情報処理学会論文誌 Vol.53 No.11 2612–2623 (Nov. 2012)

推薦論文

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[1,a)] 廣森 聡仁[1] 山口 弘純[1] 東野 輝夫[1]

受付日 2012年1月21日, 採録日 2012年9月10日

概要：本論文では，複数のサーバに分散して蓄積された環境センシング情報などの大量の情報を活用したアプリケーションサービス（以下，サービス）を，分散環境で効率良く実行するためのサービス設計手法を提案する．この手法では，センシング情報の解析処理やパターンマッチングなど基本的なデータ処理単位をコンポーネントとし，サービスはそれらコンポーネントの逐次並列結合として与えられるものとする．このもとで，各サーバの計算能力，サーバ間オーバレイネットワークのデータ転送遅延などを考慮し，最適なデータ処理速度を達成するために，どのコンポーネントをどのサーバで実行すべきかを自動で決定する．また，この手法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行えるサービス実行プラットフォームの設計および実装を行っている．さらに，PlanetLab 上で評価実験を行い，実環境においても，提案手法と提案プラットフォームにより，効率的にサービス開発が行えることを確認する．

キーワード：分散アルゴリズム，オーバーレイネットワーク/P2P，ミドルウェアアーキテクチャ，ユビキタスコンピューティング

Design and Development of Service Execution Platform for Overlay Networks

Yuki Sakai[1,a)] Akihito Hiromori[1] Hirozumi Yamaguchi[1] Teruo Higashino[1]

Received: January 21, 2012, Accepted: September 10, 2012

Abstract: 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 derive optimal allocation of components that does not overload network links and servers. Using the platform, services can be installed and executed easily on real networks. We have conducted experiments on PlanetLab to validate our method. The experimental results have shown that the proposed method could derive efficient execution sequences and they could achieve higher throughput than the other methods.

Keywords: distributed algorithms, overlay networks/P2P, middleware architecture, ubiquitous computing

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，それらを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ユビキタスサービスを提供するための研究開発がさかんに行われている．また，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，交通流解析のためのカメラなど街角に設置されるセンサも増加すると予想される．そのような多数のセンサからときどき刻々と計測される大量のセンシング情報の集約や，それらを利用した多数のユビキタスサービスの実行を効率良く行うシステムの実現が課題と

[1] 大阪大学大学院情報科学研究科
Graduate School of Information Science and Technology, Osaka University, Suita, Osaka 565–0871, Japan
[a)] y-sakai@ist.osaka-u.ac.jp

本論文の内容は 2011 年 7 月のマルチメディア，分散，協調とモバイル（DICOMO2011）シンポジウム 2011 にて報告され，マルチメディア通信と分散処理研究会主査により情報処理学会論文誌ジャーナルへの掲載が推薦された論文である．

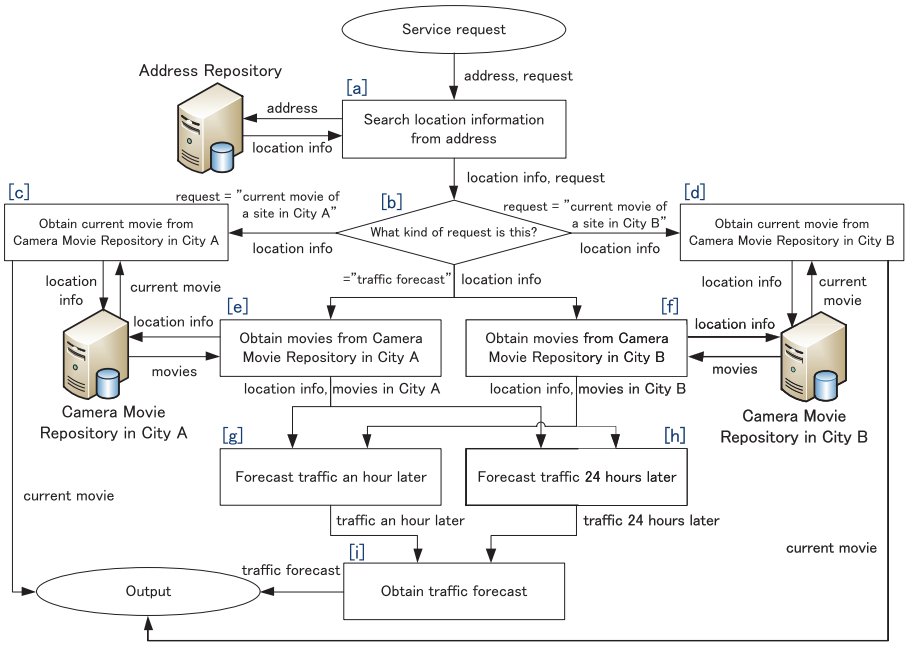

図 1 サービス例（交通解析サービス）
Fig. 1 Service example (transportation analysis services).

なっている．

たとえば，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 情報を地域ごとに設置された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，データ量，サーバ性能やリンクの容量に対する負荷を考慮し，データを蓄積するサーバ，処理を実行するサーバ，サーバ間のデータ転送を注意深く設計する必要がある．しかし，サービスやネットワークに最適な設計を行うことは設計者にとって容易でない．(2) サーバの性能やリンクの負荷は，サーバやネットワークの利用率の変化に応じて変化していく可能性がある．そのような変化に対しても性能を保証できるよう，システムをつねに管理していくことはかなりの手間をともなう．

本論文では，(1) の課題に対し，複数のローカルサーバ（以下，単にサーバとよぶ）により構成されるオーバレイネットワークにおいて，サービスを効率良く分散協調実行するためのアルゴリズムを提案する．サービスはその基本処理単位であるサービスコンポーネントの逐次，並列，分岐，同期の組合せで記述できるものとし，これをカラーペトリネット [1] でモデル化する．このモデルに対し，サーバ群の処理能力と各サーバ間の通信遅延，利用可能帯域などを考慮することにより，サーバやネットワークに与える負荷を分散しながら，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分散実行方針を機械的に決定するアルゴリズムを提案している．また (2) の課題に対し，サービスの設計から実現までを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の最適な配置を導出し，そのインストールと実行を自動で行う．さらに，サービスの実行時の性能を評価するために，実行時のサーバのスループットとネットワークトラフィックをリアルタイムで観測できる機能を実現している．これにより，負荷状況に基づきサービスの分散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したりするなど，サービス実行にともなう管理を容易にできるようにしている．

