博 士 論 文

データ量に依存した並列分散処理の効率化と性能評価

平成23年3月

九州工業大学大学院 情報工学研究科

吉永一美

目 次

第 1 章　はじめに　1

第 2 章　データ通信時間に着目した実行の効率化
-ストリーミング型分散アプリケーションのタスクスケジューリング-　3

2.1 序論 3
2.1.1 想定する分散コンピューティング環境 3
2.1.2 設計指針 4
2.2 提案するタスクスケジューリング手法 5
2.2.1 ストリーミングタスクの割り付け 5
2.2.1.1 ワーカの選択 5
2.2.1.2 タスクマイグレーション 6
2.2.2 非ストリーミングタスクの割り付け 6
2.3 性能評価 7
2.3.1 実験環境 7
2.3.2 評価実験用アプリケーション 8
2.3.2.1 ストリーミング型アプリケーション 8
2.3.2.2 非ストリーミング型アプリケーション 9
2.3.3 タスクスケジューリングプラットフォーム 9
2.3.3.1 タスクスケジューリングシステム 10
2.3.3.2 資源情報サーバ (RIS) 12
2.3.4 実験 1: 経路選択の評価実験 12
2.3.5 実験 2: マイグレーションの評価実験 14
2.4 関連研究と本研究の位置付け 14
2.5 まとめ 15

第 3 章　通信データ量の変化に対応したマルチサーバシステムの性能評価手法　16

3.1 序論 16
3.2 Mean Value Analysis を用いたマルチサーバシステムの性能評価手法 . 17
3.2.1 MVA のサーバ性能評価への適用 17
3.2.2 性能評価に用いる MVA モデルの定義 18
3.2.3 多粒度 MVA モデル 19

3.2.4 MVA モデルの正確性 20
3.2.4.1 単一待ち行列モデル 20
3.2.4.2 複数待ち行列によるネットワークモデル 24
3.2.5 実システムにおける MVA モデルの構築 28
3.3 データ量によるサービス時間の変化を考慮した拡張 MVA モデル . . . 29
3.3.1 データ量の変化がサービス時間に与える影響 29
3.3.2 既存 MVA の問題点 30
3.3.3 拡張 MVA モデルの定義 30
3.3.4 実システムにおける拡張 MVA モデルの構築 31
3.4 拡張 MVA モデルを用いた性能評価実験 32
3.4.1 クライアント–サーバ連鎖型システム 32
3.4.1.1 zero-load トランザクションでのサービス時間測定 . . 33
3.4.1.2 非 zero-load 時の応答時間測定 36
3.4.2 株価取得システム 37
3.4.2.1 zero-load トランザクションでのサービス時間測定 . . 39
3.4.2.2 非 zero-load 時の応答時間測定 40
3.5 関連研究と本研究の位置付け 41
3.6 まとめ 42

第 4 章 結論 **43**

図 目 次

2.1 分散アプリケーションのタスクグラフの例 (早稲田大学の許可を得て [12] から転載). 4
2.2 実験に用いた計算機とネットワークの構成. 7
2.3 評価用ストリーミング型アプリケーション. 8
2.4 ソフトウェアプラットフォーム. 10
2.5 タスクスケジューリング GUI. 11

3.1 開放型待ち行列ネットワークの例. 18
3.2 単一待ち行列と複数待ち行列での図 3.1 のシステムの MVA モデルによる応答時間予測比較 (パラメータ: s_1=0.001, s_2=0.002, s_3=0.004, $p_{1,2}$=0.4, $p_{1,3}$=0.2). 19
3.3 MVA ネットワークを構成するサブネットワーク: (a) 直列, (b) 並列. . 25
3.4 クライアント–サーバ連鎖型システム. 32
3.5 トランザクションメッセージフロー : クライアント–サーバ連鎖型システム. 33
3.6 サービス時間 : クライアント–サーバ連鎖型システム. 34
3.7 サービス時間 (s_1, s_2) : クライアント–サーバ連鎖型システム. 34
3.8 サービス時間 (s_3, s_4) : クライアント–サーバ連鎖型システム. 35
3.9 サービス時間 (s_5) : クライアント–サーバ連鎖型システム. 35
3.10 システム応答時間の予測値と実測値 : クライアント–サーバ連鎖型システム. 36
3.11 株価取得システム. 37
3.12 株価取得システムの動作ハードウェア構成. 38
3.13 トランザクションメッセージフロー : 株価取得システム. 39
3.14 サービス時間 : 株価取得システム. 40
3.15 システム応答時間の予測値と実測値 : 株価取得システム. 41

表 目 次

2.1 実験に用いた計算機の動作環境 (全ホスト共通). 7
2.2 各プログラムの実行時間. 13
2.3 各プログラムの実行時間 (同時実行). 13
2.4 タスクマイグレーションの効果. 14

3.1 クライアント–サーバ連鎖型システムの動作環境 (全ホスト共通). . . . 32
3.2 株価取得システムの動作環境. 37

第1章　はじめに

近年, 計算機の低価格化や高性能化, ネットワークの高速化が進み, グリッドコンピューティング [3] やクラウドコンピューティングなどの並列分散コンピューティング環境の普及が進んでいる. そのような環境上で動作する並列分散アプリケーションの実行の効率化は重要な課題であり, 盛んに研究が行われている [5] [6]. また, 高速なネットワークが低価格化し急速に普及したことで, 大量のデータ通信を含む処理が増加している [4]. 例えば, ストリーミングによる動画配信などは通信量が非常に多いサービスの典型であり, 動画データベースやフォーマット変換サーバなどでの一連の処理には大量のデータ通信を伴う. また, クラウドコンピューティングは, クラウド内の計算資源を用いて処理を行うサービスであるが, 処理に必要なデータの通信や結果のやり取りなどのネットワーク通信が発生し, データ量が増大する. このような背景から, 通信を多く含み, 処理がデータ量に依存する並列分散アプリケーションについて, そのデータ通信時間とデータ処理時間に着目して実行を効率化することを目的として研究を行った.

まずデータ通信時間に着目し, それを短縮することによって処理がデータ量に依存する並列アプリケーションの実行を効率化する方法として, ストリーミングデータ処理を含む並列分散アプリケーションを効率的に実行可能なタスクスケジューリング手法を考案した [2]. ここでは, 通信量が多く実行時間がデータ量に依存するアプリケーションの代表例として, マルチメディアストリーミングの処理や, 実験装置から連続的に生成されるデータの解析など, 近年要求が高まっているストリーミング型分散アプリケーションを取り上げる. 分散アプリケーションの効率的実行には, 効率的な資源割り当てを行うタスクスケジューリング手法が必要であるが, 既存のタスクスケジューリング手法はストリーミング処理に適応できない. そこで, ネットワーク特性と変動を考慮し, 通信形態に適した経路を利用することでデータ通信に要する時間を削減し, ストリーミング型分散アプリケーションの実行を効率化する, 新たなタスクスケジューリング手法の提案と評価を行った. 提案手法では, 複数の経路を選択利用可能な状況下において, 通信形態に適した経路を利用することで, アプリケーションの実行時間の短縮を実現した. また, タスクを別のワーカへ移動して利用経路を切り替えるタスクマイグレーションを実装したことにより, 動的な負荷変動により利用経路の性能が低下した場合にも安定した高性能を維持することを可能とした. 安定性能を維持することで, 資源利用時間の予測が高精度で可能となるため, 別アプリケーションの実行の際にも効率的な資源割り当てを行い, 効率的な実行

を行うことができるようになる.

一方, 実際のシステムにおいて, Web サーバやデータベースサーバなどを利用する場合は, 処理を行うべきワーカが決定しており, 分散アプリケーションの実行を効率化するためには, その分散アプリケーションに適したシステムを構築し, システムの性能を向上させなければならない. その際, 適当な粒度で各部分の性能を正しく把握し, ボトルネックとなる部分の性能向上をしなければ, 並列アプリケーションの実行において効果的な性能改善を得ることはできないため, 性能評価手法が必要となる. しかし, 現状ではシステムのボトルネックを特定するためには, エンジニアがログデータを経験に基づいて解析する必要があり, 容易に行うことができない. また, アプリケーションに対して応答時間の保証や許容できるリクエストの量 (キャパシティ) など, 一定の性能を満たす実行を求められる場合も数多く存在するが, 実際のシステム上で様々な状況下での性能を測定することは困難である. そこでこれらの問題点を解決するために, システムの性能を容易に把握し, 評価を行うことのできる手法として, Mean Value Analysis(MVA) という待ち行列解析手法をマルチサーバシステムの性能評価に応用した手法 [25] を拡張した, 新たなシステム性能評価手法を提案する [1]. MVA を用いた性能評価手法の特徴として, システム解析のために要するコストが少ないこと, トランザクションの到着率に対するシステム全体の応答時間を予測でき, 同時にキャパシティの予測もできること, アプリケーション動作時のシステムにおけるボトルネック部分が把握できることが挙げられるが, 既存の手法では通信するデータ量の変化によりサービス時間が変動するシステムを正しく評価することができない. 一般的なシステムにおいてはサーバ間でのファイル転送や処理に必要なデータの転送, データベースからのデータの取得など, アプリケーションの動作には様々なデータのやり取りが必要であり, そのデータ量に依存して通信時間だけでなく, 処理に要する時間も変化する. そのため, 正しく性能を評価するためにはデータ量に依存して変化するサービス時間の考慮が必須である. 本研究では, サービス時間の変化をデータ量の関数として定義することで, 既存の MVA の持つモデル構築の容易さを損なうことなく, 通信データ量の変化に対応した性能評価を行えるようになった.

本論文では, 第 2 章において, ストリーミングデータ処理を含む並列分散アプリケーションの効率的なタスクスケジューリング手法について述べる. ここではまず一般的な並列分散アプリケーションについて述べ, ストリーミングデータ処理との差異を示した後に, 提案手法について述べる. 続いて第 3 章において, 通信データ量の変化に対応したシステム性能評価手法について述べる. まず既存の MVA を用いたシステム性能評価手法を紹介し, 本研究で提案する通信データ量の変化への対応について論ずる. そして第 4 章において本論文のまとめを行う.

第2章　データ通信時間に着目した実行の効率化 -ストリーミング型分散アプリケーションのタスクスケジューリング-

2.1　序論

分散アプリケーションは, 処理の単位であるタスクの集合で構成される. タスク間には依存関係が存在しており, タスクとその依存関係を示した非循環有向グラフであるタスクグラフを用いて, 分散アプリケーションを表すことができる. タスクグラフの例を図 2.1 に示す. 図中において○印がタスクを, 矢印がタスク間の依存関係を示している. タスクをどのような順序で, どの計算機に割り付けて実行するかを決定することをタスクスケジューリングと呼ぶ. このタスクスケジューリングの手法により, 分散アプリケーションの実行時間が大きな影響を受けるため, 効率的な実行には効率的なタスクスケジューリング手法が不可欠である.

ストリーミング処理を含まない通常のタスクでは, データの入力, 処理, 出力は一度だけしか行わない. 一方, ストリーミング処理を含むタスクは, これらを繰り返し行う. この違いから, 従来から研究されているワークフロー型 [7][8] やタスクファーミング型の分散アプリケーション向けのタスクスケジューリング手法は, ストリーミング型分散アプリケーションにそのままでは適用できず, 現状では分散コンピューティング環境上で効率の良い実行ができない. そこで, ストリーミング型分散アプリケーションを効率的に実行可能とする新たなタスクスケジューリング手法を開発する.

2.1.1　想定する分散コンピューティング環境

本研究で想定する分散コンピューティング環境は, 複数の特性の異なるネットワーク経路が存在し, それらを使い分けることが可能な広域分散コンピューティング環

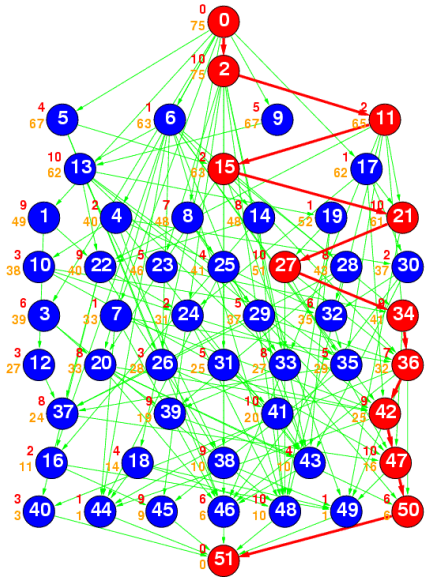

図 2.1 分散アプリケーションのタスクグラフの例 (早稲田大学の許可を得て [12] から転載).

境である. 通常ネットワークには複数の経路が存在するが, 現在一般的に利用されているネットワーク経路制御プロトコルでは経路選択の指標として静的なメトリックが使用されており, 複数の経路を選択利用することは困難である. しかしながら複数経路表を用いた方法 [13] など, 複数の経路を動的に切り替える経路制御の研究がなされており, 本研究はそのような環境を前提としている.

2.1.2 設計指針

提案するスケジューリング手法は, ストリーミング型分散アプリケーションだけでなく, 一般的な非ストリーミング処理が混在した場合にも適切な資源割り当てを行い, 両種共に効率的な実行が可能なように設計する. 2.1.1 で述べたような広域分散コンピューティング環境では, 様々な性能・特性を持つ計算機やネットワークが混在している. また, 一般的に複数のユーザによって共有されることが多いため, 動的に負荷が変動すると考えられる. 特に広域に分散している環境ではネットワークにおける負荷変動が大きい. そこで, ネットワーク特性と性能の変動に着目し, タスク間の通信に適したネットワーク経路を利用することで通信時間を削減し, 実行時間を短縮するタスクスケジューリング手法を設計する.

2.2 提案するタスクスケジューリング手法

2.2.1 ストリーミングタスクの割り付け

ストリーミング型分散アプリケーションはストリーミングデータ処理を行うタスクを含んでおり, こうしたタスクを含まない通常のアプリケーションと比較すると, 一般にタスク間で長時間に渡り多量のデータ通信を行うという特徴がある. この特徴を考慮し, 提案するスケジューラでは, ストリーミングデータの通信にできる限りバンド幅の大きなネットワークを利用するような割り付けを行い, データ通信を効率化し, 分散プログラムの実行時間の短縮を図る.

分散アプリケーションをタスクグラフとして表すと, ストリーミングデータ処理を含むタスクとしては,

1. ストリーミングデータの入力がある.
2. ストリーミングデータの出力がある.
3. ストリーミングデータの入出力ともに持ち合わせている.

の三種類がある. 提案するタスクスケジューリング手法では, ストリーミングデータの入力があるタスク, つまり 1 と 3 のタスクをストリーミングタスクとして定義し, 2.2.1.1 で述べる割り付けを行う.

2.2.1.1 ワーカの選択

スケジューラは実行可能なタスク t_x を割り付けるワーカを, 利用可能なワーカ群 $(w_1, \cdots, w_m)$ の中から選択する. ここで, 全ての利用可能なワーカについて評価式

$$\min\{bw\_pred(w_{s_1}, w_x), \cdots, bw\_pred(w_{s_n}, w_x)\} \tag{2.1}$$

の値を求め, この値が最大となる w_x を選択する. $bw\_pred(w_m, w_n)$ は w_m と w_n 間のバンド幅の予測値を示し, $w_{s_1}, \cdots, w_{s_n}$ は, タスク t_x にストリーミングデータを送信するタスク, つまり t_x の先行タスクである $t_{s_1}, \cdots, t_{s_n}$ が割り付けられているワーカである.

この式 2.1 は, タスク t_x をワーカ w_x に割り付けた場合に, 先行タスク群 $t_{s_1}, \cdots, t_{s_n}$ からのストリーミングデータ通信に利用する経路のバンド幅の予測値の中で, 最小となる値を返すものであり, 言い換えれば, 先行タスクからのストリーミングデータ通信に利用する経路の中でボトルネックとなる経路のバンド幅の予測値を返すと言える. その値が最大となるワーカ w_x を選ぶことで, 可能な限りストリーミングデータ通信でのボトルネックを回避する.

2.2.1.2 タスクマイグレーション

2.2.1 で述べたように, ストリーミングタスクは長期間に渡りデータを流し続ける. しかし, 2.1.2 で述べたように, 広域分散コンピューティング環境は多数のユーザにより共有されるという特徴があり, 負荷が動的に変動するために 2.2.1.1 で決めたワーカが長時間に渡り良いパフォーマンスを維持するとは考え難い. そのため, 性能が低下した場合の対策を講じる必要がある.

