個別通知サービスにおけるキャッシュと連動したイベント処理スケジューリング手法とその性能評価

山本 学[†] 田井秀樹[†]

近年各利用者の情報を利用した個別の情報を通知する個別通知サービスが登場している．個別通知サービスでは，システムは情報更新などのイベントを受けて，多数の利用者のための処理を実行するため，システムの処理速度が重要となる．システムの高速化を行う技術として，アプリケーションサーバで利用者のデータをキャッシュする方法があるが，性能をできる限り向上させるには，キャッシュ利用率の向上が鍵となる．我々が提案しているエージェントサーバは，Cohort Scheduling を利用し，キャッシュとエージェントの処理スケジューラを連動させることで，通知型サービスで行われる大量のエージェントを一気に処理する場合において，キャッシュ利用率を向上させ，全体の処理時間の短縮を行っている．本稿では，Cohort Scheduler を用いたエージェントサーバのキャッシュ機構とスケジューラの実装を示すとともに，その効果を実測，シミュレーションにより示す．さらにシミュレーションモデルの解析によりその効果の検証を行い，そのスケジューラの有効性を議論する．

A Cache Aware Event Process Scheduler for Personalized Notification Services and Its Performance Evaluation

Gaku Yamamoto[†] and Hideki Tai[†]

Recently, the personalized notification services that use individual users' data are emerging. In case of such services, performance is one of significant issues because such systems need to execute tasks with refering to individual users' data. A data cache technology at application servers will solve the performance issue. Moreover performance will be increased by improving cache utilization. This paper introduces the scheduling mechanism of the agent server we have been developed based on Cohort Scheduler. The scheduler utilizes cache efficiently and improves performance. This paper shows an implementation of a cache mechanism and a scheduler of an agent server and performance measurement results of the scheduler. We also describe efficiency of the scheduler on the environment where the scheduler is applied to a multiple servers by showing both simulation and analysis of the simulation model.

1. はじめに

近年各利用者の情報を利用した個別の情報を通知する個別通知サービスが登場している．たとえば，外貨レート更新時に，各利用者の外貨預金口座情報を参照して損益額を計算し，閾値を超えていたら通知する外貨預金券口座管理サービスがある．また，駅の自動改札機を通過した利用者に興味のある駅周辺情報を携帯電話にメールで配信するサービスもこの一例である[1),2)]．このような個別通知サービスでは，システムは情報更新などのイベントを受けて，多数の利用者のための処理を，個々の利用者の情報を参照・更新しながら行う．このようなのサービスでは，利用者数は数十万から数百万人規模となるため，高い処理能力が必要となるが，システムの処理能力がデータベースサーバの処理能力に縛られ，高い性能を実現できないという問題がある．これを解決する技術として，アプリケーションサーバ上でのデータのキャッシュ技術がある[3)~5)]．しかしながら，商用サービスでは 1 人の利用者のデータが数十 KB となる場合もあり，すべての利用者データをキャッシュできるとは限らない．このような状況では，キャッシュをできる限り有効利用すること，いい換えれば，キャッシュヒット率を最大にすることが鍵となる．

キャッシュヒット率を向上させるためのキャッシュ置換アルゴリズムは，Least Recently Used（LRU）をはじめ数多くの研究がなされている[6)~8)]．さらに，キャッシュ利用率の向上のために，キャッシュ内にあるデータにアクセスする可能性が高い処理を優先的に

† 日本アイビーエム株式会社東京基礎研究所
IBM Research, Tokyo Research Laboratory, IBM Japan Ltd.

行うスケジューリング手法もいくつか提案され，その有効性が示されている[9)~11)]．

我々は，利用者のデータを保持し，イベント受信時にその利用者のための処理を行う「エージェント」の概念を取り入れた「エージェントサーバ」を開発し，いくつかの個別通知サービスに適用している[1),12)~14)]．エージェントサーバでは，データベースに保存されている各利用者のデータはエージェントサーバ内のメモリに「エージェント」オブジェクトとしてキャッシュされる．エージェントサーバは，キャッシュ利用率の向上を行うスケジューリング手法の 1 つである Cohort Scheduling [9)] の考えに基づき，キャッシュとエージェントの処理スケジューラを連動させることで，キャッシュヒット率を向上させ，全体の処理時間の短縮を行っている．

本稿では，Cohort Scheduling のエージェントサーバへの適用を示し，その効果を実測とシミュレーションで示す．さらにシミュレーションモデルの解析によりその効果の検証を行う．2 章で，大規模な個別通知サービスとその課題を述べる．3 章ではエージェントサーバの概要を，4 章でイベントのスケジュール方法を説明する．5 章で実機を用いた性能計測結果を，6 章で複数台のエージェントサーバを使用した場合のシミュレーション結果を示す．7 章でシミュレーションモデルの解析による検証を行う．8 章で関連研究を，9 章でまとめる．