本プラットフォームを用いて，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を対象に，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に対する制約を守りつつ，より多くのリクエストを処理できる分散実行方針を導出できることを確認した．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も確認した．

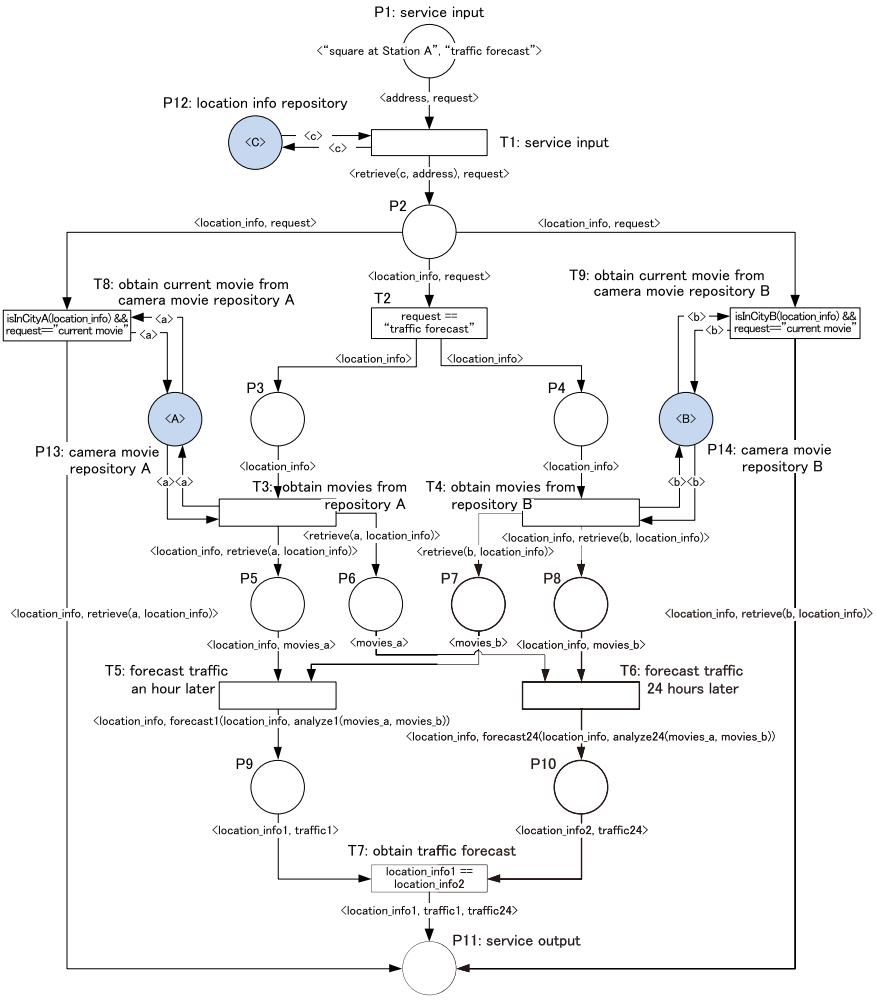

図 2 図 1 の交通解析サービスをカラーペトリネットで記述した例
Fig. 2 Transportation analysis services of Fig.1 formally written in Colored Petri net.

2. サービスの定義と記述例

本論文におけるサービスとは，センシングデータなどを入力とする 1 つの処理単位（これをサービスコンポーネントまたは単にコンポーネントとよぶ）を逐次あるいは並行に組み合わせたものを実行し，処理結果を出力するトランザクション処理とする．**図 1** は，ある 2 つの都市 A 市および B 市の街頭に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．まず，このサービスでは，交通解析の対象とする住所がリクエスト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アドレスのデータベース（以下，リポジトリ）から，最も近いカメラの位置など住所周辺の情報を検索する（コンポーネント [a]）．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から，A 市の場合はコンポーネント [e]，B 市の場合はコンポーネント [f] のように各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 時間後および 24 時間後の渋滞予測を行う．ここでは，1 時間後の渋滞予測がコンポーネント [g]，24 時間後の渋滞予測はコンポーネント [h] に対応する．最後に，コンポーネント [i] でそれらの結果をユーザに通知する．また，このサービスでは，リクエスト内容に応じて，その処理を [b] で分岐しており，渋滞予測ではなく現在の状況を知りたいユーザのために，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リクエストを受け付けることもできる．知りたい地点が A 市に属する場合はコンポーネント [c]，B 市に属する場合はコンポーネント [d] が実行される．

本論文では，上記のようなトランザクション処理が多数のユーザからリクエストされ，それらのリクエストが同時並行的に実行される状況を想定し，サービスを**図 2** のようにカラーペトリネットとして表現する．カラーペトリネットは，データ処理を表すトランジションと，データの保存場所を表すプレース，実際のデータを表すカラートークンから構成される．トランジションを実行するには，その入力プレースにカラートークンが存在しなければならず，トランジションが実行されると入力プレースのカラートーク

ンが削除され，それらの値に応じてトランジションの出力プレースに新しいカラートークンが生成される．さらに，カラートークンの値に応じて条件分岐も記述できる．なお，紙面の都合上，カラーペトリネットの詳細説明は省略する．詳細は文献 [1] を参照されたい．

本論文では，サービスに対するリクエスト（およびその引数）をカラートークンで表し，データベース検索などの各コンポーネント（a〜i）をトランジション（T1〜T9）で，トランジションがその処理に用いるデータの保管場所（データのバッファ）をプレース（P1〜P11）で表現する．たとえば，トランジション $T1$（コンポーネント [a]）について，プレース $P1$ はその入力プレースであり，プレース $P2$ はその出力プレースである．プレース $P1$ にリクエストがカラートークン $\langle$“$square\,at\,StationA$”, “$traffic\,forecast$”$\rangle$ として与えられ，トランジション $T1$ が実行されると，そのカラートークンが $P1$ から削除される．また，出力辺（アークとよぶ）($T1$, $P2$) に与えられたラベル $\langle retrieve($“c”, $address), request\rangle$ に基づき，アドレスリポジトリ “C”（プレース P12）から “$square\,at\,StationA$” が示す住所の周囲にあるカメラの位置情報が検索され，その結果がプレース $P2$ にカラートークンとして生成される．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” を表しており，プレースが保持するトークンがリポジトリのデータを表している．たとえば，リポジトリ “C” のデータは $P12$ の保持するカラートークン $\langle C\rangle$ によって表される．プレース $P2$ は位置情報の検索結果，プレース $P5$ から $P8$ 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．これらの一連のトランザクション処理の結果がプレース $P11$ に格納される．

