

## オーバレイネットワーク上でアプリケーションサービスを実行するプラットフォームの設計と実装

境 裕 樹<sup>†1</sup> 廣 森 聡 仁<sup>†1</sup>  
山 口 弘 純<sup>†1</sup> 東 野 輝 夫<sup>†1</sup>

本論文では、オーバレイネットワークを構成するサーバ群を利用し、分散して蓄積された大量のリアルタイムセンシング情報を活用するアプリケーションサービス（以下、サービス）を効率良く分散協調実行するためのアルゴリズムを提案する。このアルゴリズムでは、センシング情報の解析処理やマッチングなど基本的な処理をコンポーネントとして、サービスはそれらの組み合わせとして与えられるものとする。このもとでサーバの計算能力、ネットワークの通信遅延や利用可能帯域などを考慮し、どのコンポーネントをどのサーバで実行するかを表す分散実行方針を決定する。さらに、このアルゴリズムを用いたサービス設計から、実環境におけるサービス実行までをシームレスに行うことができるサービス実行プラットフォームの設計及び実装を行う。サービス実行プラットフォームでは、決定された分散実行方針に従って、サーバへのコンポーネント配置や分散実行を制御できる。また、ネットワークやサーバの負荷状況を収集し、そのもとで即座に最適な分散実行方針を導出できる。遠隔地に蓄積されたトラフィック情報を用いてデータ解析を行うサービスを想定した PlanetLab 上での評価実験を行い、提案手法を用いて導出した分散実行方針により、他の手法と比較して、多数のリクエストを処理でき、サービスを効率良く分散協調実行できていることを確認した。

### Design and Development of Service Execution Platform for Overlay Networks

YUKI SAKAI,<sup>†1</sup> AKIHITO HIROMORI,<sup>†1</sup>  
HIROZUMI YAMAGUCHI<sup>†1</sup> and TERUO HIGASHINO<sup>†1</sup>

In this paper, we propose a method to derive execution sequences of a given application-level service that is executed by cooperative servers on overlay networks. We also design and develop a service execution platform. The proposed method assumes that a service consists of service components, and it can de-

rive optimal sequences that do not overload network links and servers. Using the platform, a given service can be installed and executed easily on real networks. The platform can measure network loads and CPU loads continuously, and can adaptively re-derive new sequences optimizing the system load in the environment. We have conducted experiments to validate our method. These experiments have shown that the proposed method could derive efficient execution sequences and they could achieve higher throughput than the other methods.

#### 1. はじめに

人や車、モノなど社会の構成要素や環境要素をセンシングし、情報通信技術により集約、解析して人々に安全安心で利便性の高いサービスを提供するシステムが現在盛んに研究されている。今後、携帯電話や車載器などモバイル機器はますます多様なセンサを搭載し、防犯や事故記録、交通流解析などのため街角へ設置されるカメラも増加すると予想される。そのような多数のセンサーから時々刻々と蓄積される大量のリアルタイムセンシング情報を如何に効率よく集約し、どのようにして様々なサービスに結び付けるかがユビキタスなサービスを実現するうえでの課題となっている。

例えば、市街地の各所で撮影されたカメラ画像と様々な車両で計測された GPS 情報が蓄積されたデータリポジトリをネットワーク経由で利用し、各地の渋滞予測を行うサービスを考える。このサービスでは、市街地の各所で撮影されたカメラ画像と様々な車両で計測された GPS 情報をローカルサーバに蓄積し、画像認識によるトリップタイム計算や軌跡分析などを行い、広域での交通量を算出する。このように、ローカルサーバに分散して蓄積された大量のデータをサーバ間で転送しながら検索・加工を行うようなサービスの実現には、一般に以下のような課題がある。(1) 特定のサーバやリンクに対する負荷集中を回避し、高速な応答時間を確保するためには、データ量やサーバおよびリンクの容量、それらの負荷を可能な限り考慮し、どのサーバでどの処理を実行し、それらのデータをどのようにデータ転送するかのポリシーを注意深くデザインしておくことが望ましい。しかし、サービスやネットワークごとに最適なデザインを行うことは設計者にとって容易でない。(2) また、実環境ではシステムの動的な負荷変動があり、それらを考慮したデザインが望まれる。そのた

<sup>†1</sup> 大阪大学大学院情報科学研究科

Graduate School of Information Science and Technology, Osaka University

めには、実行時の負荷情報の取得や、サービスコンポーネントのサーバへの負荷分散など、これらの実現に係る一連の手間を軽減するための計算機支援が望まれる。

本論文では、(1)の課題に対し、複数のローカルサーバ（以下、単にサーバと呼ぶ）により構成されるオーバレイネットワークにおいて、サービスを効率よく分散協調実行するためのアルゴリズムを提案する。一般に上記のサービスは並行に処理される場合が多いため、提案アルゴリズムでは、形式的にサービスをカラーペトリネットモデル化する。これに対し、サーバ群の処理能力と各サーバ間の通信遅延や利用可能帯域をパラメータとして与え、サーバやネットワークに与える負荷を軽減しながら、サービスに対するリクエストの応答時間が最も短くなる実行方針を機械的に決定する。また(2)の課題に対し、サービスの設計から実環境における実現までの開発プロセスを、シームレスに支援するサービス実行プラットフォームの設計と実装を行う。本プラットフォームでは、サービスのカラーペトリネット記述を与えると、サーバへのサービスコンポーネントへの配置と実行を自動で行う。さらに、実行時のネットワークトラフィックとサーバのスループットをリアルタイムで観測できる機能を実現し、負荷状況に基づきサービスの実行方針を微修正したり、一時的に利用不能となったサーバを切り離すなどの処理を容易にできるようにすることで、開発者の負担を軽減する。

遠隔地に蓄積されたカメラ映像や車両トラフィック情報を用いてデータ解析を行うサービス想定し、本プラットフォームを用いて設計し PlanetLab 上で実行する実験を行った結果、設計者による発見的アルゴリズムと比較して、提案アルゴリズムは、サーバの処理負荷とネットワーク帯域の制約を守りつつ、多くのリクエストを処理できる実行方針を導出できた。また、開発したプラットフォームによりその実験が容易に実行でき、実行時のネットワークおよびサーバ負荷も簡単に取得できることを確認した。

## 2. サービス

本論文では、各サービスは各入力（コマンド入力、センシングデータの入力など）に対し、データベース処理など幾つかの処理を実行し、一つの出力（処理結果）を出力するプログラム（トランザクション処理）としてモデル化する。ただし、条件に応じて異なる処理を実行したり、同時に実行可能な処理を並行に実行してもよい。形式的には、その制御構造として自己ループ以外の閉路を含まないカラーペトリネットを用いる（自己ループについては後述）。以下ではカラーペトリネットの詳細を説明する前に、提案手法が想定するサービスの例を図1を用いて説明する。図1は隣接する2都市において、それぞれ各地の街頭に

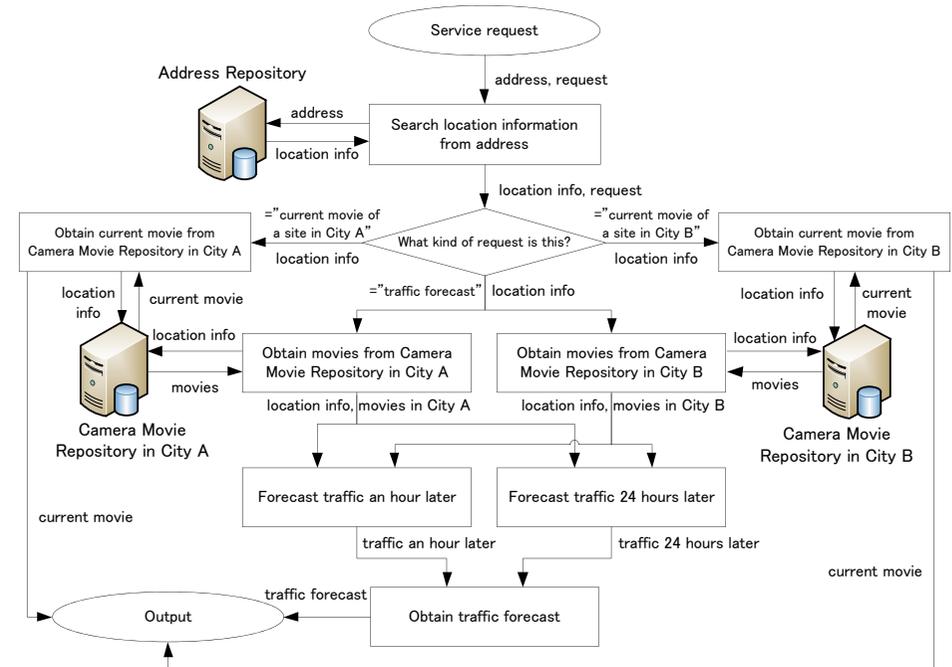


図1 オーバレイサービス例（交通解析サービス）

Fig. 1 Overlay service example (Transportation Analysis Services).

