Webサービスを用いたワークフロー管理における負荷分散手法のシミュレーションによる評価

Simulation Evaluation of a Load Balancing Method for Workflow Management with Web Services

加藤 英之♡ 小林 隆志♢ 横田 治夫♢

Hideyuki KATOH Takashi KOBAYASHI
Haruo YOKOTA

ワークフロー管理システムでは大量の処理を効率よく行うために，負荷分散を考慮したスケジューリングを行うことが重要である．本稿では，Web サービス利用を前提としたワークフロー管理システムにおいて，各ワークフロー・エンジンから観察したアクティビティ・インスタンス毎の実行時間の履歴と各アクティビティ・インスタンスの処理負荷サイズの情報を用いたスケジューリング戦略による負荷分散手法を提案する．そのため，過去の履歴から環境も含めて算出したエグゼキュータの推定処理能力と次に実行するアクティビティ・インスタンスの処理負荷サイズからエグゼキュータを割当てる方法を提案する．さらに．シミュレータを用いて．ラウンドロビンやランダム割当てと性能の比較を行い，その効果を示す．

It is important for workflow management systems handling a large amount of processes to balance loads by the efficient scheduling. In this paper, we propose a load-balancing method of scheduling activity instances with the history of the size of an activity load and it execution time observed from workflow engines in a workflow management system with the Web service properties. Therefore it allocates activity instances from the estimated processing capacity calculated from that history including environment and its activity load size.We then compare the performance of proposed methods with ordinary round-robin and random scheduling to demonstrate the effect of proposed methods.

1. はじめに

プロセス (作業) の一部もしくは全てを自動化するものとしてワークフローが提案されている．ワークフローではプロセスをアクティビティ(工程) の集合とし，実際に流れる具体的なプロセスやアクティビティをそれぞれ，プロセス・インスタンス，アクティビティ・インスタンスと呼び，その自動化された プロセスを管理するシステムとしてワークフロー管理システムが提案されている [1, 2]．ワークフロー管理システムの導入により，人的コストの削減など様々なメリットが期待できる．現在，商用・研究用を問わず多数のワークフロー管理システムが存在する．近年その普及に伴って，処理する仕事量は膨大になってきており，負荷分散の重要性が増している．

一方，ネットワークを介して HTTP と SOAP を用いて，ソフトウェアシステム間で通信を行うオープンな技術である Web サービスが注目されている [3]．Web サービスは，実行環境に依存しないアプリケーションの連携が可能なため，CORBA や DCOM などの分散オブジェクト技術よりも利用が容易である．Web サービスが登場した当初は，SOAP，WSDL，UDDI などの基盤となる技術についての検証が行われ，その後 WS-Security，XML Signature などのセキュリティ・相互接続性の確保について研究されてきた．現在，BPEL4WS[1] , WS-Transaction[2] , Web Services Choreography[3] , WS-CAF[4] などを用いた Web サービスの実用化に向けた仕様の策定が行われている．

しかし，これらの仕様は障害対策や負荷分散などに関して，まだ十分に考慮されていない．ワークフロー管理の一般的な実現方法としては，1) 集中管理手法, 2) 分散管理手法, 3) エージェントによる手法, などがあり，障害対策，負荷分散共にそれぞれで考慮すべき点が異なる．我々はこれまでに，Web サービスを用いたワークフローにおける障害対策手法として集中管理に適した手法 [4]，分散管理，エージェントによる実現に適した手法 [5] を提案してきた．しかしながら，負荷分散に関しては明らかにされていなかった．

本稿では，Web サービス利用を前提とした集中管理型ワークフローにおける負荷分散手法として，スケジューリング戦略 OXTHAS (Observed Execution Time History and Activity Size based scheduling) を提案する．OXTHAS では，エグゼキュータ毎のワークフロー・エンジンから観測した各アクティビティ・インスタンスの過去の実行時間の履歴とそのアクティビティ・インスタンスの処理負荷サイズから, 各アクティビティを実行する環境等の差異を含めた推定処理能力を算出し，その推定処理能力と各アクティビティに対する処理負荷サイズを利用して，新たなアクティビティ・インスタンスの割当て先のエグゼキュータを決めることで負荷分散を実現する．