そこで, 通信に利用しているネットワークの性能が低下してきた場合, スケジューラはタスクを他のワーカに移動 (マイグレーション) し, 利用する経路を変更することで, ネットワークの混雑を回避するようにした. タスクマイグレーションは, 以下の手法で実現している.

1. 各ストリーミングタスクの入力ストリームを監視し, 流量を取得する.
2. そのストリームが利用しているネットワーク経路のバンド幅の測定値を, 資源情報サーバ (RIS, 2.3.3.2) を通じて取得する.
3. 1 で得た流量と 2 で得たバンド幅の測定値の和が規定した閾値を下回る場合, ネットワークが混雑していると判断する.
4. ネットワークの混雑を検知すると, スケジューラはタスクの実行を中断し, そのタスクを他のワーカへ移動させる. この時, 移動先のワーカは 2.2.1.1 で述べた手法を用いて決定する.
5. タスクの移動を終えた後, ストリームの再接続などの必要な処理を行い, 中断した時点からの実行を再開する.

2.2.2 非ストリーミングタスクの割り付け

非ストリーミングタスクは, 一般的にストリーミングタスクと比較して通信するデータ量が少ない. データ通信に要する時間 t は, バンド幅を bw, 通信データサイズを d, 遅延を t_l とすると, 以下の式

$$t = \frac{d}{bw} + t_l \tag{2.2}$$

で表すことができ, 通信するデータのサイズが小さいほどネットワーク遅延の影響を大きく受けることがわかる. そこで, 非ストリーミングタスクを割り付ける際には遅延の小さなネットワークを利用するように配置することで, 実行の高速化を図る. なお, タスクは実行可能になったものから順次割り当てを開始される.

2.3 性能評価

2.3.1 実験環境

実験に用いた計算機とネットワークの構成を図 2.2 に示す．8 台の同一性能の計算機が接続されており, 1 台がスケジューラ, 残りの 7 台がワーカの役割を果たす．計算機の動作環境は表 2.1 の通りである．

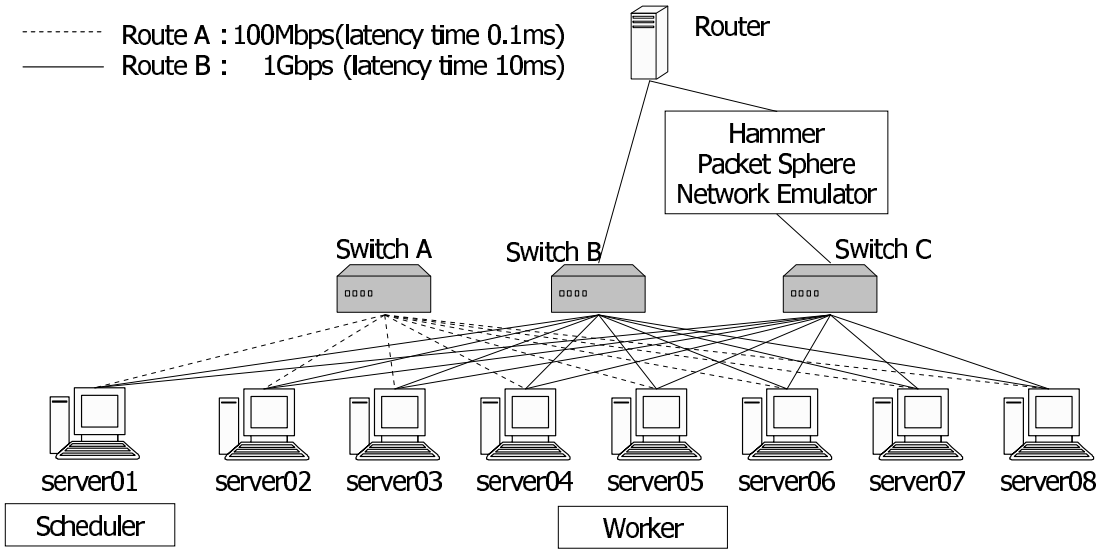

図 2.2 実験に用いた計算機とネットワークの構成.

表 2.1 実験に用いた計算機の動作環境 (全ホスト共通).

CPU	AMD Athlon^{TM}64 X2 3800+
メモリ	1GB (512B ×2) DDR 400
NIC	nVidia Corporation MCP51(100Mbps)
	Intel Corporation 82541PI Gigabit Ethernet Controller(1Gbps) × 2
ソフトウェア	Fedora Core 6 (x86_64) Java JDK 1.5.0_11 iperf 2.0.2

各計算機間の通信に利用できるネットワーク経路は二種類存在する．経路 A は 100Mbps(100Base-TX) のネットワークで, スイッチングハブ A を介して直接通信を行う経路であるため, 遅延が非常に小さい (0.1ms 程度)．経路 B は 1Gbps(1000Base-T) のネットワークであり, スイッチングハブ B, ルータ, ネットワークエミュレータ, スイッチングハブ C を経由して接続される．ネットワークエミュレータでは, 通過するデータに常に 10ms の遅延を加えており, 経路 B は経路 A と比較して, バンド幅は大きいものの遅延も大きいという経路になっている．

本研究が対象にしている環境は，多数の利用者により共有され，負荷が動的に変動する分散コンピューティング環境である．しかしながら, 実験に用いた環境は共有されておらず, 外部からの動的な負荷変動は存在しない．そこで実験では, iperf[16]を用いたスクリプトによりランダムなトラフィックを発生させ, 他の利用者によりネットワークへ負荷が加えられる状態を模擬した．スクリプト内の処理は以下の通りである．

1. 帯域制限するノードを，server03〜server07 からランダムに 2 つ選択する．
2. 帯域制限する時間 t_{load} を，60〜120 秒の間でランダムに設定する．
3. 1 で決定した 2 ホスト間に, iperf を用いて t_{load} 秒間の通信を発生させる．この通信が利用する経路は経路 B である．
4. t_{load} 秒経過後, 1 に戻る．

2.3.2 評価実験用アプリケーション

2.3.2.1 ストリーミング型アプリケーション

ストリーミング型分散アプリケーションとして, 研究室内で開発している分散型ネットワーク監視システムの一部を用いた．このシステムでは IP パケットを監視し, セキュリティチェックやコンテンツの分析等を行う．このシステムの中でコンテンツのキャッシュを行うモジュール部分を, 実験における評価用アプリケーションとして利用した．このモジュールは, セキュリティチェックを行うために必要不可欠な処理であり, ネットワーク利用の効率化を図るためにも有用なものである．

実験に用いたアプリケーションの構成を図 2.3 に示す．

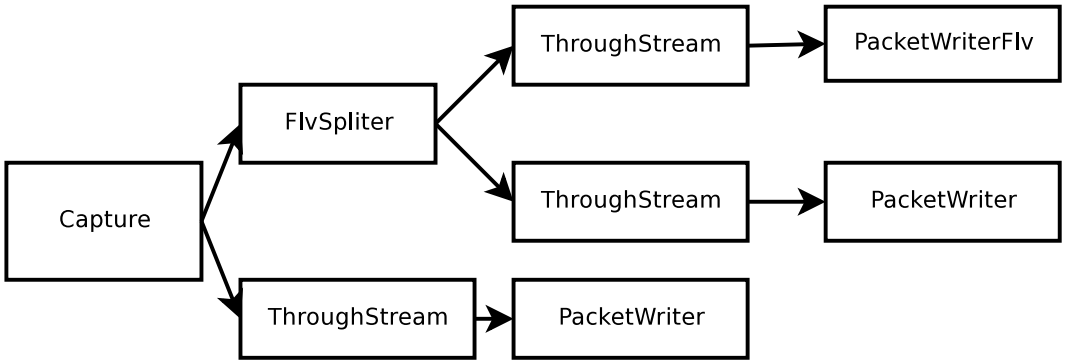

図 2.3 評価用ストリーミング型アプリケーション．

以下にそれぞれのタスクの簡単な説明を述べる．

Capture タスク： IP パケットをキャプチャし, HTTP 通信とそれ以外とに分割するタスクである. 今回は各実験で同じデータを利用するために, リアルタイムにパケットをキャプチャするのではなく, tcpdump[14] 形式で保存されたデータをファイルから読み込む仕様とし, 実験では約 2.7GB のパケットダンプデータを用いた. このタスクは常に `server02` 上で実行される.

FlvSpliter タスク： 先行タスクから HTTP 通信の IP パケットを受け取り, ヘッダを解析して flv データ (動画ストリーミングデータ) のパケットとそれ以外のパケットに分割するタスクである.

PacketWriter タスク： このタスクは, 先行タスクから IP パケットを受け取り, シーケンス番号に基づいてデータを再構築し, そのデータをローカルのディスクへ書き出す. このタスクはストリーミング入力を持っているため, 提案するスケジューリング手法では本来ストリーミングタスクとして扱われるが, ファイルの書き出しを行うため例外的に一定のホストで実行する必要がある. 今回の実験では `server08` で実行される.

PacketWriterFlv タスク： PacketWriter とほぼ同じタスクだが, こちらは HTTP ヘッダの解析を行い, 再構築されたデータの中から flv データ部分だけを取り出し, ディスクへ書き出す. このタスクも PacketWriter タスクと同様の理由で, 常に `server08` 上で実行される.

ThroughStream タスク： 特に何も処理をせず, 先行タスクから受け取った IP パケットを後続タスクへと受け流すだけのタスクである. このタスクは, 実際のアプリケーションにおいてファイル内のチェック処理, 動画変換処理などが挿入される部分に代わって配置した.

2.3.2.2 非ストリーミング型アプリケーション

非ストリーミング型分散アプリケーションとして, 早稲田大学笠原研究室で配布されている, Standard Task Graph set(STG)[12] を用いた. 今回の実験では, タスク数が 50 個の STG を 180 種類利用し, タスク間では 10KB のデータ転送を行うものとした. 評価にはこれら 180 種類の STG の平均実行時間を用いた.

2.3.3 タスクスケジューリングプラットフォーム

実験に用いた並列分散処理ソフトウェアプラットフォームの構成を図 2.4 に示す. これは大きく分けて，タスクスケジューリングシステムと資源情報サーバ (RIS) [10]

の二つの部分から構成されており，すべて Java を用いて実装されている．また, 各モジュール間の通信には JavaRMI およびソケットライブラリが用いられている．

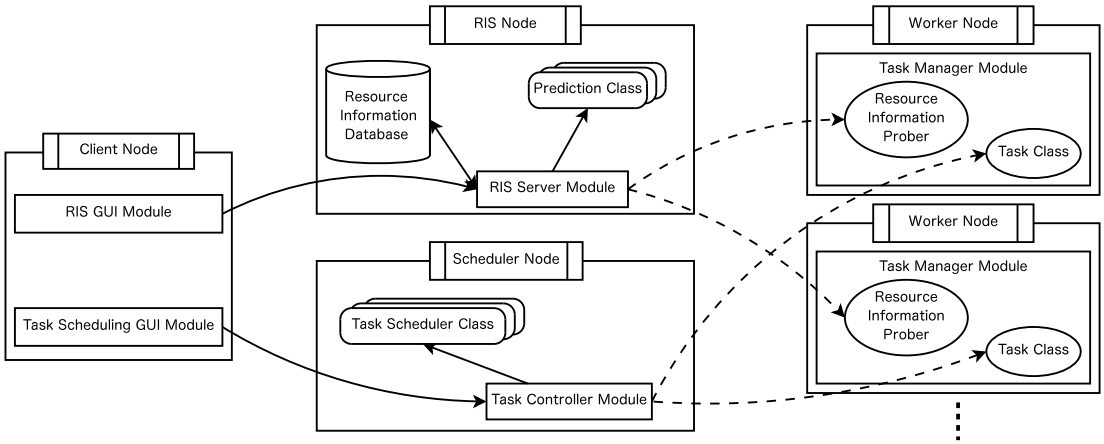

図 2.4 ソフトウェアプラットフォーム．

2.3.3.1 タスクスケジューリングシステム

タスクスケジューリングシステムは, タスクコントローラ，タスクマネージャ，タスクスケジューリング GUI の三つのモジュールから構成されており, タスクの組み合わせで構成される並列分散アプリケーションを, 並列分散環境上で実行するためのシステムである．各モジュールの役割は以下の通りである．

タスクコントローラモジュール： タスクコントローラモジュールはスケジューラノード上で実行され, ユーザが指定したタスクスケジューラクラスに基づいて，タスクをタスクマネージャモジュールへと割り付ける．

タスクマネージャモジュール： タスクマネージャモジュールは, 各ワーカノード上で動作する．このモジュールは, タスクコントローラにより割り付けられたタスクを実行するものである．

タスクスケジューリング GUI モジュール： ユーザがタスクコントローラを制御するための GUI フロントエンドであり, クライアントノード上で実行される．ユーザはこのモジュールを用いることで, タスクを組み合わせて並列分散アプリケーションを構成できる．また, 任意のタスクスケジューラを利用してその並列分散アプリケーションをシステム上で実行可能である．タスクスケジューリング GUI モジュールのスクリーンショットを図 2.5 に示す．

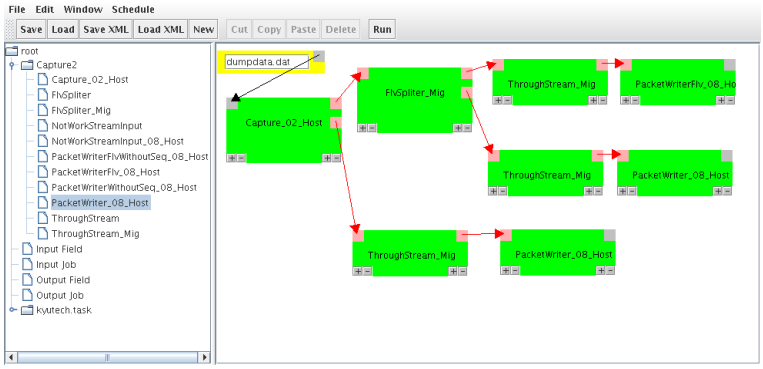

図 2.5 タスクスケジューリング GUI.

タスクスケジューリングシステムは API ライブラリを提供している．この API を用いて Java のクラスとしてタスクを定義することが可能で, そのタスクを接続することにより, システム上で動作する並列分散アプリケーションを作成することができる．また, 同様にこの API を用いることで, タスク実行順序・実行計算機の決定を行うタスクスケジューリング手法も, Java のクラスとして実装することができる．それぞれの詳細について以下に示す．

タスククラス： タスククラスは, 並列分散アプリケーション内のタスクの実装である．タスク間の通信には二種類の方法が利用可能である．一つは Java RMI を用い, タスク開始時に引数としてデータを受け取り, タスク終了時に返り値としてデータを渡す方法で, これは通常の (非ストリーミング) 通信として利用される．もう一つは Java Socket を用いて, タスクの実行中に継続的にタスク間通信を行う手法であり, ストリーミング通信として利用されるものである．なお, どちらの方法もデータの受け渡しはタスクマネージャモジュール間で直接行われる．

並列分散アプリケーション： タスクスケジューリングシステムで実行可能な並列分散アプリケーションは, タスククラスの組み合わせとして定義される．並列分散アプリケーションの実体はタスククラスの組み合わせを記述した XML であり, ユーザは XML を直接記述するか, 前述のタスクスケジューリング GUI を用いることで定義可能である．

タスクスケジューラクラス： タスクスケジューラクラスはタスクスケジューリング手法の実装である．このクラスは, 前述のタスクコントローラモジュールにより利用され, このクラスに基づいてタスクの割り付けが行われる．

本研究では，このタスクスケジューリングシステム上で動作するタスクスケジューラを開発し, 評価を行った.

2.3.3.2 資源情報サーバ (RIS)

資源情報サーバ (RIS: Resource Information Server) は，各ワーカノードのプロセッサ負荷やネットワーク負荷といった資源情報を計測・蓄積し, 過去の情報の提示や未来の資源情報の予測の提供を行う [10]. RIS はサーバモジュールと GUI モジュールから構成される. サーバモジュールは資源情報収集子を用いて資源情報を収集し, データベースにその結果を格納する. 収集可能な資源情報は CPU 負荷やメモリ情報, ホスト間をポイントツーポイントで測定したネットワークバンド幅とレイテンシであり, 取得する資源情報の種類や収集間隔は設定ファイルを記述することで自由に設定可能である. また, このモジュールは, 指定された予測手法に基づいて未来の資源情報の予測値の提供も行う. 予測手法は API ライブラリを用いて予測クラスとして自由に実装可能であり, 現在は過去一定期間の変動の平均を今後の予測値とする平均法や, 最近の変動と類似した過去のパターンを検索し, マッチしたパターンの負荷変動を今後の予測値とする類似法 [10], その改良手法 [20] などが実装されている. GUI モジュールは, 収集した資源情報の表示や, グラフによる資源情報の変動履歴の表示などが行える.