設置されたカメラ映像を利用した交通解析サービスをフローチャート風に記述した例である。このサービスは交通解析を行いたい場所の住所がユーザから入力として与えられると、マップ情報や設置されている街頭カメラの位置情報を管理するリポジトリ（データサーバ）“C”からその住所周辺の情報を検索する。得られた情報を用いて、それぞれA市、B市の各地点に設置された街頭カメラ映像を蓄積したリポジトリ“A”および“B”から交通解析に必要な全地点の街頭カメラ映像を検索し、それらの映像を解析することで、指定された住所の1時間後と24時間後の渋滞予測を行う。最後にそれらの結果をユーザに通知する。渋滞予測ではなく現在の状況を知りたいユーザには、指定された住所に最も近い街頭カメラの映像を提供することもできる。

一般にこういったサービスでは、上記のようなトランザクション処理が同時に多数のユー

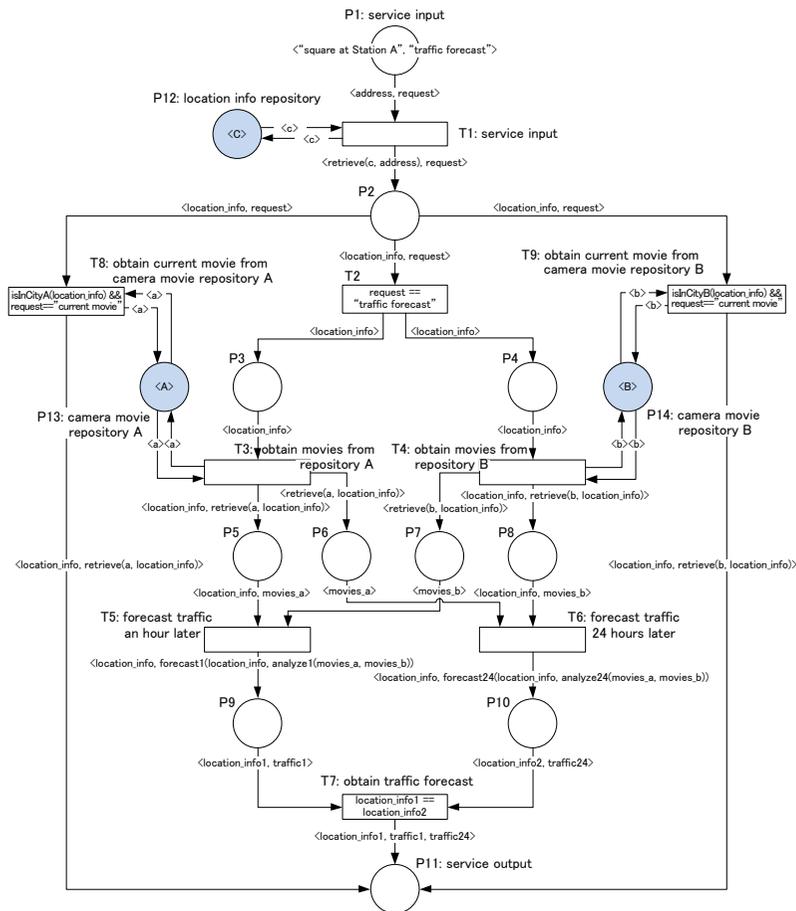


図2 図1の交通解析サービスをカラーペトリネットで記述した例

Fig.2 Transportation Analysis Services of Fig.1 formally written in Colored Petri net.

ザから送られてくるのが考えられる。そこで、本論文では、並行に多数実行される状況をカラーペトリネット<sup>1)</sup>で図2のように表現する。図2のカラーペトリネットでは、各トランザクションの入力(依頼)はカラートークンで表し、データベースの検索などの各コンポーネントはトランジションで表現する。各プレースには、次に実行するトランジションの

処理に必要なデータの組がトークンの形で保持される。

図2では、プレース P1 と P11 は、それぞれサービスに対するリクエストのエントリポイント(入力を与えるプレース)とエンドポイント(出力を受け取るプレース)を表し、プレース P12, P13 および P14 は、それぞれ位置情報リポジトリ“C”と街頭カメラ映像リポジトリ“A”および“B”を表している。プレース P2 は、位置情報の検索結果、プレース P5 から P8 までは、それぞれカメラ映像の検索結果、プレース P9 と P10 は、それぞれ1時間後と24時間後の渋滞予測結果を保持している。プレース P3 と P4 は、カメラ映像の検索処理が行われるまでのバッファを表している。

一般にユーザからのリクエストを同時並行的に処理する場合、多数のトランザクションが発生する。カラーペトリネットでは、それらの各トランザクションをカラートークンの値の違いによって識別する。トランジションでは、カラートークンの値に応じて発火条件を記述でき、それぞれのトランジションでの発火を制御できる。例えば、request で表されるカラーの値が“traffic forecast”の場合、トランジション T2 が発火可能である。一方、request が“traffic forecast”の場合で、かつ、指定された住所がA市である場合、トランジション T8 が、B市である場合、トランジション T9 が発火可能となる。

サービスを表現するために用いるカラーペトリネットは、(P, T, A) で表現される有向グラフをその構造として持ち、P, T, A は、それぞれプレース集合、トランジション集合、アーク集合に対応する。カラーペトリネットは、プレース集合 P, トランジション集合 T から構成されるラベル付き2部グラフなので、アーク集合 A は、(P × T) ∪ (T × P)、グラフの頂点集合 V は、P ∪ T と表現することができる。すべてのアークの重みは1とする。入力アークを持たないプレースと出力アークを持たないプレースは、それぞれ1つずつ持ち、サービスに対するリクエストのエントリポイントとエンドポイントを表す。さらに、本論文では自己ループを除き閉路を持たないことを仮定する。自己ループは、プレース p がトランジション t の入力かつ出力プレースとなっている状態のことを言う。これは、図2のプレース P12, P13 および P14 のように、データベースを表すプレースを記述できるようにするためのものである。

### 3. オーバレイネットワーク

我々が対象とするネットワークアーキテクチャの概念図を図3に示す。ネットワークは、データリポジトリやサーバをオーバレイノードとした、ノード群が相互にユニキャスト通信可能な論理ネットワークを想定する。各データリポジトリやサーバは、カメラ画像といっ

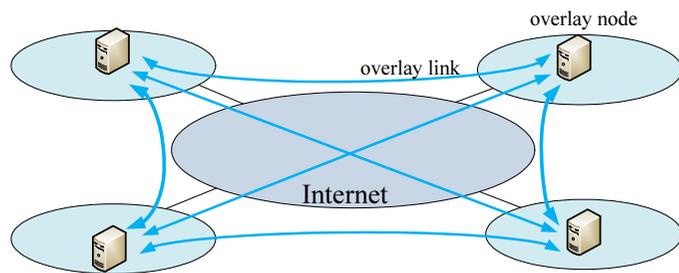


図3 ネットワークアーキテクチャの概念図  
Fig. 3 Network Architecture.

たセンシングデータを蓄積する機能（プレース）やコンポーネントを実行する機能（トランジション）をそれぞれ保持している。各ノードはサービスを構成するサービスコンポーネントを実行可能であり、その処理速度はノードごとに異なり、同様に、各ノード間のリンクにおけるデータ転送速度も異なる。各アプリケーションサービスプロバイダ（以下、ASP）では、例えばセンサネットワークのゲートウェイとしてセンサノードから収集した情報を蓄積し、コンポーネントの実行に必要なデータを提供するデータサーバと、コンポーネントを実行する機能を備えたサーバによって構成される。各ASPでは、データリポジトリやサーバが、比較的、広帯域かつ低遅延なネットワークによって連結され、それらの各ASPでのネットワークが、インターネットなどの広域ネットワークによって連結されているようなネットワークアーキテクチャを想定する。各データリポジトリやサーバは、カメラ画像といったセンシングデータを蓄積する機能やコンポーネントを実行する機能をそれぞれ保持している。また、それぞれのサーバはリクエストに応じて、それぞれのコンポーネント処理を他のサーバへ振り分けるディスパッチ機能を有する。

オーバーレイネットワークは重み付きグラフ  $G = (N, L)$  でモデル化する。オーバーレイネットワーク  $G$  はサーバ集合  $N$  およびオーバーレイリンク（以下単にリンクとよぶ）集合  $L = N \times N$  からなる完全グラフである。