2. 大規模な個別通知サービスとその課題

本稿で対象とする個別通知サービスでは，利用者はあらかじめサービスを提供するサーバに通知処理で参照される利用者情報を登録しておく．サーバは定期的に通知対象である情報を取得し，利用者情報と最新の通知対象情報を参照した処理を利用者ごとに行い，通知メールを送信すべきかを判断し，必要であれば通知対象情報をメールで送信する．たとえば，証券口座損益額通知サービスでは，利用者情報は，株銘柄名・株数・購入時株価・損益額通知の閾値である通知上下限値・通知先メールアドレスである．株銘柄名・株数・購入時株価は複数登録可能である．これらは利用者データベースに書き込まれる．通知対象情報は株価と計算された損益額である．株価は株価情報データベースに格納され，適宜更新される．サーバは一定期間ごとに株価情報データベースから最新の株価を取得し，各利用者の証券口座の損益額を計算し，その値が登録されている通知上下限値の範囲を超えたらメールで通知する．このとき，再度通知を行わないように「通知済み」の情報を利用者データベースに書き込む．この情報は利用者が再度登録情報を変更したときに「未通知」に変更される．たとえば，利用者 A は，1 株 8,500 円で A 社株を 200 株，1 株 4,200 円で B 社株を 100 株，通知上限値を +10 万円，下限値を −5 万円と登録したとする．サーバは定期的に損益額の計算を行っているが，A 社，B 社の株価がそれぞれ 9,540 円，3,210 円になると，利用者 A の損益額は +109,000 円となるため，通知メールを送り，利用者データベースの利用者 A のレコードに「通知済み」印をつける．

このような個別通知サービスでは，ある利用者に対して通知メールを送信すべきかどうかを判断するには，サーバでその利用者のデータを参照して処理を行う必要がある．先述した例では損益額計算がそれに該当する．このため，サーバは通知対象である情報の更新のたびに全利用者の処理を行う．インターネットを利用したサービスでは，利用者は数十万人以上となる．毎秒 500 件処理を行うことができるデータベースサーバを用い，データ参照のたびにデータベースから取得する場合，100 万人の利用者で 33 分以上を要する．処理時間を短縮するには，利用者情報をサーバのメモリにキャッシュさせることが鍵となる．しかしながら，我々は，キャッシュ上に全利用者の 90%の利用者のデータが載っている状況の処理性能は，全利用者のデータが載っている状況の半分の性能になることを実験で確認している．この原因は，キャッシュにデータが存在するかどうかに関係なく利用者の処理順序が決定されること，キャッシュにない利用者の処理を行うために，キャッシュされているが未処理である利用者のデータをキャッシュから破棄することのためにキャッシュヒット率が低下するためである．このため，より多くのメモリが必要となり，最終的にはサーバ計算機の増加につながる．したがって，キャッシュ利用率を向上させることは重要な課題である．

3. エージェントサーバ

3.1 プログラミングモデル

我々は「エージェント」の概念を導入したプログラミングモデルと多数のエージェントの活動を管理できる実行環境である「エージェントサーバ」を開発し，いくつかの個別通知サービスに適用している[1),12)~14)]．エージェントは利用者ごとに 1 つ以上生成され，エージェントサーバに常駐する．エージェントは，利用者のデータを持ち，メッセージを受信したときに，そのデータを参照・更新しながら処理を行う．このデータはデータベースやファイルなどの外部記憶装置に保存さ

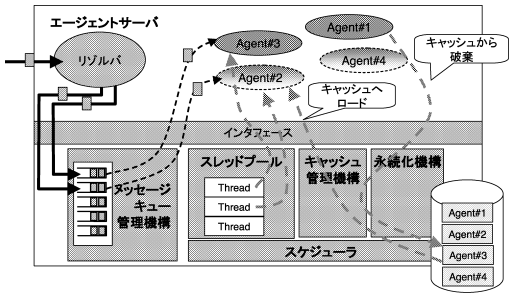


図 1 エージェントサーバ概要
Fig. 1 Overview of Agent Server.

れているもので，データが更新されたときは，実行環境がそのデータを永続領域に書き出す．あるエージェントが他のエージェントのデータにアクセスすることはない．エージェントは外部のプログラムまたは同じプログラム上の他のモジュールが送信するメッセージを受けて処理を行う．メッセージはいったん実行環境内に蓄えられ，実行環境が適当な時期にエージェントのメッセージ・ハンドラを呼び出す．複数のエージェントの処理は同時並行的に行われるが，1 つのエージェントが複数のメッセージ処理を同時並行して実行することはない．メッセージの送信方法には，宛先エージェントを特定したピア・トゥ・ピア送信と，不特定多数のエージェントへメッセージを送信するマルチキャスト送信がある．エージェントにはそれを特定するキーがあり，ピア・トゥ・ピア送信の場合は，そのキーを指定してメッセージを送信する．一方，マルチキャスト送信では，宛先エージェントを特定する「リゾルバ」オブジェクトがメッセージの内容を参照して配信すべきエージェントのリストを生成し，実行環境がそのリストを参照してメッセージをエージェントに送信する．本稿で対象とする個別通知サービスでは，マルチキャスト送信が使われる．

3.2 実行環境

エージェントサーバは，その内部にメッセージ・キュー管理機構，スレッド管理機構，キャッシュ管理機構，処理スケジューラを持つ（**図 1**）．

メッセージ・キュー管理機構は，各エージェントのメッセージ・キューを管理する．エージェントに送信されたメッセージは，対応するメッセージ・キューに格納される．この機構は優先度付メッセージをサポートしており，優先度の高いメッセージが先に処理される．

スレッド管理機構は，あらかじめ設定された数のスレッドをプールしておき，メッセージ処理を行うエージェントにスレッドを提供する．

キャッシュ管理機構は，エージェントをキャッシュするための機構である．キャッシュの大きさは，キャッシュ可能なエージェント数で指定される．エージェントはオブジェクトであり，エージェントが処理を実行するには，そのオブジェクトがメモリ上になければならない．エージェントの数が莫大であると，物理メモリに保持しきれなくなる．物理メモリに入りきらない分のエージェントはメモリから破棄され，必要に応じて外部記憶装置から読み出されキャッシュされる．同時に，他のエージェントがキャッシュから破棄される．