以下では，まず，本稿で前提とするワークフロー管理システムのモデルについて説明した後，実行した処理負荷サイズと実行時間の履歴から環境も含めたエグゼキュータの推定処理能力の算出式を示し，その推定処理能力と次に実行するアクティビティ・インスタンスの処理負荷サイズからエグゼキュータを割当てる方法について述べる．次に，シミュレータを用いて．ラウンドロビンやランダム割当てと性能の比較を行い，その効果を示す．最後に本稿をまとめ，今後の課題について言及する．

2. 前提とするワークフロー管理システムのモデル

2.1 プロセスモデル

ビジネスプロセスはグラフに例えられる．ビジネスプロセスはノードと矢印から成り立っており，ノードはアクティビティを意味し，矢印は依存関係を意味する．複数のアクティビティが矢印でつながり，それらがビジネスプロセスを構成する．この構成されたビジネスプロセスを自動化する際に，その制御や管理を行う部分がワークフロー・エンジンである．ワークフロー・エンジンはプロセスモデルの一部というよりはワークフロー管理システムのアーキテクチャの一部である．

2.2 ワークフロー管理システムのアーキテクチャ

本稿では，以下の集中管理型のワークフロー管理システムのアーキテクチャを対象とする (図 1)．図 1 のワークフロー管理システムは，主にワークフロー・エンジンとエグゼキュータから構成されている．その動作の概要は次の通りである．

1. ワークフロー・エンジン内のスケジューラは，クライアントからキューに入ってきたプロセス・インスタンスを 1 つずつ取り出し，その中で最初に実行すべきアクティビティ・インスタンスを決め，同時にそのプロセス・インスタンスの情報をプロセス・インスタンステーブルに持つ．あるいは，エグゼキュータから返ってきたアクティビティ・インスタンスが入っているキューから 1 つずつ取り出して，そのアクティビ

♡ 学生会員 東京工業大学 大学院 情報理工学研究科 計算工学専攻
katoh@de.cs.titech.ac.jp

♢ 正会員 東京工業大学 学術国際情報センター
{tkobaya,yokota}@cs.titech.ac.jp

[1] http://www-128.ibm.com/developerworks/library/specification/ws-bpel/
[2] http://www-128.ibm.com/developerworks/library/specification/ws-tx/
[3] http://www.w3.org/TR/ws-chor-reqs/
[4] http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=ws-caf

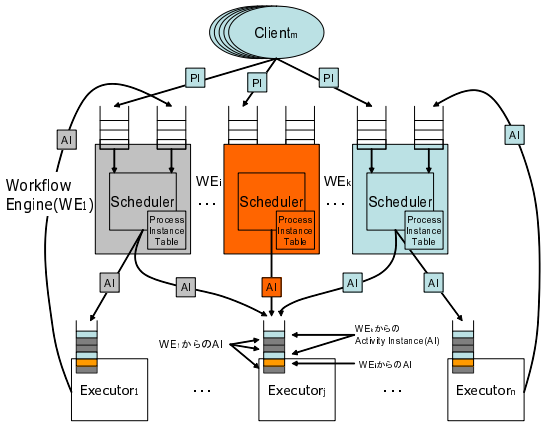

図 1: 前提とするワークフロー管理システムのアーキテクチャ
Fig.1 Assumed architecture of workflow management systems

ティ・インスタンスが属するプロセス・インスタンスに関して，プロセス・インスタンステーブルから情報を取り出し，次に実行すべきアクティビティ・インスタンスを決める．

2. スケジューラは，実行すべきアクティビティ・インスタンスの情報と，各エグゼキュータが実行可能なアクティビティの情報から，そのアクティビティ・インスタンスをどのエグゼキュータに割当てるかのスケジューリングを行い，対応するエグゼキュータの処理キューに入れる．
3. 各エグゼキュータもキューを持ち，キューの先頭にあるものから 1 つずつ処理していき，処理が終了したとき，ワークフロー・エンジンのキューにその処理結果を返す．
4. 以上を全てのプロセス・インスタンスの処理が終わるまで繰り返す．

