論　文

OXTHAS：Webサービスベースのワークフロー管理における障害を考慮した負荷分散手法***

加藤　英之†*　　小林　隆志††**　　横田　治夫††,†

OXTHAS: Failure-Aware Activity Scheduling in Web Services Based Workflow Management***

Hideyuki KATOH†*, Takashi KOBAYASHI††**, and Haruo YOKOTA††,†

あらまし　ワークフロー管理システムでは，アクティビティを適切に割り当てることが処理効率の改善につながる．本研究では，近年注目されている Web サービスを用いたワークフローに着目し，その特徴を考慮したアクティビティの割当方法を提案する．Web サービスを前提とすると，一つの Web サービスが特定のワークフローエンジンのみから利用されるとは限らないため負荷状況を正確に把握できない．提案手法では，Web サービスの過去の処理履歴を利用して負荷状況を推定し，処理負荷サイズを考慮してアクティビティを効率良く割り当てる．更に本研究では，より実際の環境に対応させるべく，ネットワークやハードウェア等の障害を考慮に入れ，タイムアウトを設定し，効率の良い処理を可能にする再割当手法を提案する．そして，シミュレーションにより，提案手法の評価を行う．

キーワード　Web サービス，ワークフロー，Web とインターネット

1. まえがき

プロセス（作業）の一部若しくはすべてを自動化するものをワークフローと呼ぶ．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，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(注1)，WS-BusinessActivity や WS-AtomicTransaction(注2)，WS-Choreography(注3)，WS-CAF(注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．

† 東京工業大学大学院情報理工学研究科計算工学専攻，東京都
Department of Computer Science, Graduate School of Information Science and Engineering, Tokyo Institute of Technology, 2–12–1 Ookayama, Meguro-ku, Tokyo, 152–8552 Japan
†† 東京工業大学学術国際情報センター，東京都
Global Scientific Information and Computing Center, Tokyo Institute of Technology, 2–12–1 Ookayama, Meguro-ku, Tokyo, 152–8550 Japan
* 現在，東芝ソリューション株式会社
** 現在，名古屋大学大学院情報科学研究科
*** 本論文はデータ工学研究専門委員会推薦論文である．

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

BPEL4WS は XML ベースのワークフロー定義言語であり，WS-BusinessActivity や WS-AtomicTransaction はアプリケーションの動作やトランザクションの信頼できる結果を調整する方法を提供する．WS-Choreography は複数の Web サービスの協調動作を実現する．また，WS-CAF 仕様は，複数の Web サービス間でトランザクションに関する重要な情報を共有できるようにシステムの設定を行う．これらの仕様の登場により，ワークフローを Web サービスを用いて実現することが盛んに研究されるようになってきた．

しかし，これらの仕様では，障害対策や負荷分散などに関しては十分に言及されていない．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，負荷分散とも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，負荷分散に関しては言及していなかった．

そこで，本研究では Web サービス利用を前提とした集中管理型ワークフロー管理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) [6]〜[8] を提案する．

OXTHAS では，エグゼキュータごと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処理能力の推定値（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インスタンス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先のエグゼキュータを決定することで負荷分散を実現する．また，エグゼキュータの故障やネットワークの障害等を考慮し，それらに対応するために OXTHAS 再スケジューリング戦略とその核となる 2 種類の調整法を提案する．ネットワークやエグゼキュータの障害に対応するために OXTHAS にタイムアウトの概念を導入し，タイムアウトが起こった場合には，そのエグゼキュータで発生したタイムアウトの回数も考慮に入れて推定処理能力を再計算，若しくはタイムアウトの値を調整し，再スケジューリングを行う．

以下では，まず，本研究の対象となるワークフロー管理のアーキテクチャについて説明した後，Web サービス利用を前提としたワークフロー管理の特徴について述べる．そして，その特徴を踏まえた上で，提案手法である OXTHAS について述べる．次に，このワークフロー管理アーキテクチャにおけるタイムアウトの概念について言及し，エグゼキュータやネットワークの障害に対応するためにタイムアウトを利用した OXTHAS 再スケジューリング戦略と 2 種類の調整法について説明する．そして，シミュレータを用いて，ランダムやラウンドロビンなどのアルゴリズムと OXTHAS の比較評価を行う．更にタイムアウトを用いた OXTHAS の性能評価を行い，その効果を示す．最後に本研究をまとめ，今後の課題について言及する．

2. 対象となるワークフロー管理アーキテクチャ

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（図 1）．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．

2.2 ワークフロー管理アーキテクチャ

本研究では，図 2 のような集中管理型のワークフロー管理アーキテクチャを対象とする．図 2 のワークフロー管理アーキテクチャは，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．ワークフローエンジンとは，前述したビジネスプロセスを自動化するための制御や管理を行う部分である．このシステムの動作の概要は次のとおりである．

2.2.1 動　　作

（1） ワークフローエンジン内のスケジューラは，

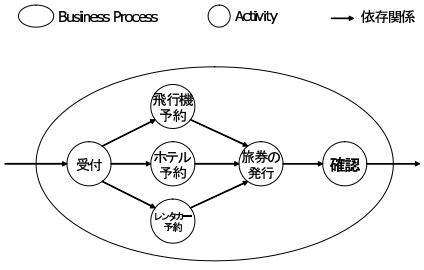

図 1　ワークフローのプロセスモデル
Fig. 1　A process model of a workflow.

