, String deleteID){
        if(!manager.getAutoMode()) return "";

        System.out.println(">> Start Agent Delete");

        IDManager id = manager.getIDManager();
        String originalID = id.getOrigID(deleteID);

        //Delete
        id.deleteID(originalID, deleteID);

        System.out.println(">> Agent Cloning Delete "+ deleteID);

        return deleteID;
    }
}
```

A.2 ランキングシステム

```
<?xml version="1.0"?>
<agentdef package="rdarank" version="rdarank1.0">
    <entities>
        <!--UserAgent Entity define-->
        <entity type="useragent">
            <attribute name="UserID" type="string" primarykey="true" maxlength="16"/>
            <attribute name="Profile" type="profile" singleentity="true"/>
            <attribute name="Data" type="long" />
            <attribute name="CommunicationID" type="string" maxlength="16"/>
            <attribute name="ConnectionCount" type="long" />
            <attribute name="MessageLatency" type="long" />
            <attribute name="MessageQueueLength" type="long" />
            <attribute name="Log" type="userlog" />
        </entity>
        <entity type="profile">
            <attribute name="UserID" type="string" primarykey="true" relationto="UserID" maxlength="16"
 />
            <attribute name="Name" type="string" maxlength="32"/>
            <attribute name="Sex" type="string" maxlength="2"/>
            <attribute name="Age" type="string" maxlength="4"/>
            <attribute name="Address" type="string" maxlength="64"/>
            <attribute name="LastAccessTime" type="timestamp" />
        </entity>
        <entity type="userlog">
            <attribute name="UserID" type="string" primarykey="true" relationto="UserID" maxlength="16"
 />
            <attribute name="AccessID" type="string" primarykey="true" maxlength="16"/>
            <attribute name="CurrentTime" type="long"/>
            <attribute name="LastAccessTime" type="timestamp" />
        </entity>

        <!--RankAgent Entity define-->
        <entity type="rankagent">
            <attribute name="RankID" type="string" primarykey="true" maxlength="16"/>
            <attribute name="RankTable" type="ranktable" />
            <attribute name="TotalUsers" type="long" />
            <attribute name="ConnectionCount" type="long" />
            <attribute name="MessageLatency" type="long" />
            <attribute name="MessageQueueLength" type="long" />
            <attribute name="Log" type="ranklog" />
        </entity>
        <entity type="ranktable">
            <attribute name="RankID" type="string" primarykey="true" relationto="RankID" maxlength="16"
 />
            <attribute name="UserID" type="string" primarykey="true" maxlength="16"/>
            <attribute name="Rank" type="long" />
            <attribute name="Data" type="long" />
            <attribute name="ConnectionCount" type="long" />
            <attribute name="CurrentTime" type="long"/>
            <attribute name="LastAccessTime" type="timestamp" />
        </entity>
        <entity type="ranklog">
            <attribute name="RankID" type="string" primarykey="true" relationto="RankID" maxlength="16"
 />
            <attribute name="AccessID" type="string" primarykey="true" maxlength="16"/>
            <attribute name="CurrentTime" type="long"/>
            <attribute name="LastAccessTime" type="timestamp" />
        </entity>

        <!--SystemManager Extension define-->
        <entity type="systemmanageragent">
            <attribute name="ID" type="string" primarykey="true"/>
        </entity>

        <!--AgentIntaraction Extension define-->
        <entity type="intaractionmanageragent">
            <attribute name="ID" type="string" primarykey="true"/>
        </entity>
    </entities>
    <messages>
        <!--system define-->
        <!--UserAgent Messages -->
        <message type="initUserAgent" class="rdarank.agent.user.message.InitUserMessage"/>
        <message type="readUserAgent" />
        <message type="updateUserAgent" class="rdarank.agent.user.message.UpdateUserMessage"/>
        <message type="updateCommunicationID" class="rdarank.agent.user.message.UpdateCommunicationIDMe
 ssage"/>
        <message type="readLogUserAgent"/>
        <message type="disposeUserAgent" />
        <message type="intaractionAgent" />


        <!--RankAgent Messages -->
        <message type="initRankAgent" class="rdarank.agent.rank.message.InitRankMessage"/>
        <message type="readRankAgent" />
        <message type="updateRankAgent" class="rdarank.agent.rank.message.UpdateRankMessage"/>
        <message type="updateRanking" class="rdarank.agent.rank.message.UpdateRankingMessage"/>
        <message type="queryRanking" class="rdarank.agent.rank.message.QueryRankingMessage"/>
```

```
084:         <message type="readLogRankAgent"/>
085:         <message type="disposeRankAgent" />
086:
087:         <!--SystemManager Extension define-->
088:         <message type="launchSystem" />
089:         <message type="createAgent" class="rda.extension.manager.message.CreateAgentMessage"/>
090:         <message type="dataGenerate" />
091:         <message type="shutdownSystem" />
092:
093:         <!--AgentIntaraction Extension define-->
094:         <message type="intaractionAgent" />
095:
096:     </messages>
097:     <agents>
098:         <!--system define-->
099:         <agent type="useragent">
100:             <handler message="initUserAgent" class="rdarank.agent.user.handler.InitUserHandler"/>
101:             <handler message="readUserAgent" class="rdarank.agent.user.handler.ReadUserHandler"/>
102:             <handler message="updateUserAgent" class="rdarank.agent.user.handler.UpdateUserHandler">
103:                 <postmessage message="intaractionAgent"/>
104:             </handler>
105:             <handler message="intaractionAgent" class="rdarank.agent.user.handler.IntaractionAgentHandl
    : er"/>
106:             <handler message="updateCommunicationID" class="rdarank.agent.user.handler.UpdateCommunicat
    : ionIDHandler"/>
107:             <handler message="readLogUserAgent" class="rdarank.agent.user.handler.ReadLogUserHandler"/>
108:             <handler message="disposeUserAgent" class="rdarank.agent.user.handler.DisposeUserHandler"/>
109:         </agent>
110:
111:         <!--system define-->
112:         <agent type="rankagent">
113:             <handler message="initRankAgent" class="rdarank.agent.rank.handler.InitRankHandler"/>
114:             <handler message="readRankAgent" class="rdarank.agent.rank.handler.ReadRankHandler"/>
115:             <handler message="updateRankAgent" class="rdarank.agent.rank.handler.UpdateRankHandler"/>
116:             <handler message="updateRanking" class="rdarank.agent.rank.handler.UpdateRankingHandler"/>
117:             <handler message="queryRanking" class="rdarank.agent.rank.handler.QueryRankingHandler"/>
118:             <handler message="readLogRankAgent" class="rdarank.agent.rank.handler.ReadLogRankHandler"/>
119:             <handler message="disposeRankAgent" class="rdarank.agent.rank.handler.DisposeRankHandler"/>
120:         </agent>
121:
122:         <!--SystemManager Extension define-->
123:         <agent type="systemmanageragent">
124:             <handler message="launchSystem" class="rda.extension.manager.handler.LaunchSystemHandler"/>
125:             <handler message="createAgent" class="rda.extension.manager.handler.CreateAgentHandler"/>
126:             <handler message="dataGenerate" class="rda.extension.manager.handler.DataGenerateHandler"/>
127:             <handler message="shutdownSystem" class="rda.extension.manager.handler.ShutdownSystemHandle
    : r"/>
128:         </agent>
129:
130:         <!--AgentIntaraction Extension define-->
131:         <agent type="intaractionmanageragent">
132:             <handler message="intaractionAgent" class="rda.extension.intaraction.handler.IntaractionAge
    : ntHandler"/>
133:         </agent>
134:     </agents>
135: </agentdef>
```

```
 1: package rdarank;
 2:
 3: import com.ibm.agent.exa.AgentException;
 4: import com.ibm.agent.exa.impl.HPAEntity;
 5: import com.ibm.agent.exa.TxID;
 6: import com.ibm.agent.exa.entity.EntitySet;
 7: import com.ibm.agent.exa.entity.EntityKey;
 8: import com.ibm.agent.exa.entity.Entity;
 9: import java.util.Iterator;
10:
11: /**
12:  * Generated code for rankagent.
13:  *
14:  * <p>entity type="rankagent tablename="rankagent <br>
15:  * attribute name="RankID" type="STRING" primarykey="true" <br>
16:  * attribute name="RankTable" type="ranktable" <br>
17:  * attribute name="TotalUsers" type="LONG" <br>
18:  * attribute name="ConnectionCount" type="LONG" <br>
19:  * attribute name="MessageLatency" type="LONG" <br>
20:  * attribute name="MessageQueueLength" type="LONG" <br>
21:  * attribute name="Log" type="ranklog" <br>
22:  **/
23: public class Rankagent extends HPAEntity {
24:     /**
25:     * Primary key size
26:     **/
27:     public static final int PKINDEXSIZE = 1;
28:
29:     /**
30:     * Primary key index of RankID
31:     **/
32:     public static final int PKINDEX_RANKID = 0;
33:
34:     /**
35:     * Column index of RankID
36:     **/
37:     public static final int RANKID = 0;
38:
39:     /**
40:     * Column index of RankTable
41:     **/
42:     public static final int RANKTABLE = 1;
43:
44:     /**
45:     * Column index of TotalUsers
46:     **/
47:     public static final int TOTALUSERS = 2;
48:
49:     /**
50:     * Column index of ConnectionCount
51:     **/
52:     public static final int CONNECTIONCOUNT = 3;
53:
54:     /**
55:     * Column index of MessageLatency
56:     **/
57:     public static final int MESSAGELATENCY = 4;
58:
59:     /**
60:     * Column index of MessageQueueLength
61:     **/
62:     public static final int MESSAGEQUEUELENGTH = 5;
63:
64:     /**
65:     * Column index of Log
66:     **/
67:     public static final int LOG = 6;
68:
69:     /**
70:      * This constructor is used by the runtime.
71:      * An application should not create an instance with this constructor
72:     **/
73:     public Rankagent() {
74:         super();
75:     }
76:
77:     /**
78:      * Get the version string
79:     **/
80:     public String getVersion() {
81:         return "rdarank1.0";
82:     }
83:
84:     /**
85:      * Get a value of RankID.
86:      * The setter method of RankID is not generated because this attribute is
a primarykey.
87:      * @return RankID
88:      **/
89:     public final String getRankID(TxID tx) {
90:         // generated code
91:         return getString(tx,0);
92:     }
93:
94:     /**
95:      * Get a set of RankTable.
96:      * Entity type of this entity set is ranktable.
97:      * The setter method of RankTable is not generated because this attribute
is a EntitySet.
98:      * @return an entity set containing Ranktable
99:      * @throws AgentException
100:      **/
101:     public final EntitySet getRankTable(TxID tx) throws AgentException {
102:         // generated code
103:         return getEntitySet(tx,1);
104:     }
105:
106:     /**
107:      * Get a value of RankTable.
108:      * @param tx a transaction context
109:      * @param UserID
110:      * @return Ranktable
111:      * @throws AgentException
112:      **/
113:     public final Ranktable getRankTable(TxID tx,String UserID) throws AgentExc
eption {
114:         // generated code
115:         EntitySet es = getEntitySet(tx,1);
116:         Entity parent = es.getParent();
117:         Object[] primaryKeys = new Object[]{parent.getObject(tx,0),UserID};
118:         EntityKey ek = new EntityKey("ranktable", primaryKeys);
119:         Ranktable entity = (Ranktable)es.getEntity(ek);
120:         return entity;
121:     }
```