この種の資源情報予測システムとして, NWS(Network Weather Service,[11]) が存在する. NWS に組み込まれる予測モジュールは, 直近の資源情報の変化を予測するものであるのに対し, RIS は任意の未来における資源情報の予測を行うものであるため, 互いに依存関係のあるタスクから構成されるアプリケーションのタスクスケジューリングには, RIS による予測がより有用である.

2.3.4 実験 1: 経路選択の評価実験

この実験は, タスクに適した特性のネットワークを選択することの有用性について評価するためのものである. この実験では, ストリーミング型アプリケーションと非ストリーミング型アプリケーションを, 以下の三種類のスケジューラを用いて実行した.

1. 全ての通信に経路 A を用いる.
2. 全ての通信に経路 B を用いる.
3. 通信に適した経路を用いる, すなわちストリーミングデータ通信には経路 B を, 通常の通信には経路 A を用いる.

各アプリケーションを各スケジューラで 10 回ずつ実行し, その平均実行時間を用いて評価を行う.

なお, この実験は通信負荷を掛けずに行った. タスクに適した特性のネットワークを選択することの有用性の確認がこの実験の目的であり, バンド幅と遅延の差異に着目した評価をするためである.

表 2.2 は, ストリーミング型アプリケーションと, 非ストリーミング型アプリケーションを, それぞれ別に動作させた場合の結果である. 非ストリーミング型アプリケーションは経路 A を, ストリーミング型アプリケーションは経路 B を利用した場合に, 効率的に実行されていることが確認できる. そして, 通信の特性を考慮し, 経路 A,B の両方を使い分けた場合, ストリーミング型アプリケーション・非ストリーミング型アプリケーション共に効率良く実行できている.

表 2.2 各プログラムの実行時間.

利用経路 / アプリケーション	A	B	A,B
ストリーミング [sec]	560	225	204
非ストリーミング [msec]	1660	2715	1660

続いて, ストリーミング型アプリケーションと非ストリーミングアプリケーションを同時に実行し, その実行時間を測定した. 表 2.3 に結果を示す. 提案手法である通信に適した経路を利用するタスクスケジューラを利用し, 経路 A, B を適切に使い分けることにより, 両アプリケーション共に効率的な実行ができていることが確認できる. 個別に動作させた場合と比較して, 全体の実行時間は増加する傾向にある. 特に経路 A のみを利用した場合の結果が大きく悪化しており, 単独実行であれば高速に処理できていた非ストリーミング型アプリケーションも非常に遅くなっている. これは, ストリーミング型アプリケーションに大きく帯域を圧迫されたことが原因だと考えられる. 経路 A, B を使い分けた場合のストリーミング型アプリケーションの実行時間が経路 B だけを用いた場合と比較して長くなっているが, これは平均をとった 10 回の試行のうち, 1 回の実行が非常に長い実行時間を要しており, その結果に足を引っ張られたためである. その 1 回の実行中にバックグラウンドでシステムプロセスなどが動作したため, 実行時間が増大したのではないかと考えている.

表 2.3 各プログラムの実行時間 (同時実行).

利用経路 / アプリケーション	A	B	A,B
ストリーミング [sec]	601	270	277
非ストリーミング [msec]	6671	2692	1865

2.3.5 実験 2: マイグレーションの評価実験

この実験は, 動的なネットワーク負荷変動に対するタスクマイグレーションの有用性を明らかにするためのものである. 実験には二種類のタスクスケジューラを利用しており, 両者ともにワーカの選択には 2.2.1.1 で示した手法を用いている. そして, 一方にはタスクマイグレーション機構を実装し, 他方では非実装である. この実験では, 2.3.1 で示したスクリプトを用いて通信負荷を加えた実験環境上で, 上記の二種のタスクスケジューラを用いてストリーミング型アプリケーションを実行し, その実行時間を比較することでタスクマイグレーションの効果を確認する. また, 実行時間の標準偏差を用い, 実行時間のばらつきについても検証する.

タスクマイグレーションを行う際に利用するネットワークバンド幅の予測値は, RIS を用いて取得する. 利用した予測手法は, 過去 1 分間のネットワークバンド幅の平均値を予測値とする手法である. 実験環境が普段あまり利用されてない環境であり, 予測に必須となる有効なログが無い上に, スクリプトを用いて完全にランダムな負荷が加えられるため, 有効な予測値の取得が難しいことから, 類似法などの高精度予測手法を利用せず, 簡易な予測手法を用いた. また, タスクマイグレーションを実装したタスクスケジューラでは, ストリーミングデータの流量を 10 秒毎に測定し, その値とネットワーク性能の実測値の和が 15MBytes/sec を切った場合にネットワークが混雑していると判断し, タスクマイグレーションを行うように設計した.

実験結果を表 2.4 に示す. マイグレーションを行うことにより, 平均実行時間が短縮できていることが確認できる. また, 実行時間の標準偏差が小さくなっている, つまり実行時間のばらつきが抑えられ, 実行速度の安定度が向上していることが確認できる.

表 2.4 タスクマイグレーションの効果.

	実行時間 [sec]			
	平均	最短	最大	標準偏差
マイグレーション実装	376.9	298	461	51.3
マイグレーション非実装	530.9	284	936	217.0

2.4 関連研究と本研究の位置付け

N. Vydyanathan らは, 本研究と同様に通信に着目してストリーミング型分散アプリケーションの実行の高速化を図っている [21]. この研究では, データ通信時間とタスク実行時間を比較し, 実行時間の方が短い場合はタスクの複製を行い通信先のタスクと同一ノードで実行することで, ノード間の通信を削減し, 全体の実行時間を短縮するスケジューリング手法を提案している. また, データ依存関係にあるタスクの同一ノードへの配置や, プロセッサのコア数に応じてタスクの配置数を増加させ, タ

スク間の通信に内部通信を用いる配置を行うなどの手法も取っている. Q. Zhu らは, 部分グラフ同型判定アルゴリズムを利用した, ストリーミング型分散アプリケーション向けのタスクスケジューリング手法を提案している [22]. この手法では, ノードの計算能力とノード間の通信能力が定義された非循環有向グラフを用いて表現されたネットワーク構造グラフと, タスクの計算コストとタスク間通信コストが定義されたタスクグラフの二種類のグラフを利用する. ネットワーク構造グラフ内に存在するタスクグラフの部分グラフをすべて求め, 各グラフのコストを用いて実行時間を計算し, 最短になる配置を行う. 両研究ともにストリーミング型分散アプリケーションを効率的に実行するためのタスクスケジューリング手法の提案であるが, 本研究との大きな違いは, 動的な負荷変動を考慮しているか否かという部分である. 両研究は実行開始時のみにタスクの配置を決定するのに対し, 本研究ではそれに加え, 実行中にリアルタイムにタスクの配置を変更する.

2.5 まとめ

本章では, ストリーミング型分散アプリケーションと非ストリーミング型分散アプリケーションの双方を, 同時に効率的に実行できる新しいタスクスケジューリング手法を提案し, その評価を行った. 評価実験の結果, 提案手法ではタスク間の通信形態に適した特性を持つネットワーク経路を選択し利用することで, 全体の実行時間を短縮できることを確認した. また, ネットワーク負荷が動的に変動する状況においても, 実行時間を短縮するとともに, そのばらつきを抑えることができることを確認した. この特徴により, アプリケーションの実行時間が高精度に予測可能となり, 資源情報の予測精度も向上する. その結果, タスクスケジューラが他のアプリケーションやタスクを実行する際に, より効率的なタスクスケジューリングを実現することが可能となるため, 今回得られた特性は非常に有用である.

第3章　通信データ量の変化に対応したマルチサーバシステムの性能評価手法

3.1　序論

システム環境の性能を向上させ, その上で動作するアプリケーションの実行速度を高速化させるためには, システム内においてボトルネックとなる部分を正しく把握し, その部分の性能を向上させる必要がある. また, アプリケーションに対して, 応答時間の保証や許容するリクエストの量など, 一定の性能を満たす実行を求められる場合は多く存在する. こういった理由から, システム性能を正しく評価するということへの要求は非常に大きい. しかしながら, 実際のシステム上でアプリケーションを動作させて性能評価を行うためには, システムに対し高負荷を加える必要があるため, 既に運用されているシステムであれば, 提供するサービスに支障をきたすことが考えられる. また, 運用前のシステムやサービスを一時停止した上で評価を行うとしても, 様々な状況を考慮して負荷を発生させなければならず, 容易に性能評価を行うことはできない.

そこで, 実際のシステムに高負荷をかけることなく, 正確な性能評価を行うことができる評価手法が必要となる. このような性能評価手法として, Mean Value Analysis(MVA)[23] という待ち行列ネットワークの解析手法をマルチサーバシステムの性能評価に適用した手法 [25] が存在する. しかし, この手法は処理するデータ量の変化を考慮しておらず, データ量に依存して処理時間及び通信時間が変化する実際的なシステムを正しく評価することができない. そこで本章では, 実行がデータ量に依存する場合においても正しく性能を評価できるように既存の MVA モデルを拡張した, 拡張 MVA モデルによる分散マルチサーバシステムの性能評価手法を提案する. そして実際にアプリケーションへ適用し, その評価を行う.

3.2 Mean Value Analysis を用いたマルチサーバシステムの性能評価手法

3.2.1 MVA のサーバ性能評価への適用

待ち行列ネットワークの解析手法である Mean Value Analysis(MVA)[23] は, たたみ込み法と比較して単純であることから広く利用されている. 当初この MVA は閉鎖型ネットワークの解析手法として発表されたが, 発表後間もなくに開放型ネットワークへも適用され [24], 計算機ネットワークシステムや医療用アプリケーション, 金融システムなどの幅広い分野において利用されてきた.

この MVA を用いて分散マルチサーバシステムのトランザクション性能評価を行う手法が, D. Cavendish らにより提案されている [25]. この手法には, 以下の特徴がある.

性能の評価に利用するトランザクションの特性を容易に把握可能:
実際のシステムで利用するためには, MVA モデルの構築を容易に行うことができなければならない. そのために必要となるトランザクションの解析にコストがかからないことは非常に重要である.

トランザクションの到着間隔やサービス時間のばらつきとは無関係に, 任意の負荷状態での応答時間と, システムのキャパシティを評価可能:
実システムにおいて到着間隔やサービス時間のばらつきを確認することは困難であるため, これらに影響されないモデルである必要性がある.

最悪の場合の応答時間を予測し, かつ過剰な予測値を返さない:
システムの評価においては最悪の場合を考慮することが重要であるため, 提案手法では最悪の場合の応答時間を予測値として返す. しかし, それを保証するために過剰な応答時間を予測してしまうと, 正しい性能の評価を行うことができないため, そのような予測は行わない.

様々な粒度のモデルに適応できる柔軟性を持つ:
分散マルチサーバシステムの管理の観点から考えれば, 性能評価を行いたい粒度は様々であるため, どのような粒度でも対応可能にする. 例えば, 各サーバ毎の性能を評価する場合もあれば, システムの中である処理を実行する際に利用する複数のノードをまとめて粗粒度に評価したい場合や, CPU 処理性能や DMA 転送性能などの非常に細粒度な性能評価を行いたい場合などが存在する.

3.2.2 性能評価に用いる MVA モデルの定義

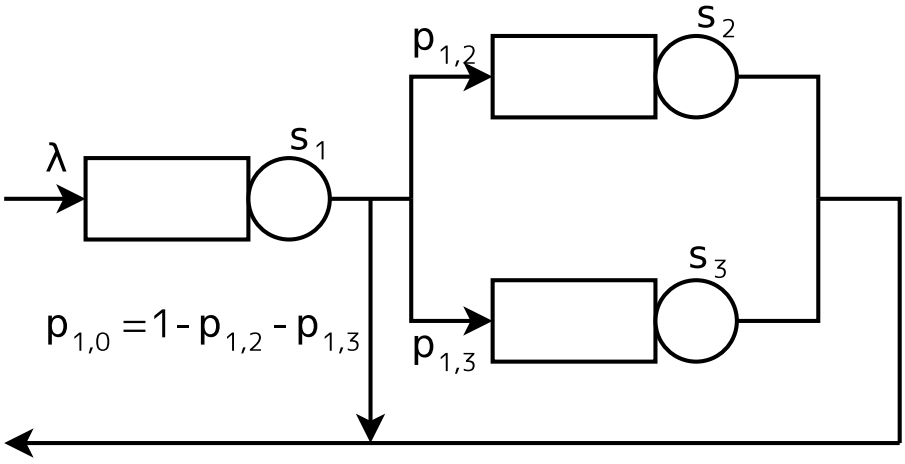

図 3.1 開放型待ち行列ネットワークの例.

図 3.1 は, フロントエンドにロードバランサがあるシステムの開放型待ち行列ネットワークの例であり, 三つの待ち行列が存在している. この図において, λ はシステムへ入力されるメッセージの平均到着率 (messages/sec) を示す. メッセージは各コンポーネントの中で処理され, その処理に要する平均サービス時間は, 各待ち行列ごとに s_1, s_2, s_3 である. また, $p_{i,j}$ は待ち行列 i で処理されたメッセージが待ち行列 j へと遷移する確率を表している. システムの外部は番号 0 で表され, $p_{i,0}$ は待ち行列 i からシステムの外部へメッセージが移動する確率を示す.

各待ち行列 i について, その待ち行列へのメッセージの平均到着率を λ_i, その待ち行列での平均サービス時間を s_i とすると, MVA を用いた解析によりその待ち行列の平均応答時間は, 以下の式を用いて予測できる.

$$R_i(\lambda_i) = \frac{s_i}{1 - \lambda_i s_i} \tag{3.1}$$

図 3.1 で示されるネットワークにトランザクション T_a が平均到着率 λ_a で到着した場合のシステム全体の応答時間は, 各待ち行列の平均応答時間を示す式 3.1 を考慮して, 以下の式で予測することができる.

$$R_{T_a}(\lambda_a) = \frac{s_1}{1 - \lambda_a s_1} + p_{1,2}\frac{s_2}{1 - \lambda_a p_{1,2} s_2} + p_{1,3}\frac{s_3}{1 - \lambda_a p_{1,3} s_3} \tag{3.2}$$

式 3.2 からわかるように, サービス時間 s_i とメッセージの遷移確率 $p_{i,j}$ を一度求めてしまえば, どのようなトランザクションの到着率に対しても応答時間の予測が行える.

3.2.3 多粒度 MVA モデル

MVA モデルを構築する際に利用した待ち行列は, マルチサーバシステムの中でトランザクション処理を行う部分を抽象化したものである. つまり, 処理を行う部分をどのように決定するかによって, 様々な MVA モデルを構築することができる. 例えば, 図 3.1 のフロントエンドロードバランサシステムにおいて, システム全体を処理を行う部分, つまり全体を一つの待ち行列として表すこともできる. この時, サービス時間 s_s を用いて応答時間は以下の式で表すことができる.

$$R_{T_a}(\lambda) = \frac{s_s}{1 - \lambda s_s} \tag{3.3}$$

この式において, $s_s = s_1 + p_{1,2}s_2 + p_{1,3}s_3$ であり, 式 3.2 と式 3.3 は低負荷時 ($\lambda = 0$) においては全く同じ応答時間を示す. しかし, 負荷が任意の場合を考えると, これら二つの式は異なる応答時間を示し, 式 3.3 は式 3.2 が示す応答時間以上の結果を示すことになる. 言い換えれば, MVA モデルを詳細なモデルとして構築すれば, より小さな平均応答時間を予測できることになる.

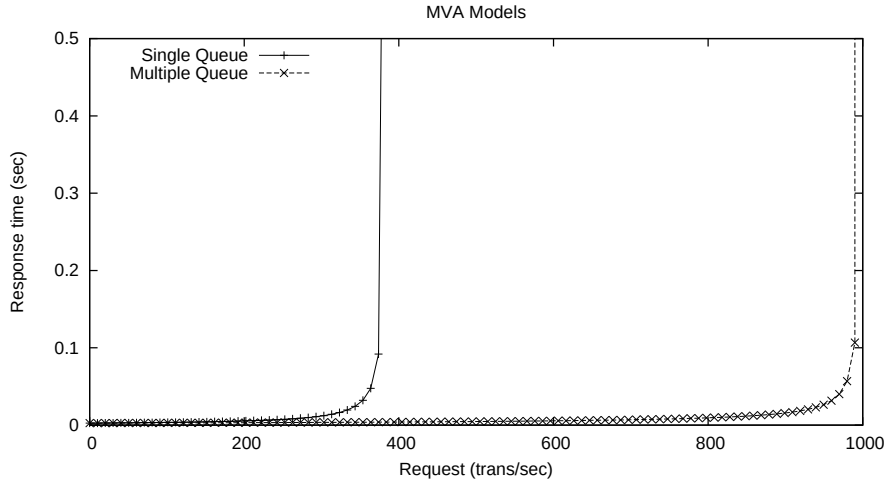

図 3.2 単一待ち行列と複数待ち行列での図 3.1 のシステムの MVA モデルによる応答時間予測比較 (パラメータ: s_1=0.001, s_2=0.002, s_3=0.004, $p_{1,2}$=0.4, $p_{1,3}$=0.2).