上記の手順の中で，各アクティビティ・インスタンスをどのエグゼキュータに割当てるかの戦略によって，システム全体の負荷分散を行うことが可能となる．

ここでは，ワークフロー・エンジンは自分が処理しているプロセス・インスタンスやアクティビティ・インスタンスに関して，どこのエグゼキュータに処理のリクエストを渡しているかについては把握しているが，他のワークフロー・エンジンが処理しているプロセス・インスタンスやアクティビティ・インスタンスに関しては知らないものとする．

3. ワークフロー管理システムの負荷分散の概念

ここでは，前述のワークフロー管理システムのアーキテクチャにおいて，負荷分散を行うためのスケジューリング戦略について考える．我々は，負荷分散を行うために，ワークフロー・エンジンから観測した各アクティビティ・インスタンスの処理時間に着目する．

以下では，まず Web サービスを用いてワークフロー管理システムを構築する際に考慮すべき点について述べる．そしてワークフロー管理システムにおける負荷分散に関する関連研究と提案手法との相違を述べる．

3.1 Web サービスを用いたワークフロー管理システムの特徴

まず，ネットワーク経由での Web サービス利用を前提としたワークフロー管理システムを考える．その場合，

1. 1 つのエグゼキュータが 1 つの Web サービスである．
2. 複数のワークフロー管理システムが共生することを考慮すると，エグゼキュータを複数のワークフロー・エンジンで共有する場合がある．

といったことが考えられる．

まず (1) については，Web サービスは 1 種類以上のアクティビティ・インスタンスの処理をサービスとして提供することが考えられる．そのため，アクティビティ・インスタンス毎に処理の時間などが異なるため，各エグゼキュータの推定処理能力をアクティビティ・インスタンス毎に考慮する必要がある．なお，ここではエグゼキュータ内で同時に処理できるアクティビティ・インスタンスは 1 つに限ることとして議論する．

また，個々のアクティビティ・インスタンスの入力処理負荷サイズが一定でないことが処理時間に影響を与える場合を想定する必要がある．本稿では，処理時間はサイズに比例すると考えて推定処理能力を考えていく．

さらに，各エグゼキュータの能力や環境が一定でないことが考えられる．これは，単純にあるアクティビティ・インスタンスの処理能力が異なるということも考えられるし，ネットワークの良否による処理の遅延やリトライなどの発生が考えられる．

(2) については，この場合，エグゼキュータに溜っているキューの中のアクティビティ・インスタンスのリクエストは複数のワークフロー・エンジンから出されたリクエストの集合である．また，1 つのエグゼキュータを複数のワークフロー・エンジンが利用するだけでなく，そのエグゼキュータが提供している機能を Web サービス以外の形態で利用する可能性もある．そのため，ワークフロー・エンジンが知ることのできる情報は，ワークフロー・エンジンがエグゼキュータに処理を要求してから，その処理が完了してそのワークフロー・エンジンに戻ってくるまでの時間である．

以上の前提を踏まえて，ワークフロー管理システムで負荷分散を行うためのスケジューリング戦略を決定する必要がある．この時，各ワークフロー・エンジンが利用できる情報は，処理が完了したアクティビティ・インスタンスの処理負荷サイズと観測される実行時間の履歴である．そこで，それらの情報からスケジューリング戦略を決定する手法を提案する．それらは，ネットワークの状況やエグゼキュータ上での他の処理といった環境も考慮したものとなる．なお，これらの情報は，ワークフロー・エンジン毎に異なるものとなる．

3.2 関連研究

Lie-jie Jin らは，プロセス・インスタンスを一定時間毎にワークフロー・エンジンに投入する際にワークフロー・エンジンでの処理前の待ち時間を減らすようにどのワークフロー・エンジンに入れるかについての負荷分散手法を提案している [6]．その中で分散ワークフロー管理システムにおけるワークフロー・エンジン間の負荷分散を行うために負荷指標を定義している．しかし，今回のように複数のワークフロー・エンジンがエグゼキュータを共用する場合には，それぞれのエグゼキュータの負荷を計算し，その負荷に応じて処理を振り分ける負荷分散が重要となってくるが，この研究ではエグゼキュータ間の負荷分散に関しては対応していない．