#### 4. ネットワーク上でのサービスの実行とその最適化アルゴリズム

分散して蓄積された大量のリアルタイムセンシング情報を集約し処理するサービスをオーバーレイネットワーク上で提供することを考えた場合、一般に、入力プレース、または、出力プレースを保持するサーバ、センシング情報を保持するサーバ、コンポーネントを処理する

（トランジションを実行する）サーバは異なる。サーバ間をトークンが移動する時、それらのサーバ間でデータ転送が発生する。例えば、図2の  $T3$  と  $P5$  に割り当てられたサーバが異なるとすると、 $T3$  で検索された映像データがネットワーク上で転送されることになる。

この状況において、例えば特定のASP間通信の帯域が制限されたり、アプリケーションサーバのスループットが制限されるなどの理由により、トランジション  $T3$  とプレース  $P5$  をそれぞれ担当するサーバ間のリンクにボトルネックが生じたとすれば、データ転送に非常に時間がかかり、サービスの実行がここで滞る。この一方で、例えばトランジション  $T3$  を実行するサーバを、ボトルネックの生じない別のサーバ、あるいは、プレース  $P5$  と同じサーバに変更した場合、トランジション  $T3$  とプレース  $P5$  との間のデータ転送におけるボトルネックを回避することができ、サービスの応答時間が改善される可能性が高い。しかしながら、トランジション  $T3$  の実行するサーバを変更した結果、プレース  $P5$  以外の入出力プレース  $P3$ ,  $P6$  および  $P13$  のいずれかのプレースとトランジション  $T3$  間のデータ通信が、新たなボトルネックとならないよう考慮しながら、トランジション  $T3$  の実行するサーバを選択する必要がある。さらに、サービス全体としてリクエストが適切な時間で処理できるようにするためには、1つのサーバに実行負荷が集中しないようにしつつ、トランジション  $T3$  のみならず、全てのプレースやトランジションの依存関係を考慮し、適切な配置を見つける必要がある。

本論文では、すべてのプレースやトランジションをどのサーバに配置するかを定めたものを分散実行方針と呼ぶ。次節以降では、最適な分散実行方針を決定するための手法について述べる。

##### 4.1 最適な分散実行方針の導出

同じサービス、同じネットワークでも複数の分散実行方針が導出可能であり、分散実行方針が異なればサービスの実行時間も異なる。サーバやネットワークのリソース制約を満たしながら、単位時間あたりに、なるべく多くのリクエストを処理できるような分散実行方針を用いて、サービスを分散化することが望ましい。

そこで、サービスにおける単位時間あたりのリクエストの処理数をサービスのスループットとし、サービスのスループットを最大化する分散実行方針を求めるための最適化アルゴリズムを提案する。サーバでのコンポーネント（トランジション）の実行やサーバ間のデータ転送は、サーバの処理能力やサーバ間のリンク容量によって、単位時間あたりに処理できる量（以降、スループット）は制限される。同様に、データベースへの参照を伴う処理に関しても、データベースの処理能力によって、スループットは制限される。これらの制約の中

でサービスのスループットを最大化することを考える。提案アルゴリズムでは、このようなサービスの最適化問題を次のような整数線形計画問題に帰着させる。

整数線形計画問題では、サービスを構成するプレース、あるいは、トランジション  $v$  を、どのサーバ  $n$  へ割り当てるかを表す 0-1 変数  $Alloc(v, n)$  を導入する。すなわち、すべての元  $v$  が、ただ一つの元  $n$  へ割り当てられることを仮定すると、すべての  $v \in V$  (ただし、 $V = P \cup T$ )、 $n \in N$  についての  $Alloc(v, n)$  は、サービスグラフの頂点集合  $V$  から、ネットワークグラフの頂点集合  $N$  に対する写像  $Alloc: V \rightarrow N$  を与える。整数線形計画問題においては、全域写像  $Alloc: V \rightarrow N$  が解であり、サービスの最適化問題における分散実行方針を表す。そして、 $Alloc(v, n)$  が与えられた時の、サービス全体のスループット  $Th(S)$  を定義し、このスループット  $Th(S)$  を最大化する写像  $Alloc: V \rightarrow N$  を求めることを考える。

まず、整数線形計画問題の入力を以下のように定義する。

#### [入力]

- サービスモデル  $S = (P, T, A)$
- ネットワーク  $G = (N, L)$
- 最大スループット  $m_{exec}(t, n)$ 、および、 $m_{msg}(v_1, v_2, n_1, n_2)$

サービスモデル  $S$  は、 $(P, T, A)$  で表現されるカラーペトリネットモデルであり、 $P$ 、 $T$ 、 $A$  は、それぞれプレース集合、トランジション集合、アーク集合に対応する。さらに、リクエストを入力するプレース  $p_{in}$  (ただし、 $|\cdot p_{in}| = 0$ ) と出力するプレース  $p_{out}$  (ただし、 $|p_{out} \cdot| = 0$ ) をそれぞれ 1 つずつ持ち、閉路を含まないことを仮定する。ここで、 $\cdot p$  と  $p \cdot$  は、それぞれプレース  $p$  の入力トランジション集合、出力トランジション集合を表す。同様に、 $\cdot t$  と  $t \cdot$  (ただし、 $t \in T$ ) は、それぞれトランジション  $t$  の入力プレース集合、出力プレース集合を表す。なお、データベースを表すための自己ループについては、4.2 節で述べるような変換方法を適用することで、本最適化問題へ適用可能なサービスモデルを得ることができる。

ネットワークモデル  $G$  は、 $(N, L)$  で表現される、ノード集合  $N$  とリンク集合  $L$  より構成される完全グラフとする。

最大スループット  $m_{exec}(t, n)$  は、トランジション  $t$  をサーバ  $n$  で実行するとき、サーバ  $n$  が本来持つトランジション  $t$  についての処理スループットの上界を与える定数である。同様に、 $m_{msg}(v_1, v_2, n_1, n_2)$  は、サーバ  $n_1$  からサーバ  $n_2$  へアーク  $(v_1, v_2)$  を通過するトークンの保持するデータが、転送されるときのスループットの上界を表す。これらの指標は、

サーバの処理能力やネットワークの通信容量、あるいは、コンポーネントを実行するために必要な計量量やコンポーネント間のデータ量などを考慮し与えられる相対的な値である。

#### [目的関数]

本問題では、単位時間あたりのリクエストの処理数を最大化するために、サービスのスループット  $Th(S)$  を定義し、この  $Th(S)$  を最大化することを目的とする。ここで、サービスのスループットとは、サービスの出力プレース  $p_{out}$  に出力されるトークンの単位時間当たりの数とする。トランジション  $t$  について、入力プレースからトークンを取得し、トランジション  $t$  の実行を行い、出力プレースにトークンを出力するまでの一連の処理についてのスループットを  $Th(t)$  としたとき、サービスのスループット  $Th(S)$  は、以下のような式で表される。

$$Th(S) = \sum_{t \in p_{out}} Th(t) \quad (1)$$

#### [スループットに関する制約式]

一般に、並行システムにおけるスループットは、システムの処理を構成するサブコンポーネントの中で、最もボトルネックとなるサービスコンポーネントのスループットが、システム全体のスループットを決定する。本論文の想定するサービスでは、下記に挙げるスループットが、サービスのスループットに大きく関係する。

- サービスコンポーネントの実行スループット
- サーバ間でのデータ転送スループット

各地のサーバに分散して蓄えられている膨大なデータを活用したサービスでは、実行方針によっては、データ転送に時間がかかることがあるため、単純に各サーバでのサービスコンポーネントの処理スループットを考慮するだけでは、スループットは最適とは限らない。ゆえに、計算機リソースだけでなくネットワークリソースも含めた、ネットワーク全体のリソースを考慮して最適化を行うため、上記のようなスループットを考える。そして、それぞれのスループットの中で、最も小さいスループットがボトルネックであり、コンポーネント処理全体のスループットとなる。

そこで、あるトランジション  $t$  の実行について考えると、以下のようなスループットの中で最も小さいスループットが、トランジション  $t$  の発火全体のスループット  $Th(t)$  となる。

- トランジション  $t$  自身の処理スループット
- トランジション  $t$  への入力に必要なトークンの転送スループット
- トランジション  $t$  から出力されたトークンの転送スループット

トランジション  $t$  の処理をサーバ  $n$  で実行したときのスループットを  $Th_{exec}(t, n)$ , アーク  $(v_1, v_2)$  のデータをサーバ  $n_1$  からサーバ  $n_2$  まで転送するときのスループットを  $Th_{msg}(v_1, v_2, n_1, n_2)$  としたとき, スループット  $Th(t)$  は以下のような式によって表される.

