Webアプリケーションのための動的適応可能な処理分担機構の設計と実装*

山之井 啓泰† 石橋 崇† 佐藤 喬† 小宮 常康†
電気通信大学 大学院情報システム学研究科‡

1. 研究背景

今までの Web アプリケーションにおいて，Web ブラウザの行う主な処理は，サーバによる処理結果を表示することであった．しかし，Google Maps[1] のように，デスクトップアプリケーションと同等の機能や操作性を持つ Web アプリケーションが増加することにより，Web ブラウザで行う処理の量も増加してきている．それらは Ajax アプリケーションと呼ばれ，Web ブラウザ上で動作する JavaScript を利用して処理を行っている．

これらの Web アプリケーションの多くは，開発者がサーバ・ブラウザ間の処理の分担を手作業で決定している．しかし，手作業で最適な分担方法を決定するのは困難であり，分担に失敗するとユーザへのレスポンスが悪化してしまう．ある環境に最適な分担を行ったとしても，サーバとブラウザの処理能力やネットワークの性能は，時間と共に変化するので，これらの環境に動的に対応することはできない．

そこで本研究では，Web アプリケーションの処理分担を半自動的に決定し，さらに処理分担の動的適応を行う機構を提案する．処理分担を行う技術として，科学技術計算を扱うような既存の動的負荷分散技術があるが，処理全体のスループット向上が目的となっている場合が多い．Web アプリケーションは，ユーザへのレスポンスが重要であり，一度の負荷分散の失敗が大きな問題となってしまう．よって，ユーザへのレスポンスを重視する手法を選択する．

2. 提案手法

提案手法は，サーバとブラウザの処理分担を，関数の単位で行う．ある関数の呼び出しをサーバで行うか，ブラウザで行うかを切り替えることにより処理の分担を実現する．

ここで，どちらで実行してもよい関数の実行場所を切り替えることにより，処理を分担させる事が可能となる．そのために，サーバとブラウザの処理を一つのプログラムとして記述する．しかし，ブラウザ側の関数からサーバ側の関数は呼び出せるが，サーバからブラウザの関数を呼び出すことはできない．

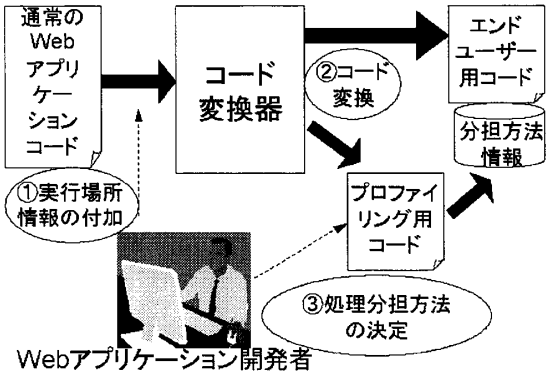

図 1: 処理分担機構の流れ

どちらでも実行していい関数を適切に分担するために，関数の実行時間やコールグラフ等のプロファイル情報を取得する．さらに，関数の組み合わせパターンを作成し，取得したプロファイル情報をもとに，サーバとブラウザの分担を決定する．

ある環境で分担を決定しても，開発環境と実際に運用する環境では異なっているので，計算機性能やネットワークを考慮にいれる必要がある．

そこで，プロファイリング時に計算機性能，ネットワークの性能を測ってパラメータ化をする．開発者が実際に運用する環境に合わせてパラメータを調節することにより，動的適応を実現する．

3. 処理分担機構

本機構は図 1 のような，Web アプリケーション開発者のための支援ツールとなっており，対象言語として JavaScript を扱っている．

開発者は，作成した通常の Web アプリケーションのコードに実行場所情報の付加を行う．次にコード変換器を使って，エンドユーザ用コードとプロファイリング用コードを生成する．さらにプロファイリング用コードを使って処理分担方法の決定する．そしてその処理分担方法を，エンドユーザ用コードに導入することで，動的適応可能な Web アプリケーションが生成することができる．

*Design and Implementation of a Dynamically Adaptive Partitioning Mechanism for Web Application Computation
†Hiroyasu Yamanoi, Sou Ishibashi, Takashi Satou, Tsuneyasu Komiya
‡Graduate School of Information Systems, The University of Electro-Communications