また，逐次で連結された一連のコンポーネント実行におけるデータの流れをフローと見なしたとき，$T5$ や $T6$ のように，入力プレースを複数持つトランジションは，それらのすべてのプレースにカラートークンが揃った場合のみに実行可能であり，複数のデータフローの同期を表す．同様に，たとえば $T2$ のようにトランジションが出力プレースを複数持つ場合，複数のデータフローの並列実行の開始を表す．$P2$ のように出力トランジションを複数持つプレースは（条件）分岐を表し，カラートークンの値に応じてどのトランジションを実行するかを記述できる．たとえば，$request$ が “traffic forecast” の文字列を持つカラートークンをプレース $P2$ が持つ場合，トランジション $T2$ の内部に記述された条件式が真となり，$T2$ が実行される．一方，$request$ が “current movie” の場合で，かつ，指定された住所が A 市である場合は，トランジション $T8$ が，B 市である場合は，トランジション $T9$ が実行可能となる．

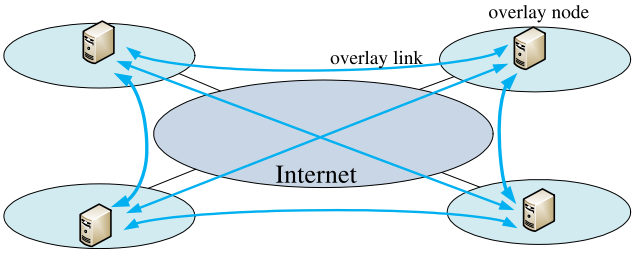

図 3 ネットワークアーキテクチャの概念図
Fig. 3 Network architecture.

本論文におけるサービスでは，カラーペトリネットの構造的制約として，リポジトリへの参照を，リポジトリを表すプレースとそれらの参照処理を表すトランジションとの自己ループとして表すこととする．自己ループとは，トランジションの入力プレースと出力プレースが同じ場合をいう．たとえば，$T1$ は，リポジトリ “C” を表すプレース $P12$ への参照を行うため自己ループの関係になっているが，$T1$ の実行の際には，リポジトリのデータを表すトークン $\langle C\rangle$ が入力として与えられ，$T1$ の実行の後，$\langle C\rangle$ がプレース $P12$ に返却されることでリポジトリ “C” への参照が完了する．なお，4 章で述べる最適化アルゴリズムのため，自己ループを除き，サービスに有向閉路は含まないこととする．また，サービスは，リクエストが到着するバッファを表すプレース p_{in} とリクエストに対するサービスの実行結果を表すプレース p_{out} を，それぞれ 1 つ保持するものとし，p_{in} は，入力トランジションを持たず，p_{out} は，出力トランジションを持たないものとする．図 2 では，$P1$ が p_{in} に，$P11$ が p_{out} にそれぞれ対応する．

3. オーバレイネットワーク

サービスが実行されるネットワークは，**図 3** のように，サーバをオーバレイノードとした，サーバ群が相互にユニキャスト通信可能な論理的なオーバレイネットワークとする．各サーバは，カメラ画像といったセンシングデータを蓄積する機能（リポジトリ）と，コンポーネントを実行する機能を保持する．コンポーネントの実行速度はサーバごとに異なり，同様に，各サーバ間のリンクのデータ転送速度も異なる．

オーバレイネットワークは完全有向グラフ $G = (N, L)$ でモデル化する．ここで，N はサーバ集合，$L = N \times N$ はオーバレイリンク（以下単にリンクとよぶ）の集合を表す．

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センシングデータなどをオーバレイネットワーク上で収集および処理するサービスを提供する際，一般に，センシングデータ（プレース）を保持するサーバ，コンポーネント（トランジション）を処理す

るサーバは異なる可能性があり，その処理に関わるサーバ間でデータ転送を行う必要がある．たとえば，図 2 のトランジション $T3$ がサーバ i に，$T3$ の出力プレース $P5$ がサーバ j にそれぞれ割り当てられた場合には，$T3$ において検索結果として得られた映像データがサーバ i からサーバ j へのオーバレイリンク上を転送されることになる．また，サーバ i，j 間のリンクの帯域が十分でない場合には，そのデータ転送がボトルネックとなる可能性もある．$T3$ をサーバ j に割り当てれば，$T3$ と $P5$ 間のデータ転送は生じないが，$T3$ の他の入出力プレース $P3$，$P6$ および $P13$ を保持するサーバ群とサーバ j 間でデータ転送を行う必要性が生じ，それが新たなボトルネックとなることも想定される．リンク帯域だけでなくサーバの処理能力についても同様の問題が想定される．各サーバの処理能力は様々であり，各トランジションの実行に要する時間も異なってくるため，サーバ j 自体の処理速度が不十分となりボトルネックとなる可能性も考えられる．

このように，サービス全体としてリクエストをなるべく速く処理するためには，プレースやトランジションの関係を考慮し，それらを適切なサーバに割り当てることが望ましい．本論文では，サービスに記述されたプレースおよびトランジションをいずれのサーバに割り当てるかを定めたものを，分散実行方針とよび，多くのリクエストを効率良く処理できるよう，サーバの処理速度やリンクの処理速度を考慮し，サービスが単位時間あたりに処理できるリクエスト数（これをサービスのスループットとよぶ）をなるべく大きくする分散実行方針を導出することを目標とする．

4.1 分散実行方針に基づくサービスの分散実行例

以下，サービスを構成するプレースの集合とトランジションの集合を，それぞれ P および T で表すと，分散実行方針は，サーバ集合 N への写像 $alc : P \cup T \to N$ として表せる．また，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 $\bullet t$，$t\bullet$ でそれぞれ表す．同様に，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 $\bullet p$，$p\bullet$ でそれぞれ表す．以下，ある分散実行方針におけるサービス実行の様子を説明する．

トランジション t は以下の 3 つのステップで実行される．

(I) トランジション t の各入力プレース $p_x \in \bullet t$ について，p_x のサーバ $alc(p_x)$ が，トランジション t のサーバ $alc(t)$ に p_x のカラートークン（データ）を転送する．この転送は各入力プレースごとに独立して実行される．