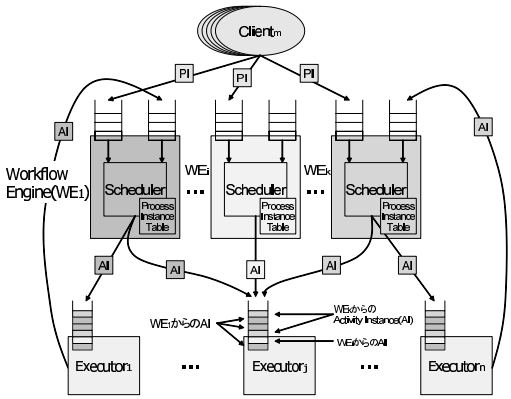

図 2　前提とするワークフロー管理アーキテクチャ
Fig. 2　Assumed workflow management architecture.

クライアントからのキューに入ってきたプロセスインスタンスを一つずつ取り出し，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る．それと同時にスケジューラは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格納する．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一つずつ取り出して，そのアクティビ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り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もち，キューの先頭にあるアクティビティインスタンスから一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） ワークフローエンジンはエグゼキュータからの処理結果について，ACK（処理結果の受理）若しくは NACK（処理結果の拒否）を同一のエグゼキュータのキューに入れる．

（5） エグゼキュータは ACK/NACK を受けたことをワークフローエンジンに知らせるために，ACK/NACK の受理をワークフローエンジンのキューに入れる．

（6） ワークフローエンジンはエグゼキュータから ACK を受けたとき，次のアクティビティインスタンスについての割当を行う．

（7） 以上をすべ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．

2. 2. 2　負荷分散と障害対策の前提

2.2.1 の手順において，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り当てるかを考慮することによって，システム全体の負荷分散を行うことが可能となる．手順 (2) のところで，一定時間を経過しても (3) が行われなかった場合，タイムアウトが発生しワークフローエンジンは別のエグゼキュータにアクティビティインスタンスを割り当てる（リトライ）．このとき，同じアクティビティインスタンスのリクエストが複数回発生することになるが，(4) により，そのうちの一つだけを採用するようにしている．そして，リトライが起こった場合，最初に割り当てたエグゼキュータから処理結果が返ってきてもそのエグゼキュータには NACK を送信し，必ず，最後に割り当てたエグゼキュータからの処理のみを採用する．このリトライの際に，タイムアウトをどのように設定するか，どのように再割当を行うかの戦略によってエグゼキュータ障害時の対策と障害を考慮した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複数存在し，各ワークフローエンジンは任意のエグゼキュータを共用することができることを前提とする．また，ワークフローエンジン同士，エグゼキュータ同士は独立しており，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の情報は知らないものとする．

2. 3　Web サービスベースのワークフロー管理で考慮すべき点

前述のワークフロー管理アーキテクチャをもととした Web サービスベースのワークフロー管理を対象とするとき，以下のような特徴が考えられる．

（1） 一つのエグゼキュータの行うアクティビティインスタンスの処理が一つの Web サービスである．

（2） 個々のワークフローエンジンはエグゼキュータの内部情報を知ることができない．

まず (1) については，エグゼキュータは 1 種類以上

のアクティビティインスタンスの処理を Web サービスとして提供することが考えられる．そのため，個々のアクティビティインスタンスごとにエグゼキュータの処理能力を考慮して割当を行う必要がある．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一つに限ることとして議論する．複数のアクティビティインスタンスを実行する環境は，物理的なプロバイダに複数の論理的なエグゼキュータをマップすることでモデル化できる．

また，個々のアクティビティインスタンスの入力データのファイル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について考慮する必要がある．本研究では，処理時間はファイルサイズに比例すると考える．入力データには様々な種類が考えられるが，あるアクティビティインスタンスにおいて入力データの種類は一つであり，処理時間は比例されると前提する．更に，各エグゼキュータの能力や環境が一定でないことが処理時間に影響を与えることが考えられる．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 Web サービスを提供するプロバイダの環境等によって異なる場合や，ネットワークの良否による処理の遅延やタイムアウトの発生などの場合が考えられ，これらを考慮する必要がある．