また, 詳細なモデルを構築することで, システムの応答時間が急激に増大する部分であるキャパシティを, より高い値で得ることができる. 図 3.2 は, 単一待ち行列 MVA モデルの式 3.3 と, 複数待ち行列 MVA モデルの式 3.2 の二つを用いて, 図 3.1

のシステムの応答時間を予測したものである. なお, 各待ち行列のサービス時間及び待ち行列間の遷移確率は, 図のキャプションに示してあるパラメータを用いている. 図 3.2 を見れば, 複数待ち行列を用いたキャパシティの予測値は, 単一待ち行列を用いた予測値の約 3 倍になっていることがわかる. したがって, より詳細な MVA モデルの構築には多数のパラメータを得て利用する必要があるが, それと引き換えに高いキャパシティの予測を行うことができると言える.

続いて, この式を待ち行列が N 個あるシステムのモデルへ一般化する. そのようなシステムでは, MVA モデルを作成するために, N 個存在する待ち行列のサービス時間 s_i と, 各待ち行列及び外部の間でのメッセージ遷移確率 $p_{i,k}$, $(0 \leq i, k \leq N)$ を求める必要がある. これらの値を用いることで, トランザクションの応答時間は以下の式で予測できる.

$$R_T(\lambda) = \sum_{i=1}^{N} \frac{s_i \sum_{j=0}^{N} p_{j,i}}{1 - \lambda s_i \sum_{j=0}^{N} p_{j,i}} \tag{3.4}$$

また, 応答時間が無限大となるトランザクション到達率をシステムのキャパシティ C_T として定義すれば, 先程の式 3.4 からキャパシティを表す式として以下を得ることができる.

$$C_T = \frac{1}{s_k \sum_{j=0}^{N} p_{j,k}}, \quad s_k \sum_{j=0}^{N} p_{j,k} = \max_i \left(s_i \sum_{j=0}^{N} p_{j,i} \right) \tag{3.5}$$

このとき, 待ち行列 k がキャパシティに直接の影響を与える待ち行列となり, システムのボトルネックとなる部分であることが判断できる.

3.2.4 MVA モデルの正確性

ここでは, MVA モデルの正確性について述べる. まず単一待ち行列のモデルについて述べ, そして複数待ち行列によるネットワークモデルについて述べる.

3.2.4.1 単一待ち行列モデル

ここで述べる中で最も重要なのは以下の定理である.

定理 3.1. 到着間隔とサービス時間の分布が任意である, 安定な待ち行列システムを考える. この待ち行列システムの平均サービス時間を s, サービス時間の標準偏差を σ_b, トランザクションの到着間隔の標準偏差を σ_a としたとき,

$$2s^2 \geq \sigma_a^2 + \sigma_b^2 \tag{3.6}$$

であれば, *MVA* モデルは応答時間の上界を示す.

定理 *3.1* の証明. MVA モデルの出発点の式である, 式 3.1 を考える. この式は, 到着率とサービス時間それぞれの平均値のみに基づいて, 任意の到着率とサービス時間の分布を持つ待ち行列 (G/G/1 待ち行列) の平均応答時間を示すものであり, 到着間隔とサービス時間の分布は考慮する必要がない. しかし, ここでは MVA モデルが示す応答時間の範囲と G/G/1 待ち行列の応答時間の範囲について論じるため, これらの分布を考える必要がある.

W_u を待ち行列における待ち時間の上限値, W_{mva} を MVA モデルで予測される待ち時間とする. 本論文では到着率の分布とサービス時間の分布は独立であると仮定するので, 式 3.3 から W_{mva} を以下の値として得ることができる.

$$W_{mva} = \frac{s}{1 - \lambda s} - s \tag{3.7}$$

ここでは MVA モデルによる応答時間の予測が上界を示す, すなわち最悪の場合の応答時間を保証する場合について考えるため, 以下の条件について議論する.

$$W_{mva} - W_u \geq 0 \tag{3.8}$$

この式 3.8 が, MVA モデルが最悪の場合の応答時間を保証するための条件である. G/G/1 待ち行列の平均待ち時間の上限値 W_u は, 以下の式で表される [26].

$$W_u = \frac{\sigma_a^2 + \sigma_b^2}{2\tau(1 - \rho)} \tag{3.9}$$

ここで, σ_a と σ_b はそれぞれ到着間隔とサービス時間の標準偏差, τ は平均到着間隔 ($\tau = 1/\lambda$), $\rho = \lambda s$ は平均利用率である. 式 3.7, 3.9 を用いて, MVA モデルが最悪の場合の応答時間を保証するための条件を示す不等式 3.8 を表せば, 以下の式になる.

$$\frac{s}{1 - \lambda s} - s - \frac{\sigma_a^2 + \sigma_b^2}{2\tau(1 - \rho)} = \frac{2s^2 - (\sigma_a^2 + \sigma_b^2)}{2\tau(1 - \rho)} \geq 0 \tag{3.10}$$

この式から, 以下の条件を満たすことで単一待ち行列の MVA モデルは最悪の場合の応答時間を示すことが保証される.

$$2s^2 \geq \sigma_a^2 + \sigma_b^2 \tag{3.11}$$

□

なお, 式 3.6 は MVA モデルが最悪の場合の応答時間を予測することを保証するための十分条件であるが, 必要条件ではないことに注意が必要である. つまり, 式 3.6 を満たさない場合でも, MVA モデルによる応答時間の予測値が待ち行列の応答時間よりも低い値を示すとは限らない.

続いて具体的に, 多くのシステムの典型的な例である, 到着間隔が指数分布であるシステム (M/G/1 システム) を考える. M/G/1 待ち行列の応答時間はポラチェック-ヒンチンの平均値公式 [26] から,

$$W = \frac{\sigma_b^2 + s^2}{2\tau(1-\rho)} \tag{3.12}$$

であるので, MVA モデルによる応答時間の予測値がこれを上回るための条件は, 式 3.7, 3.8, 3.12 より,

$$s^2 \geq \sigma_b^2 \tag{3.13}$$

となることがわかる.

もう一つの正確性に関する論点は, MVA モデルによる予測応答時間がどの程度実際の応答時間を超過するかである. しかし, G/G/1 待ち行列の平均待ち時間の下限を示す式は, 到着間隔とサービス時間の分散が 0 である時以外では存在しないため, MVA モデルによる予測値との関係を示すことができない. そこで, 到着間隔の分布を仮定してこの関係を示す.

まず最初に一つ定理を示す.

定理 3.2. 到着間隔とサービス時間の分布が任意である, 安定な待ち行列システムを考える. この待ち行列システムの平均到着率を λ, 平均サービス時間を s とすると, *MVA* モデルが示す平均応答時間と, 待ち行列の平均応答時間の差は, 最大で

$$W_{mva} = \frac{s}{1-\lambda s} - s \tag{3.14}$$

である.

定理 *3.2* の証明. 任意の分布ということで, D/D/1 待ち行列システムを取り上げて考える. D/D/1 待ち行列システムでは平均待ち時間は 0 であるため, $W_{mva} - W_{d/d/1}$ は W_{mva} となり, これが最大の差である. □

続いて, MVA モデルが予測する応答時間の範囲が到着間隔及びサービス時間の分布の平均値と分散にどのように依存するか示すため, 幾つかの特殊な到着過程について例として取り上げる.

定義 3.1. 連続型分布関数 F において, 全ての t について

$$\int_t^\infty \frac{1-F(u)}{1-F(t)} du \leq \gamma \quad 1 - F(t) > 0 \tag{3.15}$$

を満たす場合, その平均残余寿命は γ 以下である. このような確率分布を, $\gamma-MRLA$ 分布と呼ぶ.

定理 3.3. 到着間隔の分布が τ-*MRLA* 分布であり, 平均到着間隔が τ でその標準偏差が σ_a, サービス時間の分布は一般分布であり, 平均サービス時間が s でその標準偏差が σ_b である安定な待ち行列システムを考える. この待ち行列システムを示す *MVA* モデルの平均応答時間の予測精度は

$$W_{mva} - W_{mrla} \leq \frac{s\rho}{1-\rho} + \frac{\tau(1+\rho)}{2} - \frac{\sigma_a^2 + \sigma_b^2}{2\tau(1-\rho)} \tag{3.16}$$

となる. ここで, $\rho = s/\tau$ であり, W_{mva} と W_{mrla} はそれぞれ *MVA* 及び τ-*MRLA/G/1* 待ち行列の平均待ち時間である.

定理 *3.3* の証明. 到着時間が τ-MRLA 分布である, τ-MRLA/G/1 待ち行列の平均待ち時間の下限は以下の式で表される [26].

$$W_{mrla} \geq W_u - \frac{1}{2}\tau(1+\rho) \tag{3.17}$$

したがって, 式 3.3, 3.9, 3.17 より, τ-MRLA/G/1 待ち行列の応答時間の MVA モデルによる予測超過量は,

$$W_{mva} - W_{mrla} \leq \frac{s\rho}{1-\rho} + \frac{\tau(1+\rho)}{2} - \frac{\sigma_a^2 + \sigma_b^2}{2\tau(1-\rho)} \tag{3.18}$$

となる. □

式 3.16 は, 到着間隔とサービス時間の分散が大きいほど MVA モデルの予測する応答時間の超過量が小さくなることを示す. 最悪の場合の超過量を示すのは分散が共に 0 の場合であり, その超過量は,

$$W_{mva} - W_{mrla} \leq \frac{s\rho}{1-\rho} + \frac{\tau(1+\rho)}{2} \tag{3.19}$$

となる. この式より, 利用率 ρ が 1 に近づけば, 予測の超過量は無限大に近づくことがわかる. 低負荷時においては, 平均到着間隔の半分である $\tau/2$ が予測超過量となる.

定義 3.2. 連続型分布関数 F において, $t > 0$ であれば任意の $\epsilon > 0$ について

$$\frac{F(t+\epsilon) - F(t)}{1 - F(t)} \quad 1 - F(t) > 0 \tag{3.20}$$

が t の非減少関数となるならば, この分布を *IFR* 分布と呼ぶ.

定理 3.4. 到着間隔の分布が *IFR* 分布であり, 平均到着間隔が τ でその標準偏差が σ_a, サービス時間の分布は一般分布であり, 平均サービス時間が s でその標準偏差が σ_b である安定な待ち行列システムを考える. この待ち行列システムを示す *MVA* モデルの平均応答時間の予測精度は

$$W_{mva} - W_{ifr} \leq \frac{s\rho}{1-\rho} + \frac{\tau\rho}{2} - \frac{\rho\sigma_a^2 + \sigma_b^2}{2\tau(1-\rho)} \tag{3.21}$$

となる.

定理 *3.4* の証明. 到着間隔が IFR 分布である, IFR/G/1 待ち行列システムの平均待ち時間の下限は, 以下の式で示される [26].

$$W_{ifr} \geq W_u - \frac{1}{2}\tau\left(\frac{\sigma_a^2}{\tau^2} + \rho\right) \tag{3.22}$$

したがって, 式 3.3, 3.9, 3.22 より, IFR/G/1 待ち行列システムにおける平均待ち時間の予測超過量は

$$W_{mva} - W_{ifr} \leq \frac{s\rho}{1-\rho} + \frac{\tau\rho}{2} - \frac{\rho\sigma_a^2 + \sigma_b^2}{2\tau(1-\rho)} \tag{3.23}$$

となる. □

式 3.21 も式 3.16 と同様に, 到着間隔とサービス時間の分散が大きいほど MVA モデルの予測する応答時間の超過量が小さくなることを示す. 最悪の場合の超過量を示すのは分散が共に 0 の場合であり, その超過量は,

$$W_{mva} - W_{ifr} \leq \frac{s\rho}{1-\rho} + \frac{\tau\rho}{2} \tag{3.24}$$

となる. この式より, 超過量は高負荷時には無限大に, 低負荷時には 0 に近づく.

3.2.4.2 複数待ち行列によるネットワークモデル

ここまで, 到着間隔とサービス時間の分布の標準偏差が, シングルサーバシステムの MVA モデルが示す予測応答時間の精度にどのような影響を与えるかについて示してきた.

現実のシステムを考えるためには, 複数待ち行列の MVA ネットワークモデルにおいて, この精度がどのように影響を受けるのかを調査する必要がある. そこで, 到着過程とサービス過程の一次及び二次モーメント, つまり平均と分散が, MVA 待ち行列ネットワークを伝わっていくとどのように変化するかを考える. 一般的なネットワークは, 図 3.3 に示すような直列サブネットワークと並列サブネットワークの二種類の待ち行列ネットワークを複数組み合わせて構成されている. それぞれのネットワークについて応答時間がどうなるかを論じる.

直列サブネットワーク 直列サブネットワークについて考えるには, 二番目の待ち行列への到着過程の平均値および分散を把握しなければならない. ここではマルコフモデル直列ネットワークについての議論にしたがって, 一般分布についての議論をしていく.

到着間隔とサービス時間の分布が互いに独立であるとすれば, 二番目の待ち行列への到着過程は, 最初の待ち行列が空であるか否かによって場合分けして決定でき

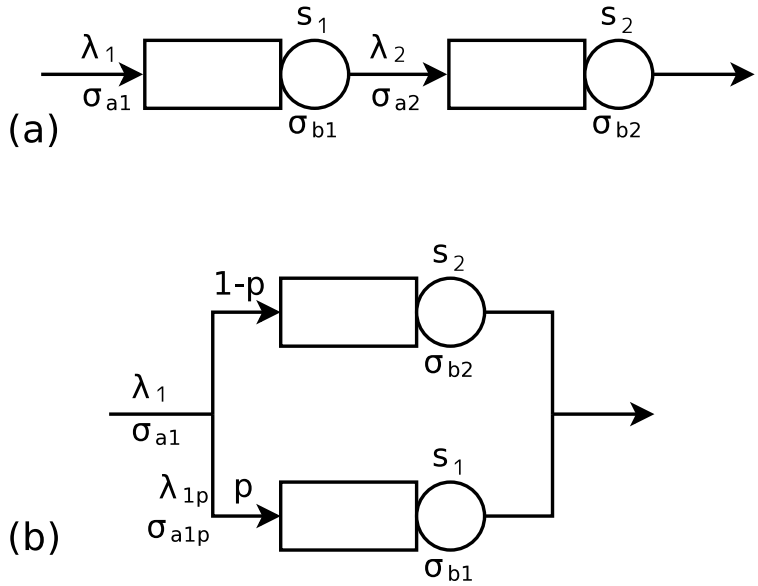

図 3.3 MVA ネットワークを構成するサブネットワーク: (a) 直列, (b) 並列.

る. もし最初の待ち行列が空でなければ, 二番目の待ち行列での到着過程の分布は最初の待ち行列でのサービス過程の分布と一致する. 一方最初の待ち行列が空である場合, 最初の待ち行列での到着間隔における残余寿命と, 最初の待ち行列でのサービス時間の合計が, 二番目の待ち行列での到着間隔の確率変数となる.

より正確に示すために, 待ち行列 1 への到着間隔の確率変数を T_1 とすれば, $P[T_1 < t]$ はあるメッセージが届いた後に, 時間 t 経過までの間に次のメッセージが待ち行列 1 に届く確率を示す. このとき, 残余寿命 T_1^r は

$$T_1^r = P[T_1\langle t + t_r | T_1 \rangle t_r] \tag{3.25}$$

と定義される. この T_1^r の一次モーメントを, T_1 の平均残余寿命と呼ぶ. T_1 が有限モーメントを持てば, T_1^r も同じく有限モーメントを持つ.