$\forall t \in T$ , について,

$$X = \{x | x = Th_{exec}(t, n) \wedge x \neq 0 \wedge n \in N\} \quad (2)$$

$$Y = \{x | x = Th_{msg}(p, t, n_1, n_2) \wedge x \neq 0 \wedge p \in \cdot t \wedge (n_1, n_2) \in N \times N\} \quad (3)$$

$$Z = \{x | x = Th_{msg}(t, p, n_1, n_2) \wedge x \neq 0 \wedge p \in t \cdot \wedge (n_1, n_2) \in N \times N\} \quad (4)$$

$$Th(t) = Min(X \cup Y \cup Z) \quad (5)$$

$Min(A)$  は, 集合  $A$  から最小である元を取り出す関数である. トランジション  $t$  がサーバ  $n$  に割り当てられない場合,  $Th_{exec}(t, n) = 0$  であるため (後述の式 (8) を参照), 式 (2) においては, 集合  $X$  から  $\{0\}$  を除外している. 式 (3) や式 (4) においても同様である. また, 目的関数は最大化関数であるため, 式 (2)~(5) は下記のような線形式で表すことができる.

$\forall t \in T, \forall n \in N$  について,

$$Th(t) - Th_{exec}(t, n) - C * (1 - Alloc(t, n)) \leq 0 \quad (6)$$

$\forall (v_1, v_2) \in A, \forall (n_1, n_2) \in N \times N$  について,

$$Th(t) - Th_{msg}(v_1, v_2, n_1, n_2) - C * (1 - Msg(v_1, v_2, n_1, n_2)) \leq 0 \quad (7)$$

$C$  は十分大きな定数とする. また,  $Msg(v_1, v_2, n_1, n_2)$  は, アーク  $(v_1, v_2)$  の  $v_1$  がサーバ  $n_1$ ,  $v_2$  がサーバ  $n_2$  に配置されている場合に発生するデータ転送の存在を表す 0-1 変数である. 式 (6) は, トランジション  $t$  がサーバ  $n$  に割り当てられない場合, 常に真となる. 式 (7) においても同様である. さらに, スループット  $Th_{exec}(t, n)$ , および,  $Th_{msg}(v_1, v_2, n_1, n_2)$  は, 次のような制約を持つ.

$\forall t \in T, \forall n \in N$  について,

$$Th_{exec}(t, n) - m_{exec}(t, n) * Alloc(t, n) \leq 0 \quad (8)$$

$\forall (v_1, v_2) \in A, \forall (n_1, n_2) \in N \times N$  について,

$$Th_{msg}(v_1, v_2, n_1, n_2) - m_{msg}(v_1, v_2, n_1, n_2) * Msg(v_1, v_2, n_1, n_2) \leq 0 \quad (9)$$

一方, プレースについて着目すると, あるプレース  $p$  におけるトークンの入出力をフローとしてみなしたとき, 以下のような関係式が成り立つ.

$\forall p \in P$  について,

$$\sum_{t \in \cdot p} Th(t) - \sum_{t \in p \cdot} Th(t) \geq 0 \quad (10)$$

ただし, プレース  $p_{in}$ , および, プレース  $p_{out}$  に関しては, 上記の制約式 (10) を持たないこととする.

[サーバ間の通信に関する制約式]

アーク  $(v_1, v_2)$  の  $v_1$  がサーバ  $n_1$ ,  $v_2$  がノード  $n_2$  に配置されている場合に発生するデータ転送の存在を表す 0-1 変数である  $Msg(v_1, v_2, n_1, n_2)$  は, 以下のように定義できる.

$$Msg(v_1, v_2, n_1, n_2) = Alloc(v_1, n_1) * Alloc(v_2, n_2) \quad (11)$$

$Alloc(v_1, n_1)$ , および,  $Msg(v_1, v_2, n_1, n_2)$  は, ともに 0-1 変数であるため, 式 (11) は, 下記のような線形式で表すことができる.

$$Alloc(v_1, n_1) - Msg(v_1, v_2, n_1, n_2) \geq 0 \quad (12)$$

$$Alloc(v_2, n_2) - Msg(v_1, v_2, n_1, n_2) \geq 0 \quad (13)$$

$$Alloc(v_1, n_1) + Alloc(v_2, n_2) - Msg(v_1, v_2, n_1, n_2) \leq 1 \quad (14)$$

[配置に関する制約式]

それぞれのプレース, および, トランジション  $v$  は, 必ずいずれかの 1 台のサーバ  $n$  へ配置されなければならないため, 以下のような制約式が成り立つ.

$\forall v \in V$  について,

$$\sum_{n \in N} Alloc(v, n) = 1 \quad (15)$$

これらの目的関数および制約式は, すべて線形式であり, かつ, 変数  $Th$ , および,  $Msg$  は, 0-1 変数であることから, 本最適化問題は整数線形計画問題に属する. 以上のように, サービスの最適化問題を線形計画問題へ帰着させ, これを解くことによって, 最適な分散実行方針を求めることができる.

#### 4.2 サービスの変換とそれに伴う制約式の追加

2章で仮定したサービスでは, データベースを表現するために自己ループを持つことができるが, 4.1 節の最適化問題で与えるサービスモデルは閉路を持たないことを仮定しているため, 2章のサービスモデルでは最適化問題への入力としてそのまま適用できない. 本節では, 最適化問題へ入力可能なサービスモデルを得るための, サービスモデルの構造的な変換と制約式の追加について述べる.

一般に, データベースへのデータの検索やマッチング処理は, データベースへの大量の

データ参照を伴うため、データベースの存在するサーバと同じサーバで行われることが多い。そこで、自己ループの関係にあるデータベースを表すプレース  $p$  とトランジション  $t$  を同じサーバ  $n$  に割り当てることを仮定する代わりに、プレース  $p$  とプレース  $p$  に接続するすべての入出力アークを除去する。

データベースを表すプレース集合を  $D = \{p | p \in P \wedge \cdot p = p\}$  (ただし,  $D \subset P$ ) とすると、以下に示される式によってサービスモデル  $S' = (P', T, A')$  を得ることができる。

$$P' = P \setminus D \quad (16)$$

$$A' = A \setminus A^- \quad (17)$$

$$A^- = \{(t, p) \in A | t \in \cdot p \wedge p \in D\} \cup \{(p, t) \in A | t \in p \cdot \wedge p \in D\} \quad (18)$$

自己ループの関係にあるプレース  $p$  とトランジション  $t$  が、同じサーバ  $n$  へ割り当てられることを表す制約は、以下のような式となる。

$$\forall p \in D, \forall t \in \cdot p, \forall n \in N \text{ について,} \\ Alloc(p, n) - Alloc(t, n) = 0 \quad (19)$$

かつ、式 (15) と同様に、プレース  $p \in D$  は、以下のような配置に関する制約を持つ。

$$\forall p \in D \text{ について,} \\ \sum_{n \in N} Alloc(p, n) = 1 \quad (20)$$

また、データベースを表すプレース  $p \in D$  がサーバ  $n$  に割り当てられたときのデータベースの処理スループットの上界を表す定数を  $m_{ab}(p, n)$  とすると、プレース  $p$  と自己ループの関係にあるトランジション  $t$  について、以下のような制約式が成立する。

$$\forall p \in D \text{ について,} \\ m_{ab}(p, n) - \sum_{t \in p \cdot} Th(t) \geq 0 \quad (21)$$

制約式 (19)~(21) は線形式であり、4.1 節と同様に整数線形計画問題に帰着できる。以上のような変換によって得られたサービスモデル  $S'$  を入力とし、かつ、これらの制約式を加えて 4.1 節の問題を解けば、プレース集合  $D$  を含めた写像  $Alloc: V \rightarrow N$  を得ることができる。

### 4.3 処理の振り分けを考慮した最適化アルゴリズムへの拡張

我々が想定するネットワークアーキテクチャのサーバは、サービスコンポーネントの処理ごとに、リクエストを他のサーバへ振り分けられることのできるリクエストのディスパッチ機能を持つ。そこで、本節では、4.1 節の最適化アルゴリズムを振り分けを考慮した最適化アル

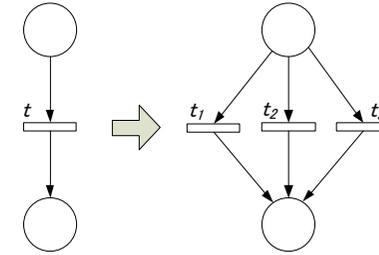


図 4 トランジションの変換例

Fig.4 Example of transformation of transition.