```
   122:
   123:     /**
   124:      * Create a value of Ranktable.
   125:      * @param tx a transaction context
   126:      * @param UserID
   127:      * @return Ranktable
   128:      **/
   129:     public final Ranktable createRankTable(TxID tx,String UserID) throws Agent
Exception {
   130:         // generated code
   131:         if (UserID.length() > 16) {
   132:             throw new AgentException("UserID > maxlength(16)");
   133:         }
   134:         EntitySet es = getEntitySet(tx,1);
   135:         Object[] primaryKeys = new Object[]{null,UserID};
   136:         Ranktable entity = (Ranktable)es.createEntity(tx,primaryKeys);
   137:         return entity;
   138:     }
   139:
   140:     /**
   141:      * Get an iterator of Ranktable.
   142:      * @param tx a transaction context
   143:      **/
   144:     public final Iterator<Entity> getRankTableIterator(TxID tx) throws AgentEx
ception {
   145:         // generated code
   146:         EntitySet es = getEntitySet(tx,1);
   147:         return es.iterator(tx);
   148:     }
   149:
   150:     /**
   151:      * @return TotalUsers
   152:      **/
   153:     public final long getTotalUsers(TxID tx) {
   154:         // generated code
   155:         return getLong(tx,2);
   156:     }
   157:
   158:     /**
   159:      * Set a value to TotalUsers.
   160:      * @param tx a transaction context
   161:      * @param value a value to be set to TotalUsers
   162:      **/
   163:     public final void  setTotalUsers(TxID tx, long value) throws AgentExceptio
n {
   164:         // generated code
   165:         setLong(tx,2,value);
   166:     }
   167:
   168:     /**
   169:      * @return ConnectionCount
   170:      **/
   171:     public final long getConnectionCount(TxID tx) {
   172:         // generated code
   173:         return getLong(tx,3);
   174:     }
   175:
   176:     /**
   177:      * Set a value to ConnectionCount.
   178:      * @param tx a transaction context
   179:      * @param value a value to be set to ConnectionCount
   180:      **/
   181:     public final void  setConnectionCount(TxID tx, long value) throws AgentExc
eption {
   182:         // generated code
   183:         setLong(tx,3,value);
   184:     }
   185:
   186:     /**
   187:      * @return MessageLatency
   188:      **/
   189:     public final long getMessageLatency(TxID tx) {
   190:         // generated code
   191:         return getLong(tx,4);
   192:     }
   193:
   194:     /**
   195:      * Set a value to MessageLatency.
   196:      * @param tx a transaction context
   197:      * @param value a value to be set to MessageLatency
   198:      **/
   199:     public final void  setMessageLatency(TxID tx, long value) throws AgentExce
ption {
   200:         // generated code
   201:         setLong(tx,4,value);
   202:     }
   203:
   204:     /**
   205:      * @return MessageQueueLength
   206:      **/
   207:     public final long getMessageQueueLength(TxID tx) {
   208:         // generated code
   209:         return getLong(tx,5);
   210:     }
   211:
   212:     /**
   213:      * Set a value to MessageQueueLength.
   214:      * @param tx a transaction context
   215:      * @param value a value to be set to MessageQueueLength
   216:      **/
   217:     public final void  setMessageQueueLength(TxID tx, long value) throws Agent
Exception {
   218:         // generated code
   219:         setLong(tx,5,value);
   220:     }
   221:
   222:     /**
   223:      * Get a set of Log.
   224:      * Entity type of this entity set is ranklog.
   225:      * The setter method of Log is not generated because this attribute is a E
ntitySet.
   226:      * @return an entity set containing Ranklog
   227:      * @throws AgentException
   228:      **/
   229:     public final EntitySet getLog(TxID tx) throws AgentException {
   230:         // generated code
   231:         return getEntitySet(tx,6);
   232:     }
   233:
   234:     /**
   235:      * Get a value of Log.
   236:      * @param tx a transaction context
   237:      * @param AccessID
   238:      * @return Ranklog
```

```
  239:      * @throws AgentException
  240:      **/
  241:     public final Ranklog getLog(TxID tx,String AccessID) throws AgentException
 {
  242:         // generated code
  243:         EntitySet es = getEntitySet(tx,6);
  244:         Entity parent = es.getParent();
  245:         Object[] primaryKeys = new Object[]{parent.getObject(tx,0),AccessID};
  246:         EntityKey ek = new EntityKey("ranklog", primaryKeys);
  247:         Ranklog entity = (Ranklog)es.getEntity(ek);
  248:         return entity;
  249:     }
  250:
  251:     /**
  252:      * Create a value of Ranklog.
  253:      * @param tx a transaction context
  254:      * @param AccessID
  255:      * @return Ranklog
  256:      **/
  257:     public final Ranklog createLog(TxID tx,String AccessID) throws AgentExcept
ion {
  258:         // generated code
  259:         if (AccessID.length() > 16) {
  260:             throw new AgentException("AccessID > maxlength(16)");
  261:         }
  262:         EntitySet es = getEntitySet(tx,6);
  263:         Object[] primaryKeys = new Object[]{null,AccessID};
  264:         Ranklog entity = (Ranklog)es.createEntity(tx,primaryKeys);
  265:         return entity;
  266:     }
  267:
  268:     /**
  269:      * Get an iterator of Ranklog.
  270:      * @param tx a transaction context
  271:      **/
  272:     public final Iterator<Entity> getLogIterator(TxID tx) throws AgentExceptio
n {
  273:         // generated code
  274:         EntitySet es = getEntitySet(tx,6);
  275:         return es.iterator(tx);
  276:     }
  277:
  278: }
```

```
   1: package rdarank;
   2:
   3: import com.ibm.agent.exa.AgentException;
   4: import com.ibm.agent.exa.impl.HPAEntity;
   5: import com.ibm.agent.exa.TxID;
   6:
   7: /**
   8:  * Generated code for ranktable.
   9:  *
  10:  * <p>entity type="ranktable tablename="ranktable <br>
  11:  * attribute name="RankID" type="STRING" primarykey="true" relationto="RankID"
<br>
  12:  * attribute name="UserID" type="STRING" primarykey="true" <br>
  13:  * attribute name="Rank" type="LONG" <br>
  14:  * attribute name="Data" type="LONG" <br>
  15:  * attribute name="ConnectionCount" type="LONG" <br>
  16:  * attribute name="CurrentTime" type="LONG" <br>
  17:  * attribute name="LastAccessTime" type="TIMESTAMP" <br>
  18: **/
  19: public class Ranktable extends HPAEntity {
  20:     /**
  21:     * Primary key size
  22:     **/
  23:     public static final int PKINDEXSIZE = 2;
  24:
  25:     /**
  26:     * Primary key index of RankID
  27:     **/
  28:     public static final int PKINDEX_RANKID = 0;
  29:
  30:     /**
  31:     * Primary key index of UserID
  32:     **/
  33:     public static final int PKINDEX_USERID = 1;
  34:
  35:     /**
  36:     * Column index of RankID
  37:     **/
  38:     public static final int RANKID = 0;
  39:
  40:     /**
  41:     * Column index of UserID
  42:     **/
  43:     public static final int USERID = 1;
  44:
  45:     /**
  46:     * Column index of Rank
  47:     **/
  48:     public static final int RANK = 2;
  49:
  50:     /**
  51:     * Column index of Data
  52:     **/
  53:     public static final int DATA = 3;
  54:
  55:     /**
  56:     * Column index of ConnectionCount
  57:     **/
  58:     public static final int CONNECTIONCOUNT = 4;
  59:
  60:     /**
  61:     * Column index of CurrentTime
  62:     **/
  63:     public static final int CURRENTTIME = 5;
  64:
  65:     /**
  66:     * Column index of LastAccessTime
  67:     **/
  68:     public static final int LASTACCESSTIME = 6;
  69:
  70:     /**
  71:      * This constructor is used by the runtime.
  72:      * An application should not create an instance with this constructor
  73:     **/
  74:     public Ranktable() {
  75:         super();
  76:     }
  77:
  78:     /**
  79:      * Get the version string
  80:     **/
  81:     public String getVersion() {
  82:         return "rdarank1.0";
  83:     }
  84:
  85:     /**
  86:      * Get a value of RankID.
  87:      * The setter method of RankID is not generated because this attribute is
a primarykey.
  88:      * @return RankID
  89:      **/
  90:     public final String getRankID(TxID tx) {
  91:         // generated code
  92:         return getString(tx,0);
  93:     }
  94:
  95:     /**
  96:      * Get a value of UserID.
  97:      * The setter method of UserID is not generated because this attribute is
a primarykey.
  98:      * @return UserID
  99:      **/
 100:     public final String getUserID(TxID tx) {
 101:         // generated code
 102:         return getString(tx,1);
 103:     }
 104:
 105:     /**
 106:      * @return Rank
 107:      **/
 108:     public final long getRank(TxID tx) {
 109:         // generated code
 110:         return getLong(tx,2);
 111:     }
 112:
 113:     /**
 114:      * Set a value to Rank.
 115:      * @param tx a transaction context
 116:      * @param value a value to be set to Rank
 117:      **/
 118:     public final void  setRank(TxID tx, long value) throws AgentException {
 119:         // generated code
 120:         setLong(tx,2,value);
 121:     }
```