また，Koji Nonobe らは，資源制約付きスケジューリング問題に関するアルゴリズムを提案している [7]．これは，ワークフロー・エンジンが 1 つであるような閉じたワークフロー管理システムにおける仕事の割り当てに関しては最短で処理が完了する割り当て方を提供してくれる．しかし，複数のワークフロー・エンジンがあり，それぞれのワークフロー・エンジンが自身の仕事にのみ割り当てや監視を行っているような場合や，ネットワークの環境が動的に変化するような場合，このアルゴリズムをそのまま利用することはできない．

4. OXTHAS スケジューリング戦略

以上の内容をふまえ，各ワークフロー・エンジンから見たエグゼキュータの環境も含めた推定処理能力を算出する方法と，その推定処理能力を用いたスケジューリング戦略である OXTHAS を提案する．

4.1 推定処理能力

まず以下のように変数を定義する．

- プロセス・インスタンス番号: $P = \{p_1, \ldots p_i, \ldots p_l\}$
- アクティビティ番号: $A = \{a_1, \ldots a_j, \ldots a_m\}$
- エグゼキュータ番号: $E = \{e_1, \ldots e_k, \ldots e_n\}$
- p_i における a_j の処理負荷サイズ: s_{p_i,a_j}
- p_i における a_j が e_k で処理可能であるかどうかを示すマッピング:m_{p_i,a_j,e_k}
  エグゼキュータで処理可能な場合は 1，そうでない場合は 0 となる．
- p_i における a_j の観測実行時間: t_{p_i,a_j}
  観測実行時間 t_{p_i,a_j} はワークフロー・エンジンがエグゼキュータにリクエストを出してから，処理が終わってワークフロー・エンジンに返ってくるまでの時間を表す．つまり，処理時間の他にエグゼキュータのキュー内での待ち時間，ネットワークの遅延やリトライの時間などを含む．

以上の変数を用いて，エグゼキュータ e_k における各アクティビティ a_j の推定処理能力 c_{e_k,a_j} を以下の式で算出する．

$$c_{e_k,a_j} = \Big\{\sum_{p_i=1}^{l}\Big(\frac{s_{p_i,a_j}}{t_{p_i,a_j}} \cdot m_{p_i,a_j,e_k}\Big)\Big\} \Big/ \Big(\sum_{p_i=1}^{l} m_{p_i,a_j,e_k}\Big)$$

この推定処理能力 c_{e_k,a_j} は単位時間当たりの仕事量の平均値を過去の処理時間から求めている．

推定処理能力 c_{e_k,a_j} は，値が大きいほど，そのエグゼキュータの処理能力が大きいことを示す．しかし，c_{e_k,a_j} の値が大きいことには他の要因も考えられる．例えば，そのエグゼキュータがあまり使用されていないために，待ち時間がほとんどない場合，他のエグゼキュータよりも c_{e_k,a_j} の値が大きくなることがある．また，そのエグゼキュータで処理をしていたアクティビティ・インスタンスはそのエグゼキュータの得意なアクティビティ・インスタンスを多く処理していたため，他のアクティビティ・インスタンスの c_{e_k,a_j} の値も相乗して大きくなってしまう場合もある．

4.2 OXTHAS-N

上述したように，推定処理能力 c_{e_k,a_j} は，基本的に値が大きいほど，そのエグゼキュータの処理能力が大きいことを示すため，アクティビティ・インスタンスを c_{e_k,a_j} の値が最も大きいエグゼキュータに投げるようにするのが良いと考えられる．