(2) に関しては，複数のワークフローエンジンが共存することを想定する場合，いくつかのワークフローエンジンがエグゼキュータを共用する可能性がある．そのため，ワークフローエンジンはエグゼキュータのキュー長を知ることができない．このとき，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リクエストを出してからエグゼキュータから処理が終わってそのワークフローエンジンに戻ってくるまでの時間になる．そして，この時間にはエグゼキュータ内の処理時間のほかにキューでの待ち時間やネットワークの遅延時間なども含まれる．

以上の前提を踏まえて，我々は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 OXTHAS を提案する．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入力データのファイルサイズと観測された実行時間（観測実行時間）の履歴であるため，OXTHAS ではそれらの情報からアクティビティインスタンスの割当を決定する．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ごとに異なるものとなる．

3. OXTHAS スケジューリング戦略

ここでは，我々が提案する OXTHAS スケジューリング戦略について説明する．**2.** で挙げた点を前提とし，本研究では，各 プロセスインスタンスごとにアクティビティインスタンスの数や種類は同一であるとする．アクティビティインスタンスの数や種類が同一でない場合についての妥当性の証明は今後の課題とするが，本手法では，該当アクテビティを実行できるエグゼキュータを対象とした推定処理能力計算と，アクティビティインスタンスごとの割当を行っており，アクティビティインスタンスの数及び種類には依存していないため，数及び種類が同じではなくても適用できると考える．

3.1 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：$P = \{p_1, \ldots p_i, \ldots p_l\}$
- プロセスインスタンス番号：i
- アクティビティインスタンス：$A = \{a_1, \ldots a_j, \ldots a_m\}$
- アクティビティインスタンス番号：j
- エグゼキュータ：$E = \{e_1, \ldots e_k, \ldots e_n\}$
- エグゼキュータ番号：k
- p_i における a_j の処理負荷サイズ：$s_{i,j}$
- 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：$m_{i,j,k}$

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：$t_{i,j}$

処理負荷サイズ $s_{i,j}$ は，アクティビティインスタンス a_j の入力データのファイルサイズを表し，観測実行時間 $t_{i,j}$ 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，エグゼキュータで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実際に観測した時間を表す．つまり，処理時間のほか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の時間などを含む．また，リトライが起こった場合，そのアクティビティインスタンス処理がそのエグゼキュータから返ってくるまでは，タイムアウトの値を暫定的な観測実行時間として推定処理能力を計算する．アクティビティインスタンスの処理時間はエ

グゼキュータによって異なるが，$t_{i,j}$ は過去に処理されたアクティビティインスタンスの観測実行時間であるため，k には依存されない．

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 $c_{k,j}$ を以下の式で算出する．

$$c_{k,j} = \frac{\sum_{i=1}^{l}(\frac{s_{i,j}}{t_{i,j}} \cdot m_{i,j,k})}{\sum_{i=1}^{l} m_{i,j,k}}$$

この推定処理能力 $c_{k,j}$ は単位時間当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 $c_{k,j}$ 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，$c_{k,j}$ 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 $c_{k,j}$ 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 $c_{k,j}$ の値も相乗して大きくなってしまう場合もある．

3.2 OXTHAS-N

アクティビティインスタンスの処理負荷サイズが大きい場合は，当然 $c_{k,j}$ の値が最も大きいところに割り当てるべきだが，逆に処理負荷サイズが小さい場合は，必ずしも $c_{k,j}$ の値が最も大きいところに割り当てなくても，次に $c_{k,j}$ 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，OXTHAS-N では候補となる $c_{k,j}$ の値の高い N 台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しきい値を設け，処理負荷サイズに応じた割当を行う．

このとき，しきい値の決定方法は等分割，整数比分割，$c_{k,j}$ の値に応じた分割（c 値分割）等いくつか考えられる．

整数比分割とは，OXTHAS-N に対して処理負荷サイズを大きい方から N:N-1: ･･･: 1 と分割する方法である．c 値分割とは，分割する際の比率を候補となるエグゼキュータの推定処理能力の値によって決定する方法である．整数比分割（逆）とは，処理負荷サイズの大きい方からではなく，小さい方から例えば 3:2:1 に分割する方法である．c 値分割（逆）も同様である．等分割は，処理負荷サイズの大きい方から 1:1:1 ･･･ と分割する方法である．

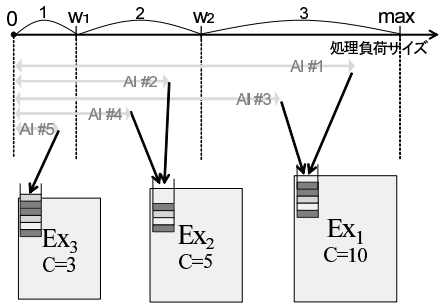

図 3　OXTHAS-N スケジューリングの例 ($N = 3$)
Fig. 3　An example of OXTHAS-N scheduling. ($N = 3$)