```
   122:
   123:     /**
   124:      * @return Data
   125:      **/
   126:     public final long getData(TxID tx) {
   127:         // generated code
   128:         return getLong(tx,3);
   129:     }
   130:
   131:     /**
   132:      * Set a value to Data.
   133:      * @param tx a transaction context
   134:      * @param value a value to be set to Data
   135:      **/
   136:     public final void  setData(TxID tx, long value) throws AgentException {
   137:         // generated code
   138:         setLong(tx,3,value);
   139:     }
   140:
   141:     /**
   142:      * @return ConnectionCount
   143:      **/
   144:     public final long getConnectionCount(TxID tx) {
   145:         // generated code
   146:         return getLong(tx,4);
   147:     }
   148:
   149:     /**
   150:      * Set a value to ConnectionCount.
   151:      * @param tx a transaction context
   152:      * @param value a value to be set to ConnectionCount
   153:      **/
   154:     public final void  setConnectionCount(TxID tx, long value) throws AgentExc
eption {
   155:         // generated code
   156:         setLong(tx,4,value);
   157:     }
   158:
   159:     /**
   160:      * @return CurrentTime
   161:      **/
   162:     public final long getCurrentTime(TxID tx) {
   163:         // generated code
   164:         return getLong(tx,5);
   165:     }
   166:
   167:     /**
   168:      * Set a value to CurrentTime.
   169:      * @param tx a transaction context
   170:      * @param value a value to be set to CurrentTime
   171:      **/
   172:     public final void  setCurrentTime(TxID tx, long value) throws AgentExcepti
on {
   173:         // generated code
   174:         setLong(tx,5,value);
   175:     }
   176:
   177:     /**
   178:      * @return LastAccessTime
   179:      **/
   180:     public final java.sql.Timestamp getLastAccessTime(TxID tx) {
   181:         // generated code
   182:         return getTimestamp(tx,6);
   183:     }
   184:
   185:     /**
   186:      * Set a value to LastAccessTime.
   187:      * @param tx a transaction context
   188:      * @param value a value to be set to LastAccessTime
   189:      **/
   190:     public final void  setLastAccessTime(TxID tx, java.sql.Timestamp value) th
rows AgentException {
   191:         // generated code
   192:         setTimestamp(tx,6,value);
   193:     }
   194:
   195: }
```

```
   1: package rdarank;
   2:
   3: import com.ibm.agent.exa.AgentException;
   4: import com.ibm.agent.exa.impl.HPAEntity;
   5: import com.ibm.agent.exa.TxID;
   6:
   7: /**
   8:  * Generated code for ranklog.
   9:  *
  10:  * <p>entity type="ranklog tablename="ranklog <br>
  11:  * attribute name="RankID" type="STRING" primarykey="true" relationto="RankID"
<br>
  12:  * attribute name="AccessID" type="STRING" primarykey="true" <br>
  13:  * attribute name="CurrentTime" type="LONG" <br>
  14:  * attribute name="LastAccessTime" type="TIMESTAMP" <br>
  15: **/
  16: public class Ranklog extends HPAEntity {
  17:     /**
  18:     * Primary key size
  19:     **/
  20:     public static final int PKINDEXSIZE = 2;
  21:
  22:     /**
  23:     * Primary key index of RankID
  24:     **/
  25:     public static final int PKINDEX_RANKID = 0;
  26:
  27:     /**
  28:     * Primary key index of AccessID
  29:     **/
  30:     public static final int PKINDEX_ACCESSID = 1;
  31:
  32:     /**
  33:     * Column index of RankID
  34:     **/
  35:     public static final int RANKID = 0;
  36:
  37:     /**
  38:     * Column index of AccessID
  39:     **/
  40:     public static final int ACCESSID = 1;
  41:
  42:     /**
  43:     * Column index of CurrentTime
  44:     **/
  45:     public static final int CURRENTTIME = 2;
  46:
  47:     /**
  48:     * Column index of LastAccessTime
  49:     **/
  50:     public static final int LASTACCESSTIME = 3;
  51:
  52:     /**
  53:      * This constructor is used by the runtime.
  54:      * An application should not create an instance with this constructor
  55:     **/
  56:     public Ranklog() {
  57:         super();
  58:     }
  59:
  60:     /**
  61:      * Get the version string
  62:     **/
  63:     public String getVersion() {
  64:         return "rdarank1.0";
  65:     }
  66:
  67:     /**
  68:      * Get a value of RankID.
  69:      * The setter method of RankID is not generated because this attribute is
a primarykey.
  70:      * @return RankID
  71:      **/
  72:     public final String getRankID(TxID tx) {
  73:         // generated code
  74:         return getString(tx,0);
  75:     }
  76:
  77:     /**
  78:      * Get a value of AccessID.
  79:      * The setter method of AccessID is not generated because this attribute i
s a primarykey.
  80:      * @return AccessID
  81:      **/
  82:     public final String getAccessID(TxID tx) {
  83:         // generated code
  84:         return getString(tx,1);
  85:     }
  86:
  87:     /**
  88:      * @return CurrentTime
  89:      **/
  90:     public final long getCurrentTime(TxID tx) {
  91:         // generated code
  92:         return getLong(tx,2);
  93:     }
  94:
  95:     /**
  96:      * Set a value to CurrentTime.
  97:      * @param tx a transaction context
  98:      * @param value a value to be set to CurrentTime
  99:      **/
 100:     public final void  setCurrentTime(TxID tx, long value) throws AgentExcepti
on {
 101:         // generated code
 102:         setLong(tx,2,value);
 103:     }
 104:
 105:     /**
 106:      * @return LastAccessTime
 107:      **/
 108:     public final java.sql.Timestamp getLastAccessTime(TxID tx) {
 109:         // generated code
 110:         return getTimestamp(tx,3);
 111:     }
 112:
 113:     /**
 114:      * Set a value to LastAccessTime.
 115:      * @param tx a transaction context
 116:      * @param value a value to be set to LastAccessTime
 117:      **/
 118:     public final void  setLastAccessTime(TxID tx, java.sql.Timestamp value) th
rows AgentException {
 119:         // generated code
```

```
120:         setTimestamp(tx,3,value);
121:     }
122:
123: }
```

```java
package rdarank;

import com.ibm.agent.exa.AgentException;
import com.ibm.agent.exa.impl.HPAEntity;
import com.ibm.agent.exa.TxID;
import com.ibm.agent.exa.entity.EntitySet;
import com.ibm.agent.exa.entity.EntityKey;
import com.ibm.agent.exa.entity.Entity;
import java.util.Iterator;

/**
 * Generated code for useragent.
 *
 * <p>entity type="useragent tablename="useragent <br>
 * attribute name="UserID" type="STRING" primarykey="true" <br>
 * attribute name="Profile" type="profile" <br>
 * attribute name="Data" type="LONG" <br>
 * attribute name="CommunicationID" type="STRING" <br>
 * attribute name="ConnectionCount" type="LONG" <br>
 * attribute name="MessageLatency" type="LONG" <br>
 * attribute name="MessageQueueLength" type="LONG" <br>
 * attribute name="Log" type="userlog" <br>
 **/
public class Useragent extends HPAEntity {
    /**
     * Primary key size
     **/
    public static final int PKINDEXSIZE = 1;

    /**
     * Primary key index of UserID
     **/
    public static final int PKINDEX_USERID = 0;

    /**
     * Column index of UserID
     **/
    public static final int USERID = 0;

    /**
     * Column index of Profile
     **/
    public static final int PROFILE = 1;

    /**
     * Column index of Data
     **/
    public static final int DATA = 2;

    /**
     * Column index of CommunicationID
     **/
    public static final int COMMUNICATIONID = 3;

    /**
     * Column index of ConnectionCount
     **/
    public static final int CONNECTIONCOUNT = 4;

    /**
     * Column index of MessageLatency
     **/
    public static final int MESSAGELATENCY = 5;

    /**
     * Column index of MessageQueueLength
     **/
    public static final int MESSAGEQUEUELENGTH = 6;

    /**
     * Column index of Log
     **/
    public static final int LOG = 7;

    /**
     * This constructor is used by the runtime.
     * An application should not create an instance with this constructor
     **/
    public Useragent() {
        super();
    }

    /**
     * Get the version string
     **/
    public String getVersion() {
        return "rdarank1.0";
    }

    /**
     * Get a value of UserID.
     * The setter method of UserID is not generated because this attribute is a primarykey.
     * @return UserID
     **/
    public final String getUserID(TxID tx) {
        // generated code
        return getString(tx,0);
    }

    /**
     * Get a set of Profile.
     * Entity type of this entity set is profile.
     * A returned entity set has a single entity.
     * The setter method of Profile is not generated because this attribute is a EntitySet.
     * @return an entity set containing Profile
     * @throws AgentException
     **/
    public final EntitySet getProfileSet(TxID tx) throws AgentException {
        // generated code
        return getEntitySet(tx,1);
    }