(II) 各入力プレースのカラートークンがサーバ $alc(t)$ に揃い次第，サーバ $alc(t)$ がトランジション t を実行する．

(III) トランジション t の各出力プレース $p_y \in t\bullet$ について，サーバ $alc(t)$ は，p_y のサーバ $alc(p_y)$ に対し，p_y に生成されるべきカラートークンを転送する．(I) と同様に，この転送は各出力プレースごとにそれぞれ独立して実行される．

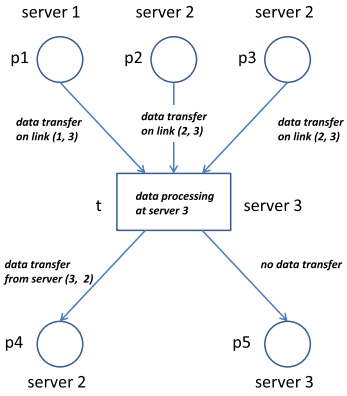

図 4 トランジションの分散実行例
Fig. 4 Example of a transition execution.

サービスへのリクエストが連続的に到着する場合，サービスの各トランジション t における上記の (I)〜(III) は，パイプライン処理のように実行されるため，(I)〜(III) のうち，1 つのリクエストを処理するために必要な遅延時間が最も大きな手順がボトルネックとなり，t の単位時間あたりに処理できるリスエスト数も決定される．そこで，手順 (I)〜(III) のそれぞれの単位時間あたりに処理できるリクエスト数のうち，最小の値をトランジション t のスループット（これを $th(t)$ と表す）とする．

たとえば，**図 4** に示すトランジション t の分散実行例を考える．図 4 では，プレース $p1$ が server1，プレース $p2$，$p3$，$p4$ が server2，プレース $p5$ およびトランジション t が server3 に配置されている．t の実行では，まず，(I) の手順において，(a) server1 から server3 へ，$p1$ が保持するトークンのデータ転送が行われ，(b) server2 から server3 へ，$p2$ と $p3$ の保持するトークンのデータ転送が行われる．すべてのデータ転送が完了すれば，(II) の手順において，(c) server3 で t の実行が行われ，$p4$ と $p5$ のトークンが生成される．最後に，(III) の手順において，(d) server3 から server2 へ $p4$ のトークンのデータ転送が行われる．t と $p5$ は，ともに server3 に配置されており，サーバ間のデータ転送は生じない．

図 4 のトランジション t のスループットは，(a)〜(d) において，それぞれの単位時間あたりに処理できるリクエスト数のうち，最小の値となる．サービス全体のスループット（これを $th(S)$ で表す）は，p_{out} への入力トランジションのスループットの総和（出力される結果のスループット）として定義し，これを最小化することを目的とする．

4.2 問題定義

サービスのスループットを最大とする，サービスの最適化問題を定義する．まず，前述のステップ (I) と (III) のス

ループットは，サーバ間のリンク帯域とサーバ間で転送するトークンのデータ量で制限されるため，その制限を表すために，サーバ i からサーバ j（$i,j \in N$）へのリンク帯域と，アーク (u,v)（$u,v \in P \cup T$）を通過するトークンのデータ量によって決まる最大スループット $TH_{msg}(u,v,i,j)$ を与える．同様に，ステップ (II) のスループットは，サーバの処理能力やコンポーネントの処理に関する必要計算量で制限されるため，その制限を表すために，サーバ i の処理能力とトランジション t の必要計算量によって決まる最大スループット $TH_{exec}(t,i)$ を与える．また，複数のトランジションが 1 つのリポジトリ（プレース）を参照する場合，そのリポジトリを参照する各トランジションのスループットは，リポジトリを保持するサーバ i の処理能力によって制限されるため，その制限を表すために，サーバ i にリポジトリ r（$r \in P$）を割り当てたときの最大スループット $TH_{db}(r,i)$ を与える．なお，リポジトリが存在するサーバとそれを参照するトランジションを保持するサーバが異なる場合，2 つのサーバ間でリポジトリのデータ全体の受け渡しを行う必要があり，一般的には現実的でない．そこで最適化問題では，リポジトリの参照を行うトランジションはリポジトリが存在するサーバで実行されるものと仮定する．本来，複数のトランジションが 1 つのサーバに割り当てられた場合など，これらの最大スループットはサーバへの割り当て方の状況に応じて変化するが，ここでは問題の簡単化のために定数として与える．

提案アルゴリズムでは，このサービス最適化問題を線形計画問題に帰着させる．線形計画問題では，カラーペトリネットによるサービス記述 S，ネットワークグラフ $G=(N,L)$，ならびに最大スループット $TH_{exec}(t,i)$（$\forall t \in T, \forall i \in N$），$TH_{msg}(u,v,i,j)$（$\forall u,v \in P \cup T, \forall i,j \in N$），および，$TH_{db}(r,i)$（$\forall r \in P, \forall i \in N$）に対し，分散実行方針 $alc : P \cup T \to N$ を，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（$v \in P \cup T$）をサーバ i（$i \in N$）へ割り当てるときに 1，そうでないときに 0 である 0-1 変数 $alc(v,i)$ を導入する．また，アーク $(u,v) \in (P \times T) \cup (T \times P)$ とリンク $(i,j) \in L$ について，u がサーバ i，v がサーバ j に配置されている場合 1，そうでないときに 0 である 0-1 変数 $msg(u,v,i,j)$ を導入する．

4.3 サービス最適化問題のための線形計画問題

［目的関数］

本問題では，サービスのスループット $th(S)$ 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} に単位時間あたりに出力されるトークンの数とする．このとき，目的関数は以下で表される．

$$max\ th(S) = \sum_{t \in \bullet p_{out}} th(t) \tag{1}$$

［スループットに関する制約式］

まず，各トランジション $t \in T$ のスループット $th(t)$ は以下を満たす必要がある．

$$th(t) \leq min(X \cup Y \cup Z) \tag{2}$$

$min(A)$ は，集合 A から最小である元を取り出す関数であり，X，Y，Z を下記のように定義する．

$$X = \{TH_{msg}(p,t,i,j) \mid msg(p,t,i,j) = 1 \wedge p \in \bullet t \wedge (i,j) \in L\} \tag{3}$$

$$Y = \{TH_{exec}(t,i) \mid alc(t,i) = 1 \wedge i \in N\} \tag{4}$$

$$Z = \{TH_{msg}(t,p,i,j) \mid msg(t,p,i,j) = 1 \wedge p \in t \bullet \wedge (i,j) \in L\} \tag{5}$$