本研究では，どの方法が適しているか実験を行い，整数比分割が最適であったため，以下ではこれを用いる．実験の詳細な結果に関しては，**5.2.1** で述べる．

整数比分割では，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しきい値は，w_i = (処理負荷サイズのボーダライン)，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) とし，次のように定める．

$$w_i = \frac{\sum_{j=1}^{i} j}{\sum_{k=1}^{n} k} \cdot S_{max}$$

例えば，OXTHAS-3 では，推定処理能力の値が高い上位 3 台のエグゼキュータを対象として，過去の履歴から最も大きい処理負荷サイズを取り出して，処理を担当する処理負荷サイズの幅が 3:2:1 の比率になるように処理負荷サイズのしきい値を決定する（図 3 を参照）．

4. 障害の考慮

3. で説明した OXTHAS スケジューリング戦略では，障害対策に関して言及しなかった．そこで，この章ではエグゼキュータやネットワークの障害に対応するための OXTHAS 再スケジューリング戦略と調整法に関して説明を行う．

4.1 OXTHAS 再スケジューリング戦略

エグゼキュータからの応答がない場合，その理由としてネットワークの混雑とネットワーク若しくはエグゼキュータの故障が考えられる．その理由がネット

ワークの混雑の場合，処理が遅くなるだけで，処理自体はそのうち行われ，その混雑の程度に関しては既に推定処理能力で対応されている．しかし，ネットワークやエグゼキュータの故障が理由の場合，処理が停止してしまうことが予想される．そこで，我々はタイムアウトの概念を導入し，ネットワークやエグゼキュータの故障に対応する．このとき，タイムアウト発生時の再割当方法を OXTHAS 再スケジューリング戦略と呼ぶ．

OXTHAS 再スケジューリング戦略では，タイムアウトが発生してアクティビティインスタンスを再び割り当てるときに，再割当を行うアクティビティインスタンスを既に割り当てたエグゼキュータには基本的には割り当てないようにする．そのため，もしエグゼキュータが 10 台の場合，同じアクティビティインスタンスで再割当が発生するたびに割当対象となるエグゼキュータの数も 9 台，8 台 … と減っていく．そして，再割当対象エグゼキュータ台数 N_{exe} と OXTHAS-N の N の関係が，

$$N_{exe} < N$$

となってしまった場合，OXTHAS-N_{exe} で再割当を行う．

4.2 調 整 法

タイムアウトを導入する際，故障の影響をなるべく抑える必要がある．故障の影響を抑える方法としては，故障の可能性があるエグゼキュータにアクティビティインスタンスをなるべく割り当てないようにする方法と，たとえ故障の可能性のあるエグゼキュータに割り当てられてもすぐにタイムアウトを発生させることで故障の影響を抑える方法が考えられる．前者を推定処理能力調整法，後者をタイムアウト値調整法と呼ぶ．

4.2.1 推定処理能力調整法

先ほど定義した推定処理能力にタイムアウトの発生時を考慮に入れて，$c'_{k,j}$ を以下のように再定義する（R_k はそのエグゼキュータでタイムアウトが発生した回数．α は定数）．

$$c'_{k,j} = \frac{c_{k,j}}{1 + \alpha \cdot R_k}$$

タイムアウトの発生時，その推定処理能力は $\alpha = 1$ とすると 2 分の 1，3 分の 1 … となり，そのエグゼキュータにアクティビティインスタンスを割り当てないようになる．

4.2.2 タイムアウト値調整法

タイムアウトが発生した場合はそれ以降に割り当てるアクティビティインスタンスに関して，タイムアウトの値 T_k を次のように定める．（β, T_{init} は定数）

$$T_k = T_{init} - \beta \cdot R_k$$

タイムアウトの値は $T_{init} = 1000$，$\beta = 100$ とすると 900，800 … と減っていき，故障したエグゼキュータに割り当てられてもすぐにタイムアウトが発生し，再割当が起こる．

4.2.3 故障傾向の考慮

もし，タイムアウトが発生した後にそのエグゼキュータから処理が返ってきた場合，故障検知ミスか故障からの復旧を意味するため，$c'_{k,j}$ や T_k の値を最初に定義した値に戻す．この方法をとることで，一時的な故障に対応することが可能となり，更に故障傾向が高い場合は，タイムアウトのペナルティを大きくするという効果がある．

5. 評　　価

5.1 シミュレーション内容

前述した OXTHAS スケジューリング戦略の有効性と OXTHAS におけるタイムアウトの利用方法の検証と再スケジューリング戦略の有効性を示すために **2.2** で説明した対象となるワークフロー管理アーキテクチャに基づいたシミュレータを作成し，シミュレーションを行った．最初に，OXTHAS-N の領域分割方法の違いによる総処理ステップ数を比較し，最適な領域分割方法を決定する．次に，OXTHAS スケジューリング戦略とランダムやラウンドロビンを比較する．まずエグゼキュータの処理能力の値をいろいろ変化させたときについて，総処理ステップ数を比較し，その後プロセスインスタンスの数を変化させたときの総処理ステップ数について比較する．各シミュレーションに関しては，それぞれ 20 回の測定を行い，その平均をとる．また，ここまでのシミュレーションでは故障を考慮しないため，ワークフローエンジンがエグゼキュータに ACK/NACK を通知する部分を省いた処理プロトコルになっている．

