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

加藤　英之† 　小林　隆志†† 　横田　治夫††,†

† 東京工業大学 大学院 情報理工学研究科 計算工学専攻　〒 152–8552 東京都目黒区大岡山 2–12–1
†† 東京工業大学 学術国際情報センター　〒 152–8550 東京都目黒区大岡山 2–12–1
E-mail: †katoh@de.cs.titech.ac.jp, ††tkobaya@gsic.titech.ac.jp, ††, †yokota@cs.titech.ac.jp

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. 我々はこれまでに，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り当て方法 OXTHAS を提案し，その有効性を示した．Web サービスを前提とすると，1 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．OXTHAS 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．本稿では，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り当て手法を提案する．さらに本稿では，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

A Failure-aware Activity Scheduling Method for Web-Service Based Workflow Management

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

† 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
†† Global Scientific Information and Computing Center, Tokyo Institute of Technology
2–12–1 Ookayama, Meguro-ku, Tokyo, 152–8550 Japan
E-mail: †katoh@de.cs.titech.ac.jp, ††tkobaya@gsic.titech.ac.jp, ††, †yokota@cs.titech.ac.jp

Abstract　In a workflow management system, appropriate allocation of its activities greatly contributes to improvement of its efficiency. We have proposed OXTHAS, a load-balancing method of scheduling the activities in workflow management systems using Web services. The OXTHAS makes the activities be allocated to appropriate executors based on the estimation of their processing capacity using execution histories in workflow engines. In this paper, we propose a failure-aware re-scheduling method for the OXTHAS using process timeouts under the network and system failures. To allocate activities appropriately, the estimated processing capacity and timeout duration are re-calculated with considering the penalty for a failure when a process timeout occurs. We then evaluate the effectiveness of the proposed methods through simulations.

Key words　Web Service, Workflow, Web and Internet

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，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．

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ている．

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．また，我々は集中管理に適した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], [7] を提案してきた．しかし, [6], [7] では，障害時におけるスケジューリングについては言及していなかった．

本稿では，我々が提案したスケジューリング戦略 OXTHAS に関して，サービス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，タイムアウトを用いた OXTHAS 再スケジューリング戦略を提案する．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推定処理能力を算出し，アクティビティ・インスタンスを適切なエグゼキュータに割り当てることで負荷分散を実現する．このとき，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，も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．

(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

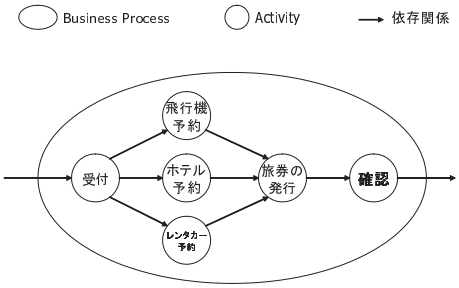

図 1　ワークフローのプロセスモデル

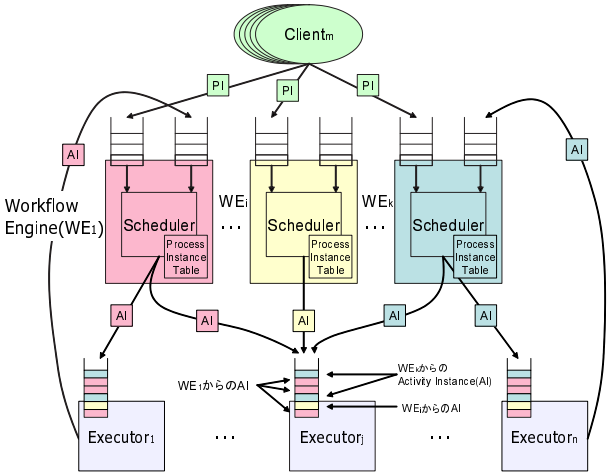

図 2　ワークフロー管理システムのアーキテクチャ

以下では，まず，本稿の対象とな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，前述の OXTHAS について簡単に述べる．次に，このワークフロー管理モデルにおけるタイムアウトの概念について言及し，タイムアウトを利用した OXTHAS 再スケジューリング戦略について説明する．そして，シミュレータを用いて，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．最後に本稿をまとめ，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 (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