スケジューラは，メッセージ処理を行うべきエージェントの選択とキャッシュから破棄するエージェントの選択を行う．スケジューラは，エージェントの状態を管理し，メッセージの優先度を考慮しつつも，キャッシュヒット回数をできるだけ高くするスケジュールを行う．スケジューリング方式は次章で詳しく述べる．

エージェントサーバが複数あるシステムでは，あるエージェントは，そのエージェントのキーのハッシュ値から 1 つのエージェントサーバを決定し，そのエージェントサーバ上でだけ活動するように管理される．

4. エージェントのスケジューリング

4.1 概　　要

エージェントサーバのスケジューラは Cohort Scheduling [9] の考えに基づき，キャッシュの状態を考慮したスケジュールを行う．Cohort Scheduling は，似たような処理または同じデータにアクセスするタスク群を「Cohort」としてまとめ，ある Cohort のタスクを連続的に行い，プロセッサなどのキャッシュ利用率を向上させる．エージェントサーバでは，キャッシュにあるエージェント群とないエージェント群を別々の Cohort とし，スケジューラはキャッシュにあるエージェント群の Cohort を先に処理し，その後にキャッシュにないエージェント群の Cohort の処理を行う．エージェントサーバでは，エージェントの優先度を考慮してスケジュールを行うので，Cohort はエージェントの優先度ごとにキャッシュにあるエージェント群とないエージェント群の Cohort が 2 つずつ作られる．ここで，エージェント優先度とは，そのメッセージ・キュー内にある最も優先度の高いメッセージの優先度である．なお，本稿では簡単のために優先度は最高優先度と通常優先度の 2 つとして説明する．

文献 9) では，プログラムをいくつかの Stage に分割する Staged Computing のプログラミングモデルを用い，Stage ごとに Cohort を生成する．一方，エージェントサーバでは，エージェントという形で処理とデータが利用者ごとに独立しており，さらに実行環境

表 1 エージェントの状態
Table 1 States of an Agent.

RUNNUNG	現在スレッドが割り当てられ，メッセージ処理中である
IN_FILLED	キャッシュ上にあり，メッセージ・キューに処理待ちメッセージがある
IN_EMPTY	キャッシュ上にあり，メッセージ・キューが空である
OUT_FILLED	キャッシュ上になく，メッセージ・キューに処理待ちメッセージがある
OUT_EMPTY	キャッシュ上になく，メッセージ・キューが空である
MAX_PRI_FILLED	メッセージ・キューに最高優先度の処理待ちメッセージがある

がエージェントの優先度やエージェントがキャッシュにあるかどうかを実行環境が把握できるので，実行環境が自動的に Cohort を作ることができる．これを行うには，エージェントがキャッシュにあるかどうか，未処理のメッセージを持つかどうかを示すエージェントの状態を管理する必要がある．スケジューラはこの状態を参照して，エージェントの処理順序を決定する．

キャッシュ利用率を向上させるには，キャッシュから破棄するエージェントの選択も重要になる．未処理のメッセージを持つエージェントを破棄すると，そのエージェントの処理を行うときにキャッシュミスが発生する．プロセッサの L2 キャッシュに関して同様のことが文献 10) で報告されている．Cohort Scheduling ではキャッシュ破棄のスケジュールは含んでいないが，エージェントサーバのキャッシュ破棄のスケジュールでは，未処理メッセージを持たないエージェント群からエージェントを選択して破棄を行うことで，さらなるキャッシュヒット率の向上を行う．

4.2 エージェントの状態管理

エージェントの優先度やエージェントがキャッシュにある･なしを考慮してスケジュールを行うには，エージェントの状態を管理する必要がある．エージェントの状態を**表 1** に示す．状態の変化は，エージェントのメッセージ処理が完了し，他のエージェントの処理に切り替えるとき，メッセージをエージェントのメッセージ・キューに入れるとき，および，エージェントがキャッシュから破棄されるときにおきる．**図 2** に状態遷移図を示す．

エージェントは多数存在するため，同じ優先度の同じ状態にあるエージェントも多数存在する．同じ優先度で同じ状態にあるエージェントは，その状態になった時期が古い順で順序付けされ，リスト化される．つまり，エージェントの優先度が変化した場合，そのエージェントは，該当する優先度の該当する状態のエージェントのリストの最後尾に追加される．

4.3 スケジュールアルゴリズム

エージェントサーバのスケジュールを**図 3** に示す．このアルゴリズムでは，最高優先度のメッセージを持つエージェントは，それがキャッシュ上にあるなしにかかわらず，優先的に処理される．また，いったんあるエージェントが選択されると，スケジューラのエージェント切替えのオーバヘッドを削減するために，そのエージェントよりも高い優先度を持つエージェントがあっても，できるだけ連続して同じエージェントの処理を行う．

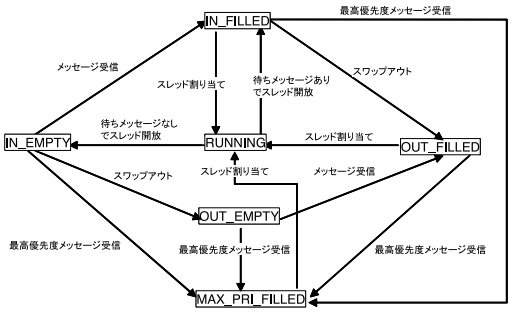


図 2 状態遷移図
Fig. 2 State transitions of an Agent.