次に，1 台のエグゼキュータを途中で故意に故障させたときのタイムアウトの効果を，OXTHAS-N ($N = 3$) において通常の OXTHAS-N と提案した二つの調整法を用いた OXTHAS-N を比較する．更に調整法の効果について検証する．

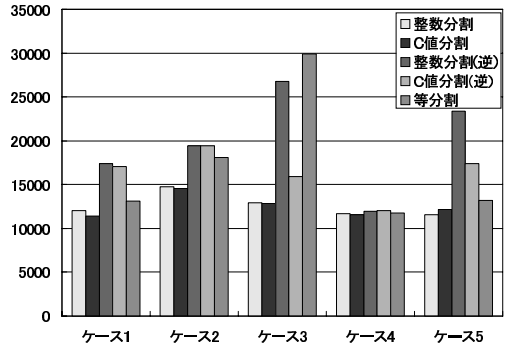

図 4　しきい値 w の設定の違いによる総処理ステップ数の比較
Fig. 4　Comparison of the number of total processing steps between load balancing strategies.

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，はじめの 1000 ステップまでは，ランダム割当を用いる．それぞれ 10 回測定を行い，その平均をとる．誤差棒は 95 パーセント信頼区間を表す．

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（プロセスインスタンス数は 200 個）を処理し，プロセスインスタンスは 10 ステップに一つ到着し，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も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．また今回は，エグゼキュータの部分的な故障は考えず，故障のときは，そのすべてのアクティビティインスタンスの処理が不可能であるとする．

5.2　結果と考察

5.2.1　OXTHAS-N に関する結果と考察

図 4 は，OXTHAS-N における処理負荷サイズの領域分割方法の違いによる総処理ステップ数を表す．エグゼキュータの性能を表 1 から表 5 の 5 種類の特徴別に分けて測定を行った．各表の数値は 1 ステップに処理する処理負荷サイズを表す．プロセスインスタンスの数は 1000 とした．

ケース 1 は表 1 のように各エグゼキュータに一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．処理内容によって得意であるかそうでないかが分かれるケースである．ケース 2 は表 2 のように 1 台のエグゼキュータだけ処理性能が他のより良い場合である．ケース 3 は表 3 のように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いう設定にした場合である．ケース 4 は表 4 のように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．

ケース 5 は表 5 のようにエグゼキュータごとに，すべて同等性能・アクティビティごとに多少性能差があるもの・性能差が大きいもの，の 3 パターンが混在しているケースの場合である．

この 5 種類のケースに対して，**3.2** で説明した五つの領域分割方法を用いて比較を行った．

グラフから分かるように整数比による分割と推定処理能力の値の比による分割が効果を上げている．等分

表 1　エグゼキュータの性能設定（ケース 1）
Table 1　Settings of executor's performance. (case1)

	e_1	e_2	e_3	e_4	e_5	e_6
a_1	10	2	4	4	5	5
a_2	4	5	4	10	4	1
a_3	5	10	3	2	5	10
a_4	1	5	3	5	10	4
a_5	5	4	10	5	1	4

表 2　エグゼキュータの性能設定（ケース 2）
Table 2　Settings of executor's performance. (case2)

	e_1	$e_2 - e_6$
$a_1 - a_5$ の平均	8	3

表 3　エグゼキュータの性能設定（ケース 3）.
Table 3　Settings of executor's performance. (case3)

	e_1, e_4	e_2, e_5	e_3, e_6
$a_1 - a_5$ の平均	8	5	2

表 4　エグゼキュータの性能設定（ケース 4）
Table 4　Settings of executor's performance. (case4)

	$e_1 - e_6$
$a_1 - a_5$ の値	4 - 6

表 5　エグゼキュータの性能設定（ケース 5）
Table 5　Settings of executor's performance. (case5)

	e_1	e_2	e_3	e_4	e_5	e_6
a_1	9	1	5	7	3	5
a_2	1	9	5	3	7	5
a_3	9	1	5	7	3	5
a_4	1	9	5	3	7	5
a_5	9	1	5	7	3	5

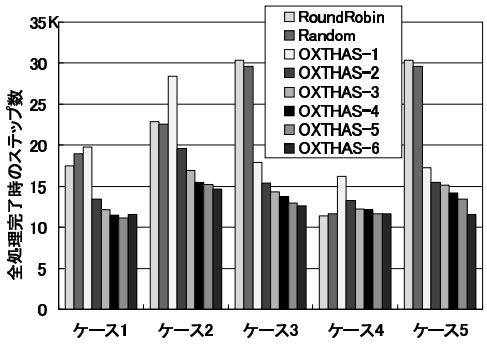

図 5 エグゼキュータの性能差の違いによる総処理ステップ数

Fig. 5 Comparison of the number of total processing steps between executor's performance settings.