表 1: 組み合わせパターン例

	関数 f	関数 g	関数 h	予測実行時間
パターン 1	*Client*	*Client*	*Client*	$C_f + C_g + C_h$
パターン 2	*Client*	*Client*	*Server*	$C_f + C_g + S_h$
パターン 3	*Client*	*Server*	*Client*	$C_f + S_g + C_h$
パターン 4	*Client*	*Server*	*Server*	$C_f + S_g + S_h$

3.1. 実行場所情報の付加

サーバかクライアントのどちらかでのみ実行すべき関数に「*Client*」と「*Server*」というアノテーションを，開発者が関数に付加する．アノテーションが付加されない関数はどちら側でも実行される可能性がある関数となる．このような実行場所が自由な関数が処理分担の対象となる．

3.2. コード変換器

付加した実行場所情報をもとに，プロファイリング用コードとエンドユーザ用コードを生成する．プロファイリング用コードには，プロファイル情報の取得，パラメータの計算，関数の組み合わせパターンの作成，処理分担方法の決定，を行うためのコードが生成される．エンドユーザ用コードには，プロファイリング用コードで作られた処理分担方法の活用と，動的適応を行うためのコードが生成される．

3.3. プロファイリング用コード

開発者が，実行場所が自由な関数が全てクライアントで実行するように，実際に Web アプリケーションを動作させ，コールグラフと実行時間を取得する．

次に，パラメータの測定を行う．パラメータとは，クライアントとサーバで同じプログラムで負荷を測定し，同時にネットワークの時間も測ることで，実際の性能の実時間を調べた値である．調べた時間を *server*，*client*，*network* とする．さらに，関数 f がサーバやクライアントで動作したときの実行時間を S_f，C_f とし，関数 f が呼び出される回数を $count_f$ とする．プロファイル情報の取得で，$count_f$ と C_f がわかっているので，サーバ側の関数 f がクライアントから呼び出された場合の時間 S_f は，次の式で予測することができる．

$$S_f = count_f \times network + C_f \times \frac{server}{client}$$

S_f を求めることで，関数の組み合わせパターンと，パターンの実行時間の予測を行う．表 *1* は，f の実行場所が *Client* で g と h の実行場所が決められてない場合の，組み合わせパターンと予測実行時間を示す例である．

あるパラメータの環境 A ではパターン 2 が最適な処理分担で，別のパラメータの環境 B ではパターン 4 が最適な処理分担だと仮定する．環境 A と環境 B のパラメータの値や，どちらの環境を重視するか，といった情報を開発者が手動で調整を行うことで，半自動的に処理分担を決定することができる．

3.4. エンドユーザ用コード

エンドユーザ用コードには動的適応を行うために，関数の実行場所を変更する機能と，実行環境の変化を調べる機能がある．

関数の実行場所を変えるためには，関数が実行される毎に実行場所の確認を行う．実行中の環境の変化は，数 10 秒の間隔で負荷情報を調べるプログラムを実行することで測定する．プロファイリング用コードで決定した処理分担方法を使うことで，環境毎に動的適応可能な Web アプリケーションを生成することができる．

4. 関連研究

Hop[2] と Links[3] は，Web アプリケーション開発用に設計されたプログラミング言語である．どちらも，サーバ側の処理とブラウザ側の処理を一つの言語で扱うので，シームレスに Web アプリケーションを記述することができる．しかし，これらの言語は，サーバとブラウザ間での処理分担に関する機能は存在しない．

5. おわりに

本研究では，Web アプリケーションの処理分担を半自動的に決定し，さらに処理分担の動的適応を行う機構の設計，実装を行った．これより，処理分担方法の決定の支援と共に，動的適応によって環境毎に適した処理を行うことが可能となった．

今後の課題は，非同期通信を扱った Web アプリケーションへの対応方法や，制約の部分を解決し，大規模な Web アプリケーションへの適応を行うことである．

謝辞 本研究の一部は，科学研究費補助金（20500028）の補助を受けている．

参考文献

[1] Google Maps, `http://maps.google.co.jp/`

[2] Manuel, S., Erick, G. and Florian, L.: Hop, a Language for Programming the Web 2.0, *Companion to the 21st ACM SIGPLAN conference on OOPSLA'06*, pp.975–985, (2006).

[3] Ezra, C., Sam, L,. Philip, W. and Jeremy, Y.: Links: Web Programming Without Tiers, *5th International Symposium on Formal Methods for Components and Objects* (2006).