ゴリズムへ拡張する。

まず、サービスモデル  $S = (P, T, A)$  のある一つのトランジション  $t \in T$  を  $k$  台のサーバへ振り分けることを考える。トランジションの振り分けは、基本的にトランジション  $t$  を  $k$  個の元からなるトランジション集合に置き換え、トランジション  $t$  の入出力プレースから対応するアークをそれぞれ競合の関係に連結することと同じである。例えば、図 4 は、トランジション  $t$  を、3 台のサーバへ振り分けた場合の変換の例を表している。トランジション  $t$  について、置き換えるトランジション集合を  $T_d(t) = \{t_1, t_2, \dots, t_k\}$  (ただし,  $|T_d(t)| = k$ ) とすると、下記に示される式によってサービスモデル  $S' = (P, T', A')$  が得られる。

$$T' = T \setminus \{t\} \cup T_d(t) \quad (22)$$

$$A' = A \setminus A^- \cup A^+ \quad (23)$$

$$A^- = \{(p, t) \in A | p \in \cdot t\} \cup \{(t, p) \in A | p \in t \cdot\} \quad (24)$$

$$A^+ = \{(p, t') \in A | p \in \cdot t \wedge t' \in T_d(t)\} \cup \{(t', p) \in A | p \in t \cdot \wedge t' \in T_d(t)\} \quad (25)$$

さらに、すべてのトランジション  $t \in T$  について、下記に示される式を繰り返し適用し、結果として得られるサービスモデル  $S' = (P, T', A', d(t))$  を入力とすれば、4.1 節のサービス最適化問題をそのまま適用できる。

## 5. サービス実行プラットフォームの設計と実装

本論文では、提案アルゴリズムを利用しサービスの分散実行方針を導出するだけでなく、その実行方針に基づき、実環境においてサービスを分散協調実行するサービス実行プラットフォームの設計と実装を行った。

本プラットフォームに対し、サービス記述と利用する分散環境に関する情報を与えると、

分散実行方針の導出だけでなく、その実行方針に基づいたサービスコンポーネントの配置やリンク確立など分散実行環境の構築を行い、効率よくサービスを実行できる環境を自動的に整備する。このプラットフォームは、主に、分散化サービス導出部、サービス実行部の二つのモジュールから構成されている。分散化サービス導出部は、カラーペトリネットで作成したサービスモデルから、それぞれの処理サーバで実行する分散化サービスを自動的に導出するためのモジュールである。また、サービス実行部は、導出部で導出した分散化サービスを利用して、実際にサービスをそれぞれのサーバへ配備、および、実行するためのモジュールである。以下、分散化サービス導出部とサービス実行部の詳細について述べる。

### 5.1 分散化サービス導出部

分散化サービス導出部では、カラーペトリネットで記述されたサービスモデルとネットワーク情報から分散化サービスを導出する。分散化サービスを導出するまでのデータフローを図5に示す。サービスモデルの記述には、カラーペトリネットを記述可能なCPN Toolsを利用し、CPN Toolsの出力ファイルが利用可能である。また、ネットワーク情報は、3章で示したネットワーク環境情報の内容を記述したXMLファイルを利用する。図5に分散化サービス導出部のデータフローを示す。

モデル解析では、サービスモデルとネットワーク環境情報から4.1節に従った目的関数と制約式、および、カラーペトリネットの実行に必要なサービスモデルに関する情報を生成する。モデル解析後、生成した目的関数と制約式を整数線形計画問題として解き、分散実行方針を決定する。整数線形計画問題の解により、まず、各ブレースがどのサービスサーバに配置されるかが決定される。

### 5.2 サービス実行部

サービス実行部では、分散化サービス導出部で導出した分散化サービスに基づき、サービスを実行できる環境の整備を行う。サービス実行部は、インターフェイスサーバ、制御サーバ、実行サーバから構成されている(図6)。各サーバの役割は次の通りである。インターフェイスサーバとは、サービスを利用するユーザからの要求受付や、ユーザへ要求結果を通知する役割をもつ。制御サーバは、サービス全体のデータフロー制御、および実行サーバが保持するサービスコンポーネントの実行タイミングを制御する役割を持つ。実行サーバは、制御サーバの要求に従い、サービスコンポーネントの実行を行う役割を持つ。

図6では、サーバの分類を三種類としているが、この概念は論理的なもので、必ずしもこれらのサーバを物理的に別々のサーバに分ける必要はない。最適化を行った結果、ボトルネックにならないのであれば、制御サーバと受付ノードのプログラムを同じサーバで実行される

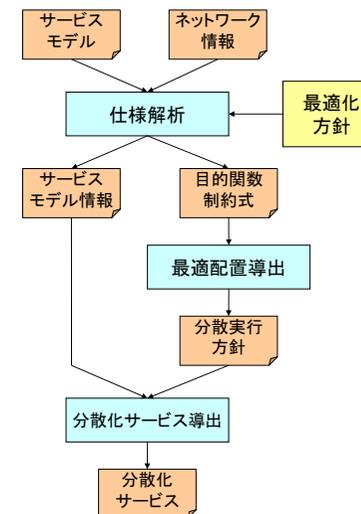


図5 分散化サービス導出部のデータフロー

Fig. 5 Dataflow Description of Protocol Specifications.

ような配置が導出される場合も考えられる。次項では、各サーバの設計について詳細を述べる。

#### 5.2.1 インターフェイスサーバ

要求受付では、ユーザから、サービス利用要求と共に、サービス実行に必要な情報を受け取る。インターフェイスサーバは、ユーザからの要求を受け付けた後、制御サーバにその要求の制御を委譲するとともに、サービス実行に必要なデータを実行サーバに配置する。また、サービス実行が終了した際には、制御サーバからサービス実行終了通知を受け、実行サーバから出力データを取得し、ユーザに要求結果の通知を行う。

インターフェイスサーバは、主にユーザインターフェイスとサービスインターフェイスから構成されている。ユーザインターフェイスを変更可能で、今回の実装では、Webベースのインターフェイスを採用している。また、テスト用にダミーインターフェイスを実装しており、ユーザインターフェイスの一つとして置き換えが可能である。ダミーインターフェイスは、サービスの負荷テストなどのために、仮想的にユーザからの要求を生成するモジュールで、例えば、指数分布に従ったユーザ要求を発生させることができる。

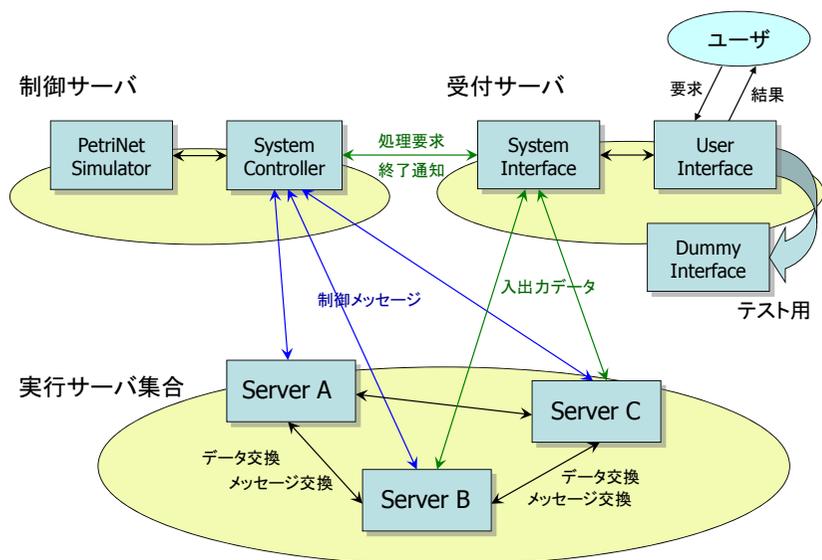


図 6 サービス実行部全体図

Fig. 6 Overview of Service Execution System.