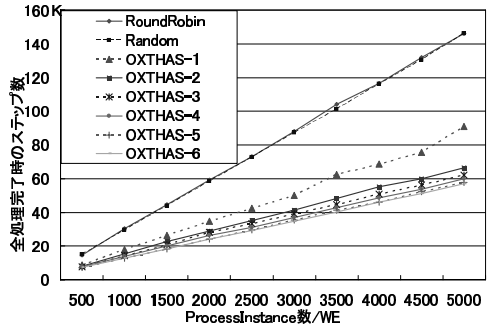

図 6 各プロセスインスタンス数における総処理ステップ数

Fig. 6 Comparison of the number of total processing steps for each method with varying number of process instance.

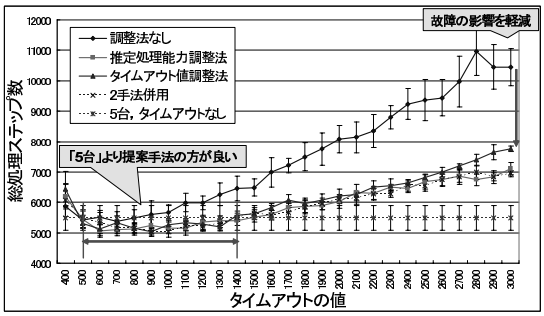

図 7 故障時における再割当手法の違いによる総処理ステップ数の比較

Fig. 7 Comparison of the number of total processing steps for each adjustment method with varying setting of time-out steps.

割に比べて良かった理由としては，推定処理能力の高いエグゼキュータに多くのアクティビティインスタンスを割り当てているからであるといえる．また，処理負荷サイズの領域を処理負荷サイズが小さい方から領域を分割する方法より処理効率が良かったことから，処理負荷サイズの大きいアクティビティインスタンスは推定処理能力の高いエグゼキュータに割り当てる方がよいということが分かる．整数比による分割と推定処理能力の値の比による分割については，ケース 1 では推定処理能力の値の比による分割の方が若干効果を上げているが，ケース 5 のように各アクティビティインスタンスについてエグゼキュータの処理性能が全く異なるようなケースにおいては整数比による分割の方が効果を上げており，処理性能にばらつきが多い場合には整数比による分割が適していることが分かる．以下のシミュレーションでは整数比分割を用いることとする．

図 5 は，エグゼキュータの性能を先ほどのようにケース別に分けて処理ステップを測定した結果である．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いえない．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 台のときよりも，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分かる．

図 6 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．エグゼキュータの性能はケース 3 を用いた．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．更に，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分かる．図 5 と図 6 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いえる．

5. 2. 2 調整法に関する結果と考察

図 7 は，故障が発生する場合にタイムアウトを変化させていったときの調整法を用いない OXTHAS と二つの調整法をそれぞれ用いた OXTHAS について比較したものである．エグゼキュータ 6 台中の 1 台（エグゼキュータ 2）を，2000 ステップ目に故意に故障させて測定を行った．推定処理能力調整法は $\alpha = 1$ と

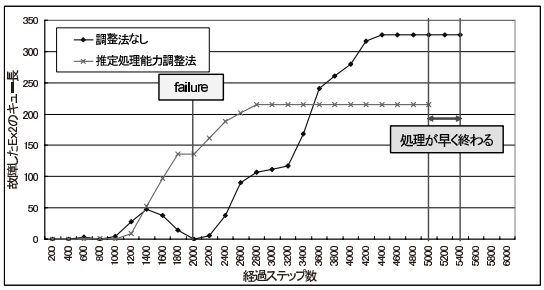

図 8　推定処理能力調整法の有無による OXTHAS の故障したエグゼキュータ 2 のキュー長の変化
Fig. 8　Transition of queue length in the crashed executor.

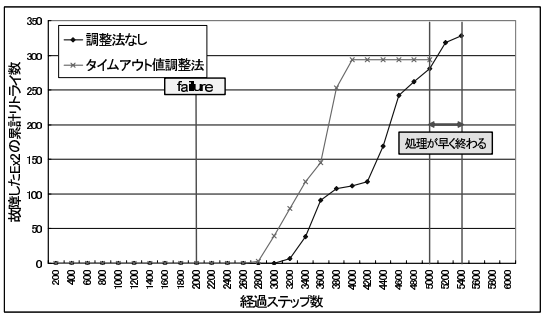

図 9　タイムアウト値調整法の有無による OXTHAS の故障したエグゼキュータ 2 の累計リトライ数の変化
Fig. 9　Transition of aggregated number of retries in the crashed executor.