1. MAX_PRI_FILLED の状態のエージェントがあれば，そのリストの先頭のエージェントを選択
2. 直前までに処理していたエージェントのメッセージ・キューにメッセージがあり，かつ，連続処理回数が一定値以下であれば，そのエージェントを再度選択
3. IN_FILLED 状態のエージェントがあれば，そのリストの先頭のエージェントを選択
4. OUT_FILLED 状態のエージェントがあれば，そのリストの先頭のエージェントを選択

図 3 エージェントのスケジューリングアルゴリズム
Fig. 3 Agent scheduling algorithm.

4.4 キャッシュ破棄

キャッシュヒット回数を向上させるには，キャッシュから破棄するエージェントの選択も重要である．このアルゴリズムを**図 4** に示す．このアルゴリズムは，IN_EMPTY 状態のエージェント群から破棄するエージェントを選択するときは，LRU に基づき最も以前にアクセスされたエージェントを選択する．一方，IN_FILLED のエージェント群から破棄するエージェントを選択するときは，処理頻度の公平性から，最も

1. IN_EMPTY 状態のエージェントがあれば，そのリストの先頭のエージェントを選択
2. IN_FILLED 状態のエージェントがあれば，そのリストの最後尾のエージェントを選択
3. MAX_PRI_FILLED 状態のエージェントがあれば，そのリストの最後尾のエージェントを選択

図 4 キャッシュからに破棄のアルゴリズム
Fig. 4 Cache eviction algorithm.

最近処理が完了したエージェントを選択する．この方法では，利用者からの Web ブラウザからのアクセスによるメッセージを未処理メッセージとして持つエージェントが，利用者が連続的にアクセスを行うにもかかわらずキャッシュから破棄されことがありうるが，この問題は利用者からのアクセスによるメッセージの優先度を最高優先度にすることで解決される．

5. 性能評価

5.1 評価対象アプリケーション

本稿では，エージェントサーバが 1 つのメッセージを受けてすべての利用者の処理を行うための処理時間を，スケジューラがある場合とスケジューラがない場合の 2 種類に対して，キャッシュサイズを変えながら計測し，性能を比較する．性能評価のアプリケーションは 2 章で示した証券口座損益額通知サービスである．

利用者データベースは**表 2** で示すテーブル StockAccount で定義される．Userid がプライマリキーであり，エージェントの識別に使用される．UPPER と LOWER は損益額通知の上限値と下限値，STOCK0 は株銘柄名，STOCKNUM0 は株数，STOCKPRICE0 は購入時株価である．利用者は株銘柄名・株数・購入時株価のセットを最大 10 まで登録できるため，StockAccount レコードにはあらかじめ 10 株分の領域を確保している．株価情報データベースは，VARCHAR(10) である株銘柄名と INTEGER である株価をレコードに持つテーブルで定義される．**図 5** にエージェントサーバが行う通知処理の流れを，**図 6** に各エージェントの処理を示す．

性能計測環境を**表 3** に示す．この評価では通知メール送信処理は行わない．すべての利用者は株銘柄名・株数・購入時株価のセットを 10 セット持つ．また，損益額通知の上限と下限の範囲から超える利用者数は，全利用者数の 10%とする．

5.2 計測結果

図 7 に，スケジューラありの場合となしの場合のキャッシュ可能なエージェント数を変化させたときの性能と全エージェント数に対するキャッシュヒット率の計測結果を示す．ここで，キャッシュ率とは全エージェント数に対するキャッシュ上のエージェント数の比である．

表 2 StockAccount のテーブル構成
Table 2 Schema of the StockAccount table.

Column Name	Data Type
Userid	VARCHAR(10)
Username	VARCHAR(50)
Mailaddress	VARCHAR(50)
Upper	INTEGER
Lower	INTEGER
Notified	CHAR(1)
STOCK0	VARCHAR(10)
STOCKNUM0	INTEGER
STOCKPRICE0	INTEGER
From STOCK1··· to STOCKPRICE9	

1. 最新の株価をデータベースから読み込み，メモリ中に保持する．
2. Notified の値が 0 の Userid を StockAccount から取得する．ただし，性能計測ではすべての Userid が取得される．
3. 利用者ごとに図 6 で示す処理を開始する．この処理は複数のスレッドで行う．

図 5 通知処理の流れ
Fig. 5 Notification procedure of the Agent Server.

1. 変数 total に 0 を代入
2. Stock0 から Stock9 に対して 3 から 5 の処理を繰り返す (StockX は Stock0 から Stock9 を示す)
3. もし StockX が NULL なら 4，5 をスキップする
4. StockX の最新株価を取得する
5. StockX の損益額を求め，変数 total に加算する
6. もし変数 total の値が Upper より大きいか，Lower より小さければ Notified の値を 1 にする
7. トランザクションをコミットする

図 6 エージェントの処理
Fig. 6 Notification handler of the Agent.