本稿では，図 2 のような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 2 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．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通りである．

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，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 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 1 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持ち，キューの先頭にあるアクティビティ・インスタンスから 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） 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) もしくは 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 を受けた時，次のアクティビティ・インスタンスについての割り当てを行う．

（7） 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．

2.2.2 概　　要

2.2.1 節の手順において，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 1 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，どのように再割り当てを行うかの戦略によってエグゼキュータ障害時の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．

2.3 Web サービスベースのワークフロー管理

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の行うアクティビティ・インスタンス処理が 1 つの Web サービスである．

（2） エグゼキュータの行うアクティビティ・インスタンス処理を提供する Web サービス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，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，各エグゼキュータの処理能力を考える際には，処理内容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．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を想定する必要がある．本稿では，処理時間はファイルサイズに比例すると考える．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 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．あるワークフロー・エンジンが知ることができる情報は自身が管理するアクティビティ・インスタンスに関してだけであり，エグゼキュータの処理性能を知るためにキューの全長を把握することは困難である．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の観測値である．それは，キューでの待ち時間とエグゼキュータ内の処理時間，およびネットワーク遅延等の合計の値である．

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 を提案した．OXTHAS では各ワークフロー・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る実行時間の履歴であるため，それらの情報からアクティビティ・インスタンスの割り当てを決定する．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・エンジン毎に異なるものとなる．また，障害時の対応に関しては後に説明をする．

3. OXTHAS スケジューリング戦略

ここでは，我々が提案した OXTHAS スケジューリング戦略について簡潔に説明する．詳細は [6], [7] を参照されたい．

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}

  処理負荷サイズ s_{p_i,a_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_{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，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の時間などを含む．また，リトライが起こった場合，タイムアウトの値を処理がそのエグゼキュータから返ってくるまでの暫定的な観測実行時間として推定処理能力を計算する．

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．

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

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．

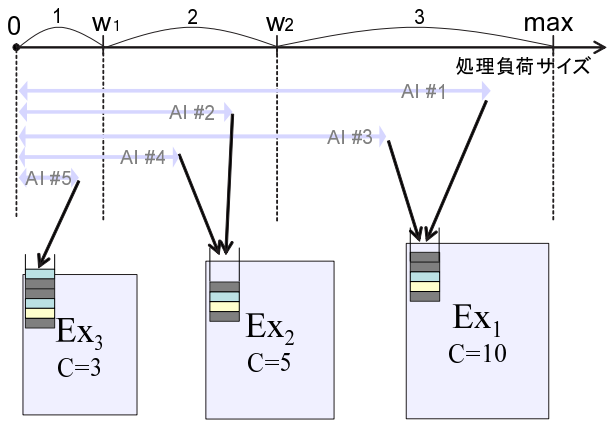

図 3　OXTHAS-N の例 (N=3)

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_{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，逆に処理負荷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，OXTHAS-N では候補となる c_{e_k,a_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閾値を設け，処理負荷サイズに応じた割り当てを行う．閾値の決定方法は等分割，整数比分割，c_{e_k,a_j} の値に応じた分割等いくつか考えられる．実験結果から整数比分割が最適であった [8] ため，以下ではこれを用いる．

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閾値は，w_i = (処理負荷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) とし，次のように定める．

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

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処理負荷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処理負荷サイズをとりだして，OXTHAS-2 はその処理負荷サイズを 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処理負荷サイズの幅が 2:1 の割り当てになるように処理負荷サイズの閾値を決定する．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処理負荷サイズの閾値を決定する (図 3 を参照).