し，タイムアウト値調整法は $\beta = T_{init}/100$ とした．「5 台，タイムアウトなし」の線は故障させた 1 台を最初から抜かしてエグゼキュータを 5 台でタイムアウトなしで測定したときの結果である．調整法を用いない OXTHAS よりも調整法を用いた場合の方が，タイムアウトの値が長くなっていったときの総処理ステップ数の低下を抑えていることが分かる．更に，タイムアウトの値が 500 ステップから 1400 ステップの間では，「5 台，タイムアウトなし」よりも提案した調整法の方が効率良く処理が行われている．このことは，低信頼なエグゼキュータを含んだ 6 台のエグゼキュータの処理が高信頼な 5 台のエグゼキュータの処理に匹敵，若しくは勝る場合もあるということを示している．

図 8 は各エグゼキュータのキュー長の変化を表し，図 9 は故障したエグゼキュータ 2 の累計リトライ数の変化を表す．これらの図はタイムアウトの値が 1000 ステップのときの 10 回の試行のうちのある 1 回の試行についてのグラフである．他の試行についても同様の傾向を示している．まず，推定処理能力調整法の効果について説明する．処理前半のキュー長の変化の傾向については，処理されたアクティビティインスタンスの数が少ないため，割当にばらつきが生じている．推定処理能力調整法を用いない OXTHAS の場合の図 8 では，故障に対して早期に反応できていないため，故障後も長いステップ数の間，エグゼキュータ 2 のキュー長が増加している．一方，推定処理能力調整法を用いた場合では，故障に対して早く反応し，そのエグゼキュータにアクティビティインスタンスが割り当たらず，別のエグゼキュータに割り当てられ（この図の場合はエグゼキュータ 1，3，4），全体の処理が早く終わる．また，タイムアウト値調整法を用いない場合の図 9 では，累計リトライ数が全処理が終わるステップ数まで増加している．一方，タイムアウト値調整法を用いる場合では，故障したエグゼキュータ 2 累計リトライ数の増加は 4000 ステップほどで止まっている．これは，タイムアウト値調整法により，タイムアウトの値が短くなりエグゼキュータ 2 から他のエグゼキュータに早期に再割当が行われ，その結果全体の処理も早く終わる．

以上のことから推定処理能力調整法とタイムアウト値調整法がエグゼキュータの故障時に有効であるということが分かる．また，二つの調整法の間に有意な差は見られないため，推定処理能力調整法とタイムアウト値調整法の間に相乗効果は見られない．

6. 関 連 研 究

Lie-jie Jin らは，プロセスインスタンスを一定時間ごと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 [9]．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 らが提案している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 [10] や Eitan Franchtenberg が提案している並列ジョブスケ

ジューリングにおいて，ジョブをその性質に応じていくつかのクラスに分類し，割当時のクラスタのワークロードに応じて柔軟にジョブを割り当てるアルゴリズム [11] などスケジューリングに関する研究がある．これらは，ワークフローエンジンが一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当に関しては最短で処理が完了する割当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当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．

更に，ワークフローにおいてプロセスインスタンスの処理の順番を動的に変更することにより期日に遅れるプロセスインスタンスの数を減らすスケジューリング手法を提案している [12]．これは処理時間とルートを推測しプロセスインスタンスの処理の順番を適切に決定している．しかし，本研究が目的としている負荷分散や障害対策については述べていない．

その他に，分野は異なるが，Wireless ネットワークにおける場所依存の通信路の容量とエラーを考慮したスケジューリングアルゴリズムを提案し，遅延とパケットロスを改善する研究 [13] や，Web サーバにおいて，小さなファイルを求めるリクエストから優先的に処理することによりキューでの待ち時間を減らし，パフォーマンスを改善する研究 [14] や，分散システムにおいて，遅延予測方法と遅延時間のマージンの決定方法を提案し，それらを組み合わせ障害を検知する研究などがある [15]．しかし，エグゼキュータが複数存在する場合や依存関係のある連続した処理について言及していないため，Web サービスベースのワークフロー管理システムにこれらの手法をそのまま適用することはできない．

7. むすび

本研究では，Web サービスベースの集中管理型ワークフロー管理における障害を考慮した負荷分散手法を提案した．まず，ワークフロー管理アーキテクチャを説明し，ワークフローと Web サービスの特徴について説明した．そして，その特徴を利用したエグゼキュータ間の負荷分散手法として OXTHAS スケジューリング戦略を提案した．更に，OXTHAS に基づいて，エグゼキュータの障害を考慮に入れて，タイムアウトを考慮して再スケジューリング戦略として推定処理能力調整法とタイムアウト値調整法を提案した．そして，シミュレーションを行うことにより，OXTHAS スケジューリング戦略では推定処理能力の高いエグゼキュータに処理負荷サイズの大きいアクティビティインスタンスを多く割り当てる整数比分割が最適な領域分割方法であることを示した．また，OXTHAS スケジューリング戦略はランダムやラウンドロビンよりも効率良く処理を行うことが可能であり，調整法を用いることで障害の発生時にもその処理効率をあまり低下することなく処理可能なことを示した．更に，適切なタイムアウトの値を設定することにより，低信頼なエグゼキュータを含んでいても調整法を用いることで効率の良い処理を行うことが可能であることを示した．