スケジューラありのキャッシュヒット率は，エージェントのキャッシュ率と同じであり，理論上最適なキャッシュヒット率となっている．一方，スケジューラなしの場合では，キャッシュ内にあるエージェントとないエージェントの区別をしていないため，キャッシュ内にあるが未処理であるエージェントを破棄してしまい，キャッシュヒット率が低下する．データベースサーバからの読み込みコストはメモリ中のデータに参照するコストよりもはるかに大きいため，その影響が性能に大きく現れている．なお，スケジューラなしでは，データベースから Userid 取得後，それらの結果順をランダムに入れ替える処理を施しているためにキャッシュ率 100%時で性能が若干低くなっている．リゾルバによる Userid の検索結果は毎回同じ順番になるた

表 3 性能計測環境
Table 3 Performance measurement environment.

Agent	数	10 万
	サイズ	約 1 KB/agent
Agent Server	Middleware	IBM WebSphere Application Server Enterprise Edition v5.0
	Java	IBM Java version 1.3 最小・最大ヒープ 1.5 GB
	OS	Windows 2000 Server
	H/W	Pentium III Xeon 2.2 GHz dual processor, 4 GB Memory
Database Server	DBMS	IBM Universal Database Server v7.2
	OS	Windows 2000 Server
	H/W	Pentium III Xeon 2.2 GHz dual processor, 4 GB Memory

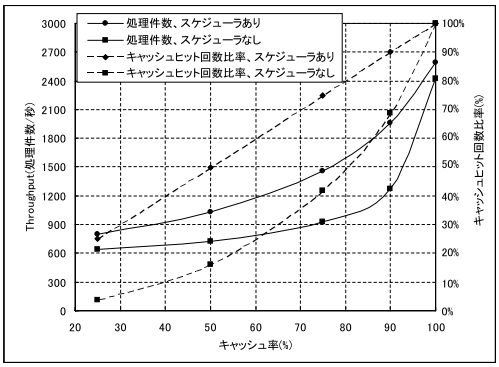


図 7 キャッシュ率と処理性能およびキャッシュヒット率
Fig. 7 Cache ratio vs. throughput and cache hit ratio.

表 4 シミュレーションモデルのパラメータ
Table 4 Parameters of the simulation model.

N	システム内にある全エージェント数
M	エージェントサーバ計算機台数
T_{ag}	キャッシュ内のエージェントに対する エージェントサーバの最大処理件数/秒
T_{agdb}	エージェントサーバのデータベース サーバへの最大アクセス件数/秒
T_{db}	データベースサーバの最大処理件数/秒
C_{no}	スケジューラなしの場合のエージェント のキャッシュヒット回数比率
C_{ag}	スケジューラありの場合のエージェント のキャッシュヒット回数比率
U	通知処理を行うエージェントの全エージェント 数に対する比率

め，この処理を行わないと，1 回目の計測後のキャッシュ上のエージェントは検索結果の後ろの方の Userid のエージェントになり，2 回目のイベント処理でキャッシュヒット率が極端に低下してしまう．

6. クラスタシステムにおける性能

6.1 シミュレーションモデル

次に，複数台のエージェントサーバ計算機を用いる場合のスケジューラの効果をシミュレーションにより検証する．システムは複数台のエージェントサーバ計算機と 1 台のデータベースサーバからなる．1 つのエージェントは 1 つのエージェントサーバ計算機でだけ活動する．イベントは全エージェントサーバに転送され，各サーバで図 5 と図 6 の処理が実行される．以下にスケジューラなしとスケジューラありのそれぞれのシミュレーションモデルを示す．

スケジューラなしの場合の全エージェントの処理時間 P_{no} は下記の式で表される．

$$P_{no} = max(P^1_{no}, P^2_{no})$$
$$max(x, y) \text{ は } x \text{ と } y \text{ の大きい値をとる関数}$$
$$P^1_{no} = \frac{N}{M}\left(\frac{1}{T_{ag}} + \frac{1 - C_{no} + U}{T_{agdb}}\right)$$
$$P^2_{no} = \frac{N(1 - C_{no} + U)}{T_{db}}$$

ここで，P^1_{no} はエージェントサーバでの処理時間であり，P^2_{no} はデータベースサーバでの処理時間である．システム全体の処理時間は，エージェントサーバとデータベースサーバの処理時間の長い方になる．このシミュレーションでは，リゾルバにおける Userid の順序入替えのコストは無視する．

スケジューラありの場合の全エージェントの処理時間 P は下記の式で表される．

$$P = max(P^{A1}, P^{A2}) + max(P^{B1}, P^{B2})$$
$$P^{A1} = \frac{N}{M}\left(\frac{C_{ag}}{T_{ag}} + \frac{C_{ag}U}{T_{agdb}}\right)$$
$$P^{A2} = \frac{NC_{ag}U}{T_{db}}$$
$$P^{B1} = \frac{N(1 - C_{ag})}{M}\left(\frac{1}{T_{ag}} + \frac{1 + U}{T_{agdb}}\right)$$
$$P^{B2} = \frac{N(1 - C_{ag})(1 + U)}{T_{db}}$$

ここで P^{A1} はキャッシュにある全エージェントに対するエージェントサーバでの処理時間，P^{A2} はキャッシュにある全エージェントに対するデータベースサーバでの処理時間，P^{B1} はキャッシュにない全エージェ

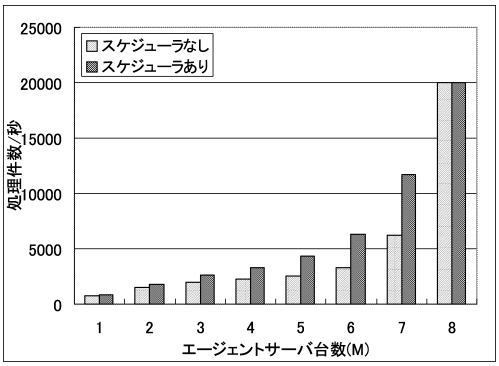


図 8 エージェントサーバ台数と処理性能
Fig. 8 Number of Agent Servers vs. throughput.