しかし、単に c_{e_k,a_j} の値が大きいところにだけアクティビティ・インスタンスを割り当てていくということをしていくと，プロセス・インスタンスの数が多く，長い期間のスケジューリングを考えた場合には，最良の戦略とは言えない．それは，c_{e_k,a_j} この値は，過去の情報からの値であるため，ある種の傾向を示してはくれるが，その時その時で柔軟に変化をしてくれる値ではないため，ある時刻に，たくさんの同一種のアクティビティ・インスタンスのリクエストがワークフロー・エンジンに要求された場合，c_{e_k,a_j} の値が最も高い 1 つのエグゼキュータに集中してしまうことになり，結果としてキュー長に偏りができてしまう．そして，しばらくすると，そのエグゼキュータの c_{e_k,a_j} の値は極端に下がってしまうことになる．そこで，本稿ではより効率的にアクティビティ・インスタンスの処理を行うために，アクティビティ・インスタンスの処理負荷サイズに注目する．

アクティビティ・インスタンスのサイズが大きい場合は，当然 c_{e_k,a_j} の値が最も大きいところに割り当てるべきだが，逆にサイズが小さい場合は，必ずしも c_{e_k,a_j} の値が最も大きいところに割り当てなくても，次に c_{e_k,a_j} の値が良いところに割り当てる方が偏りが減って効率的である．そこで，候補となる N 個のエグゼキュータに対し，アクティビティ・インスタンスの処理負荷サイズを分割するための閾値を設け，処理負荷サイズに応じた割り当てを行う負荷分散手法 OXTHAS-N を提案する．

アクティビティ・インスタンスの処理負荷サイズを分割するための閾値は，w_i = (サイズのボーダーライン)，n = (分割数)，S_{max} = (過去の履歴の中でのアクティビティ・インスタンスの最大サイズ)

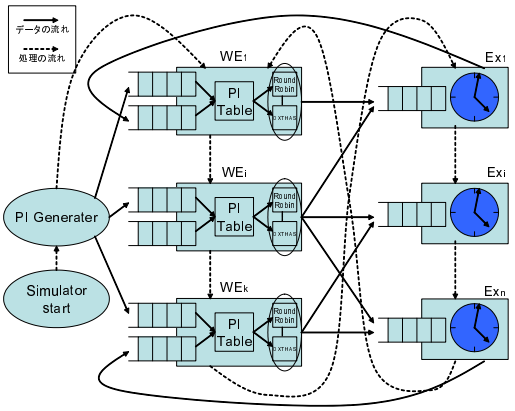

図 2: 実現したシミュレーションシステムの制御とデータの流れ
Fig.2 Control and data flow of the implemented simulation system

とすると以下のように定められる．

$$w_i = \Big(\sum_{j=1}^{i} j \Big/ \sum_{k=1}^{n} k\Big) \cdot S_{max}$$

例えば，2 つのエグゼキュータに負荷を分散する OXTHAS-2 では，割り当てを決定する区切りのサイズの決定方法に関しては，過去の履歴から最も大きいサイズをとりだして，そのサイズを 1 番目と 2 番目に処理能力の高いエグゼキュータに処理を担当するサイズの幅が 2:1 の割り当てになるようにサイズの閾値する．3 つのエグゼキュータ間の負荷分散を行う OXTHAS-3 では同様に，3:2:1 の比率になるようにサイズの閾値を決定する．

5. シミュレーションによる実験

5.1 シミュレータの構成

先ほど示した推定処理能力とそれを用いた戦略の有効性を示すためにシミュレーションを行った．1 台の計算機上で図 2 のようなワークフロー管理システムの制御とデータの流れを擬似的に再現した．このシミュレーションで，生成した全プロセス・インスタンスが終了する時間をステップ数で計測した．図 2 の破線の 1 周が 1 ステップである．

最初に，PI Generator がプロセス・インスタンスを生成し，各ワークフロー・エンジンに送る．ワークフロー・エンジン，エグゼキュータでは，前に説明したアーキテクチャにおけるワークフロー・エンジン，エグゼキュータの動作と同様である．これをステップを 1 つずつ増やしながら破線の流れで繰り返していって，すべてのプロセス・インスタンスが終了したときの処理ステップを測定する．なお、今回はネットワークの遅延やリトライについては考えないものとし，処理は失敗せず，必ず完了するものとする．