今後の課題としては，より効率的な再スケジューリング戦略にするための提案手法の改良や，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，様々なプロセスインスタンスの到着率やプロセスインスタンスごとにアクティビティインスタンスの数や種類が異なるとき，アクティビティインスタンスの処理負荷サイズに偏りがある場合等に対応する必要がある．更に，エグゼキュータが複数のアクティビティインスタンスを同時に処理できる場合，ビジネスプロセスが分岐などを含む複雑なものである場合，エグゼキュータの故障が多発する場合やエグゼキュータが一部の種類のアクティビティインスタンスの処理ができないような部分故障が発生する場合などの様々な状況に対応するようにすることなどが挙げられる．そして，実際にこのワークフロー管理システムを実現し，実際のシステムで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，及び東京工業大学 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，日科技連出版社，1998.
[3] (株) 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,” DEWS2004, I-12-02, 2004.
[5] N.B. Lakhal, T. Kobayashi, and H. Yokota, “Throws: An architecture for highly available distributed execution of web services compositions,” Proc. RIDE 2004, pp.103–110, March 2004.
[6] 加藤英之，小林隆志，横田治夫，“Web サービスを用いたワークフローにおける負荷分散手法,” 信学技報，DE2005-46(2005-7), July 2005.
[7] 加藤英之，小林隆志，横田治夫，“Web サービスベースのワークフロー管理における障害を考慮したアクティビティスケジューリング手法,” DEWS2006, 7C-o1, 2006.
[8] T. Kobayashi, H. Katoh, and H. Yokota, “Activity scheduling in web-service based workflow management for balancing load and handling failures,” FMUIT2006, IEEE, 2006.
[9] L.J. Jin, F. Casati, M. Sayal, and M.-C. Shan, “Load balancing in distributed workflow management system,” SAC2001, Aug. 2001.
[10] K. Nonobe and T. Ibaraki, “Formulation and tabu search algorithm for the resource constrained project scheduling problem,” in Essays and Surveys in Metaheuristics, ed. C.C. Ribeiro and P. Hansen, pp.557–588, Kluwer Academic Publishers, 2002.
[11] E. Frachtenberg, D.G. Feitelson, F. Petrini, and J. Fernandez, “Adaptive parallel job scheduling with flexible coscheduling,” IEEE Trans. Parallel Distrib. Syst., vol.16, no.11, pp.1066–1077, Nov. 2005.
[12] G. Baggio, J. Wainer, and C. Ellis, “Applying scheduling techniques to minimize the number of late jobs in workflow systems,” SAC2004, March 2004.
[13] S. Lu, V. Bharghavan, and R. Srikant, “Fair scheduling in wireless packet networks,” IEEE/ACM Trans. Netw., vol.7, no.4, pp.473–489, Aug. 1999.
[14] M. Harchol-Balter, B. Schroeder, N. Bansal, and M. Agrawal, “Size-based scheduling to improve web performance,” TOCS2003, May 2003.
[15] R.C. Nunes and I. Jansch-Porto, “Qos timeout-based self-tuned failure detectors: The effects of the communication delay predictor and the safety margin,” DSN2004, June 2004.

（平成 19 年 3 月 22 日受付，10 月 29 日再受付）

加藤　英之

平 16 東工大・工・情報工学卒．平 18 同大大学院・情報理工・計算工・修士課程了．現在，東芝ソリューション（株）．ワークフロー管理，Web サービス連携に関する研究に従事．

小林　隆志　（正員）

平 9 東工大・工・情報工卒．平 16 同大大学院・情報理工・計算工・博士課程了．平 14 同大学術国際情報センター助手．平 19 より名大大学院・情報科学・特任准教授，現在に至る．工博．ソフトウェア開発方法論，ソフトウェア再利用技術，複合メディアコンテンツの管理・検索，Web サービス連携などの研究に従事．情報処理学会，日本ソフトウェア科学会，日本データベース学会，ACM 各会員．

横田　治夫　（正員：フェロー）

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 (ICOT)．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・大学院情報理工・助教授．平 13 より同大学術国際情報センター教授，現在に至る．工博．主として分散インデクシング，データ工学向けアーキテクチャ，高機能ストレージシステム，ディペンダブルシステム等に関する研究に従事．日本データベース学会理事．ACM SIGMOD 日本支部長．情報処理学会，人工知能学会，IEEE，ACM 各会員．