式 (3)〜(5) は，それぞれ手順 (I)，手順 (II)，手順 (III) における，スループットの最大値を表す制約である．式 (2) において，手順 (I)〜(III) の中で最小となる値がスループット $th(t)$ とすることを表す．さらに，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．

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

$$th(t) - TH_{exec}(t,i) - C \cdot (1 - alc(t,i)) \leq 0 \tag{6}$$

$\forall t \in T, \forall p \in P, \forall (i,j) \in L$ について，

$$th(t) - TH_{msg}(p,t,i,j) - C \cdot (1 - msg(p,t,i,j)) \leq 0 \tag{7}$$

$$th(t) - TH_{msg}(t,p,i,j) - C \cdot (1 - msg(t,p,i,j)) \leq 0 \tag{8}$$

C は十分大きな定数とする．したがって，$alc(t,i) = 0$ あるいは $msg(u,v,i,j) = 0$ の場合，これらの制約式はつねに真となる．式 (6) は，式 (4) に対応し，式 (7) と式 (8) は，それぞれ式 (3) と式 (5) に対応する．式 (6) は，トランジション t がサーバ i に割り当てられる場合のみ，$th(t) - TH_{exec}(t,i) \leq 0$ の制約を満たす必要があることを表す．式 (7) および式 (8) においても同様である．

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見なしたとき，以下のような式が成り立つ．

$\forall p \in P \setminus \{p_{in}, p_{out}\}$ について，

$$\sum_{t \in \bullet p} th(t) - \sum_{t' \in p \bullet} th(t') \geq 0 \tag{9}$$

［リポジトリへの参照をともなうトランジションに関する制約式］

リポジトリを表すプレース集合を $R \subset P$ とし，リポジトリ $r \in R$ への参照をともなうトランジション集合を $T'(r)$ とすると，仮定よりこれらのトランジションは同じサーバ

へ割り当てるため，各リポジトリ r について，以下のような制約が成立する．

$\forall r \in R, \forall i \in N, \forall t \in T'(r)$ について，

$$alc(t,i) - alc(r,i) = 0 \tag{10}$$

さらに，スループットに関し，以下のような制約式が成り立つ必要がある．

$\forall r \in R, \forall i \in N$ について，

$$\sum_{t \in T'(r)} th(t) - TH_{db}(r,i) - C \cdot (1 - alc(r,i)) \leq 0 \tag{11}$$

式 (6) と同様に，C は十分大きな定数とする．

［サーバ間の通信に関する制約式］

変数 $msg(u,v,i,j)$ は，以下の式で表すことができる．

$\forall (u,v) \in (P \times T) \cup (T \times P), (i,j) \in L$ について，

$$msg(u,v,i,j) = alc(u,i) \cdot alc(v,j) \tag{12}$$

式 (12) は，u，v が i，j にそれぞれ割り当てられているときに限り $msg(u,v,i,j)$ が 1 となることを表している．$alc(u,i)$ および $alc(v,i)$ は，すべて 0-1 変数であるため，式 (12) は，等価な下記の線形式で置換することができる．

$\forall (u,v) \in (P \times T) \cup (T \times P), \forall (i,j) \in L$ について，

$$alc(u,i) - msg(u,v,i,j) \geq 0 \tag{13}$$

$$alc(v,j) - msg(u,v,i,j) \geq 0 \tag{14}$$

$$alc(u,i) + alc(v,j) - msg(u,v,i,j) \leq 1 \tag{15}$$

［配置に関する制約式］

各プレースまたはトランジションは，必ずいずれかのサーバ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．

$\forall v \in P \in T$ について，

$$\sum_{i \in N} alc(v,i) = 1 \tag{16}$$

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，alc を求めることによって，最適な分散実行方針を求めることができる．0-1 整数計画問題は NP 困難に属する問題であるが，本論文でのサービスは，たかだか十数個のプレースやトランジションで構成され，ネットワークに関しても数十台程度を想定しており現実的な時間で解ける．また，さらにサービスやネットワークの規模を大きくした場合においても，整数計画問題は多数の近似解法が提案されており，準最適解を比較的高速に求めることができるため，提案手法は実用的に適用可能と考えられる．

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ている．

本プラットフォームに対し，サービス記述とネットワーク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コンポーネントのインストールやネットワークの構築など分散実行環境の構築を行い，効率良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，サービス導出部，サービス実行部の 2 つのモジュールから構成されている．サービス導出部は，4 章で提案したアルゴリズムを実装し，カラーペトリネットで作成したサービスモデルから，実行サーバ群で分散実行するサービスを自動的に導出するためのモジュールである．サービス実行部は，導出部で得られたサービスを基に，サービス実行のための準備および実行を担当するモジュールである．以下，サービス実行部の詳細について述べる．

5.1 サービス実行部

サービス実行部では，サービス導出部で導出したサービスに基づき，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，受付サーバ，制御サーバ，実行サーバから構成されている（**図 5**）．各サーバの役割は次のとおりである．受付サーバとは，サービスを利用するユーザからのリクエスト受付や，ユーザへリクエストの処理結果を通知する役割を持つ．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 5 では，サーバの分類を 3 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．次項では，各サーバの設計について詳細を述べる．

5.1.1 受付サーバ

リクエスト受付では，ユーザからのリクエストとともに，サービス実行に必要な情報を受け取る．受付サーバは，ユーザからのリクエストを受け付けた後，制御サーバに通知するとともに，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，ユーザにリクエストの処理結果の通知を行う．

受付サーバは，主にユーザインタフェースとシステムイ

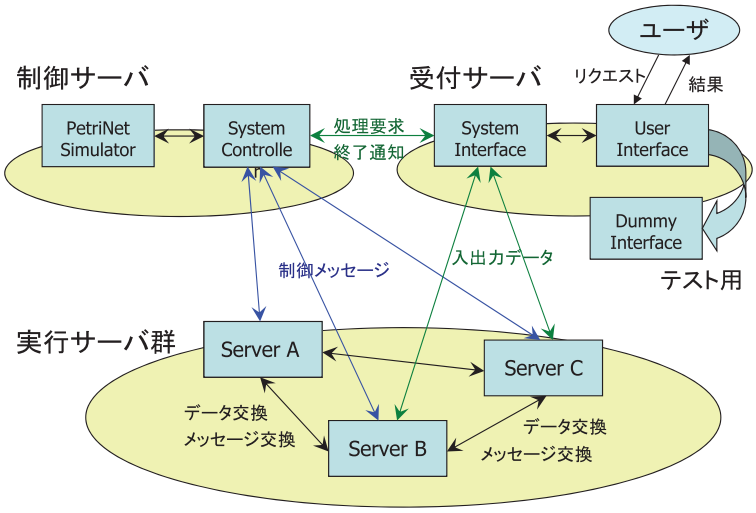

図 5 サービス実行部全体図
Fig. 5 Overview of service execution system.