5.2 シミュレーションの内容

最初に，プロセス・インスタンスの数を変えていって，アルゴリズム間で比較を行った．ラウンドロビンとランダムなどの一般的なアルゴリズムと OXTHAS-N($N = 1, \cdots, 6$) で比較を行った．推定処理能力は過去の履歴から，求められるものであるため，初めの 1000 ステップまでは，ランダムアルゴリズムを用いた．それぞれ 20 回測定を行い，その平均を取った．また，エグゼキュータの性能の違いによる処理ステップ数を測定し，同様の比較を行った．このときのプロセス・インスタンスの数は 1000 とした．

エグゼキュータは 6 台，ワークフロー・エンジンは 2 台とし，各ワークフロー・エンジンでは同じ数のプロセス・インスタンスを処理し，それぞれのプロセス・インスタンスをどこで処理するかなどはそのワークフロー・エンジンしか管理していない．ビジネスプロセスは分岐などのない単純なものとした．エグゼキュー

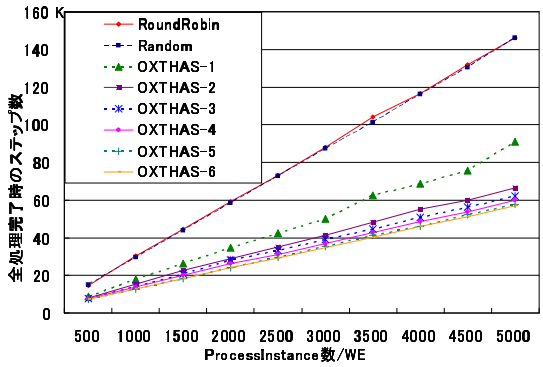

図 3: 各プロセス・インスタンス数における総処理ステップ数
Fig.3 Total execution steps of each method under different number of process instances

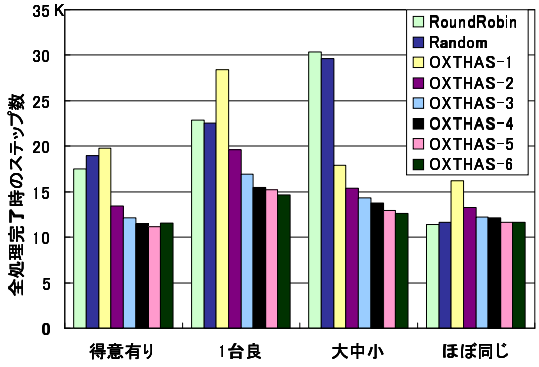

図 4: エグゼキュータの性能差の違いによる総処理ステップ数
Fig.4 Total execution steps of each method under different settings of executor performance

タの処理性能や，プロセス・インスタンス，アクティビティ・インスタンスのサイズなどはランダムで決定される．また，すべてのプロセス・インスタンスが持つアクティビティ・インスタンスの数は 5 として，すべて同じにしてある．そして，各エグゼキュータではすべてのアクティビティ・インスタンスに関して処理可能であるとする．

5.3 結果と考察

図 3 は，各手法についてプロセス・インスタンスの数を変えて処理時間を計測した結果である．図 3 を見ると，提案手法である OXTHAS-N はラウンドロビンやランダムを用いた手法よりも良い値を示している．さらに，OXTHAS-N の N の値が大きいほど，その効果はより大きいことがわかる．これにより，提案した OXTHAS スケジューリング戦略が有効であることが言える．

図 4 では，エグゼキュータの性能を特徴別に分けて処理ステップを測定した結果である．プロセス・インスタンスの数は 1000 とした．「1 台良」は 1 台のエグゼキュータだけ処理性能が他のよりよい場合であり，「大中小」はエグゼキュータの処理性能を 2 台ずつ良いものと悪いものとその中間のものとした場合である．「得意有り」というのは，各エグゼキュータに 1 つのアクティビティに関してのみ，処理性能を非常に高くしてある．「ほぼ同じ」はエグゼキュータの処理性能がどれもほとんど同じであることを表している．グラフから，OXTHAS-1 はラウンドロビンやランダムの手法と比較すると良い結果とは言えない．しかし，エグゼキュータが 3 台のときはラウンドロビンやランダムの手法とはそれほど差異は見られなかった．そのため，これは前述したキュー長の偏りが出たと考えられ，エグゼキュータが 6 台であったため，3 台の時よりも，その傾向が強く出てしまったと考えられる．N=2 以降の場合は，OXTHAS-N の方が有効であり，特にエグゼキュータ間の性能が大きく異なる場合，提案手法が有効であることがわかる．