4. OXTHAS 再スケジューリング戦略

OXTHAS-N は，エグゼキュータの負荷分散のみを考慮しているため，ネットワークやサーバの障害などによりエグゼキュータが故障した際，処理自体が途中で止まってしまう．そこで，エグゼキュータが故障しても処理が止まらず，さらに処理効率の低下を最小限に防ぐ必要がある．この問題点を解決するために，本稿では OXTHAS 再スケジューリング戦略を提案する．以下では，OXTHAS 再スケジューリング戦略としてタイムアウトの利用方法について説明し，OXTHAS に適用するタイムアウトの利用方法を示し，OXTHAS における再割り当て手法について説明する．

4.1 タイムアウトの利用方法

タイムアウトを用いることには大きく分けて次の 2 通りの目的がある．以下では，その 2 つについて詳しく議論する．

- 処理効率の向上
- 障害検知

一つ目の処理効率の向上に関しては，処理性能の悪いエグゼキュータにアクティビティ・インスタンスが集中してしまった場合，処理効率を改善させる場合がある．これは，あるアクティビティ・インスタンスでタイムアウトが発生し，別のエグゼキュータにリクエストをし直した場合，その処理時間は (タイムアウトの値 + 処理時間) となるため，タイムアウトの値だけ処理時間が増加するため，タイムアウトによる時間のロスを補うことが可能となる．

一方，障害検知に関しては，処理が途中で停止することなく終了させることが目的である．まず障害にはエグゼキュータの故障やネットワークの故障，混雑による速度低下等が考えられる．また，故障にも一時的なものとすぐには復旧しないものがある．ネットワークが混雑することによる速度低下の場合，通常より時間はかかるが処理は行われる．しかし，エグゼキュータやネットワークの故障の場合，アクティビティ・インスタンスの処理自体が行われず，プロセス・インスタンスの処理がいつまでも終了しないことになる．そのため，どのような割り当て方法を用いた場合であっても，タイムアウトを利用して別のエグゼキュータに再割り当てをすることが必要である．また，故障から復旧した場合，そのエグゼキュータにもアクティビティ・インスタンスを割り当てることが望ましい．

4.2 OXTHAS におけるタイムアウトの利用

OXTHAS では，その時その時で不確定要素を踏まえて推定処理能力を計算することでキュー長の偏りまでを把握してアクティビティ・インスタンスを割り当てている．キュー長が長くなる場合は推定処理能力が低下し，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ないようになる．その結果，同じエグゼキュータのキュー長が常に長くなってしまうことを防ぎ，処理効率の向上を実現している．そのため，前述のタイムアウトの利用方法として 2 つ挙げたが，処理効率の向上のためのタイムアウトの利用は OXTHAS では必要ない．

そこで，OXTHAS では障害検知としてタイムアウトを利用する．ネットワークの混雑による速度低下の場合は，OXTHAS では推定処理能力に反映されているため，ここでは故障した場合について考慮する．故障検知のみを目的とする場合，ある程度長くタイムアウトを取ることで，故障検知を行うことができる．故障を検知する時刻は故障する前に割り当てられたアクティビティ・インスタンスがタイムアウトした時であるため，タイムアウトの値が短い方がより早く故障を検知することができる．しかし，あまりに短いタイムアウトを設定する場合，処理時間が遅いアクティビティ・インスタンスに対してもタイムアウトを行ってしまい，検知ミスを起こすことを考慮に入れなくてはならない．

4.3 再スケジューリング戦略

以上のことを踏まえて，故障検知の対応のみを目的とするため,「タイムアウトの発生=エグゼキュータもしくはネットワークの故障」と考える．そして，タイムアウトが起こった場合，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てられないようにするために，推定処理能力 c_{e_k,a_j} の値を下げる．もしくは，割り当てられても，すぐにタイムアウトが発生するようにタイムアウトの値を下げる．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し，以下のように定義する．