ンタフェースから構成されている．ユーザインタフェースはユーザに対するリクエストの受付や結果の通知を行い，システムインタフェースは制御サーバや実行サーバに対するメッセージやデータのやりとりを行う．また，テスト用にダミーインタフェースを実装しており，ユーザインタフェースの 1 つとして置き換えが可能である．ダミーインタフェースは，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リクエストを生成するモジュールで，たとえば，リクエストの到着時間間隔が指数分布に従ったユーザリクエストを発生させることができる．

5.1.2 制御サーバ

制御サーバでは，実行サーバごと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カラーペトリネットシミュレータを保持している．このシミュレータによるサービスの動作結果に基づいて，システムコントローラが制御メッセージを発行し，各実行サーバへ通知することでサービス全体の制御を行う．

[カラーペトリネットシミュレータ]

制御サーバは，それぞれの実行サーバごとの挙動をシミュレートするためのサブシミュレータとその実行サーバが実行すべきサービスを記述したカラーペトリネットモデルを，それぞれ実行サーバの数だけ保持する．トークンはユーザのリクエストおよびデータの流れを表している．あるサブシミュレータから，対応する別のサブシミュレータへトークンが移動することが，実行サーバ間のデータ転送を意味する．制御サーバで実行したシミュレータの結果に基づいて，実際にそれぞれのサブシミュレータが対応する実行サーバ間でデータを移動させ，その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，トランジションの実行では，サブシミュレータが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実行サーバに送信し，実行サーバではこの制御メッセージによってコンポーネントを実行する．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，トランジションの実行を完了する．

[システムコントローラ]

システムコントローラは，サービス全体の制御を行うとともに，実行サーバとの接続を確保し入出力インタ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．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，担当する実行サーバのペトリネットモデルを入力としてそれぞれのサブ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，実行サーバに対してプログラムなど必要なデータを転送する．すべての実行サーバで実行サーバプログラムが起動された後，オーバレイネットワークの構築を行う．このオーバレイネットワークは，カラーペトリネットシミュレータにおいて，トークンが移動するサブシミュレータ間に対応する実行サーバ間の TCP コネクションにより構成される．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．次にすべ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

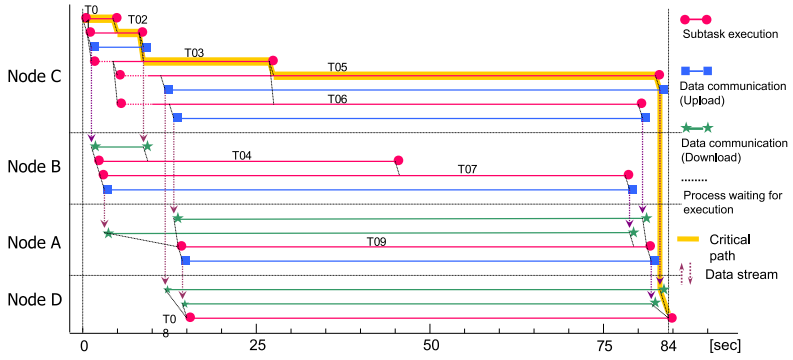

図 6 タイムチャート
Fig. 6 Timing chart.

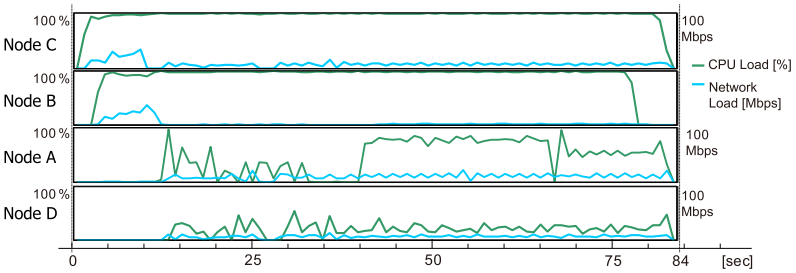

図 7 サーバやネットワークの負荷
Fig. 7 CPU and network load.

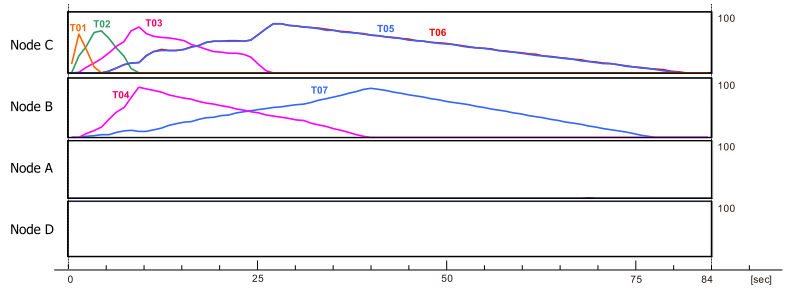

図 8 処理待ちトークン数
Fig. 8 # of tokens to be processed.