ントに対するエージェントサーバでの処理時間，P^{B2} はキャッシュにない全エージェントに対するデータベースサーバでの処理時間である．スケジューラありの場合では，キャッシュにあるエージェントの処理を先に行い，それらが完了した後にキャッシュにないエージェントの処理を行う．したがって，システム全体の処理時間は，キャッシュにある全エージェントの処理時間とキャッシュにない全エージェントの処理時間の合計となる．

6.2 シミュレーション結果

ここでは，評価アプリケーションに対して次の状況下でのエージェントサーバ計算機台数を変化させた場合の全エージェントの秒あたりの処理件数を示す．これは $1/P_{no}$ ならびに $1/P$ で与えられる．

- 1 台で 20 万エージェントまでキャッシュ
- N は 150 万
- T_{ag} は実測値より 3757.4 件/秒
- T_{agdb} は実測値をもとにした計算から 1000 回/秒
- T_{db} は 2000 件/秒

図 8 に，シミュレーション結果を示す．この結果から，キャッシュ率が低い場合，および，すべてのエージェントがキャッシュされる場合は，スケジューラありとなしで差がないが，それ以外ではスケジューラありの方がスケジューラなしに比べ，処理時間が短く効果が大きいことが分かる．スケジューラなしで計算機台数が 1〜2 までに比べ 3〜6 までの間では，エージェントサーバ計算機台数の増加にともなうスループットの増加量が鈍くなっているが，この原因は，エージェントサーバ台数が 1〜2 では，エージェントサーバでの処理がボトルネックとなっているが，3 以上では全体の処理量増加にともないデータベースサーバがボトルネックとなったためである．スケジューラありの場合では，つねにエージェントサーバとデータベースサーバのそれぞれの処理時間の合計で全体の処理時間が決定されるため，このような増加量の変化は見られない．

表 5 P_{no} と P の場合分け
Table 5 Case classification by P_{no} and P.

$P^1_{no} \geq P^2_{no}$	$P^{A1} \geq P^{A2}$	$P^{B1} \geq P^{B2}$	Case1
		$P^{B1} < P^{B2}$	Case2
	$P^{A1} < P^{A2}$	$P^{B1} \geq P^{B2}$	不成立
		$P^{B1} < P^{B2}$	不成立
$P^1_{no} < P^2_{no}$	$P^{A1} \geq P^{A2}$	$P^{B1} \geq P^{B2}$	不成立
		$P^{B1} < P^{B2}$	Case3
	$P^{A1} < P^{A2}$	$P^{B1} \geq P^{B2}$	不成立
		$P^{B1} < P^{B2}$	Case4

「不成立」となっている箇所は条件が成立しない場合である

7. 性能解析

図 8 では，つねにスケジューラありの方がスケジューラなしよりも高い性能を示していた．しかしながら，スケジューラなしでは，エージェントサーバとデータベースサーバがつねに同時並行的に動くが，スケジューラありでは，はじめにキャッシュにあるエージェントの処理を行い，その後にキャッシュにないエージェントの処理を行うため，エージェントサーバとデータベースサーバの並行処理性が低下し，結果としてスケジューラなしの方が高い性能を示すことも考えられる．ここではシミュレーションモデルからこの点を検証する．

P_{no} と P は**表 5** に示す 4 つの場合に分けられる．

ここで Case1 で P_{no} と P の大小比較を行う．

$$\begin{aligned} P_{no} - P &= P^1_{no} - (P^{A1} + P^{B1}) \\ &= \frac{N}{M(C_{ag} - C_{no})T_{agdb}} \end{aligned}$$

キャッシュが全エージェントを保持できない状況では，$C_{ag} > C_{no}$ なので $P_{no} - P > 0$ となる．よって，スケジューラありの方が処理時間が短いことが示される．同様に計算すると，Case4 でスケジューラありの方が処理時間が短いことが示される．一方，Case2 および 3 は，スケジューラなしの方が処理時間が短い場合がありうる．以下にこれらの場合に関して考える．

Case2 条件 $P^1_{no} \geq P^2_{no}$ かつ $P^{A1} \geq P^{A2}$ かつ $P^{B1} < P^{B2}$ から，Case2 となる M の上限値 M_{max} と下限値 M_{min} が下記のように与えられる．

$$M_{max} = \frac{T_{db}}{(1 - C_{no} + U)T_{ag}} + \frac{T_{db}}{T_{agdb}}$$

$$M_{min} = \frac{1/T_{ag} + 1/T_{agdb}}{(1+U)/T_{db} - U/T_{agdb}}$$

スケジューラありの方が処理時間が短い場合では，$P_{no} - P = P^1_{no} - P^{A1} - P^{B2} \geq 0$ であり，この条件を満たす M の上限値 M_{max2} は下記のように

なる．

$$M_{max2} = \frac{T_{db}}{(1+U)T_{ag}} + \frac{(1-C_{no}+U-C_{ag}U)T_{db}}{(1+U)(1-C_{ag})T_{agdb}}$$