    /**
     * Get a value of Profile.
     * @param tx a transaction context
     * @return Profile
     * @throws AgentException
     **/
    public final Profile getProfile(TxID tx) throws AgentException {
        // generated code
        EntitySet es = getEntitySet(tx,1);
        if (es == null) return null;
```

```
123:         return (Profile)es.getSingleEntity();
124:     }
125:
126:     /**
127:      * Create a value of Profile.
128:      * @param tx a transaction context
129:      * @return Profile
130:      **/
131:     public final Profile createProfile(TxID tx) throws AgentException {
132:         // generated code
133:         EntitySet es = getEntitySet(tx,1);
134:         Profile entity = (Profile)es.createEntity(tx,new Object[1]);
135:         return entity;
136:     }
137:
138:     /**
139:      * @return Data
140:      **/
141:     public final long getData(TxID tx) {
142:         // generated code
143:         return getLong(tx,2);
144:     }
145:
146:     /**
147:      * Set a value to Data.
148:      * @param tx a transaction context
149:      * @param value a value to be set to Data
150:      **/
151:     public final void  setData(TxID tx, long value) throws AgentException {
152:         // generated code
153:         setLong(tx,2,value);
154:     }
155:
156:     /**
157:      * @return CommunicationID
158:      **/
159:     public final String getCommunicationID(TxID tx) {
160:         // generated code
161:         return getString(tx,3);
162:     }
163:
164:     /**
165:      * Set a value to CommunicationID.
166:      * @param tx a transaction context
167:      * @param value a value to be set to CommunicationID
168:      **/
169:     public final void  setCommunicationID(TxID tx, String value) throws AgentE
xception {
170:         // generated code
171:         if (value != null && value.length() > 16) {
172:             throw new AgentException("CommunicationID > maxlength(16)");
173:         }
174:         setString(tx,3,value);
175:     }
176:
177:     /**
178:      * @return ConnectionCount
179:      **/
180:     public final long getConnectionCount(TxID tx) {
181:         // generated code
182:         return getLong(tx,4);
183:     }
184:
185:     /**
186:      * Set a value to ConnectionCount.
187:      * @param tx a transaction context
188:      * @param value a value to be set to ConnectionCount
189:      **/
190:     public final void  setConnectionCount(TxID tx, long value) throws AgentExc
eption {
191:         // generated code
192:         setLong(tx,4,value);
193:     }
194:
195:     /**
196:      * @return MessageLatency
197:      **/
198:     public final long getMessageLatency(TxID tx) {
199:         // generated code
200:         return getLong(tx,5);
201:     }
202:
203:     /**
204:      * Set a value to MessageLatency.
205:      * @param tx a transaction context
206:      * @param value a value to be set to MessageLatency
207:      **/
208:     public final void  setMessageLatency(TxID tx, long value) throws AgentExce
ption {
209:         // generated code
210:         setLong(tx,5,value);
211:     }
212:
213:     /**
214:      * @return MessageQueueLength
215:      **/
216:     public final long getMessageQueueLength(TxID tx) {
217:         // generated code
218:         return getLong(tx,6);
219:     }
220:
221:     /**
222:      * Set a value to MessageQueueLength.
223:      * @param tx a transaction context
224:      * @param value a value to be set to MessageQueueLength
225:      **/
226:     public final void  setMessageQueueLength(TxID tx, long value) throws Agent
Exception {
227:         // generated code
228:         setLong(tx,6,value);
229:     }
230:
231:     /**
232:      * Get a set of Log.
233:      * Entity type of this entity set is userlog.
234:      * The setter method of Log is not generated because this attribute is a E
ntitySet.
235:      * @return an entity set containing Userlog
236:      * @throws AgentException
237:      **/
238:     public final EntitySet getLog(TxID tx) throws AgentException {
239:         // generated code
240:         return getEntitySet(tx,7);
241:     }
```

```
  242:
  243:     /**
  244:      * Get a value of Log.
  245:      * @param tx a transaction context
  246:      * @param AccessID
  247:      * @return Userlog
  248:      * @throws AgentException
  249:      **/
  250:     public final Userlog getLog(TxID tx,String AccessID) throws AgentException
 {
  251:         // generated code
  252:         EntitySet es = getEntitySet(tx,7);
  253:         Entity parent = es.getParent();
  254:         Object[] primaryKeys = new Object[]{parent.getObject(tx,0),AccessID};
  255:         EntityKey ek = new EntityKey("userlog", primaryKeys);
  256:         Userlog entity = (Userlog)es.getEntity(ek);
  257:         return entity;
  258:     }
  259:
  260:     /**
  261:      * Create a value of Userlog.
  262:      * @param tx a transaction context
  263:      * @param AccessID
  264:      * @return Userlog
  265:      **/
  266:     public final Userlog createLog(TxID tx,String AccessID) throws AgentExcept
ion {
  267:         // generated code
  268:         if (AccessID.length() > 16) {
  269:             throw new AgentException("AccessID > maxlength(16)");
  270:         }
  271:         EntitySet es = getEntitySet(tx,7);
  272:         Object[] primaryKeys = new Object[]{null,AccessID};
  273:         Userlog entity = (Userlog)es.createEntity(tx,primaryKeys);
  274:         return entity;
  275:     }
  276:
  277:     /**
  278:      * Get an iterator of Userlog.
  279:      * @param tx a transaction context
  280:      **/
  281:     public final Iterator<Entity> getLogIterator(TxID tx) throws AgentExceptio
n {
  282:         // generated code
  283:         EntitySet es = getEntitySet(tx,7);
  284:         return es.iterator(tx);
  285:     }
  286:
  287: }
```

```
   1: package rdarank;
   2:
   3: import com.ibm.agent.exa.AgentException;
   4: import com.ibm.agent.exa.impl.HPAEntity;
   5: import com.ibm.agent.exa.TxID;
   6:
   7: /**
   8:  * Generated code for profile.
   9:  *
  10:  * <p>entity type="profile tablename="profile <br>
  11:  * attribute name="UserID" type="STRING" primarykey="true" relationto="UserID"
<br>
  12:  * attribute name="Name" type="STRING" <br>
  13:  * attribute name="Sex" type="STRING" <br>
  14:  * attribute name="Age" type="STRING" <br>
  15:  * attribute name="Address" type="STRING" <br>
  16:  * attribute name="LastAccessTime" type="TIMESTAMP" <br>
  17: **/
  18: public class Profile extends HPAEntity {
  19:     /**
  20:     * Primary key size
  21:     **/
  22:     public static final int PKINDEXSIZE = 1;
  23:
  24:     /**
  25:     * Primary key index of UserID
  26:     **/
  27:     public static final int PKINDEX_USERID = 0;
  28:
  29:     /**
  30:     * Column index of UserID
  31:     **/
  32:     public static final int USERID = 0;
  33:
  34:     /**
  35:     * Column index of Name
  36:     **/
  37:     public static final int NAME = 1;
  38:
  39:     /**
  40:     * Column index of Sex
  41:     **/
  42:     public static final int SEX = 2;
  43:
  44:     /**
  45:     * Column index of Age
  46:     **/
  47:     public static final int AGE = 3;
  48:
  49:     /**
  50:     * Column index of Address
  51:     **/
  52:     public static final int ADDRESS = 4;
  53:
  54:     /**
  55:     * Column index of LastAccessTime
  56:     **/
  57:     public static final int LASTACCESSTIME = 5;
  58:
  59:     /**
  60:      * This constructor is used by the runtime.
  61:      * An application should not create an instance with this constructor
  62:      **/
  63:     public Profile() {
  64:         super();
  65:     }
  66:
  67:     /**
  68:      * Get the version string
  69:      **/
  70:     public String getVersion() {
  71:         return "rdarank1.0";
  72:     }
  73:
  74:     /**
  75:      * Get a value of UserID.
  76:      * The setter method of UserID is not generated because this attribute is
a primarykey.
  77:      * @return UserID
  78:      **/
  79:     public final String getUserID(TxID tx) {
  80:         // generated code
  81:         return getString(tx,0);
  82:     }
  83:
  84:     /**
  85:      * @return Name
  86:      **/
  87:     public final String getName(TxID tx) {
  88:         // generated code
  89:         return getString(tx,1);
  90:     }
  91:
  92:     /**
  93:      * Set a value to Name.
  94:      * @param tx a transaction context
  95:      * @param value a value to be set to Name
  96:      **/
  97:     public final void  setName(TxID tx, String value) throws AgentException {
  98:         // generated code
  99:         if (value != null && value.length() > 32) {
 100:             throw new AgentException("Name > maxlength(32)");
 101:         }
 102:         setString(tx,1,value);
 103:     }
 104:
 105:     /**
 106:      * @return Sex
 107:      **/
 108:     public final String getSex(TxID tx) {
 109:         // generated code
 110:         return getString(tx,2);
 111:     }
 112:
 113:     /**
 114:      * Set a value to Sex.
 115:      * @param tx a transaction context
 116:      * @param value a value to be set to Sex
 117:      **/
 118:     public final void  setSex(TxID tx, String value) throws AgentException {
 119:         // generated code
 120:         if (value != null && value.length() > 2) {
 121:             throw new AgentException("Sex > maxlength(2)");
 122:         }
```