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'_{e_k,a_j} を以下のように再定義する (R_{exe} 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)．

$$c'_{e_k,a_j} = \frac{c_{e_k,a_j}}{1 + \alpha \cdot R_{exe}}$$

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．

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_{exe} を次のように定める．(β, T_{init} は定数)

$$T_{exe} = T_{init} - \beta \cdot R_{exe}$$

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，再割り当てが起こる．

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'_{e_k,a_j} や T_{exe} の値を最初に定義した値に戻す．この方法を取ることで，一時的な故障に対応することが可能となり，さら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．

再割り当てを行う時は，再割り当てを行うアクティビティ・インスタンスをすで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り当て

が発生する度に割り当て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，再割り当て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} で再割り当てを行う．

5. 評　　価

5.1 シミュレーション内容

前述した 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 [6] のシミュレータを用いて，シミュレーションを行った．最初に，アクティビティ・インスタンスに設けるタイムアウトの値を変えていったときの OXTHAS-N(N=3) とランダムにおけるタイムアウトの効果を調べ，OXTHAS では処理効率改善のためのタイムアウトの利用は必要がないことを検証する．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 と提案した調整法を用いた OXTHAS-N を比較する．

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，ランダム割り当てを用いる．それぞれ 10 回測定を行い，その平均を取る．実験結果の各グラフの誤差棒は 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 1 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．また今回は，エグゼキュータの部分的な故障は考えず，故障の時は，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

図 4 は，タイムアウトの値の変化に伴う総処理ステップ数を測定した結果である．OXTHAS は再割り当て手法を含まない通常の OXTHAS-N である．なお，タイムアウトの値が 200 ステップ以下の場合，タイムアウトを永遠に繰り返し処理が終了しなかった．この図から，ランダムではタイムアウトの効果があることがわかるが，適切なタイムアウトの値は 300 から 400 ステップという非常に狭い範囲であることがわかる．パラメータが異なった場合には処理が終了しない場合も考えられるため，常に適切なタイムアウトの値を設定することは難しいことがわかる．一方，OXTHAS-N では，そのグラフに極小値が存在しないため，タイムアウトの値によって処理効率が向上することはないということが分かる．さらに，ランダムにおける最小値より，OXTHAS-N の方が早く処理されているため，処理効率の向上のためのタイムアウトの値の設定は必要ない．