スケジューラなしの方が処理時間が短くなる M の領域は $M_{max} \geq M \geq M_{max2}$ で与えられる領域である．ここで M_{max}，M_{max2} は T_{ag}，T_{agdb}，T_{db}，C_{ag}，C_{no} で与えられること，図 7 より $C_{no} = F(C_{ag})$ であること，を考慮すると，T_{ag}，T_{agdb}，T_{db} が与えられると，$M_{max} \geq M_{max2}$ となる C_{ag}，M_{max}，M_{max2} の範囲が決まり，C_{ag} から利用者数 N の領域も決定される．ここで，$1 \geq C_{ag} \geq 0$ を満たさない場合，または，M_{max} と M_{max2} の間に整数値が存在しない場合はつねにスケジューラありの方が処理時間が短いということである．

Case3 条件 $P^1_{no} < P^2_{no}$ かつ $P^{A1} \geq P^{A2}$ かつ $P^{B1} < P^{B2}$ から，Case3 となる M の上限値 M_{max} と下限値 M_{min} が下記のように与えられる．

$$M_{max} = \frac{1/T_{ag} + 1/T_{agdb}}{(1+U)/T_{db} - U/T_{agdb}}$$

$$M_{min} = \frac{T_{db}}{(1-C_{no}+U)T_{ag}} + \frac{T_{db}}{T_{agdb}}$$

スケジューラありの方が処理時間が短い場合を考えると，$P_{no} - P = P^2_{no} - P^{A1} - P^{B2} \geq 0$ なので，スケジューラありの方が処理時間が短くなる M の下限値 M_{min2} は下記のようになる．

$$M_{min2} = \frac{C_{ag}T_{db}}{(C_{ag}(1+U)-C_{no})T_{ag}} + \frac{C_{ag}UT_{db}}{(C_{ag}(1+U)-C_{no})T_{agdb}}$$

したがって，スケジューラなしの方が処理時間が短くなる M の領域は $M_{min2} \geq M \geq M_{min}$ で与えられる領域である．

前章で行ったシミュレーションで $T_{ag} = 1500$ と 4500 に変更した 2 つのケースの Case1〜4 の M の境界，Case2 の M_{max2} と Case3 の M_{min2} を**図 9** と**図 10** に示す．Case2 の領域で M_{max2} よりも大きい領域と Case3 の領域で M_{min2} よりも小さい領域がスケジューラなしの方が性能が高い領域である．

$T_{ag} = 1500$ の場合では，$M = 8$〜15 でキャッシュ率が高い場合でスケジューラなしの方が処理時間が短くなる場合がある．たとえば，$M = 8$ かつキャッシュ率が 0.970 では，Case2 の領域に含まれ，スケジューラなしとありの秒あたりの処理件数はそれぞれ 9170.88, 9135.89 である．このときの利用者数は 1649485 である．$M = 10$ かつキャッシュ率が 0.980 では，Case3 の領域に含まれ，スケジューラなしとありの秒あたりの処理件数はそれぞれ 11766.09，11609.92 である．このときの利用者数は 2040816 である．

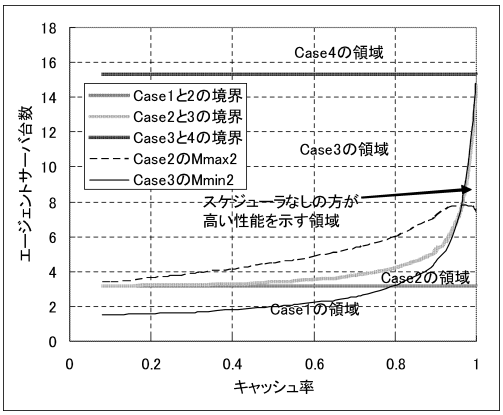


図 9 各ケースの領域（$T_{ag} = 1500$）
Fig. 9 Regions of each cases ($T_{ag} = 1500$).

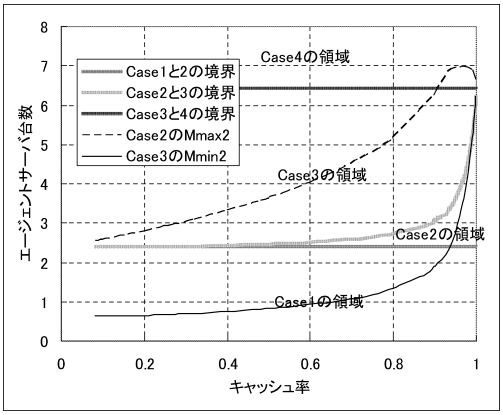


図 10 各ケースの領域（$T_{ag} = 4500$）
Fig. 10 Regions of each cases ($T_{ag} = 4500$).

$T_{ag} = 4500$ の場合では，Case2 の領域はつねに M_{max2} よりも下にあるので，つねにスケジューラありの方が高い性能を示すことが分かる．また，Case3 の領域もつねに M_{min2} よりも上にあるので，つねにスケジューラありの方が高い性能を示すことが分かる．つまり，$T_{ag} = 4500$ の場合ではつねにスケジューラありの方が処理時間が短いということである．

8. 関 連 研 究

通知型サービスの要素技術に Publish Subscribe 技術がある．サブスクライブの指定には，トピック，タイプ，条件などがある[15]．また，The Object Management Group は Notification Service として標準

化作業も行っている[16]．これらはイベントを配信すべき宛先を特定するための技術である．本稿で対象としているアプリケーションも通知型サービスであり，Publish Subscribe 技術の一般化と見ることもできるが，技術の核は個別の利用者のための処理の効率化であり，配信先を特定する技術ではない．