以上をまとめて, 直列サブネットワークでの二番目の待ち行列における到着過程について示せば, 以下のようになる.

$$\lambda_2 = \lambda_1 \tag{3.26}$$

$$\sigma_{a2}^2 = \sigma_{b1}^2 \quad \text{待ち行列 1 が空でない場合} \tag{3.27}$$

$$\sigma_{a2}^2 = \sigma_{b1}^2 + \sigma_{T_1^r}^2 \quad \text{待ち行列 1 が空の場合} \tag{3.28}$$

特に, 到着過程が指数分布であれば, その無記憶性から $\sigma_{T_1^r}^2 = \sigma_{a1}^2$ である. 式 3.28 は待ち行列が多数直列に接続されるネットワークで表されるシステムにおいて, 低

負荷時には到着間隔の分散が後ろの待ち行列へ行くほど大きくなることを示している. しかし, 式 3.27 からわかるように多くの負荷が加わるシステムでは, 到着間隔の分散は, 前の待ち行列でのサービス時間の分散と等しくなる. 待ち行列 i でのサービス時間の分散が, その待ち行列のサービス時間及び次の待ち行列 $i+1$ でのサービス時間と比較して小さいならば, 各待ち行列において式 3.6 を満たす事となる. そのため, MVA モデルによる直列サブネットワーク部分の応答時間の予測は, 実際のシステムの応答時間の上界を示す. (ただし, 非常に高負荷が加わる場合においては, 単一待ち行列の場合において議論したように大きな予測超過を伴う可能性がある.)

結論として, N 段の直列サブネットワークの応答時間について, 最悪の応答時間を示すことを維持するように単一 MVA モデルの式を用いて表せば以下のようになる.

定理 3.5. N 段に直列に接続された待ち行列ネットワークの応答時間 R_t^N は

$$R_t^N(\lambda) = \frac{s_t}{1-\lambda s_t} \leq \sum^N \frac{s_i}{1-\lambda s_i} \quad if\ s_t = \Sigma s_i \tag{3.29}$$

である.

定理 *3.5* の証明. $s_t/(1-\lambda s_t) - \sum^N s_i/(1-\lambda s_i) \geq 0$ であることを証明する.

$$\frac{s_t}{1-\lambda s_t} - \sum^N \frac{s_i}{1-\lambda s_i} = \frac{s_t \prod_i (1-\lambda s_i) - (1-\lambda s_t)\sum_j s_j \prod_{i\neq j}(1-\lambda s_i)}{(1-\lambda s_t)\prod_i (1-\lambda s_i)} \tag{3.30}$$

$0 \leq \lambda s_i < 1$, $\prod_{i\neq j}(1-\lambda s_i) \leq 1$ であるから,

$$\frac{s_t}{1-\lambda s_t} - \sum^N \frac{s_i}{1-\lambda s_i} \geq \frac{s_t \prod_i (1-\lambda s_i) - (1-\lambda s_t)\sum_i s_i}{(1-\lambda s_t)\prod_i (1-\lambda s_i)} \tag{3.31}$$

$s_t = \sum s_i$ と置き換えて,

$$\frac{s_t}{1-\lambda s_t} - \sum^N \frac{s_i}{1-\lambda s_i} \geq \frac{s_t(\prod_i (1-\lambda s_i) - (1-\lambda s_t))}{(1-\lambda s_t)\prod_i (1-\lambda s_i)} \tag{3.32}$$

ここで, $0 \leq a_i \leq 1$ であれば $\prod(1-a_i) \geq 1 - \sum a_i$ となる性質を利用すれば, $\prod(1-\lambda s_i) - (1-\lambda s_t) = \prod(1-\lambda s_i) - (1-\sum \lambda s_i) \geq 0$ であるので,

$$\frac{s_t}{1-\lambda s_t} - \sum^N \frac{s_i}{1-\lambda s_i} \geq 0 \tag{3.33}$$

□

並列サブネットワーク　待ち行列が並列に接続されている場合について考える際には, 到着間隔 $A(t)$ ではなく, 到着の計数過程 $A(n,t)$ を利用する. $A(n,t)$ は時間 t の間に n 個のメッセージがシステムに到着する確率である. メッセージが待ち行列 1 へ遷移確率 p で, 待ち行列 2 へ遷移確率 $1-p$ で送られる場合, 待ち行列 1 への到着の計数過程 $A_{1p}(n,t)$ を知るために $A(n,t)$ を利用することができる. $A(z,t)$ を $A(n,t)$ の z 変換とすれば, $A_{1p}(z,t)$ は以下のようになる [27].

$$A_{1p}(z,t) = A_1(1-p(1-z),t) \tag{3.34}$$

サブネットワークへの到着の計数過程 $A_1(n,t)$ と, 待ち行列 1 への到着の計数過程 $A_{1p}(n,t)$ の間での, 平均および分散それぞれの関係は, 式 3.34 を $z=1$ の点で z について一次微分および二次微分することで計算でき [26], 以下のようになる.

$$E[A_{1p}(n,t)] = pE[A_1(n,t)] \tag{3.35}$$

$$\sigma^2[A_{1p}(n,t)] = pE[A_1(n,t)](1-p) + p^2\sigma^2[A_1(n,t)] \tag{3.36}$$

ここで, 分割後の到着の計数過程の分散が元の到着の計数過程の分散よりも大きい時, つまり

$$\sigma^2[A_{1p}(n,t)] - \sigma^2[A_1(n,t)] > 0 \tag{3.37}$$

である状況下について考える. 到着の計数過程の分散が増加するということは, その到着間隔の分散が減少することと等価である [28].

式 3.37 に式 3.36 を代入すると, 以下の関係が得られる.

$$E[A_1(n,t)] > \frac{p+1}{p}\sigma^2[A_1(n,t)] \tag{3.38}$$

高負荷状況では, $E[A_1(n,t)] \gg \sigma^2[A_1(n,t)]$ であり, 式 3.37 の条件が成り立つ.

したがって, どんな小さな p でも, 到着の計数過程の分散が, 分割先では増加するということになり, 言い換えれば, 到着過程の分散が分割先では減少するということになる. したがって, 高負荷な状況下では, 並列サブネットワークの各待ち行列において式 3.6 を満たす事となる.

結論として, N 個の待ち行列が並列接続されたサブネットワークの応答時間について, 最悪の応答時間を示すことを維持するように単一 MVA モデルの式を用いて表せば以下のようになる.

定理 3.6. 並列に接続された N 個の待ち行列の応答時間 R_t^N は

$$R_t^N(\lambda) = \frac{s_t}{1-\lambda s_t} \geq \sum^N \frac{p_i s_i}{1-\lambda p_i s_i} \quad if\ s_t = \sum p_i s_i \tag{3.39}$$

となる.

定理 *3.6* の証明. $s_t/(1-\lambda s_t) - \sum^N p_i s_i/(1-\lambda p_i s_i) \geq 0$ であることを証明する.

$$\frac{s_t}{1-\lambda s_t} - \sum^N \frac{p_i s_i}{1-\lambda p_i s_i} = \frac{s_t \prod_i (1-\lambda s_i) - (1-\lambda s_t)\sum_j p_j s_j \prod_{i\neq j}(1-\lambda p_i s_i)}{(1-\lambda s_t)\prod_i (1-\lambda p_i s_i)} \tag{3.40}$$

$0 \leq \lambda p_i s_i < 1$, $\prod_{i\neq j}(1-\lambda p_i s_i) \leq 1$ であるから,

$$\frac{s_t}{1-\lambda s_t} - \sum^N \frac{p_i s_i}{1-\lambda p_i s_i} \geq \frac{s_t \prod_i (1-\lambda s_i) - (1-\lambda s_t)\sum_i p_i s_i}{(1-\lambda s_t)\prod_i (1-\lambda p_i s_i)} \tag{3.41}$$

$s_t = \sum p_i s_i$ と置き換えて,

$$\frac{s_t}{1-\lambda s_t} - \sum^N \frac{p_i s_i}{1-\lambda p_i s_i} \geq \frac{s_t(\prod_i (1-\lambda s_i) - (1-\lambda s_t))}{(1-\lambda s_t)\prod_i (1-\lambda p_i s_i)} \tag{3.42}$$

ここで, $0 \leq a_i \leq 1$ であれば $\prod(1-a_i) \geq 1 - \sum a_i$ となる性質を利用すれば, $\prod(1-\lambda p_i s_i) - (1-\lambda s_t) = \prod(1-\lambda p_i s_i) - (1-\sum \lambda p_i s_i) \geq 0$ であるので,

$$\frac{s_t}{1-\lambda s_t} - \sum^N \frac{p_i s_i}{1-\lambda p_i s_i} \geq 0 \tag{3.43}$$

□

実際のシステムで, ある部分のサービス時間 s_e を, 後述する zero-load と呼ばれる状況で測定すれば, $\lambda = 0$ であるから式 3.29, 3.39 が真となり, 並列サブネットワーク部であれば $s_e = \sum^N p_i s_i$, 直列サブネットワーク部であれば $s_e = \sum^N s_i$ のそれぞれの条件を満たすことになる. つまり, 式 3.29, 3.39 は, MVA モデルの多粒度性を示しており, 粒度を粗くしても MVA モデルによる予測は最悪値を保証することになる.

3.2.5 実システムにおける MVA モデルの構築

実システムにおいて性能評価のために MVA モデルを構築する手順は, 以下の通りである.

1. 性能評価を行う粒度の決定: 3.2.1 で述べたように, この手法では様々な粒度で性能評価を行うことが可能である. モデルを構築するためにはまずこの粒度を決定し, 要素の境界を決定する必要がある.

2. MVA パラメータの決定: 1 で決定した要素ごとに, MVA モデルを作成するために必要なパラメータである平均サービス時間と遷移確率を測定する. この測定は zero-load トランザクションを用いて行い, 複数回実行し平均値を求める. zero-load トランザクションとは, 他に負荷のかかっていない状況においてただ単一に発生させるトランザクションのことであり, 当然のことながらトランザクションを複数発生させて並列に動作させてはならず, 投入したトランザクションの応答が返ってきたことを確認してから, 次のトランザクションを投入しなければならない.

3. MVA モデルの構築: 2 で求めたパラメータを式 3.4, 3.5 に適用し, MVA モデルの構築を行う.

MVA モデルを構築するにあたり, 実際にシステムに負荷を加えることになるのは, 手順 2 の部分のみである. ここで加える負荷は単一トランザクションであるため, システムが利用されていない状況を利用して計測することも可能であるし, システムを停止させたとしても, 実際に様々な状況を考慮してアプリケーションを動作させる場合と比較して, 短時間の停止で済む. また, このトランザクションは故意にかけたものでなくとも, あるユーザが利用した際に発生したトランザクションが zero-load であること, つまり, そのトランザクションの処理中に他のトランザクション処理のリクエストが無かったことが保証されていれば, その結果を用いても良い. これは, DTrace[29] 等のカーネルプローブや, スニファ等のネットワークログを調査することで保証することができる. このような理由から, 実際に負荷を加えて性能を測定する場合と比較して, MVA を用いた性能評価は非常に容易に行うことができると言える.

3.3 データ量によるサービス時間の変化を考慮した拡張 MVA モデル

3.3.1 データ量の変化がサービス時間に与える影響

多くの分散型システムにおいて, サーバやネットワークなどの各コンポーネントがメッセージを処理するために要する時間は, メッセージのデータ量に大きく依存する. 以下にデータ量がどのようにサービス時間に影響するかを示す.

データ処理時間への影響

マルチサーバシステム上の各サーバは, メッセージに付加されているデータに対して様々な処理を施すが, この処理に要する時間はデータのサイズにより変動する. 例えば, SOAP では XML ベースのメッセージを用いて通信を行うが, 一連の処理には一般的に XML の構文解析・バリデーションチェック (例えば, MSXML[30] や javax.xml.{parsers, validation}[31] 等による) を必要とする. 構文解析・バリデーションチェックに要する処理時間は, XML データのサイズに依存する. また別の例として, マルチサーバシステムがセキュリティ向上のために通信の暗号化・復号などを行う場合も, データ量によりその処理に要する時間が変動する.

データ転送時間への影響

メッセージが持つデータ量が変化することで, 当然のことながらデータを転送するために要する時間にも影響を及ぼす. 利用可能なネットワークのバンド幅

が通信するデータサイズと比較して十分に大きいならば, データ転送に要する時間は無視できる程度になる. しかし, 通信データサイズの大きなアプリケーションでは, この転送時間の変動も考慮しなければならない.

3.3.2 既存 MVA の問題点

現実のシステムを考えると, 様々なデータ量を含んだトランザクションが発生することは一般的である. 例えば, Web サーバへクライアントから HTTP リクエストが送られてきた場合, 取得するコンテンツによって返却するデータのサイズは様々である. 3.3.1 に述べたように, データ量はサービス時間と密接な関係があり, データ量が変化すればサービス時間も変化する. しかし, 3.2.2 で述べた既存 MVA モデルは, トランザクションにおけるデータ量の変化を考慮していないためにそれに伴なうサービス時間の変化に対応することができず, 正しい予測を行うことができない.

この問題を解決するために, データ量の変化に対応するような既存 MVA モデルの拡張を提案する.

3.3.3 拡張 MVA モデルの定義

3.3.1 で述べたように, サービス時間 s_i はデータ量 l_i に依存して変化する. そこで, 既存 MVA 式において定数として定義されていたサービス時間 s_i をデータ量 l_i に依存する値として考え, 式 3.4 に適用することで, 以下の式

$$R_T(\lambda) = \sum_{i=1}^{N} \frac{s_i(l_i) \sum_{j=0}^{N} p_{j,i}}{1 - \lambda s_i(l_i) \sum_{j=0}^{N} p_{j,i}} \tag{3.44}$$

をデータ量の変化に対応した MVA モデルとして得ることができる. 同様にキャパシティ C_T を表す式は,

$$C_T = \frac{1}{s_k(l_k) \sum_{j=0}^{N} p_{j,k}}, \quad s_k(l_k) \sum_{j=0}^{N} p_{j,k} = \max_i \left(s_i(l_i) \sum_{j=0}^{N} p_{j,i} \right) \tag{3.45}$$

となる.

しかし, このようにサービス時間 s_i をデータサイズ l_i に依存する単一の値として定義することは, MVA モデル作成の利便性を損ねてしまう. なぜなら, サーバ上へ投入されうるトランザクションにおける全てのデータサイズについて, そのサービス時間を求めるという作業が必要となるためである. 全てのデータサイズに対応するサービス時間を求めるということは, 非常に多くのパターンのトランザクションを用いて zero-load 時の負荷を測定, 解析をする必要があるために非現実的である. そこで, サービス時間 s_i をデータサイズ l_i に依存する関数 $f_i(l_i)$ として定義し, 拡張

MVA モデルを構成する. 応答時間を示す MVA モデルの式およびキャパシティを示す式は, 式 3.44, 3.45 から

$$R_T(\lambda) = \sum_{i=1}^{N} \frac{f_i(l_i) \sum_{j=0}^{N} p_{j,i}}{1 - \lambda f_i(l_i) \sum_{j=0}^{N} p_{j,i}} \tag{3.46}$$

$$C_T = \frac{1}{f_k(l_k) \sum_{j=0}^{N} p_{j,k}}, \quad f_k(l_k) \sum_{j=0}^{N} p_{j,k} = \max_i \left(f_i(l_i) \sum_{j=0}^{N} p_{j,i} \right) \tag{3.47}$$

となる.

本研究では, MMSE(Minimum Mean Square Error : 最小平均二乗誤差法) を用いて s_i の近似曲線を求め, その曲線の式を関数 $f_i(l_i)$ とする. ここで, x 軸をデータサイズ l_i,y 軸をサービス時間 s_i とし, 一般的な二次曲線 $y = ax^2 + bx + c$ を近似曲線として仮定した. 測定した l_i の個数を n とすると, 以下の式を解くことで係数 a,b,c を得られる.

$$\begin{aligned}
\frac{\partial U}{\partial a} &= 2\left(a\sum^{n} x_i^4 + b\sum^{n} x_i^3 + c\sum^{n} x_i^2 - \sum^{n} x_i^2 y_i \right) = 0 \\
\frac{\partial U}{\partial b} &= 2\left(a\sum^{n} x_i^3 + b\sum^{n} x_i^2 + c\sum^{n} x_i - \sum^{n} x_i y_i \right) = 0 \\
\frac{\partial U}{\partial c} &= 2\left(a\sum^{n} x_i^2 + b\sum^{n} x_i + nc - \sum^{n} y_i \right) = 0
\end{aligned} \tag{3.48}$$

近似曲線の次数を高次にすることで, 近似精度をより高めることが可能であるが, 本研究において拡張 MVA モデルの評価に使用したシステムでは, 二次曲線近似で十分な近似関数を得ることができた. 近似関数 f_i は実際のサービス時間の上限を正確に示さなければならない. しかし, この近似関数では上限を保証する形にならないため, 厳密に言えば 3.2.1 で論じた MVA モデルは応答時間の見積りは少なくとも最悪値を示すという特性を保証できなくなる. しかしながら, 二種類のプロトタイプを用いた実験の結果, 近似関数を利用した拡張 MVA モデルにおいても, その近似関数が充分に精度の良い値を示すことがわかったため, これまでと同様に最悪応答時間の見積りを示すと考えられる.