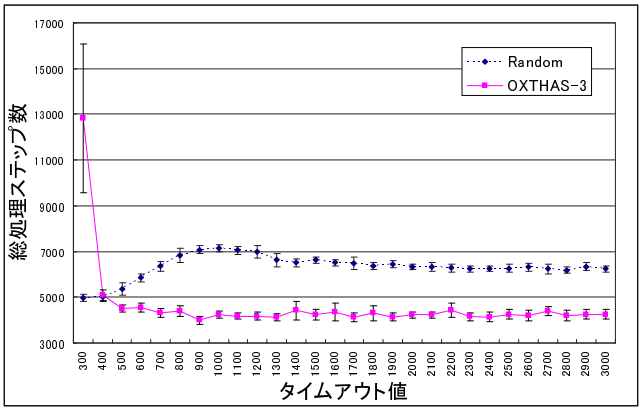

図 4　タイムアウトの値を変化させた時の総処理ステップ数

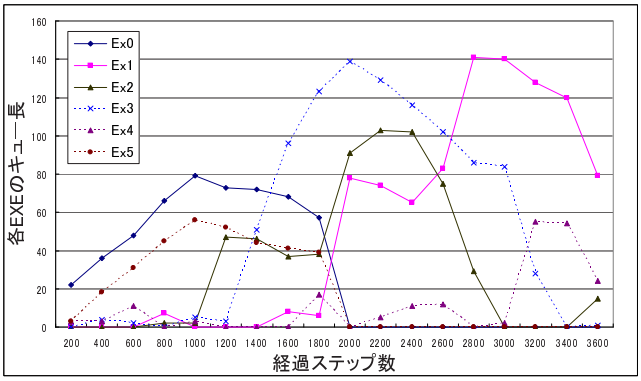

図 5　OXTHAS-3 におけるタイムアウトなしのときの各エグゼキュータのキュー長の変化

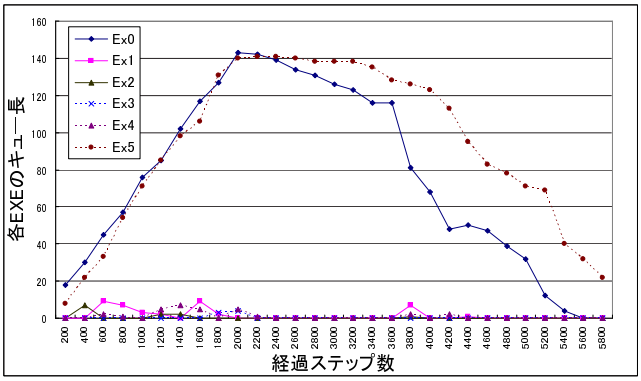

図 6　ランダムにおけるタイムアウトなしのときの各エグゼキュータのキュー長の変化

また，図 4 における OXTHAS とランダムそれぞれについてタイムアウトを設定しない場合の各エグゼキュータのキュー長の遷移を比較するために，10 回の試行のうちのある 1 回のそれぞれのキュー長の遷移を図 5，6 に示す．このシミュレーションでは，処理性能が高い順に (エグゼキュータ 2，3)，(エグゼキュータ 1，4)，(エグゼキュータ 0，5) となっている．図 6 から分かるように，ランダムでは，ほぼ等分に処理が割り当てられるため，