```
123:         setString(tx,2,value);
124:     }
125:
126:     /**
127:      * @return Age
128:      **/
129:     public final String getAge(TxID tx) {
130:         // generated code
131:         return getString(tx,3);
132:     }
133:
134:     /**
135:      * Set a value to Age.
136:      * @param tx a transaction context
137:      * @param value a value to be set to Age
138:      **/
139:     public final void  setAge(TxID tx, String value) throws AgentException {
140:         // generated code
141:         if (value != null && value.length() > 4) {
142:             throw new AgentException("Age > maxlength(4)");
143:         }
144:         setString(tx,3,value);
145:     }
146:
147:     /**
148:      * @return Address
149:      **/
150:     public final String getAddress(TxID tx) {
151:         // generated code
152:         return getString(tx,4);
153:     }
154:
155:     /**
156:      * Set a value to Address.
157:      * @param tx a transaction context
158:      * @param value a value to be set to Address
159:      **/
160:     public final void  setAddress(TxID tx, String value) throws AgentException
 {
161:         // generated code
162:         if (value != null && value.length() > 64) {
163:             throw new AgentException("Address > maxlength(64)");
164:         }
165:         setString(tx,4,value);
166:     }
167:
168:     /**
169:      * @return LastAccessTime
170:      **/
171:     public final java.sql.Timestamp getLastAccessTime(TxID tx) {
172:         // generated code
173:         return getTimestamp(tx,5);
174:     }
175:
176:     /**
177:      * Set a value to LastAccessTime.
178:      * @param tx a transaction context
179:      * @param value a value to be set to LastAccessTime
180:      **/
181:     public final void  setLastAccessTime(TxID tx, java.sql.Timestamp value) th
rows AgentException {
182:         // generated code
183:         setTimestamp(tx,5,value);
184:     }
185:
186: }
```

```
    1: package rdarank;
    2:
    3: import com.ibm.agent.exa.AgentException;
    4: import com.ibm.agent.exa.impl.HPAEntity;
    5: import com.ibm.agent.exa.TxID;
    6:
    7: /**
    8:  * Generated code for userlog.
    9:  *
   10:  * <p>entity type="userlog tablename="userlog <br>
   11:  * attribute name="UserID" type="STRING" primarykey="true" relationto="UserID"
<br>
   12:  * attribute name="AccessID" type="STRING" primarykey="true" <br>
   13:  * attribute name="CurrentTime" type="LONG" <br>
   14:  * attribute name="LastAccessTime" type="TIMESTAMP" <br>
   15: **/
   16: public class Userlog extends HPAEntity {
   17:     /**
   18:     * Primary key size
   19:     **/
   20:     public static final int PKINDEXSIZE = 2;
   21:
   22:     /**
   23:     * Primary key index of UserID
   24:     **/
   25:     public static final int PKINDEX_USERID = 0;
   26:
   27:     /**
   28:     * Primary key index of AccessID
   29:     **/
   30:     public static final int PKINDEX_ACCESSID = 1;
   31:
   32:     /**
   33:     * Column index of UserID
   34:     **/
   35:     public static final int USERID = 0;
   36:
   37:     /**
   38:     * Column index of AccessID
   39:     **/
   40:     public static final int ACCESSID = 1;
   41:
   42:     /**
   43:     * Column index of CurrentTime
   44:     **/
   45:     public static final int CURRENTTIME = 2;
   46:
   47:     /**
   48:     * Column index of LastAccessTime
   49:     **/
   50:     public static final int LASTACCESSTIME = 3;
   51:
   52:     /**
   53:      * This constructor is used by the runtime.
   54:      * An application should not create an instance with this constructor
   55:     **/
   56:     public Userlog() {
   57:         super();
   58:     }
   59:
   60:     /**
   61:      * Get the version string
   62:     **/
   63:     public String getVersion() {
   64:         return "rdarank1.0";
   65:     }
   66:
   67:     /**
   68:      * Get a value of UserID.
   69:      * The setter method of UserID is not generated because this attribute is
a primarykey.
   70:      * @return UserID
   71:      **/
   72:     public final String getUserID(TxID tx) {
   73:         // generated code
   74:         return getString(tx,0);
   75:     }
   76:
   77:     /**
   78:      * Get a value of AccessID.
   79:      * The setter method of AccessID is not generated because this attribute i
s a primarykey.
   80:      * @return AccessID
   81:      **/
   82:     public final String getAccessID(TxID tx) {
   83:         // generated code
   84:         return getString(tx,1);
   85:     }
   86:
   87:     /**
   88:      * @return CurrentTime
   89:      **/
   90:     public final long getCurrentTime(TxID tx) {
   91:         // generated code
   92:         return getLong(tx,2);
   93:     }
   94:
   95:     /**
   96:      * Set a value to CurrentTime.
   97:      * @param tx a transaction context
   98:      * @param value a value to be set to CurrentTime
   99:      **/
  100:     public final void  setCurrentTime(TxID tx, long value) throws AgentExcepti
on {
  101:         // generated code
  102:         setLong(tx,2,value);
  103:     }
  104:
  105:     /**
  106:      * @return LastAccessTime
  107:      **/
  108:     public final java.sql.Timestamp getLastAccessTime(TxID tx) {
  109:         // generated code
  110:         return getTimestamp(tx,3);
  111:     }
  112:
  113:     /**
  114:      * Set a value to LastAccessTime.
  115:      * @param tx a transaction context
  116:      * @param value a value to be set to LastAccessTime
  117:      **/
  118:     public final void  setLastAccessTime(TxID tx, java.sql.Timestamp value) th
rows AgentException {
  119:         // generated code
```

```
120:         setTimestamp(tx,3,value);
121:     }
122:
123: }
```

```
    1: package rdarank.agent.rank.handler;
    2:
    3: import java.sql.Timestamp;
    4:
    5: import com.ibm.agent.exa.Message;
    6: import com.ibm.agent.exa.MessageHandler;
    7: import com.ibm.agent.exa.TxID;
    8: import rdarank.Rankagent;
    9: import rdarank.Ranklog;
   10: import rdarank.agent.rank.message.InitRankMessage;
   11:
   12: public class InitRankHandler extends MessageHandler {
   13:
   14:     @Override
   15:     public Object onMessage(Message msg) throws Exception {
   16:         try {
   17:             InitRankMessage initMsg = (InitRankMessage) msg;
   18:
   19:             Rankagent rank = (Rankagent) getEntity();
   20:
   21:             TxID tx = getTx();
   22:
   23:             rank.setTotalUsers(tx, 0);
   24:
   25:             rank.setConnectionCount(tx, 0L);
   26:
   27:             Long time = System.currentTimeMillis();
   28:             Timestamp registerTime = new Timestamp(time);
   29:
   30:             Ranklog log = rank.createLog(tx, "init");
   31:
   32:             log.setLastAccessTime(tx, registerTime);
   33:             log.setCurrentTime(tx, time);
   34:
   35:             //System.out.println("InitHandler of Agent:" + getAgentKey() + " w
as initialized");
   36:             return "hello from agent:" + getAgentKey();
   37:         } catch (Exception e) {
   38:             throw e;
   39:         }
   40:     }
   41: }
```

```
    1: package rdarank.agent.rank.handler;
    2:
    3: import java.sql.Timestamp;
    4:
    5: import com.ibm.agent.exa.Message;
    6: import com.ibm.agent.exa.MessageHandler;
    7: import com.ibm.agent.exa.TxID;
    8: import com.ibm.agent.exa.entity.Entity;
    9: import java.util.HashMap;
   10: import java.util.Iterator;
   11: import java.util.Map;
   12: import rda.agent.queue.MessageObject;
   13: import rdarank.Rankagent;
   14: import rdarank.Ranklog;
   15: import rdarank.Ranktable;
   16: import rdarank.agent.rank.message.UpdateRankMessage;
   17:
   18: public class UpdateRankHandler extends MessageHandler {
   19:
   20:     @Override
   21:     public Object onMessage(Message msg) throws Exception {
   22:         UpdateRankMessage updateMsg = (UpdateRankMessage) msg;
   23:         MessageObject msgObj = (MessageObject) updateMsg.messageData;
   24:
   25:         Rankagent agent = (Rankagent) getEntity();
   26:
   27:         TxID tx = getTx();
   28:
   29:         Map tableMap = new HashMap();
   30:         for (Object data : msgObj.data) {
   31:             Map user = (Map) data;
   32:             tableMap.putAll(user);
   33:         }
   34:
   35:         Long time = System.currentTimeMillis();
   36:         Timestamp updateTime = new Timestamp(time);
   37:
   38:         for (Object id : tableMap.keySet()) {
   39:             Iterator<Entity> it = agent.getRankTableIterator(tx);
   40:             Integer rank = 1;
   41:             while(it.hasNext()){
   42:                 Ranktable table = (Ranktable) it.next();
   43:                 if((table.getData(tx) > (Long)tableMap.get(id)) && table.getUs
erID(tx) != id)
   44:                     rank = rank + 1;
   45:             }
   46:
   47:             Ranktable table = agent.getRankTable(tx, (String) id);
   48:             long count = 1;
   49:             if (table == null) {
   50:                 table = agent.createRankTable(tx, (String) id);
   51:                 agent.setTotalUsers(tx, agent.getTotalUsers(tx) + 1);
   52:             } else {
   53:                 count = table.getConnectionCount(tx) + 1;
   54:             }
   55:
   56:             if((Long)tableMap.get(id) > table.getData(tx)){
   57:                 //Update Data
   58:                 table.setData(tx, (Long) tableMap.get(id));
   59:
   60:                 //Update Ranking
   61:                 table.setRank(tx, rank);
   62:             }
   63:
   64:             //Table Status
   65:             table.setConnectionCount(tx, count);
   66:             table.setCurrentTime(tx, time);
   67:             table.setLastAccessTime(tx, updateTime);
   68:         }
   69:
   70:         //Agent Status
   71:         //Connection
   72:         agent.setConnectionCount(tx, agent.getConnectionCount(tx) + 1);
   73:         //Message Latency
   74:         agent.setMessageLatency(tx, msgObj.latency());
   75:         //Agent Message Queue
   76:         //agent.setMessageQueueLength(tx, msgObj.getLength());
   77:
   78:         // Update Log Records
   79:         Ranklog log = agent.getLog(tx, "update");
   80:         if (log == null) {
   81:             log = agent.createLog(tx, "update");
   82:         }
   83:
   84:         // Update LastAccessTime
   85:         log.setLastAccessTime(tx, updateTime);
   86:         log.setCurrentTime(tx, time);
   87:
   88:         //Long message = avgLatency;
   89:         return 0L;
   90:     }
   91: }
```