3.3.4 実システムにおける拡張 MVA モデルの構築

実際のシステム上での拡張 MVA モデルの構築は, 3.2.5 で述べた既存 MVA モデルの構築とほぼ同様にして行われる. 以下に拡張 MVA モデルの構築手順を示す.

1. 性能評価を行う粒度の決定: 既存 MVA モデルと同様に, モデルを構築するために要素の境界を決定する.

2. 拡張 MVA パラメータの決定: 既存 MVA モデルを構築する場合と異なり, データ量の変化に対するサービス時間の変化を求めるために, 内部のデータ量が異なるようなトランザクションを数種類発生させ, それぞれのデータ量について平均サービス時間を求めなければならない. また, 求めた平均サービス時間を用いて関数 f_i を生成する.

3. 拡張 MVA モデルの構築: 2 で求めた遷移確率と平均サービス時間を示す関数 f_i を, 式 3.46, 3.47 に適用し, モデルの構築を行う.

3.4 拡張 MVA モデルを用いた性能評価実験

式 3.46 で示される拡張 MVA モデルを用いて, 二種類のマルチサーバシステムを利用して性能評価実験を行った. 実験ではまず zero-load トランザクションを用いてサービス時間を測定し, 拡張 MVA モデルの構築を行う. そして非 zero-load, つまりトランザクションを多数発生させ, 実際にシステムに負荷が加わった際の応答時間を測定し, 拡張 MVA モデルによる応答時間及びキャパシティの予測値と比較する.

3.4.1 クライアント–サーバ連鎖型システム

このシステムは, 様々な通信データサイズに対する拡張 MVA モデルの特性を確認するために作成した, 評価用マルチサーバシステムである. システムの構成を図 3.4 に示す. また, 実際の動作環境は表 3.1 の通りである.

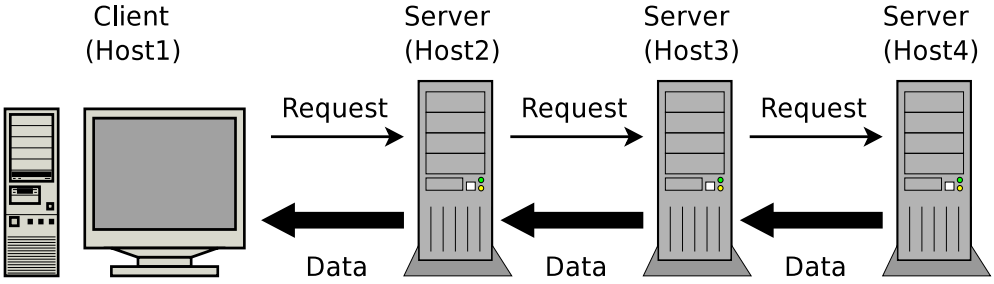

図 3.4 クライアント–サーバ連鎖型システム.

表 3.1 クライアント–サーバ連鎖型システムの動作環境 (全ホスト共通).

CPU	PowerPC G4 1.42GHz
メモリ	512MB DDR SDRAM 333MHz
NIC	100BASE-TX Onboard
OS	MAC OS X v10.5 (Leopard)

このシステムではトラフィックの生成に Java によるトラフィックジェネレータを利用している. クライアント上でこのトラフィックジェネレータを動作させ, フロントエンドサーバ (Host2) にリクエストを送信する. Host2 は受け取ったデータを Host3 へ渡し, Host3 も同様に Host4 へと転送する. Host4 では受け取ったリクエストを確認し, 指定されたサイズのランダム文字列を生成する. そしてそのデータを Host3, Host2 と転送していき, クライアントへ返却する. この一連のメッセージフローを図 3.5 に示す. この実験では, 3.3.4 で示した拡張 MVA モデルを構築するために必要となる性能評価の要素範囲を各メッセージに対する処理として定義した. この定義に基づいて算出するサービス時間も, メッセージフロー図 3.5 に示す.

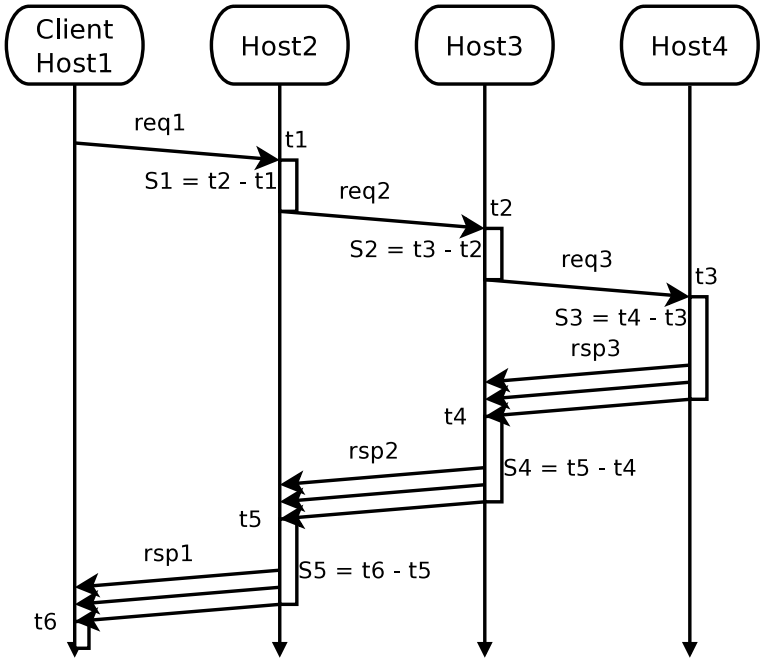

図 3.5 トランザクションメッセージフロー : クライアント–サーバ連鎖型システム.

このシステム内では, 各ホストにおいてデータの受信時刻と送信時刻を記録し, その値を送信するデータに付加していく. つまり, 最終的にクライアントには, 各ホストでのデータ受信時刻・送信時刻が負荷されたメッセージが戻ってくる. この値を用いてサービス時間の測定を行う.

3.4.1.1 zero-load トランザクションでのサービス時間測定

図 3.6, 3.7, 3.8, 3.9 にリクエストするデータサイズとサービス時間の関係を示す. また, 拡張 MVA モデルの構築に用いる近似曲線も同時に示す. なお, 図 3.7, 3.8, 3.9 は図 3.6 を見やすくするために分割して表示したものである. サービス時間は, リ

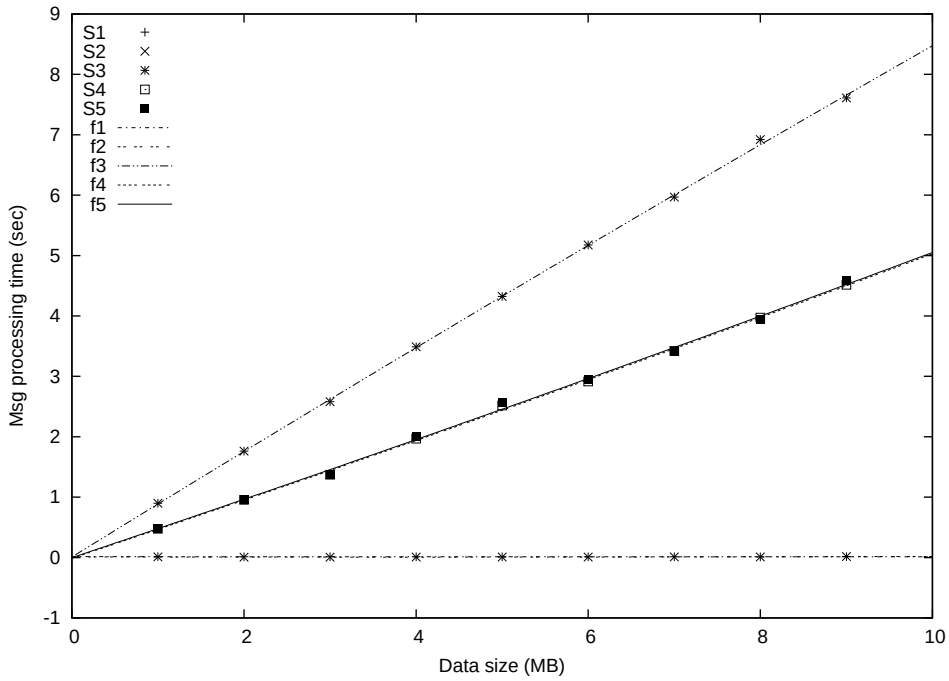

図 3.6 サービス時間：クライアント–サーバ連鎖型システム.

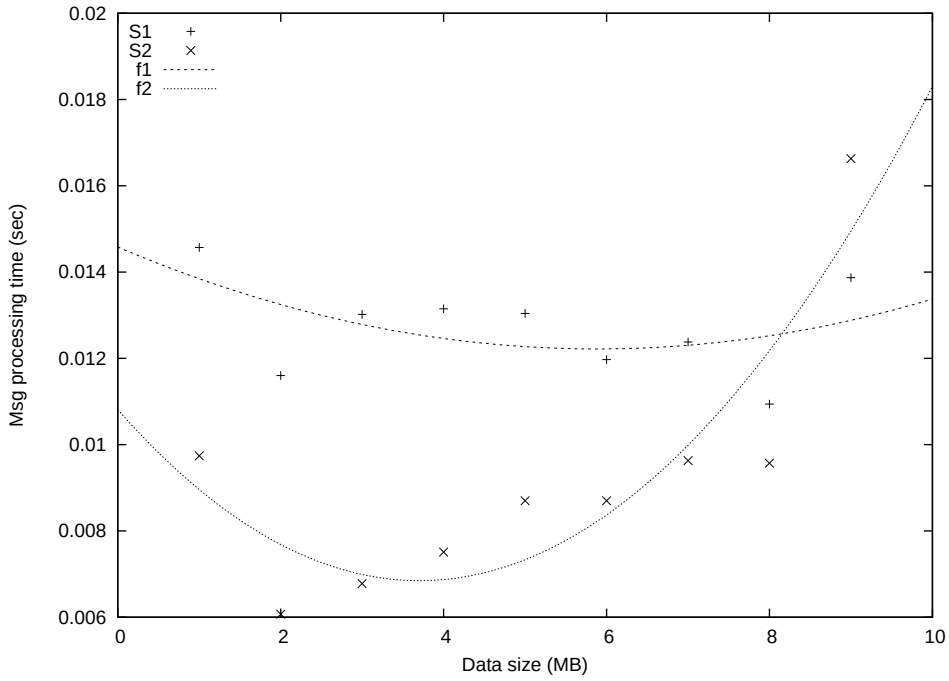

図 3.7 サービス時間 (s_1, s_2)：クライアント–サーバ連鎖型システム.

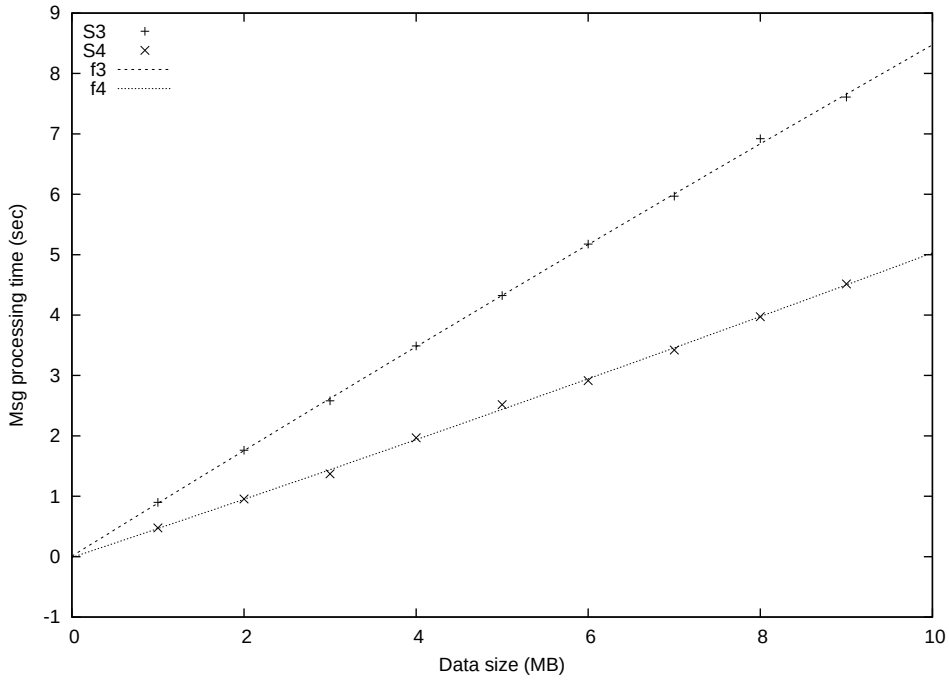

図 3.8 サービス時間 (s_3, s_4) : クライアント–サーバ連鎖型システム.

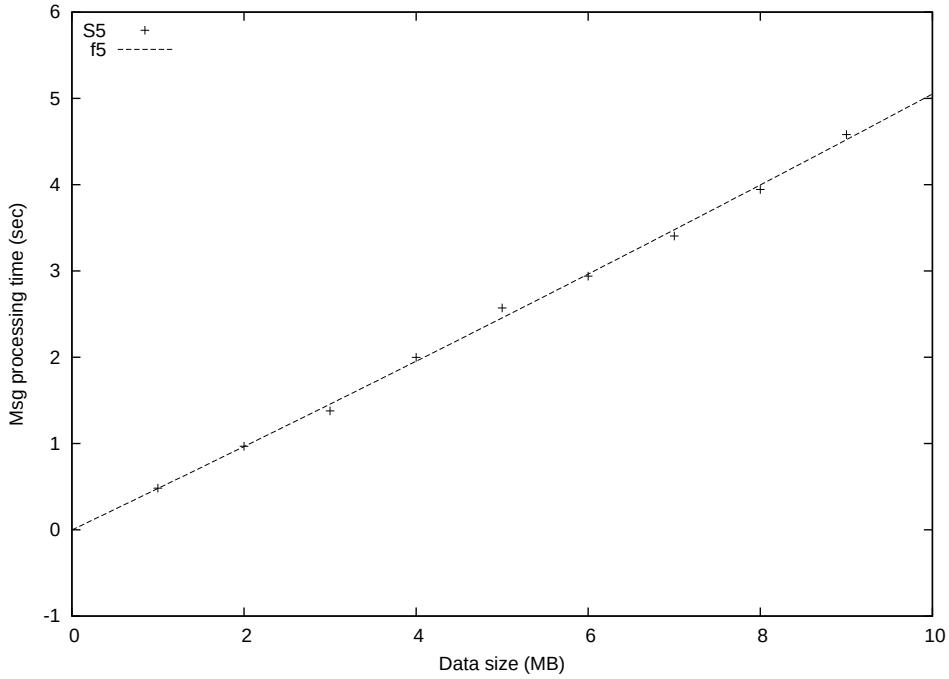

図 3.9 サービス時間 (s_5) : クライアント–サーバ連鎖型システム.

クエストするデータサイズを 1MB から 9MB まで 1MB 刻みに変化させて測定した. サービス時間 s_1 と s_2 はリクエストされたデータを転送するだけの処理に要する時間であり, 非常に小さな値であるため, 誤差範囲が大きく充分な近似が得られていないが, 他のサービス時間と比較して, 値の小ささから全体に与える影響はあまりないと考えられる. 一方でサービス時間 s_3, s_4, s_5 は, 図 3.8, 3.9 を見てわかるとおり, リクエストされるデータサイズに対して線形に増加している.

3.4.1.2 非 zero-load 時の応答時間測定

図 3.10 に非 zero-load 時の応答時間の測定値と, 拡張 MVA モデルによる予測を示す. ここでは, リクエストをするデータサイズを 4.5MB, 9MB, 10MB に変化させて測定した. 4.5MB, 10MB でのサービス時間はモデル作成時には測定していないが, 近似関数を用いることで予測ができている. 拡張 MVA モデルの予測値は, 既存 MVA モデルが持つ最悪値を保証するという特性をそのまま保持している. そのため, 実際の応答時間の平均測定値と比較して拡張 MVA モデルによる予測はより大きな応答時間を示している.