### 5.2.2 制御サーバ

制御サーバでは、サービスサーバ毎にその挙動をカラーペトリネットで表現し、それらの挙動と連携を実現するシミュレータを保持している。このシミュレータによる分散化サービスの動作結果に基づいて、サービス全体の制御を行う。このように、制御サーバではシミュレータモジュールとサービス制御モジュールの二つのモジュールから構成される。

#### [カラーペトリネットシミュレータ]

図 7 にカラーペトリネットシミュレータの構成を示す。サービスサーバ毎にサブシミュレータとカラーペトリネットモデルを 1 組ずつ保持する。トークンはユーザの要求およびデータの流れを表している。このトークンがあるカラーペトリネットのコミュニケーションプレースに移動し、対応する別のカラーペトリネットへトークンが移動することで、サービスサーバ間の連携が実現される。また、この処理と同時に、対応する実行サーバ間でデータを移動させるための、制御メッセージを実行サーバへ送信する。データ移動完了後、データを受け取った実行サーバから送信完了通知を受信した段階で、シミュレータ内のトークン移

動を完了する。

同様に、トランジションの発火では、実行サーバに対しサービスコンポーネントを実行させるための制御メッセージを送信する。実行サーバでは、この制御メッセージによってサービスコンポーネントを実行する。サービスコンポーネントの実行完了後、実行サーバからの実行完了通知を受信し、トランジションの発火を完了する。

イベントチェッカは、シミュレータと実際の実行を橋渡しする役割を持つ。シミュレータの状態を実際の実行に反映するために、トランジション発火毎に、上記のような実行サーバへのデータの転送要求やサービスコンポーネントの実行要求が可能となった状態であるかをチェックし、それぞれ実行サーバ間のデータ転送、実行サーバへのサービスコンポーネントの実行要求を行う。また、実際の実行結果をシミュレータに反映するために、データ転送要求、サービスコンポーネント実行要求が実行サーバで完了したことを通知するメッセージに基づいて、シミュレータのコミュニケーションプレース間でのトークンの移動を完了したり、トランジションの発火を完了する。

カラーペトリネットシミュレータでは、カラーペトリネットモデルに存在するアークラベルやトランジションの条件式を解釈を行う必要がある。また、トークン集合が与えられた際、トランジションに対する条件式の評価や、出力アークのアークラベルから生成される新たなトークンの評価を行う必要がある。そのため、入出力アークのアークラベルやトランジションの条件式を、解釈・評価することのできるインタープリタを実装している。実行ロジックは、カラーペトリネットシミュレータをどのように動かすか（どのトランジションを発火させるか）を決定する役割を持つ。

#### [サービスコントローラ]

サービスコントローラは、サービス全体の制御を行う役割と、実行サーバとの接続を確保し入出力インターフェースをカラーペトリネットシミュレータに対して提供する役割（サーバコントローラ）の二つの役割をもつ。

サービス全体の制御では、本プラットフォームが起動してからサービスを自動的に展開し実行可能となるまでの実行制御（起動シーケンス）と終了コマンドを受けてから、各実行サーバのプログラムを終了させ、本プラットフォームを終了させるまでの実行制御（終了シーケンス）を行う。

起動シーケンスでは、まず分散化サービスを入力としてペトリネットシミュレータを初期化する。その過程で、サービス実行に必要な実行サーバ数など実行サーバに関する情報が確定され、その情報を利用して実行サーバに対して接続を行う。このとき、実行サーバに対し

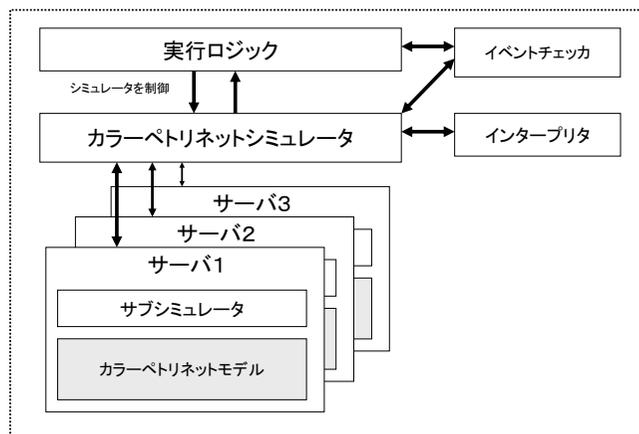


図 7 カラーペトリネットシミュレータの構造  
Fig. 7 Colored Petri Net Simulator.

てプログラムなど必要なデータを転送する。全ての実行サーバで実行サーバプログラムの起動後、オーバーレイネットワークの構築を行う。このオーバーレイネットワークは、カラーペトリネットシミュレータのコミュニケーションプレースによって結び付けられているサーバ間を全て相互に接続する。最後に、カラーペトリネットシミュレータの動作を開始して起動シーケンスは終了する。

終了シーケンスでは、まずシミュレータの動作を停止する。次に全ての実行サーバに対して終了メッセージを送信し、実行サーバプログラム終了後、終了スクリプトを実行する。終了スクリプトは、実行サーバプログラムが終了した後に、実行サーバ側で実行するコマンドを記述したシェルスクリプトである。回収したいログファイルなどがある場合、ログデータを転送するコマンドをここに記述することができる。同様に、受付サーバのプログラムも終了させ、最後に、制御サーバで実行しているプログラムを終了し、終了シーケンスは完了する。

サーバコントローラの構成を図 8 に示す。サーバコントローラでは、制御サーバと実行サーバ間の接続を管理し、カラーペトリネットシミュレータでの実行サーバと実際に実行サーバプログラムを実行するホストをマッピングする。Node Maintainer は接続の管理、Node Mapper はホストとのマッピング管理を担当している。Message Processor は、制御

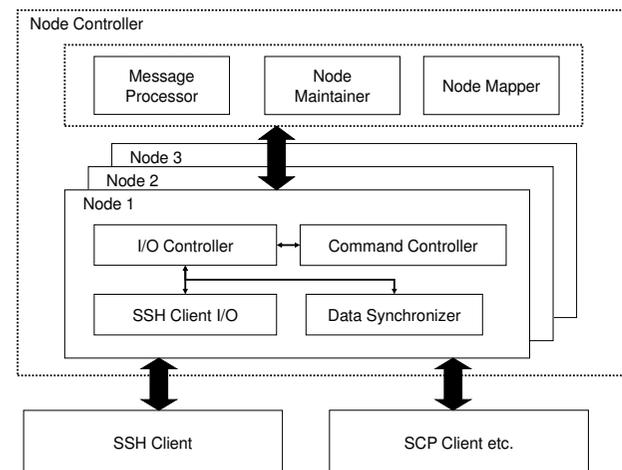


図 8 サーバコントローラの構造  
Fig. 8 Node Controller.

サーバと実行サーバ間で通信される制御メッセージに関する処理を行う。制御サーバと実行サーバ間の通信には SSH を採用し、サーバコントローラでは、SSH クライアントの標準入出力に対してプロセス間通信で入出力を行うことで、実行サーバとの通信を行っている。Data Synchronizer は、SCP や RSYNC 等を利用し、実行サーバ間とのファイルの同期を行っている。Command Controller は、起動シーケンスでの起動スクリプトや終了シーケンスの終了スクリプトなど、実行サーバ側で実行するコマンドの送信を担当している。

### 5.2.3 実行サーバ

実行サーバでは、制御サーバからの要求に応じて処理を行う。処理とは、主にサービスコンポーネントの実行やサービスコンポーネント実行に必要なデータの転送の二種類ある。サービスコンポーネントの実行スケジューリングやタイミングは制御サーバが全て行うため、実行サーバは制御サーバからの要求を受けて処理を実行し、実行が終了すれば、完了通知を制御サーバへ送信する。データの転送は、指定された実行サーバ間で直接行われる。サービスコンポーネントの実行と同様に、転送が完了すると、制御サーバへ通信結果が通知される。

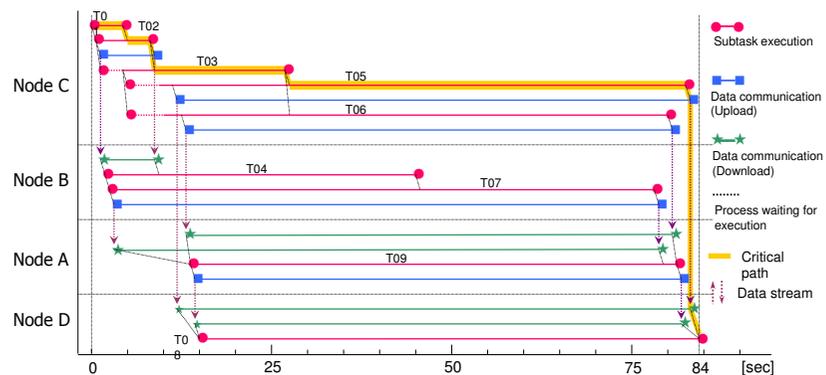


図 9 タイムチャート  
Fig. 9 Timing chart.

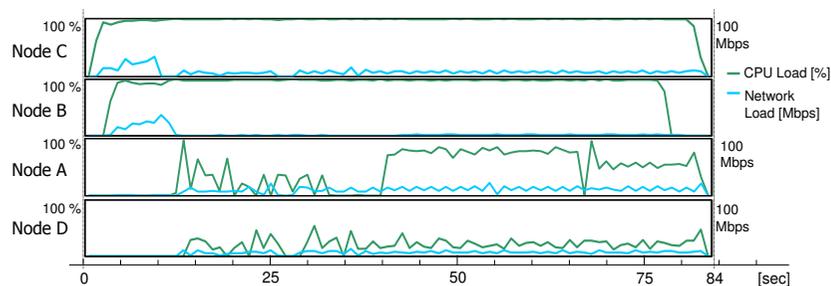


図 10 サーバやネットワークの負荷  
Fig. 10 CPU and network load.