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する．

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各サブシミュレータと実際の実行サーバとをマッピングする．また，カラーペトリネットシミュレータの実行結果に基づいて，制御メッセージを発行し，実行ノードに送信する役割を持つ．

5.1.3 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，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を行う．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すべて行うため，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，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．

5.2 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，サーバやネットワークの負荷および処理待ちトークン数をモニタする機能を持つ．**図 6** に，モニタしたデータをタイムチャート形式で表示した例を示す．タイムチャートでは，各トランジションの実行のタイミングやサービス全体のクリティカルパスなどが分かる．トランジション間の縦向きの破線は依存関係を表す．この例では，複数のリクエストが同時並行的に実行されており，各サーバの処理やデータ転送が同時に実行されている様子を示している．**図 7** 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．たとえば，13 秒あたりで NodeB や NodeC のネットワーク負荷が減少した直後に，NodeA や NodeD の負荷が大きく増加しており，各サーバで実行している処理やデータ転送がその時間に変化したことが分かる．**図 8** 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ことは，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，提案アルゴリズムにより導出した分散実行方針の評価，および提案プラットフォームの機能を利用した性能改善を行った．

この実験では，世界中に展開された計算機間で構築されたネットワークテストベット PlanetLab [2] を利用した．PlanetLab 上の異なるドメインに属する 30 台のサーバを用い，本プラットフォームを利用し，TCP 接続によるフルメッシュのオーバレイネットワークを構築した．PlanetLab では様々なアプリケーションが実行され，各サーバの負荷は基本的に高い状態であるため，なるべく負荷の低いサーバを用いるように選択した．ただし，図 2 のプレース $P13$ と $P14$ 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た．このオーバレイネットワーク上で，提案アルゴリズムにより導出した分散実行方針だけでなく，比較対象であるランダムな分散実行方針に基づき，図 2 の交通解析サービスを実行した．提案手法により分散実行方針を導出する際，提案プラットフォームの機能を利用し，各サーバの CPU 処理能力とサーバ間のリンクの利用可能帯域を計測し，この値をもとに各サーバが最低限処理できるであろうと推測したスループットを，提案アルゴリズムの入力である最大スループットとして与えた．

提案手法によって設計された分散実行方針が，様々な分散実行方針の中でどれくらいの性能を持っているのかを評価するために，比較対象としてランダムに分散実行方針を抽出したものを可能な限り網羅的に比較した．今回の実験では，このランダムな分散実行方針を 100 パターン用意し，これらの分散実行方針に対し 300 個のリクエストを投入した．この 300 個のリクエストは，トランジション $T2$ の渋滞予測処理が 60 個，トランジション $T8$ の映像検索処理が 120 個，トランジション $T9$ の映像検索処理が 120 個から構成されており，これらのリクエストがそれぞれの分散実行方針に対しランダムに入力される．このもとで分散実行方針ごとのサービス全体のスループットを計測した．

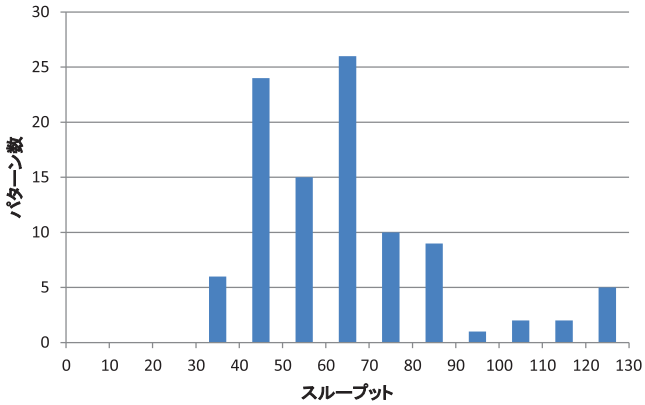

図 9 各分散実行方針におけるスループットの度数分布
Fig. 9 Frequency distribution of throughput.

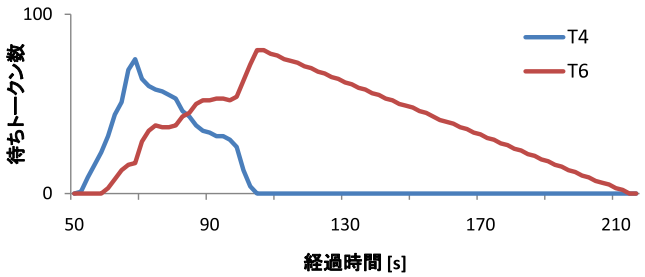

図 10 $T4$ と $T6$ における処理待ちトークン数の経緯
Fig. 10 The number of waiting tokens at $T4$ and $T6$.

6.1 サービス全体のスループット

ランダムな分散実行方針において，100 秒間に処理が完了したリクエスト数（スループット）の平均を度数分布表としてまとめた結果を**図 9** に示す．この度数分布表では，それぞれ x 軸に示された各値 x に対し，$(x-10, x]$ のスループットであった分散実行方針の数を y 軸に示している．また，最大スループットは 116.2，最小スループット 23.6 であった．この図から分かるように，分散実行方針の違いがスループットに大きく影響している．一方，提案手法による分散実行方針のスループットは 114 で，ランダムな分散実行方針の 100 パターン中 3 番目に高い値となっていた．このように，提案アルゴリズムにより性能の高い分散実行方針が導出できていることが分かる．一方，1 番目ではなく 3 番目となっている原因としては，ネットワークの負荷状況が最適化の処理の間に変化したため，サービスが実行される際にはすでに最適解でなくなっている可能性や，提案手法ではネットワークをフルメッシュオーバレイとしてモデル化しているが，実際のネットワークでは一部のリンクが競合し帯域を取り合う関係にあるなど，モデルが十分に実際のネットワークを表現できていない可能性が考えられる．

6.2 サービス全体のボトルネック

次に，提案プラットフォームの機能を利用し，サービス全体のスループットを決定するボトルネックを発見する

過程を示す．そのような個所では，処理待ちのトークンが発生していると考えられるため，処理待ちのトークン数に着目する．ある分散実行方針において，トランジション $T4$ と $T6$ に対する処理待ちトークン数の推移を**図 10** に示す．いずれのトランジションにおいても処理待ちトークンが発生しており，割り当てられたサーバの処理能力を超えるトークンが流入していることが分かる．一方，流入するトークンがない場合には，トークンの減少量の傾きが大きいほど，トークンを処理する能力は大きいといえる．つまり，待ちトークン数が頭打ちとなり減少していく際の傾きを見ると，$T4$ は $T6$ よりも待ちトークンが早く処理されていることが分かる．このことから，ボトルネックは $T4$ ではなく $T6$ であり，$T6$ の処理に関わるリソースを追加する，あるいは適切なサーバの割当てを行うことにより，サービス全体のスループットを向上させることができる可能性がある．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を容易に発見できる．

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 [3] や無閉路有向グラフ（DAG）を用いるもの [4], [5] 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したりするといった記述が可能となる．

文献 [6], [7], [8], [9], [10] などでは計算機支援による分散システムの設計手法が提案されている．iOverlay [6] 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良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 [7] 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．文献 [8], [9] では，分散サービスを設計する際の設計コストの低減に焦点を当てている．MACEDON [10] 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．文献 [11] では，オーバレイネットワーク上での分散実行環境に対し，Guarded Horn Clauses を拡張したプログラミング言語を提案している．