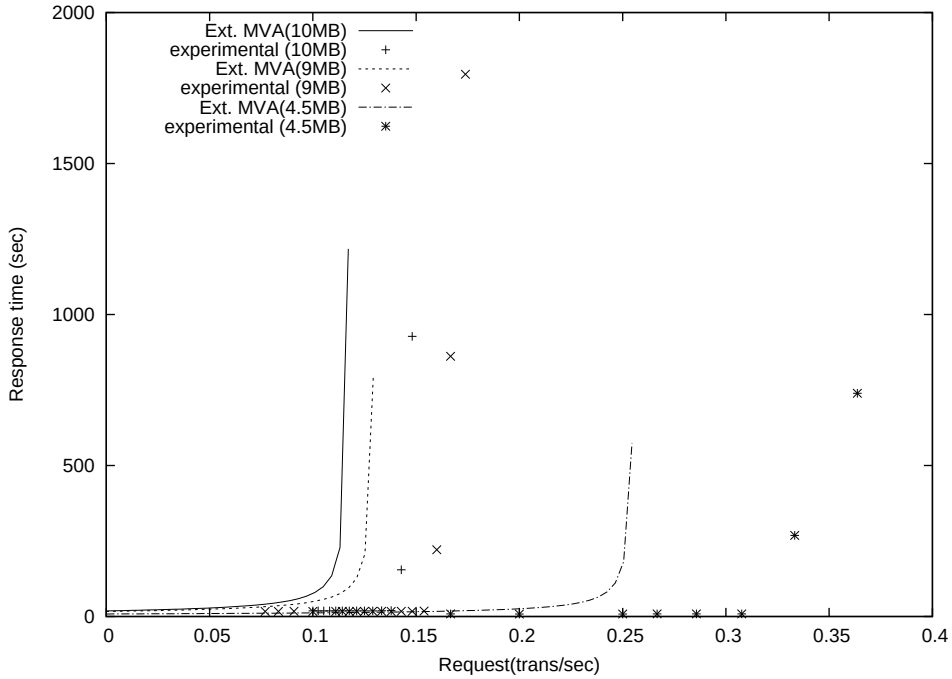

図 3.10 システム応答時間の予測値と実測値 : クライアント–サーバ連鎖型システム.

3.4.2 株価取得システム

株価取得システムは, クライアント, プロキシサーバ, Web サーバ, データベースサーバで構成されるシステムである. システムの構成図を図 3.11 に示し, 動作環境を図 3.12 及び表 3.2 に示す. このシステムは Google Web Toolkit 2.0.3[32] を用いた AJAX アプリケーションとして作成しており, プロキシサーバには delegated 9.9.7[17] を, Web サーバには GlassFish v3 (74.3)[18] を, データベースサーバには Apache Derby Network Server 10.5.3.0[19] を利用している. プロキシサーバを利用した理由は, より実際的なシステムとして構成するためであり, 今回のプロキシでは通信の圧縮のみを行っている.

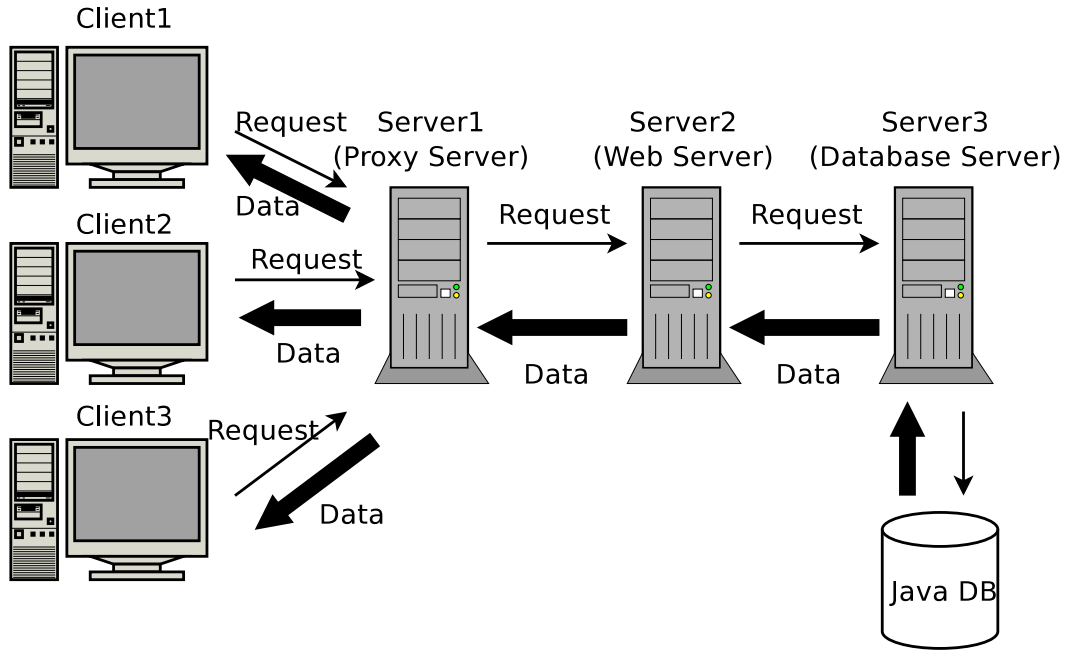

図 3.11 株価取得システム.

表 3.2 株価取得システムの動作環境.

	クライアント, サーバ	スニファ
CPU	Core^{TM}2 Quad Q9650	Core^{TM}2 Duo E8400
メモリ	4GB DDR2 SDRAM	4GB DDR2 SDRAM
NIC	1000BASE-T (Intel 82541PI)	1000BASE-T (Intel 82571EB) 4 Port ×2
OS	Fedora 13 x86_64	Fedora 13 x86_64

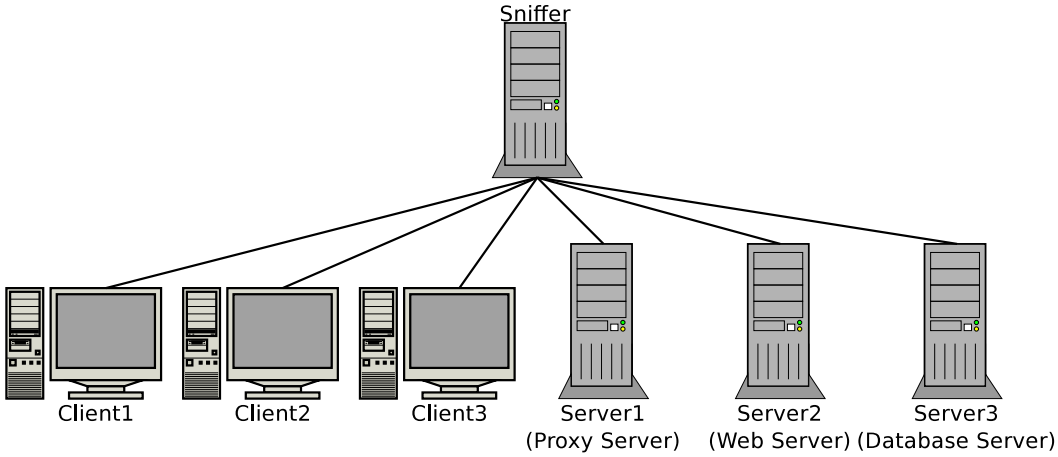

図 3.12 株価取得システムの動作ハードウェア構成.

このシステムではまず, ユーザがクライアント上の Web ブラウザを用い, 変動を監視したい株価を登録する. そしてブラウザから更新リクエストを送信すると, プロキシサーバを経由して Web サーバへとリクエストが届く. Web サーバは受け取ったリクエストに基づき, データベースサーバへ最新の株価を問い合わせ, その結果をまたプロキシサーバを経由してクライアントへ戻す. このシステムのリクエスト/応答のトランザクションフローを図 3.13 に示し, 同時に図中にサービス時間の定義を記す. 全てのホストはスニファを経由して接続されており, システムでの一連のトラフィックを tcpdump[14] 及び wireshark[15] を用いて監視し, それを解析して得た各ホストでのメッセージの到着時間を用いてサービス時間を求めている. 3.4.1 ではデータサイズを直接指定してサービス時間の測定を行ったが, このシステムでは取得する株価の個数を変化させることで, 様々なデータサイズのメッセージを作成し, サービス時間の測定を行った.

本システム上での実験において, トランザクションは, クライアントのブラウザ上で JavaScript を用いて発生させている. zero-load のトランザクションは, 一つのトランザクション発生リクエストを投入し, そのレスポンスが戻ったのを確認してから次のリクエストを投入することで実現している. 非 zero-load 時においては, トランザクションの投入間隔を指定し, 指定された間隔ごとにリクエストの発生と送信を行っている. 高負荷を加える場合はこの間隔を短くするわけであるが, クライアント一台では性能が追いつかず投入間隔を守ることができなくなるため, クライアントを三台同時に利用して分散処理することでこれを回避している.

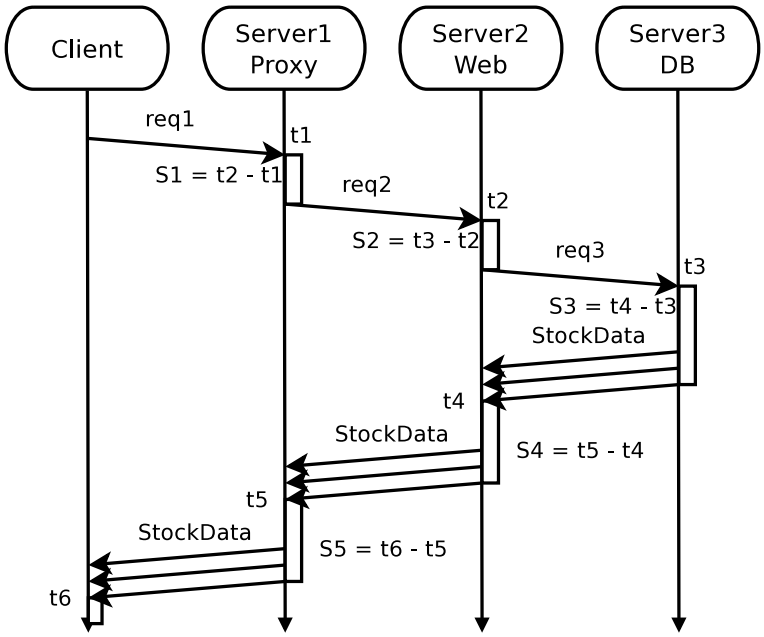

図 3.13 トランザクションメッセージフロー：株価取得システム.

3.4.2.1 zero-load トランザクションでのサービス時間測定

サービス時間のデータサイズとの依存関係を確認するために, 取得する株価のエントリ数を 100, 300, 500, 1000, 2000, 3000 と変化させて zero-load トランザクションを発生させ, サービス時間の測定を行った. なお, データベースサーバから返されるデータのサイズは, 1 エントリあたりおよそ 1KB である. 図 3.14 に株価エントリ数 (データサイズ) ごとのサービス時間の測定値と, 二次曲線による近似関数 $f_i(l_i)$ を示す. この図より, 近似関数 $f_i(l_i)$ がそれぞれの関係を正しく示せていることがわかる.

また, このグラフから, サービス時間 s_3 は株価エントリ数の二次関数になっていることが読み取れる. s_3 内の処理はデータベースへ問い合わせるための SQL 文の発行と, その SQL 文の通信である. SQL の発行では, 問い合わせる株価エントリを where 節に追加していく処理を行っている. この処理では, Java の String 型を用い, 問い合わせるエントリのキーを+演算子で連結することで生成している. コードを示すと

```
String whereClause = "";
for (String key : updateRequestKeys){
    if(whereClause.length()==0){
        whereClause = "where Name = '" + s + "'";
    }else{
        whereClause = whereClause + " or Name = '" + s + "'";
    }
```

```
}
String sqlStatement = "SELECT * FROM StockData " + whereClause;
```

という処理で行っており, Java では String の+演算子による連結は文字列の長さに線形に依存するため, 株価エントリ問い合わせ用の SQL 文の発行に要する時間が二次関数となるのだと考えられる.

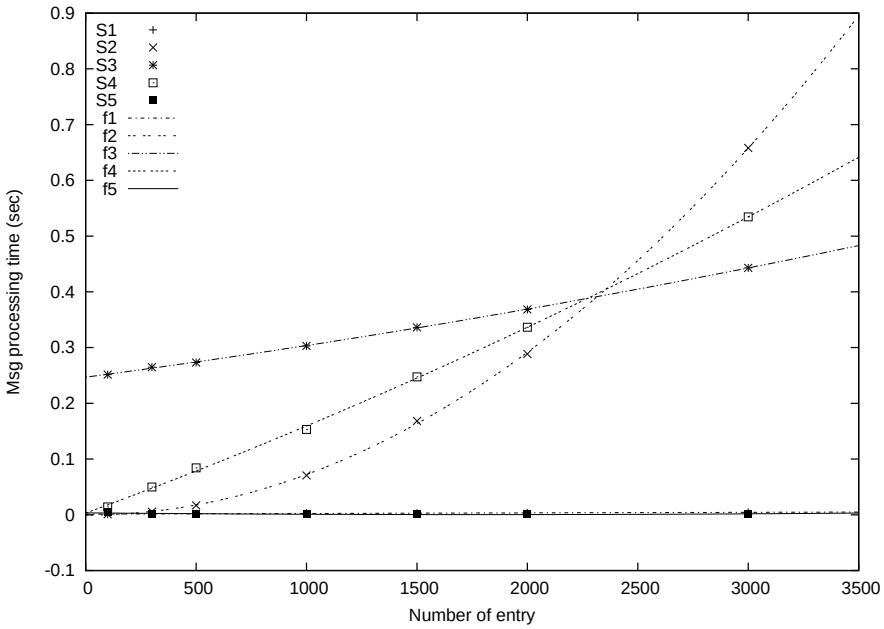

図 3.14 サービス時間：株価取得システム.

3.4.2.2 非 zero-load 時の応答時間測定

非 zero-load 時の応答時間の測定では, 取得する株価エントリ数を 1000, 2500 に設定してトランザクションを発生させ, トランザクションの発生間隔を変化させて平均実行時間を測定し, その値と拡張 MVA による予測値の比較を行った. 結果を図 3.15 に示す. 拡張 MVA モデルの生成時には, 図 3.14 に示したサービス時間の近似関数 f_i を用いている. この図より, 拡張 MVA モデルはシステムのキャパシティが急に悪化する点を良好に予測できていると言える. 取得する株価エントリ数が 2500 の場合のサービス時間は zero-load 時に測定していない. しかし, 近似関数を用いて拡張 MVA モデルを生成したことで, 測定していない株価エントリ数であっても応答時間が予測できていることが確認できる.

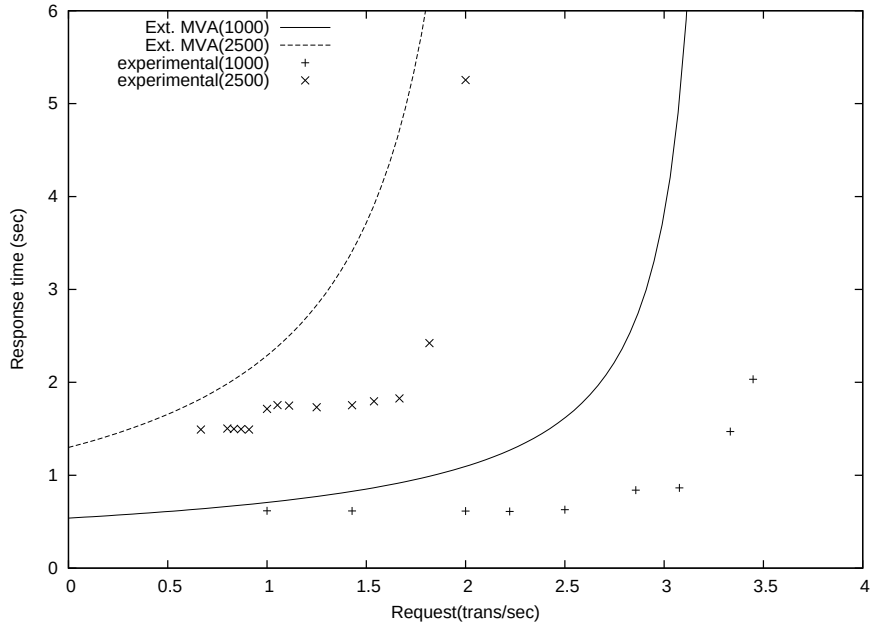

図 3.15 システム応答時間の予測値と実測値：株価取得システム.