### 5.3 サービス実行時のモニタリング機能

本プラットフォームでは、サービスの実行時において、下記のような統計情報がリアルタイムにモニタリングできる機能を持つ。

- タイムチャート
- サーバやネットワークの負荷
- 処理待ちトークン数

図 9 に、タイムチャートの例を示す。タイムチャートでは、各トランジションの実行のタ

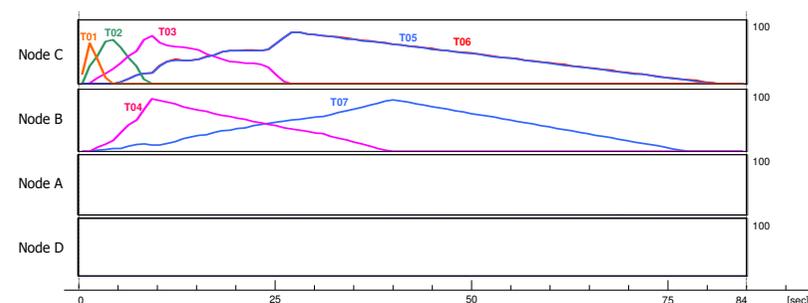


図 11 処理待ちトークン数  
Fig. 11 # of tokens to be processed.

イミングやサービス全体のクリティカルパスなどがわかる。トランジション間の縦向き破線は依存関係を表す。図 10 に、サーバやネットワークの負荷の例を示す。それぞれのサーバにおける負荷状況が把握でき、複数のサーバの負荷状況と見比べることで、分散処理の推移が視覚的に把握できる。図 11 に、処理待ちトークン数の例を示す。それぞれのトランジションにおいて、処理待ちのトークンがどのくらいあるかを表す。この値が増加するという事は、サーバの処理能力以上のトークンが流入しているということを意味する。

これらの情報から、サービスの実行状況の把握やボトルネックなどサービスの性能の評価を行うことができる。

## 6. 評価実験

評価実験では、図 2 の交通解析サービスを対象に、本プラットフォームを利用して、本論文が提案したサービスの最適な分散実行方針の導出手法について評価するために実環境上で性能試験を行った。

### 6.1 実験環境

互いに同じドメインに含まれないように選択した PlanetLab 上のサーバ 30 台を用い、本プラットフォーム上でユニキャストリンクによるフルメッシュオーバーレイを構築し、その上でサービスを実行した。さらに、その中のサーバ 1 台をネットワーク外部からリクエストを受け付けることのできる特別なサーバとみなし、ただ一つのインターフェースサーバとした。また、図 2 のプレース P13 とプレース P14 は、地理的に依存するリポジトリであるため、それぞれ特定のサーバに割り当てられるように設定した。

## 6.2 実験手法

図2の交通解析サービスに対して300個のリクエストを投入し、その際の実行状況を観察した。300個のリクエストのうち、それぞれ60個についてトランジション *T2* の渋滞予測処理、120個についてトランジション *T8* の映像検索、120個についてトランジション *T9* の映像検索が行われるリクエストによって構成されているものとする。これらのリクエストは疑似的にインターフェースサーバにて発行した。

様々なパターンが考えられる分散実行方針の中で、提案手法による分散実行方針はどの程度の性能を発揮するかを評価するために、様々な分散実行方針に基づいてサービスの実行を行い網羅的に比較した。具体的には、比較対象として、ランダムに配置した分散実行方針を100パターン用意した。その際、それぞれのトランジションについて高々3台のサーバまで振り分け処理を行うように生成した。

提案手法を用いた分散実行方針の導出では、同じく高々3台の振り分け処理を許可し、入力を与える最大スループットは、本プラットフォームの機能を利用し、それぞれのサーバのCPU処理能力、および、サーバ間のリンクの利用可能帯域を測定した上で、その値に基づいて我々が与えた。

## 6.3 スループット

図12に、100秒当りに処理できた平均リクエスト数（スループット）に関する度数分布表を示す。一方、提案手法による分散実行方針では、スループットは114であった。図12と比較すると、提案手法による分散実行方針は、ランダムな配置100パターン中、上位3番目のスループットであり、明らかに最適化の効果が出ていることが確認できる。

また、サービスはPlanetLabのサーバ上で実行しているため、様々なアプリケーションがサーバ上で実行されており、サーバやネットワークの負荷状況が刻一刻と変化する環境である。そのため、負荷状況によってサービスのスループットも変化するものと考えられる。

そこで、同じ分散実行方針で複数回サービスの実行を行い、スループットがどのように変化するか実験した。図13では、連続で実行とあるのが、あるパターンの分散実行方針を10回連続で実行したときのスループット分布である。一方、間隔をあけて実行とあるのが、図13と同じ分散実行方針ではあるが、実行間隔を2時間以上あけて実行したときのスループットの分布である。両者を比較すると、ともに実行するごとにスループットが分散しているが、実行間隔をあけて実行したほうがより分散していることがわかる。これは、時間間隔が開くほどサーバやネットワークの負荷状況が変動するためだと考えられる。

連続で実行した場合でも、スループットは20程度の差がみられるので、提案手法による

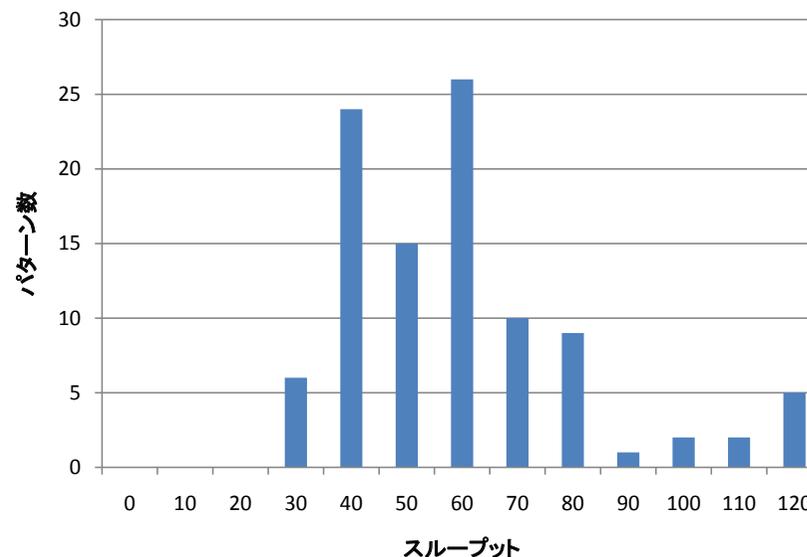


図12 各分散実行方針におけるスループットの度数分布  
Fig.12 Frequency distribution of throughput.

分散実行方針のスループットは最大ではないが、上位1番目のスループットと比較しても差は2程度であり、ほぼ最適なスループットと言える。

## 6.4 ボトルネック

図14は、ある分散実行方針についてのトランジション *T4* と *T6* における処理待ちトークン数の推移を表している。トランジション *T4* と *T6* ともに処理待ちトークンが発生しているため、トランジションが割り当てられたサーバの処理能力を超えるトークンが投入されていることがわかる。一方、処理待ちトークン数の傾きは、トークンの流入量に対する相対的なスループットを表す。流入するトークンがないと仮定すれば、減少量の傾きが急なほど処理スループットは大きいといえる。つまり、トランジション *T4* は、*T6* よりも処理スループットは大きい。

このことから、ともにサーバの処理能力以上のトークンが投入されているが、ボトルネックになっている個所は、トランジション *T4* ではなく、*T6* であることがわかる。この場合、

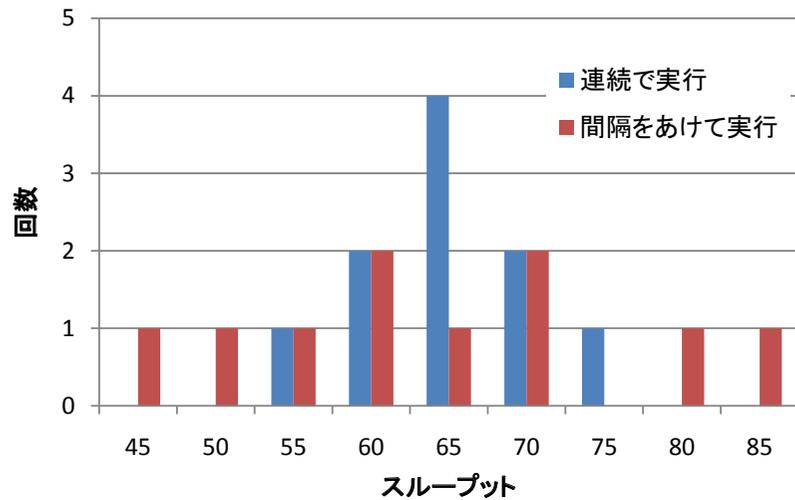


図 13 同じ分散実行方針におけるスループットの分布  
Fig. 13 Distribution of throughput on a service.