4.1 節で述べたように，本稿で示したスケジューラは，Cohort Scheduling[9] を基本としている．Cohort Scheduling の考えに基づいたものとして文献 10) がある．これは，複数のタスクが並行処理されるサーバプログラムを静的に解析し，Stage の抽出と各 Stage で必要なメモリを把握し，スケジューラがその情報を利用することでプロセッサの L2 キャッシュのヒット率を向上させるものである．これは L2 キャッシュの利用効率向上を行うので，このようなプログラムの静的解析が鍵となる．一方，本稿はキャッシュはメモリであるので，実行環境はエージェントという単位での制御で十分であり，プログラムの静的解析は必要としない．

9. ま と め

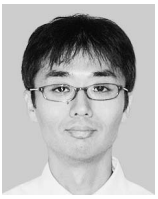
本稿では，個別通知サービスで，各利用者のための処理を大量に行う機構の高速化技術として，エージェントサーバのキャッシュを考慮したスケジューラを述べ，その性能評価を行った．これは Cohort Scheduling を基本とし，キャッシュにあるエージェントを優先的に処理することで，キャッシュヒット率を向上させ，性能向上を行うものである．本稿の評価では，スケジューラを使用しない場合と比べ最大 57%の性能向上を確認できた．また，エージェントサーバを複数台使用するシステムに対する効果をシミュレーションにより評価した．その結果，キャッシュヒット率が向上し，データベースサーバへのアクセス数を削減できるため，スケジューラなしの場合に高い性能を示し，89%の性能向上を示す場合もあることが確認された．さらに，シミュレーションモデルの解析により，スケジューラなしの方が高い性能を示す状況もあるが，多くの場合でスケジューラありの方が高い性能を示すことを検証した．

参 考 文 献

1) Koyanagi, T., Kobayashi, Y., Miyagi, S. and Yamamoto, G.: Agent Server for a Location-aware Personalized Notification Service, *International Workshop on Massively Multi-Agent Systems 2004*, pp.224–238, Springer (2004).
2) 野地英昭，宮城幸子：改札通過情報に連動した携帯電話向け e-mail 情報配信システムにおける Agent Framework を活用した高速化の実現，*OMRON TECHNICS*, Vol.46, OMRON (2005).
3) Tangosol, Inc.: Tangosol (2006). http://www.tangosol.com
4) JBoss Inc.: JBoss Cache (2006). http://www.jboss.org/products/jbosscache
5) IBM Corporation: WebSphere Exteded Deployment (2006). http://www.ibm.com
6) Robinson, J.T. and Devarakonda, M.V.: Data Cache Management Using Frequency-Based Replacement, *ACM SIGMETRICS Performance Evaluation Review*, Vol.18, No.1, pp.134–142 (1990).
7) Megiddo, N. and Modha, D.S.: Outperforming LRU with an Adaptive Replacement Cache Algorithm, *IEEE Computer*, Vol.37, No.4, pp.58–65 (2004).
8) O'Neil, E.J., O'Neill, P.E. and Weikum, G.: The LRU-K Page Replacement Algorithm For Database Disk Buffering, *Proc.1993 ACM SIGMOD international conference on Management of data*, pp.297–306, ACM (1993).
9) Larus, J.R. and Parkes, M.: Using Cohort Scheduling to Enhance Server Performance, *Proc. ACM SIGPLAN workshop on Languages, compilers and tools for embedded systems*, pp.182–187, ACM (2001).
10) Bhatia, S., Consel, C. and Lawall, J.: Memory-manager/Scheduler Co-design: Optimizing Event-driven Servers to Improve Cache Behavior, *Proc. 2006 International Symposium on Memory Management* (to be published), ACM (2006).
11) Koide, H. and Oie, Y.: A New Task Scheduling Method for Distributed Programs which Require Memory Management in Grids, *Proc. 2004 International Symposium on Applications and the Internet Workshops*, pp.666–676, ACM (2004).
12) Yamamoto, G.: Agent Server Technology for Managing Millions of Agents, *International Workshop on Massively Multi-Agent Systems 2004*, pp.1–12, Springer (2004).
13) Yamamoto, G. and Nakamura, Y.: Architecture and performance evaluation of a massive multi-agent system, *Proc. 3rd annual conference on Autonomous Agents*, pp.319–325, ACM (1999).
14) Yamamoto, G. and Tai, H.: Performance evaluation of an agent server capable of hosting large numbers of agents, *Proc. 5th international conference on Autonomous agents*, pp.363–369,

ACM (2001).
15) Eugster, P.T., Felber, P.A., Guerraoui, R. and Kermarrec, A.-M.: The Many Faces of Publish/Subscribe, *ACM Computing Surveys*, Vol.35, No.2, pp.114–131 (2003).
16) The Object Management Group: Notification Service Specification (2004). http://www.omg.org

(平成 18 年 5 月 19 日受付)
(平成 18 年 11 月 2 日採録)



山本 学（正会員）

1991 年東京工業大学大学院修士課程修了．同年日本 IBM（株）入社，東京基礎研究所配属．エージェント基盤技術および業務システム基盤技術の研究に従事．

田井 秀樹（正会員）

1997 年筑波大学大学院修士課程修了．同年日本 IBM（株）入社，東京基礎研究所配属．分散システムのミドルウェア等の業務システム基盤技術の研究に従事．