```
    1: package rdarank.agent.rank.handler;
    2:
    3: import com.ibm.agent.exa.Message;
    4: import com.ibm.agent.exa.MessageHandler;
    5: import com.ibm.agent.exa.TxID;
    6: import java.util.Iterator;
    7: import rdarank.Rankagent;
    8: import rdarank.Ranktable;
    9: import rdarank.agent.rank.message.UpdateRankingMessage;
   10:
   11: public class UpdateRankingHandler extends MessageHandler {
   12:
   13:     @Override
   14:     public Object onMessage(Message msg) throws Exception {
   15:         UpdateRankingMessage updateMsg = (UpdateRankingMessage) msg;
   16:
   17:         Rankagent agent = (Rankagent) getEntity();
   18:
   19:         TxID tx = getTx();
   20:
   21:         Iterator it = agent.getRankTableIterator(tx);
   22:         while(it.hasNext()){
   23:             Ranktable table = (Ranktable) it.next();
   24:             Iterator subit = agent.getRankTableIterator(tx);
   25:             Long rank = 1L;
   26:             while(subit.hasNext()){
   27:                 Ranktable subtable = (Ranktable) subit.next();
   28:                 if(table.getUserID(tx).equals(subtable.getUserID(tx))) continu
e;
   29:                 if(table.getData(tx) < subtable.getData(tx)) rank++;
   30:             }
   31:             table.setRank(tx, rank);
   32:         }
   33:
   34:         return 0L;
   35:     }
   36:
   37: }
```

```
    1: package rdarank.agent.rank.handler;
    2:
    3: import com.ibm.agent.exa.Message;
    4: import com.ibm.agent.exa.MessageHandler;
    5: import com.ibm.agent.exa.TxID;
    6: import com.ibm.agent.exa.entity.Entity;
    7: import java.util.HashMap;
    8: import java.util.Iterator;
    9: import rdarank.Rankagent;
   10: import rdarank.Ranktable;
   11: import rdarank.agent.rank.message.QueryRankingMessage;
   12:
   13: public class QueryRankingHandler extends MessageHandler {
   14:
   15:     @Override
   16:     public Object onMessage(Message msg) throws Exception {
   17:         QueryRankingMessage queryMsg = (QueryRankingMessage) msg;
   18:
   19:         Rankagent agent = (Rankagent) getEntity();
   20:
   21:         TxID tx = getTx();
   22:
   23:         Ranktable table = agent.getRankTable(tx, queryMsg.id);
   24:         if(queryMsg.type.equals("")){
   25:             return table.getRank(tx);
   26:         }else if(queryMsg.type.equals("delete")){
   27:             table.remove(tx);
   28:         }else if(queryMsg.type.equals("sync")){
   29:             HashMap map = new HashMap();
   30:             map.put(table.getUserID(tx), table.getLastAccessTime(tx)+","+table
.getRank(tx));
   31:             long rank = table.getRank(tx);
   32:
   33:             Iterator<Entity> it = agent.getRankTableIterator(tx);
   34:             while(it.hasNext()){
   35:                 table = (Ranktable) it.next();
   36:                 if((table.getRank(tx)) > rank)
   37:                     map.put(table.getUserID(tx), table.getLastAccessTime(tx)+"
,"+table.getRank(tx));
   38:             }
   39:
   40:             return map;
   41:         }
   42:
   43:         return null;
   44:     }
   45:
   46: }
```

rdarank/agent/rank/handler/ReadRankHandler.java


```
 1: package rdarank.agent.rank.handler;
 2:
 3: import com.ibm.agent.exa.Message;
 4: import com.ibm.agent.exa.MessageHandler;
 5: import com.ibm.agent.exa.TxID;
 6: import rdarank.Rankagent;
 7:
 8: public class ReadRankHandler extends MessageHandler {
 9:
10:     @Override
11:     public Object onMessage(Message arg0) throws Exception {
12:         Rankagent rank = (Rankagent) getEntity();
13:
14:         TxID tx = getTx();
15:
16:         return rank.getRankID(tx);
17:     }
18:
19: }
```

```java
package rdarank.agent.rank.handler;

import java.util.Iterator;

import com.ibm.agent.exa.Message;
import com.ibm.agent.exa.MessageHandler;
import com.ibm.agent.exa.TxID;
import com.ibm.agent.exa.entity.Entity;
import rdarank.Rankagent;
import rdarank.Ranklog;
import rdarank.agent.rank.logger.LogInfo;

public class ReadLogRankHandler extends MessageHandler {

    @Override
    public Object onMessage(Message msg) throws Exception {
        Rankagent rank = (Rankagent) getEntity();

        TxID tx = getTx();

        StringBuilder accessIDList = new StringBuilder();
        StringBuilder accessTimeList = new StringBuilder();
        Iterator<Entity> it = rank.getLogIterator(tx);
        while (it.hasNext()) {
            Ranklog log = (Ranklog) it.next();

            accessIDList.append(log.getAccessID(tx));
            accessIDList.append(",");

            accessTimeList.append(log.getLastAccessTime(tx));
            accessTimeList.append(",");
        }

        LogInfo info = new LogInfo(
                /*rankID*/rank.getRankID(tx),
                /*connectCount*/ rank.getConnectionCount(tx),
                /*accessLog*/ accessIDList.toString(),
                /*accessTime*/ accessTimeList.toString());

        return info;
    }

}
```

rdarank/agent/rank/handler/DisposeRankHandler.java


```
 1: package rdarank.agent.rank.handler;
 2:
 3: import com.ibm.agent.exa.Message;
 4: import com.ibm.agent.exa.MessageHandler;
 5:
 6: public class DisposeRankHandler extends MessageHandler {
 7:
 8:     @Override
 9:     public Object onMessage(Message msg) throws Exception {
10:         try {
11:             disposeAgent();
12:             return getAgentKey() + " was disposed";
13:         } catch (Exception e) {
14:             throw e;
15:         }
16:     }
17: }
```

```
    1: package rdarank.agent.user.handler;
    2:
    3: import java.sql.Timestamp;
    4:
    5: import rdarank.agent.user.message.InitUserMessage;
    6:
    7: import com.ibm.agent.exa.Message;
    8: import com.ibm.agent.exa.MessageHandler;
    9: import com.ibm.agent.exa.TxID;
   10: import rdarank.Profile;
   11: import rdarank.Useragent;
   12: import rdarank.Userlog;
   13:
   14: public class InitUserHandler extends MessageHandler {
   15:     @Override
   16:     public Object onMessage(Message msg) throws Exception {
   17:         try {
   18:             InitUserMessage initMsg = (InitUserMessage)msg;
   19:
   20:             Useragent user = (Useragent)getEntity();
   21:
   22:             TxID tx = getTx();
   23:             Profile prof = user.createProfile(tx);
   24:
   25:             //Set User Profile
   26:             prof.setName(tx, initMsg.name);
   27:             prof.setSex(tx, initMsg.sex);
   28:             prof.setAge(tx, initMsg.age);
   29:             prof.setAddress(tx, initMsg.address);
   30:
   31:             Long time = System.currentTimeMillis();
   32:             Timestamp registerTime = new Timestamp(time);
   33:             prof.setLastAccessTime(tx, registerTime);
   34:
   35:             user.setData(tx, 0);
   36:
   37:             user.setCommunicationID(tx, initMsg.comAgentID);
   38:
   39:             user.setConnectionCount(tx, 0);
   40:
   41:             Userlog log = user.createLog(tx, "init");
   42:
   43:             log.setLastAccessTime(tx, registerTime);
   44:             log.setCurrentTime(tx, time);
   45:
   46:             //System.out.println("InitHandler of Agent:" + getAgentKey() + " w
as initialized");
   47:
   48:             return "hello from agent:" + getAgentKey();
   49:         } catch(Exception e) {
   50:             throw e;
   51:         }
   52:     }
   53: }
```

```java
package rdarank.agent.user.handler;

import java.sql.Timestamp;

import com.ibm.agent.exa.Message;
import com.ibm.agent.exa.MessageHandler;
import com.ibm.agent.exa.TxID;

import rda.agent.queue.MessageObject;

import rdarank.Useragent;
import rdarank.Userlog;
import rdarank.agent.user.message.UpdateUserMessage;

public class UpdateUserHandler extends MessageHandler{

    @Override
    public Object onMessage(Message msg) throws Exception {
        UpdateUserMessage updateMsg = (UpdateUserMessage) msg;
        MessageObject msgObj = (MessageObject) updateMsg.messageData;

        Useragent agent = (Useragent)getEntity();

        TxID tx = getTx();

        long updateData = 0;
        for(Object data : msgObj.data){
            updateData =  updateData + (int)data;
        }
        agent.setData(tx, updateData + agent.getData(tx));

        //Agent Status
        //Connection
        agent.setConnectionCount(tx, agent.getConnectionCount(tx) + 1);
        //Message Latency
        agent.setMessageLatency(tx, msgObj.latency());
        //Agent Message Queue
        //agent.setMessageQueueLength(tx, msgObj.getLength());

        // Update Log Records
        Userlog log = agent.getLog(tx, "update");
        if(log == null)
            log = agent.createLog(tx, "update");

        // Update LastAccessTime
        Long time = System.currentTimeMillis();
        Timestamp updateTime = new Timestamp(time);
        log.setLastAccessTime(tx, updateTime);
        log.setCurrentTime(tx, time);