T6 の処理を振り分けるサーバを追加するなどによって、サービスのスループットが向上する可能性がある。

以上のように、本プラットフォームでは、ログ収集機能を用いることによって、このようなボトルネックの発見が容易に可能である。

## 7. 関連研究

コンポーネント間の連携によるサービスを表現するためのモデルとして単純な逐次実行パスを仮定するもの<sup>2)</sup>や無閉路有向グラフ (DAG) を用いるもの<sup>3),4)</sup>などが知られている。これらの逐次実行パスや無閉路有向グラフと比較して、ペトリネットをモデルとしたアルゴリズムを用いることで、提案システムはより現実的なサービスを扱うことを可能としている。カラーペトリネットでは、これらのモデルと比較して、ループや条件分岐といったより複雑な制御が表現可能で、DB の処理を記述したり、あるいは、ユーザからのリクエストの内容に応じて、サービスの実行内容を変更するといった記述が可能となる。

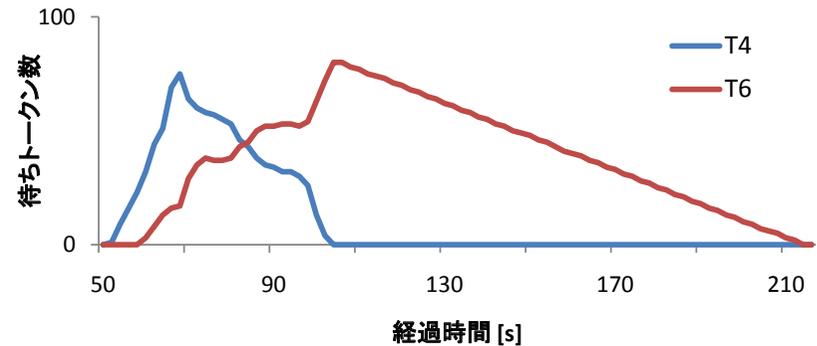


図 14 ボトルネックの例  
Fig. 14 Example of bottleneck.

P2P コンピューティング<sup>5)</sup> やグリッドコンピューティング<sup>6),7)</sup>, あるいは、サービスオーバレイ<sup>2)-4)</sup> といった新たな分散コンピューティングパラダイムが考案されてきている。これらの研究では、オーバレイネットワークは、様々な処理能力を持つ計算機によって構成され、ネットワークを構成する計算機が動的に変化するため、動的に変化するネットワークリソースをどのように管理するのかという問題がある。さらに、そのようなネットワーク上で、計算機間の依存関係を考慮しつつサービスを構成するといった複雑な分散プログラミングをどのように行うかという問題もある。これらに対し、文献 5), 8)–11) などにおいて、計算機支援による分散システム的设计手法が提案されている。iOverlay<sup>8)</sup> では、オーバレイサービスの実装と評価を行うための機能を持ったオーバレイネットワーク上での分散プラットフォームを提供している。メッセージの送受信を効率よく行うための仕組みを導入し、サービス設計者がオーバレイネットワーク上でのアプリケーションの実装に集中できるようになっている。また、Arigatoni<sup>9)</sup> は、サービスを実行するために必要なリソースを自動で発見する仕組みを提供することによって、分散サービスの実装を支援する。5), 10) では、分散サービスを設計する際的设计コストの低減に焦点を当てている。MACEDON<sup>11)</sup> は、シミュレータと PlanetLab のようなテストベッドの両方でサービスを実行できるようにすることで、P2P ネットワーク上での分散サービスの解析と評価ができる。これらに対し提案手法では、ネットワークの環境情報とサービスの記述を与えるだけで、自動的に分散

化サービスを導出することができる。またその際、サービスの性能を最適化することができる。結果として、分散協調サービスの設計コストを軽減できると期待される。

## 8. おわりに

本論文では、オーバーレイネットワークを構成するサーバ群を利用し、アプリケーションサービスを効率良く分散協調実行するためのアルゴリズムを提案した。ノード間がそれぞれ論理的に直接結合されたオーバーレイネットワーク上において、複数のコンポーネントの逐次、並列、分岐による結合により記述されたサービスを、サーバの計算能力、ネットワークの通信遅延や利用可能帯域などを考慮し、どのコンポーネントをどのサーバで実行するかを表す分散実行方針を決定できる。また、本手法を用いて、実環境におけるサービス実行までをシームレスに行うことができるサービス実行プラットフォームの設計及び実装を行った。サービス実行プラットフォームでは、決定された分散実行方針に従って、サーバへのコンポーネント配置や分散実行を制御できる。また、ネットワークやサーバの負荷状況を収集し、そのもとで即座に最適な分散実行方針を導出できる。遠隔地に蓄積されたトラフィック情報を用いてデータ解析を行うサービスを対象に評価実験を行い、高いスループットでサービスを効率良く分散協調実行できる実行方法を実現できることを確認した。

近い将来、モバイル機器に環境センサが搭載され、それらが互いにネットワーク接続されれば、そのセンサ情報を収集、加工して利用する様々なサービスが生まれる可能性がある。そのようなサービスを、モバイル P2P ネットワーク内で自律的に処理する過程で、本論文で提案したシステム概念が必要になると思われる。今後の研究では、そのような環境下での課題を整理し、モバイルアーキテクチャに適した方法論とシステムプロトタイプ開発を行っていく予定である。

## 参考文献

- 1) K.Jensen. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use Volume 1: Basic Concepts*. Monographs in Theoretical Computer Science. An EATCS Series. Springer-Verlag, 1997.
- 2) D.Xu and K.Nahrstedt. Finding service paths in an overlay media service proxy network. In *Proc. of Int. Conf. on Multimedia Computing and Networking 2002 (MMCN2002)*, 2002.
- 3) M.Wang, B.Li, and Z.Li. sFlow: Towards resource-efficient and agile service federation in service overlay networks. In *Proc. of 24th Int. Conf. on Distributed*

- Computing Systems (ICDCS2004)*, 2004.
- 4) X. Gu, K. Nahrstedt, and B. Yu. Spidernet: An integrated peer-to-peer service composition framework. In *Proc. of IEEE Int. Symposium on High-Performance Distributed Computing (HPDC-13)*, 2004.
- 5) Kazuyuki Shudo, Yoshio Tanaka, and Satoshi Sekiguchi. Overlay weaver: An overlay construction toolkit. *Computer Communications*, 31(2):402–412, February 2008.
- 6) E. Korpela, D. Werthimer, D. Anderson, J. Cobb, and M. Leboisky. Seti@home-massively distributed computing for seti. *Computing in Science & Engineering*, 3(1):78–83, 2001.
- 7) Stefan M. Larson, Christopher D. Snow, Michael R. Shirts, and Vijay S. Pande. Folding@home and genome@home: Using distributed computing to tackle previously intractable problems in computational biology. *Computational Genomics*, 2002.
- 8) Baochun Li, Jiang Guo, and Mea Wang. iOverlay: A lightweight middleware infrastructure for overlay application implementations. In *Proceedings of the Fifth ACM/IFIP/USENIX International Middleware Conference (Middleware 2004)*, also *Lecture Notes in Computer Science*, pages 135–154, 2004.
- 9) Didier Benza, Michel Cosnard, Luigi Liquori, and Marc Vesin. Arigatoni: A simple programmable overlay network. In *JVA '06: Proceedings of the IEEE John Vincent Atanasoff 2006 International Symposium on Modern Computing*, pages 82–91. IEEE Computer Society, 2006.
- 10) Boon T. Loo, Tyson Condie, Joseph M. Hellerstein, Petros Maniatis, Timothy Roscoe, and Ion Stoica. Implementing declarative overlays. *SIGOPS Oper. Syst. Rev.*, 39(5):75–90, December 2005.
- 11) Adolfo Rodriguez, Charles Killian, Sooraj Bhat, Dejan Kostic, and Amin Vahdat. MACEDON: Methodology for automatically creating, evaluating, and designing overlay networks. In *Proceedings of the USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI2004)*, pages 267–280, 2004.