6. おわりに

本稿では，Web サービスを用いた集中管理型のワークフロー管理システムにおける負荷分散手法を提案した．具体的には，ワークフロー管理システムのモデルを説明し，それに基づいてワークフローと Web サービスの特徴を考慮に入れた負荷指標の概念について説明し，各アクティビティ・インスタンスに対する作業量と過去の実行時間の履歴に基づいて，各アクティビティ・インスタンスを実行する環境等の差異を吸収する負荷指標負荷指標を提案した．さらに，提案した推定処理能力用いて，アクティビティ・インスタンスの処理負荷サイズを考慮に入れた負荷分散手法である OXTHAS を提案した．そして，シミュレーションを行うことで提案手法の有効性を示した．

今後の課題としては，ネットワークの遅延やエグゼキュータの信頼性を考慮していくことや，OXTHAS-N における N 分割する際の分割方法について，より良い方法を検討することが挙げられる．また，実際にこのワークフロー管理システムを実現して，実際のシステムにおいても同様の有効性がいえるかを検証することが重要である．

[謝辞]

本研究の一部は，文部科学省科学研究費補助金特定領域研究 (16016232)，独立行政法人科学技術振興機構 CREST，および 21 世紀 COE プログラム「大規模知識資源の体系化と活用基盤構築」の助成により行なわれた．

[文献]

[1] Workflow Management Coalition. http://www.wfmc.org/.

[2] 戸田保一, 飯島淳一, 速水治夫, 堀内正博. ワークフロー -ビジネスプロセスの変革に向けて-. 株式会社日科技連出版社, 1998.

[3] 株式会社日本ユニテック DigitalXpress 編集部. SOAP UDDI WSDL Web サービス技術基礎と実践 徹底解説. 技術評論社, 2002.

[4] 加藤英之, 小林隆志, 横田治夫. ワークフローの自動実行における障害対策. In *Proc. of DEWS2004*. I-12-02, 3 2004.

[5] Neila Ben LAKHAL, Takashi Kobayashi, and Haruo Yokota. Throws: An architecture for highly available distributed execution of web services compositions. In *Proc. of RIDE WS-ECEG'2004*, pp. 103–110. IEEE, 3 2004.

[6] Li jie Jin, Fabio Casati, Mehmet Sayal, and Ming-Chien Shan. Load balancing in distributed workflow management system. In *Proc. of SAC2001*, 8 2001.

[7] Koji Nonobe and Toshihide Ibaraki. Formulation and tabu search algorithm for the resource constrained project scheduling problem. In C.C. Ribeiro and P. Hansen, editors, *Essays and Surveys in Metaheuristics*, pp. 557–588. Kluwer Academic Publishers, 2002.

加藤 英之 Hideyuki KATOH
平 16 東工大・工・情工卒．同大大学院・情報理工・計算工・修士課程在学中．日本データベース学会学生会員．

小林 隆志 Takashi KOBAYASHI
平 9 東工大・工・情報工学卒．平 11 同大大学院・情報理工・計算工学・修士課程了．平 16 同大大学院・同専攻・博士課程了．平 14 同大学術国際情報センター・助手．工博．日本データベース学会，日本ソフトウェア科学会，情報処理学会，ACM 各会員．

横田 治夫 Haruo YOKOTA
昭 55 東工大・工・電物卒．昭 57 同大大学院・情報・修士課程了．同年富士通 (株)．同年 6 月 (財) 新世代コンピュータ技術開発機構研究所．昭 61(株) 富士通研究所．平 4 北陸先端大・情報・助教授．平 10 東工大・情報理工・助教授．平 13 東工大・学術国際情報センター・教授．工博．日本データベース学会，電子情報通信学会，情報処理学会，人工知能学会，IEEE，ACM 各会員．