        //Long message = avgLatency;


        return 0L;
    }
}
```

```java
package rdarank.agent.user.handler;

import com.ibm.agent.exa.Message;
import com.ibm.agent.exa.MessageHandler;
import com.ibm.agent.exa.TxID;
import rdarank.Profile;
import rdarank.Useragent;
import rdarank.agent.user.reader.UserInfo;

public class ReadUserHandler extends MessageHandler {

    @Override
    public Object onMessage(Message arg0) throws Exception {
        Useragent user = (Useragent) getEntity();

        TxID tx = getTx();

        Profile prof = user.getProfile(tx);

        UserInfo info = new UserInfo(
                /*UserID*/user.getUserID(tx),
                /*Name*/ prof.getName(tx),
                /*Sex*/ prof.getSex(tx),
                /*Age*/ prof.getAge(tx),
                /*address*/ prof.getAddress(tx),
                /*data*/ user.getData(tx),
                /*count */ user.getConnectionCount(tx)
        );

        return info;
    }

}
```

```java
package rdarank.agent.user.handler;

import java.util.Iterator;

import com.ibm.agent.exa.Message;
import com.ibm.agent.exa.MessageHandler;
import com.ibm.agent.exa.TxID;
import com.ibm.agent.exa.entity.Entity;
import rdarank.Useragent;
import rdarank.Userlog;
import rdarank.agent.user.logger.LogInfo;

public class ReadLogUserHandler extends MessageHandler{

    @Override
    public Object onMessage(Message msg) throws Exception {
        Useragent user = (Useragent)getEntity();

        TxID tx = getTx();

        StringBuilder accessIDList = new StringBuilder();
        StringBuilder accessTimeList = new StringBuilder();
        Iterator<Entity> it = user.getLogIterator(tx);
        while(it.hasNext()){
            Userlog log = (Userlog) it.next();

            accessIDList.append(log.getAccessID(tx));
            accessIDList.append(",");

            accessTimeList.append(log.getLastAccessTime(tx));
            accessTimeList.append(",");
        }

        LogInfo info = new LogInfo(
            /*userID*/                  user.getUserID(tx),
            /*connectCount*/            user.getConnectionCount(tx),
            /*accessLog*/               accessIDList.toString(),
            /*accessTime*/              accessTimeList.toString());

        return info;
    }

}
```

```java
package rdarank.agent.user.handler;

import com.ibm.agent.exa.Message;
import com.ibm.agent.exa.MessageHandler;

public class DisposeUserHandler extends MessageHandler {

    @Override
    public Object onMessage(Message msg) throws Exception {
        try {
            disposeAgent();
            return getAgentKey() + " was disposed";
        } catch (Exception e) {
            throw e;
        }
    }
}
```

```java
package rdarank;

import com.ibm.agent.exa.impl.HPAEntity;
import com.ibm.agent.exa.TxID;

/**
 * Generated code for intaractionmanageragent.
 *
 * <p>entity type="intaractionmanageragent tablename="intaractionmanageragent
<br>
 * attribute name="ID" type="STRING" primarykey="true" <br>
 **/
public class Intaractionmanageragent extends HPAEntity {
    /**
     * Primary key size
     **/
    public static final int PKINDEXSIZE = 1;

    /**
     * Primary key index of ID
     **/
    public static final int PKINDEX_ID = 0;

    /**
     * Column index of ID
     **/
    public static final int ID = 0;

    /**
      * This constructor is used by the runtime.
      * An application should not create an instance with this constructor
     **/
    public Intaractionmanageragent() {
        super();
    }

    /**
      * Get the version string
     **/
    public String getVersion() {
        return "rdarank1.0";
    }

    /**
      * Get a value of ID.
      * The setter method of ID is not generated because this attribute is a pr
imarykey.
      * @return ID
      **/
    public final String getID(TxID tx) {
        // generated code
        return getString(tx,0);
    }

}
```

```
    1: package rdarank.agent.user.handler;
    2:
    3: import com.ibm.agent.exa.Message;
    4: import com.ibm.agent.exa.MessageHandler;
    5: import com.ibm.agent.exa.TxID;
    6: import rdarank.Useragent;
    7: import rdarank.agent.user.data.UserData;
    8: import rda.extension.intaraction.AgentIntaractionExtension;
    9:
   10: public class IntaractionAgentHandler extends MessageHandler{
   11:     private AgentIntaractionExtension extension = AgentIntaractionExtension.ge
tInstance();
   12:
   13:     @Override
   14:     public Object onMessage(Message msg) throws Exception {
   15:         Useragent agent = (Useragent)getEntity();
   16:
   17:         TxID tx = getTx();
   18:
   19:         //UserData
   20:         UserData userData = new UserData(agent.getUserID(tx), agent.getCommuni
cationID(tx));
   21:         userData.setData(agent.getData(tx));
   22:
   23:         //Agent Communication
   24:         extension.communicateAgent(userData);
   25:
   26:         return null;
   27:     }
   28:
   29: }
```

```
    1: package rda.extension.intaraction;
    2:
    3: import com.ibm.agent.exa.AgentException;
    4: import com.ibm.agent.exa.AgentKey;
    5: import com.ibm.agent.exa.AgentManager;
    6: import com.ibm.agent.exa.MessageFactory;
    7: import com.ibm.agent.exa.SimpleMessage;
    8: import com.ibm.agent.exa.client.AgentClient;
    9: import com.ibm.agent.exa.client.AgentExecutor;
   10: import java.io.Externalizable;
   11: import java.io.IOException;
   12: import java.io.ObjectInput;
   13: import java.io.ObjectOutput;
   14: import java.util.Collection;
   15: import rda.agent.queue.MessageObject;
   16:
   17: public class Transport implements AgentExecutor, Externalizable{
   18:     private static final String AGENT_TYPE ="intaractionmanageragent";
   19:     private static final String MESSAGE_TYPE="intaractionAgent";
   20:
   21:     public Transport(){}
   22:
   23:     AgentKey agentKey;
   24:     MessageObject msg;
   25:     public Transport(AgentKey agentKey, MessageObject msg) {
   26:         this.agentKey = agentKey;
   27:         this.msg = msg;
   28:     }
   29:
   30:     @Override
   31:     public void writeExternal(ObjectOutput out) throws IOException {
   32:         out.writeObject(agentKey);
   33:         out.writeObject(msg);
   34:     }
   35:
   36:     @Override
   37:     public void readExternal(ObjectInput in) throws IOException, ClassNotFound
Exception {
   38:         this.agentKey = (AgentKey) in.readObject();
   39:         this.msg = (MessageObject) in.readObject();
   40:     }
   41:
   42:     @Override
   43:     public Object complete(Collection<Object> results) {
   44:         Object[] ret = results.toArray();
   45:         return ret[0];
   46:     }
   47:
   48:     @Override
   49:     public Object execute() {
   50:         try {
   51:             AgentManager agentManager = AgentManager.getAgentManager();
   52:
   53:             SimpleMessage smsg = (SimpleMessage)MessageFactory.getFactory().ge
tMessage(MESSAGE_TYPE);
   54:             smsg.set("message", msg);
   55:
   56:             //Sync Message
   57:             //Object ret = agentManager.sendMessage(agentKey, smsg);
   58:             agentManager.putMessage(agentKey, smsg);
   59:
   60:             //return ret;
   61:         } catch (IllegalAccessException | InstantiationException e) {
   62:             return e;
   63:         } catch (AgentException ex) {
   64:             return ex;
   65:         }
   66:
   67:         return null;
   68:     }
   69:
   70:     public void sendMessage(AgentClient client, MessageObject msg){
   71:         try{
   72:             agentKey = new AgentKey(AGENT_TYPE, new Object[]{AGENT_TYPE});
   73:
   74:             Transport executor = new Transport(agentKey, msg);
   75:             Object reply = client.execute(agentKey, executor);
   76:
   77:             //System.out.println(reply);
   78:         } catch (Exception ex) {
   79:             ex.printStackTrace();
   80:         }
   81:     }
   82: }
```

```java
package rda.extension.intaraction;

import com.ibm.agent.exa.AgentException;
import com.ibm.agent.exa.AgentKey;
import com.ibm.agent.exa.AgentManager;
import com.ibm.agent.soliddb.extension.Extension;

import java.util.Properties;

import rda.agent.queue.MessageObject;
import rda.agent.queue.MessageQueue;
import rda.agent.queue.MessageQueueEvent;
import rda.data.test.DataTemplate;
import rda.window.WindowController;

import rdarank.agent.rank.manager.RankAgentManager;

public class AgentIntaractionExtension implements Extension {

    private static AgentIntaractionExtension extention = new AgentIntaractionExtension();

    private WindowController windowCTRL = new WindowController(1000, 10L, 1);

    public static AgentIntaractionExtension getInstance() {
        return extention;
    }

    AgentKey extensionAgentKey;

    public AgentIntaractionExtension() {
    }

    // ã\203ªã\203¼ã\202¸ã\203§ã\203³å\220\215
    String regionName;

    @Override
    public void initialize(String serverTypeName, Properties params) throws Exception {
    }

    @Override
    public void primaryChanged() throws Exception {
    }

    @Override
    public void regionChanged(int serverRole, String regionName) throws Exception {
        this.regionName = regionName;

        if (serverRole == Extension.ROLE_PRIMARY) {
            startService();
        }
    }

    @Override
    public void roleChanged(int serverRole) throws Exception {
        if (serverRole == Extension.ROLE_PRIMARY) {
            startService();
        }
    }

    @Override
    public void shutdown() {
    }

    @Override
    public void start(int serverRole, String regionName) throws Exception {
        this.regionName = regionName;

        if (serverRole == Extension.ROLE_PRIMARY) {
            startService();
        }
    }

    private void startService() {
        System.out.println("> Start AgentIntaraction Extension");

        System.out.println("       *****              ********* ");
        System.out.println("      *******             ********* ");
        System.out.println("     ***   ***                ***    ");
        System.out.println("    ***     ***               ***    ");
        System.out.println("   *************             ***    ");
        System.out.println("  ***************            ***    ");
        System.out.println(" ***            ***        ********* ");
        System.out.println("***              ***       ********* ");