一方，オーバレイネットワーク上でのサービス性能を評価する方法の 1 つとして，ネットワークシミュレーションを利用する方法が提案されている．たとえば，オーバレイネットワークに特化したネットワークシミュレータとして，OverSim [12] が提案されており，ネットワークシミュレーションによるサービスの開発支援も考慮したプラットフォームもいくつか提案されている [13], [14]．特に，文献 [13] で提案されているプラットフォームでは，従来の API レベルでのサービス開発だけでなく，より高いレベルでサービス記述が可能な behavior レベルでのサービス開発が可能である．

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，自動的に各サーバで実行可能かつ最適なサービスを導出できる点が本質的に異なる．

8. おわりに

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，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．複数のコンポーネントの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ればスループットが最適化できるか自動で決定できる．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および実装を行った．サービス実行プラットフォームでは，サーバへのコンポーネント配置や分散実行を制御でき，ネットワークやサーバの負荷状況をリアルタイムに収集できる．PlanetLab を用いた評価実験を行い，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．

参考文献

[1] Jensen, K.: *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] Planetlab: An open plat form for developong, deploying, and accessing planetary-scale services, available from ⟨http://www.planet-lab.org⟩.

[3] Xu, D. and Nahrstedtm, K.: Finding service paths in an overlay media service proxy network, *Proc. Int. Conf. Multimedia Computing and Networking* (*MMCN2002*) (2002).

[4] Wang, M., Li, B. and Li, Z.: sFlow: Towards resource-efficient and agile service federation in service overlay networks, *Proc. 24th Int. Conf. on Distributed Computing Systems* (*ICDCS2004*) (2004).

[5] Gu, X., Nahrstedt, K. and Yu, B.: Spidernet: An integrated peer-to-peer service composition framework, *Proc. IEEE Int. Symposium on High-Performance Distributed Computing* (*HPDC-13*) (2004).

[6] Li, B., Guo, J. and Wang, M.: iOverlay: A lightweight

middleware infrastructure for overlay application implementations, *Proc. 5th ACM/IFIP/USENIX International Middleware Conference* (*Middleware 2004*), also Lecture Notes in Computer Science, pp.135–154 (2004).
[7] Benza, D., Cosnard, M., Liquori, L. and Vesin, M.: Arigatoni: A simple programmable overlay network, *JVA '06: Proc. IEEE John Vincent Atanasoff 2006 International Symposium on Modern Computing*, pp.82–91, IEEE Computer Society (2006).
[8] Loo, B.T., Condie, T., Hellerstein, J.M., Maniatis, P., Roscoe, T. and Stoica, I.: Implementing declarative overlays, *SIGOPS Oper. Syst. Rev.*, Vol.39, No.5, pp.75–90 (2005).
[9] Shudo, K., Tanaka, Y. and Sekiguchi, S.: Overlay weaver: An overlay construction toolkit, *Computer Communications*, Vol.31, No.2, pp.402–412 (2008).
[10] Rodriguez, A., Killian, C., Bhat, S., Kostic, D. and Vahdat, A.: MACEDON: Methodology for automatically creating, evaluating, and designing overlay networks, *Proc. USENIX/ACM Symposium on Networked Systems Design and Implementation* (*NSDI2004*), pp.267–280 (2004).
[11] Saito, K. and Miyazawa, K.: Rapid P2P overlay network programming on a distributed reduction machine, *Proc. 6th IEEE Conference on Consumer Communications and Networking Conference*, *CCNC'09*, pp.1262–1266, IEEE Press, Piscataway, NJ, USA (2009).
[12] Baumgart, I., Heep, B. and Krause, S.: Oversim: A scalable and flexible overlay framework for simulation and real network applications, *IEEE Ninth International Conference on Peer-to-Peer Computing, P2P '09*, pp.87–88 (Sep. 2009).
[13] Pujol-Ahulló, J., García-López, P., Sànchez-Artigas, M. and Arrufat-Arias, M.: An extensible simulation tool for overlay networks and services, *Proc. 2009 ACM Symposium on Applied Computing*, *SAC '09*, pp.2072–2076, ACM, New York, NY, USA (2009).
[14] Bischofs, L. and Hasselbring, W.: Analyzing and implementing peer-to-peer systems with the peerse experiment environment, *Euromicro Conference on Parallel, Distributed, and Network-Based Processing*, 0:311–315 (2009).

推薦文

提案手法の解の精度や分散実行方針の比較などについては今後の検討を期待するが，サービスオーバレイを形式的に記述し，最適な割当ノードを選択する方法を提案し，PlanetLab 上で評価しており，完成度の高い論文である．よって，本論文は推薦に値する．

（マルチメディア通信と分散処理研究会主査　勝本道哲）

境 裕樹　（学生会員）

平成 21 年大阪大学大学院情報科学研究科情報ネットワーク学専攻博士前期課程修了．同年同大学院博士後期課程進学．分散システムに関する研究に従事．

廣森 聡仁　（正会員）

平成 16 年大阪大学大学院基礎工学研究科博士後期課程修了．平成 17 年株式会社エヌ・ティ・ティ・ドコモ入社．平成 20 年より大阪大学大学院情報科学研究科助教．博士（工学）．モバイルアプリケーションやモバイルネットワークの設計および性能評価に関する研究に従事．IEEE 会員．

山口 弘純　（正会員）

平成 6 年大阪大学基礎工学部情報工学科卒業．平成 10 年同大学大学院基礎工学研究科博士後期課程修了．同年オタワ大学客員研究員．平成 11 年大阪大学大学院基礎工学研究科助手．平成 14 年同大学院情報科学研究科助手．平成 19 年より同大学院情報科学研究科准教授．博士（工学）．分散システムや通信プロトコルの設計および実装に関する研究に従事．IEEE，電子情報通信学会各会員．

東野 輝夫　（フェロー）

昭和 54 年大阪大学基礎工学部情報工学科卒業．昭和 59 年同大学大学院基礎工学研究科博士後期課程修了．同年同大学助手．現在，同大学大学院情報科学研究科教授．平成 19 年 10 月より，独立行政法人科学技術振興機構，CREST，研究代表者．博士（工学）．分散システム，通信プロトコル，モバイルコンピューティング等の研究に従事．電子情報通信学会，ACM 各会員．IEEE Senior Member．