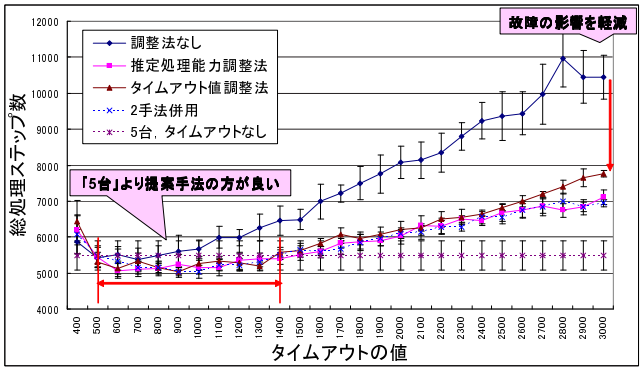

図 7　故障時における再割り当て手法の違いによる総処理ステップ数の比較

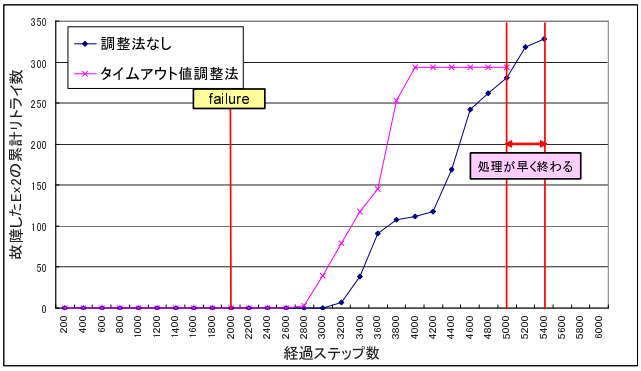

図 9　タイムアウト値調整法の有無による OXTHAS の e2 の累計リトライ数の変化

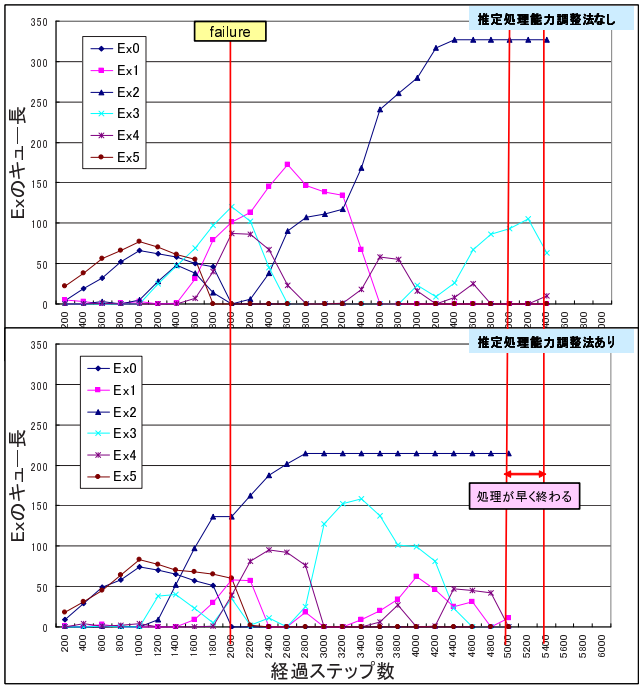

図 8　推定処理能力調整法の有無による OXTHAS の各エグゼキュータのキュー長の変化

処理性能の低いエグゼキュータのキュー長が長くなってしまい，処理性能の高いエグゼキュータは逆に Idle 状態になってしまっている．一方，図 5 から分かるように，OXTHAS-N では処理性能の高いエグゼキュータにアクティビティ・インスタンスがより多く割り当てられ，すべてのエグゼキュータを効率よく使用している．また，あるエグゼキュータのキュー長が長くなった場合，別のエグゼキュータに割り当てられるように推定処理能力が柔軟に働いていることも分かる．そのため，OXTHAS においては処理効率改善のためのタイムアウトは必要ないことが分かる．

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 2 つの調整法を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時の総処理ステップ数の低下を抑えていることがわかる．さらに，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よ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，もしくは勝る場合もあるということを示している．また，2 つの調整法の間では差が見られず，推定処理能力調整法とタイムアウト値調整法の両手法を併用した場合にも差は見られないため，2 つの調整法の間に相乗効果は見られない．

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．まず，推定処理能力調整法の効果について説明する．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り当てが行われ，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わかる．

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 [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 [10]．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．

さらに，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 [11]．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，本稿で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，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 [12] 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 [13]．しかし，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. おわりに

本稿では，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．具体的には，ワークフロー管理モデルを説明し，ワークフローと Web サービスの特徴を考慮に入れたスケジューリング戦略 OXTHAS について説明した．さらに，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の 2 つの調整法を提案した．そして，シミュレーションを行うことにより，提案手法は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．

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，様々なプロセス・インスタンスの到着率やプロセス・インスタンス毎にアクティビティ・インスタンスの数や種類が異なる時，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ことや，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，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．また，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言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 *DEWS2004*, 電子情報通信学会, Mar. 2004. I-12-02.

[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, Mar 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), 電子情報通信学会, Jul. 2005.

[7] 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. *DBSJ Letters*, Vol. 4, No. 2, pp. 25–28, Sep. 2005.

[8] 加藤英之. Web サービスベースのワークフロー管理におけるアクティビティスケジューリングに関する研究. 修士論文, 東京工業大学, Jan. 2006.

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

[10] 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.

[11] Gregorio Baggio, Jacques Wainer, and Clarence Ellis. Applying scheduling techniques to minimize the number of late jobs in workflow systems. In *Proc. of SAC2004*, Mar. 2004.

[12] Mor Harchol-Balter, Bianca Schroeder, Nikhil Bansal, and Mukesh Agrawal. Size-based scheduling to improve web performance. In *Proc. of TOCS2003*, May 2003.

[13] Raul Ceretta Nunes and Ingrid Jansch-Porto. Qos timeout-based self-tuned failure detectors: the effects of the communication delay predictor and the safety margin. In *Proc. of DSN2004*, Jun. 2004.