        //Set Agent
        AgentManager am = AgentManager.getAgentManager();
        try {
            //AgentIntaractionManager Extension AgentKey
            extensionAgentKey = new AgentKey("intaractionmanageragent", new Object[]{"intaractionmanageragent"});

            //CreateAgent
            if (am.exists(extensionAgentKey)) {
                System.out.println("IntaractionManagerAgent already exists");
            } else {
                am.createAgent(extensionAgentKey);
            }
        } catch (AgentException ex) {
        }

    }

    public WindowController getWindowController() {
        return this.windowCTRL;
    }

    public Boolean transport(MessageObject msg) {
        //Transport OtherServer Extension
        // ***
        //Get MessageQueue
        MessageQueue mq = (MessageQueue) RankAgentManager.getInstance().getMQMap().get(msg.getID());

        if (mq == null) {
            System.out.println("Not Found Destination !");
            return true;
        }

        try {
            mq.put(msg);
        } catch (MessageQueueEvent mqev) {
```

```
120:             mqev.printEvent();
121:             return false;
122:         }
123:
124:         return true;
125:     }
126:
127:     public void startConnectionAgents() {
128:         System.out.println("rda.extension.intaraction.AgentIntaractionExtensio
n.startConnectionAgents() >> " + "start intaractions!");
129:         //AgentIntaraction Thread
130:         AgentIntaractionThread thread = new AgentIntaractionThread();
131:         thread.start();
132:     }
133:
134:     public void communicateAgent(DataTemplate data) {
135:         if (data == null) {
136:             return;
137:         }
138:
139:         windowCTRL.pack(RankAgentManager.getInstance(), data);
140:     }
141: }
```

```java
package rda.extension.intaraction;

import com.ibm.agent.exa.client.AgentClient;
import rda.agent.client.DistributedAgentConnection;
import rda.agent.queue.MessageObject;
import rda.extension.manager.SystemManagerExtension;
import rda.manager.AgentManager;
import rda.window.Window;
import rdarank.agent.rank.manager.RankAgentManager;

public class AgentIntaractionThread extends Thread {

    private static final String name = "AgentIntaraction Thread";
    private static final AgentIntaractionExtension extension = AgentIntaractionExtension.getInstance();

    private SystemManagerExtension manager;
    private AgentManager agent;
    public AgentIntaractionThread() {
        this.manager = SystemManagerExtension.getInstance();
        this.agent = RankAgentManager.getInstance();
    }

    @Override
    public void run() {
        System.out.println(name + " Start !");
        Transport trans = new Transport();

        while (manager.getState()) {
            try{
            Window window = extension.getWindowController().get();
            if (window == null) {
                try {
                    Thread.sleep(1L);
                } catch (InterruptedException ex) {
                }
            } else {
                //Translation Window To Message
                MessageObject msg = new MessageObject(window.getDestID(), window.unpack());

                //local
                if (manager.getDeployPattern() != 1) {
                    if (extension.transport(msg))
                        extension.getWindowController().remove();
                    else extension.getWindowController().returnExecutable(window);

                //dist deploy
                }else {
                    DistributedAgentConnection agcon = agent.getConnection(msg.rankID());

                    AgentClient client = agcon.getClient();

                    //System.out.print(">> Server <"+agcon.toString()+"> ");
                    trans.sendMessage(client, msg);
                    extension.getWindowController().remove();

                    agcon.returnConnection(client);
                }
            }
            }catch(Exception e){e.printStackTrace();}
        }

        extension.getWindowController().close();
        System.out.println(name + " Stop !");

    }
}
```

```java
package rda.clone;

import rda.manager.AgentManager;
import rda.manager.IDManager;

public class AgentCloning {
    public static String cloning(AgentManager manager, String sourceID, Object originalState){
        if(!manager.getAutoMode()) return "";

        System.out.println(">> Start Agent Cloning");

        IDManager id = manager.getIDManager();
        String originalID = id.getOrigID(sourceID);

        //Clone
        String cloneID = manager.createCloneAgent(originalID, originalState);
        id.regID(originalID, cloneID);

        System.out.println(">> Agent Cloning New Copy From "+ originalID);

        return cloneID;
    }

    public static String delete(AgentManager manager, String deleteID){
        if(!manager.getAutoMode()) return "";

        System.out.println(">> Start Agent Delete");

        IDManager id = manager.getIDManager();
        String originalID = id.getOrigID(deleteID);

        //Delete
        id.deleteID(originalID, deleteID);

        System.out.println(">> Agent Cloning Delete "+ deleteID);

        return deleteID;
    }
}
```

```
    1: package rdarank.deploy;
    2:
    3: import com.ibm.agent.exa.client.AgentClient;
    4: import java.util.ArrayList;
    5: import java.util.HashMap;
    6: import java.util.List;
    7: import java.util.Map;
    8: import rda.agent.client.DistributedAgentConnection;
    9: import rdarank.agent.rank.manager.RankAgentManager;
   10: import rdarank.agent.user.manager.UserAgentManager;
   11: import rdarank.manager.LaunchCreateAgent;
   12:
   13: public class DeployStrategy {
   14:     private Integer pattern = 0;
   15:     public DeployStrategy(Integer pattern) {
   16:         this.pattern = pattern;
   17:     }
   18:
   19:     public void deploy(UserAgentManager user, RankAgentManager rank){
   20:         switch(pattern){
   21:             case 0: localDeploy(user, rank);
   22:                     break;
   23:             case 1: agentTypeDeploy(user, rank);
   24:                     break;
   25:             case 2: agentCloneDeploy(user, rank);
   26:                     break;
   27:         }
   28:     }
   29:
   30:     private void localDeploy(UserAgentManager user, RankAgentManager rank){
   31:         LaunchCreateAgent agent = new LaunchCreateAgent();
   32:
   33:         Map agentGroup = new HashMap();
   34:         agentGroup.put("useragent", user.getIDList());
   35:         agentGroup.put("rankagent", rank.getIDList());
   36:
   37:         //System.out.println(agentGroup);
   38:
   39:         DistributedAgentConnection agcon = UserAgentManager.getInstance().getC
onnection("0");
   40:         AgentClient client = agcon.getClient();
   41:
   42:         agent.create(client, agentGroup);
   43:         agcon.returnConnection(client);
   44:     }
   45:
   46:     private void agentTypeDeploy(UserAgentManager user, RankAgentManager rank)
{
   47:         LaunchCreateAgent agent = new LaunchCreateAgent();
   48:
   49:         //User Agent
   50:         Map agentGroup = new HashMap();
   51:         Map<DistributedAgentConnection, List> serverGroup = new HashMap();
   52:
   53:         for(String userID : user.getIDList()){
   54:             DistributedAgentConnection agcon = user.getConnection(userID);
   55:
   56:             if(serverGroup.get(agcon) == null) serverGroup.put(agcon, new Arra
yList<>());
   57:
   58:             List<String> agList = (List<String>) serverGroup.get(agcon);
   59:             agList.add(userID);
   60:             serverGroup.put(agcon, agList);
   61:         }
   62:
   63:         for(DistributedAgentConnection agcon : serverGroup.keySet()){
   64:             System.out.println(">> Deploy Agent Server = "+agcon.toString());
   65:
   66:             agentGroup.put("useragent", serverGroup.get(agcon));
   67:
   68:             AgentClient client = agcon.getClient();
   69:             agent.create(client, agentGroup);
   70:             agcon.returnConnection(client);
   71:         }
   72:
   73:         //Rank Agent
   74:         agentGroup = new HashMap();
   75:         serverGroup = new HashMap();
   76:
   77:         for(String rankID : rank.getIDList()){
   78:             DistributedAgentConnection agcon = rank.getConnection(rankID);
   79:
   80:             if(serverGroup.get(agcon) == null) serverGroup.put(agcon, new Arra
yList<>());
   81:
   82:             List<String> agList = (List<String>) serverGroup.get(agcon);
   83:             agList.add(rankID);
   84:             serverGroup.put(agcon, agList);
   85:         }
   86:
   87:         for(DistributedAgentConnection agcon : serverGroup.keySet()){
   88:             System.out.println(">> Deploy Agent Server = "+agcon.toString());
   89:
   90:             agentGroup.put("rankagent", serverGroup.get(agcon));
   91:
   92:             AgentClient client = agcon.getClient();
   93:             agent.create(client, agentGroup);
   94:             agcon.returnConnection(client);
   95:         }
   96:     }
   97:
   98:     private void agentCloneDeploy(UserAgentManager user, RankAgentManager rank
){
   99:         LaunchCreateAgent agent = new LaunchCreateAgent();
  100:
  101:         //User Agent
  102:         Map agentGroup = new HashMap();
  103:         Map<DistributedAgentConnection, List> serverGroup = new HashMap();
  104:
  105:         for(String userID : user.getIDList()){
  106:             DistributedAgentConnection agcon = user.getConnection(userID);
  107:
  108:             if(serverGroup.get(agcon) == null) serverGroup.put(agcon, new Arra
yList<>());
  109:
  110:             List<String> agList = (List<String>) serverGroup.get(agcon);
  111:             agList.add(userID);
  112:             serverGroup.put(agcon, agList);
  113:         }
  114:
  115:         for(DistributedAgentConnection agcon : serverGroup.keySet()){
  116:             System.out.println(">> Deploy Agent Server = "+agcon.toString());
  117:
  118:             agentGroup.put("useragent", serverGroup.get(agcon));
```

```
119:             //Rank Agent
120:             agentGroup.put("rankagent", rank.getIDList());
121:
122:             AgentClient client = agcon.getClient();
123:             agent.create(client, agentGroup);
124:             agcon.returnConnection(client);
125:         }
126:     }
127: }
```