3.5 関連研究と本研究の位置付け

MVA を用いた評価を本研究と重なる分野へと適応した例としては, S. Elnikety らによる研究 [33] がある. この研究では, データベースシステムを閉鎖型ネットワークを用いた MVA モデルとして表現し, スケーラブルなデータベースシステムの応答時間とスループットを予測している. この研究ではトランザクション破棄率を考慮に入れているが, 通信データサイズは考慮していない. MVA のマルチサーバシステムへの適用例としては, J. W. J. Xue らによる研究 [35] が存在する. この研究では, クラスタ上でのマルチティアなインターネットサービスの実行を, 閉鎖型ネットワークを用いた MVA モデルとして表現し, それを利用して収入を最大にするスケジューリングアルゴリズムを考案している. また, B. Urgaonkar らによる研究 [34] もマルチサーバシステムへの適用例である. この研究では, 電子商取引を行う Web アプリケーションを閉鎖型ネットワークの MVA モデルとして表し, 訪問者の割合, サービス時間, 及び利用者が考える時間をパラメータとして用いた性能評価を行っている. ここに挙げた全ての研究では, 著者らは低負荷時の応答時間の予測について提言しているのに対し, 本研究はどのような負荷の状況にも対応し, 様々なマルチサーバシステムに適用可能な手法を提案している.

3.6 まとめ

本章では, 既存の MVA を基にしたマルチサーバシステムの性能評価手法を, トランザクション内で発生するデータ転送においてデータ量が変動した場合にも対応可能にする拡張を提案した. 一連のトランザクション内において, サーバ間でのファイル転送やデータベースへのアクセスなどのデータ転送が発生することは一般的であり, 要求するファイルやデータによってそのサイズは変動する. 本章で提案した拡張を既存の MVA モデルに適用することにより, このような様々なデータサイズの通信を含むトランザクションに対応し, 平均応答時間やキャパシティなどの性能評価を行うことが可能となるため, 本手法は非常に有用であると言える. また, 近似関数を用いたことで, サーバ間でやり取りされる全てのデータサイズの通信について測定せずともモデルの構築が可能となり, 性能評価のために必要となるコストの増加を抑え, 容易に利用可能なモデルを提案できた. MVA モデルによる性能評価は様々な粒度へと対応可能であるため, 必要に応じた範囲の性能を評価し, ボトルネックを把握することが可能であるし, そこからさらに詳細にモデル化することで問題箇所を絞り込むこともできる. その反対に, 詳細なモデルにより得られた性能が, より粒度の粗いモデルへと拡張した場合にどのような範囲に, どのくらいの影響を及ぼすのかを評価することもできるため, 様々な評価の目的に対して汎用的に利用することが可能である.

提案手法により, データ量に依存するアプリケーションの性能評価を正しく行うことができ, ボトルネック部分の性能を向上させることによる動作速度の向上や, 一定の性能を保証した動作など, 分散アプリケーションの効率的な実行が可能になる. 今回は比較的単純なシステムを用いた実験で拡張 MVA モデルによる性能評価手法の有用性を示したが, 今後はさらに複雑なシステムを用いての評価を行う. また, プロセッサのコア数など他のパラメータもモデルに組み込むことで, 更なる予測精度の向上, 汎用性の向上を目指す.

第4章　結論

本論文では, 通信を多く含み, その処理がデータ量に依存する並列分散アプリケーションについて, データ通信時間とデータ処理時間に着目して実行を効率化することを目的とした研究について述べた.

第 2 章ではデータ通信時間に着目し, それを短縮することで実行を効率化する手法として, 通信を多く含み処理がデータ量に依存する並列分散アプリケーションの例としてストリーミング型分散アプリケーションを取り上げ, 効率的な実行を実現する新たなタスクスケジューリング手法を提案し, 実験を行った.

提案手法では, 複数の経路を選択利用可能な環境下において通信の特性に適したネットワーク経路を利用することで, 並列分散アプリケーションの実行の効率化を図った. ストリーミング型分散アプリケーションは, タスク間で長時間に渡りタスク間の通信を行うという特徴があるため, 通信にはできる限りバンド幅の大きなネットワークを利用する. 一方, 非ストリーミング型分散アプリケーションは通信するデータ量が少ないため, 遅延の小さなネットワークを利用するようにタスクを配置する. この手法により, ストリーミング型, 非ストリーミング型分散アプリケーションの双方を効率的に実行できることを確認した.

また, ストリーミング型分散アプリケーションは長時間に渡りタスク間で通信を行うため, 外的要因によって利用しているネットワークの性能が低下した場合の対策を考える必要がある. そこで提案手法では, 常に高性能なネットワーク経路を利用するために, 通信に利用しているネットワークの性能が低下したことを検知し, タスクを別のワーカへ移動し経路を変更する, タスクマイグレーション機構を実装した. 実験の結果, タスクマイグレーション機構を実装することで, ネットワーク上に動的な負荷が存在する環境上においてもネットワークの混雑を回避し, 実行時間を約 30%短縮し, さらにそのばらつきを抑えられることを確認した.

ばらつきを抑えたことで, そのアプリケーションの実行終了時間の予測, 言い換えれば並列分散コンピューティング環境の資源利用を終了する時間を予測する精度が向上する. その結果, 他アプリケーションの実行の際に適切かつ正確な資源割り当てをすることが可能となり, 並列分散アプリケーションの実行の更なる効率化が図れる.

第 3 章では, 分散アプリケーションの実行に適したシステムを構築するために必要となる, システム性能評価手法の提案を行った.

並列分散アプリケーション内において Web サーバやデータベースサーバなどを利

用するタスクが存在する場合, その処理を行うべきワーカが一意に決定する. この時, アプリケーションの実行を効率化するためには, その並列分散アプリケーションに適したシステムを構築し, システムの性能を向上させなければならない. その際に効果的な性能改善を得るためには, 各部分の性能を正しく把握し, ボトルネックとなる部分の性能を向上させなければならず, システムの性能評価手法が必要となる. そこで, 実際のシステムに高負荷をかけることなく, 容易かつ正確な性能評価を行うことができる手法である, MVA を用いたマルチサーバシステムの評価手法を, 処理がデータ量に依存するアプリケーションに適用するための拡張を行った.

既存の MVA 手法では, 通信データ量により変化する分散アプリケーションのデータ通信時間とデータ処理時間を考慮していない. そこで提案手法では, 既存 MVA 手法のモデルにおけるサービス時間を通信データ量に依存する値として定義することで, この問題の解決を図った. しかし, サービス時間をデータ量に依存する離散値として定義すると, サーバ上へ投入されうるトランザクションにおける全てのデータサイズについてそのサービス時間を測定するという作業が必要となるため, 既存 MVA 手法のもつ構築の容易さを損ってしまい実用的ではない. そこで, サービス時間を通信データ量の関数として定義した. その結果, 幾つかのデータサイズについてサービス時間を測定すればサービス時間の近似関数を求めることが可能となり, 性能評価に必要となる MVA モデルの構築コストの増加を抑えた実用的な手法が考案できた.

実験では二種類のマルチサーバシステムを用い, 提案手法を用いることで, 処理がデータ量に依存するアプリケーションについてデータ量に応じた応答時間とキャパシティを予測できることを示した. また, 近似関数を用いたことで, 既存の MVA 手法の持つ容易さを損なわずに性能評価が行えることを確認した. 更に, この性能評価手法を用いることにより, システムのボトルネックを正しく把握しその部分の性能を向上させることで, 分散アプリケーションを効率的に実行できることも確認した. 例えば, 3.4.2 で述べた株価取得システムでは, 取得する株価エントリの数が 2300 個程度までであればデータベースサーバでの処理時間 s_3 が, それより大きければ SQL クエリ発行処理を行う Web サーバでの処理時間 s_2 が, それぞれボトルネックとなっていることが確認できる. そのため, 取得エントリ数が非常に多いトランザクションの実行を効率化したい場合には, Web サーバの性能を向上させることが最も効果的であり, 取得エントリ数の少ないトランザクションを大量に投入する場合は, データベースサーバの性能を向上させることで劇的な性能向上が得られることがわかる.

これら本研究の成果から, 通信を多く含みその処理がデータ量に依存する並列分散アプリケーションを実行する際に, アプリケーションに適したシステムの設計と, そのシステムの資源を効率的に利用する資源割り当てが行えるため, アプリケーションの効率的な実行が可能となる.

謝辞

本研究を進めるにあたり, 多くの有益な御教示・御助言を賜りました, 九州工業大学情報工学研究院 江島 俊朗教授, 吉田 隆一教授, 安永 卓生教授, 小出 洋准教授に深く御礼を申し上げます.

特に指導教官である小出先生には, 学部時代から通算して 8 年という長期間に渡りお世話になりました. 研究の進め方で悩んでいた時にも, 研究に行き詰まり挫折しかけていた時にも, 絶えず熱心に御指導頂きました. 多くのご迷惑をおかけしましたが, こうして博士論文として研究を纏めることができました. 心より感謝いたします. どうもありがとうございました.

また, 九州工業大学情報工学研究院の光来 健一准教授, 近藤 秀樹助教には, ゼミ等を通じて研究に対し多角的な視点から御意見を頂きました. おかげさまで研究の視野を広げることができました. ここに深謝いたします.

九州工業大学ネットワークデザイン研究センター (NDRC) の Dirceu Cavendish 特任教授には, MVA の研究を進めるにあたり, 基礎から御教授頂きました. 時差もあり御多忙の中, メール・インスタントメッセンジャーでのやり取りが主でしたが, 多くの的確な御助言を頂きました. ここに感謝いたします.

また, 尾家 祐二教授をはじめ, NDRC 教員の方々には, 様々な機会でお世話になり, 多くの御助言を頂きました. 厚く御礼を申し上げます.

小出研究室および共同研究室の光来研究室の学生の方々に感謝します. 情報工学部に小出先生が赴任された年に研究室に配属され, これまで 8 年間の間, 多くの先輩・同輩・後輩と共に過ごしてきました. 研究に関する意見交換はもちろんですが, みんなのおかげで私生活においても充実した日々を送ることができました. どうもありがとうございました.

最後になりましたが, 長きに渡る学生生活を精神的にも経済的にも支えてくれた両親に, この場を借りて感謝を伝えたいと思います. 本当にありがとうございました.

参考文献

[1] K. Yoshinaga, S. Washizu, Y. Uratani, H. Koide, D. Cavendish and Y. Oie: Characterizing Transaction with Data Transfer on Multi-Server Systems, *Proceedings of the 3rd International Workshop on Information Network Design (WIND2010)*, (2010).

[2] K. Yoshinaga, Y. Uratani and H. Koide: Utilizing Multi-Networks Task Scheduler for Streaming Applications, *Proceedings of the Fourth International Workshop on Scheduling and Resource Management for Parallel and Distributed Systems (SRMPDS'08)*, pp. 25-30 (2008).

[3] I. Foster: What is the Grid? A Three Point Checklist, *GRIDtoday*, Vol. 1, No. 6, (2002).

[4] JPIX - テクニカル情報:トラヒック: `http://www.jpix.ad.jp/jp/technical/traffic.html` (2010年10月19日確認).

[5] M. Gallet, L. Marchal and F. Vivien: Efficient Scheduling of Task Graph Collections on Heterogeneous Resources, *Proceedings of 23rd IEEE Parallel and Distributed Processing Symposium (IPDPS2009)*, pp. 1-11 (2009).

[6] L. Tomás, C. Carrión B. Caminero and A. Caminero: Exponential Smoothing for Network-aware Meta-scheduler in Advance in Grids, *Proceedings of the Sixth International Workshop on Scheduling and Resource Management for Parallel and Distributed Systems (SRMPDS'10)*, pp. 323-330 (2010).

[7] Y. K. Kwok and I. Ahmad: Static Scheduling Algorithms for Allocating Directed Task Graphs to Multiprocessors, *ACM Computing Surveys*, Vol. 31, No. 4, pp. 406-471 (1999).

[8] H. Koide, H. Hirayama, A. Murasugi, T. Hayashi and H. Kasahara: Meta-scheduling for a Cluster of Supercomputers, *Proceedings of International Conference on Supercomputers Workshop, Scheduling Algorithms for Parallel/Distributed Computing - From Theory to Practice* -, pp. 63-69 (1999).

[9] H. Yamamoto, K. Kawahara, T. Takine and Y. Oie: Investigation and Fundamental Analysis of Process Allocation Scheme on Grid Computing Environment, *Techical Report of IECE*, IN2002-8, pp. 43-48 (2002).

[10] 小出洋, 山岸信寛, 武宮博, 笠原博徳: 資源情報サーバにおける資源情報予測の評価, 情報処理学会論文誌 プログラミング, Vol. 42, No. SIG3(PRO 10), pp. 65-73 (2001).

[11] R. Wolski, N. Spring and J. Hayes: The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing, *Journal of Future Generation Computing Systems*, Vol. 15(5-6), pp. 757-768 (1999).

[12] Standard Task Graph Set: Kasahara Laboratory, Waseda University, http://www.kasahara.elec.waseda.ac.jp/schedule/ (2010 年 10 月 5 日確認).

[13] 樋田 博信, 池永 全志, 尾家 祐二: 複数経路表を用いたトラヒックエンジニアリング機構の実装と評価, 電子情報通信学会技術研究報告, NS2003-308, IN2003-263, pp. 65-70 (2004).

[14] TCPDUMP/LIBPCAP public repository, http://www.tcpdump.org/ (2010 年 10 月 5 日確認).

[15] Wireshark.org, http://www.wireshark.org/ (2010 年 10 月 5 日確認).

[16] Iperf (SourceForge.net), http://sourceforge.net/projects/iperf/ (2010 年 10 月 8 日確認).

[17] DeleGate Home Page, http://www.delegate.org/delegate/ (2010 年 10 月 8 日確認).

[18] GlassFish Project, https://glassfish.dev.java.net/ (2010 年 10 月 8 日確認).

[19] Apache Derby, http://db.apache.org/derby/ (2010 年 10 月 8 日確認).

[20] 柴田慎也, 小出洋: 資源情報サーバにおける類似予測手法の改良と評価, 情報処理学会九州支部 火の国シンポジウム 2007, A-6-1, CD-ROM, (2007).

[21] N. Vydyanathan, U. Catalyurek, T. Kurc, P. Sadayappan and J. Saltz: A Duplication Based Algorithm for Optimizing Latency Under Throughput Constraints for Streaming Workflows, *37th International Conference on Parallel Processing (ICPP 2008)*, pp. 254-261 (2008).

[22] Q. Zhu and G. Agrawal: Resource Allocation for Distributed Streaming Applications, *37th International Conference on Parallel Processing (ICPP 2008)*, pp. 414-421 (2008).

[23] M. Reiser and S. S. Lavenberg: Mean-Value Analysis of Closed Multichain Queuing Networks, *Journal of the ACM*, Vol. 27(2), pp. 313-322 (1980).

[24] J. Zahorjan and E. Wong: The solution of separable queueing network models using mean value analysis, *Proceedings of ACM SIGMETRICS Performance Evaluation Review*, Vol. 10(3), pp. 80-85 (1981).

[25] D. Cavendish, Y. Oie, H. Koide and M. Gerla: A Mean Value Analysis approach to transaction performance evaluation of multi-server systems, *Wiley Concurrency and Compution: Practice and Experience*, Vol. 22, No. 10, pp. 1267-1285 (2010).

[26] L. Kleinrock: *Queueing Systems Volume II: Computer Applications*, John Wiley & Sons, Inc., New York (1976).

[27] H-W. Ferng: Modeling of Split Traffic Under Probabilistic Routing, *IEEE Communications Letters*, Vol. 8, No. 7, pp. 470-472 (2004).

[28] Robert G. Gallager: *Discreate Stochastic Processes* Springer, Berlin (1997).

[29] DTrace, http://hub.opensolaris.org/bin/view/Community+Group+dtrace/WebHome# (2010 年 10 月 19 日確認).

[30] MSXML (MSDN), http://msdn.microsoft.com/en-us/library/ms763742.aspx (2010 年 10 月 19 日確認).

[31] Java^{TM}Platform, Standard Edition 6 API Secification http://download.oracle.com/javase/6/docs/api/ (2010 年 10 月 19 日確認).

[32] Google Web Toolkit, http://code.google.com/intl/ja/webtoolkit/doc/latest/tutorial/gettingstarted.html (2010 年 10 月 5 日確認).

[33] S. Elnikety, S. Dropsho, E. Cecchet and W. Zwaenepoel: Predicting Replicated Database Scalability from Standalone Database Profiling, *Proceedings of the 4th ACM European conference on Computer systems (Eurosys09)*, pp. 303-316 (2009).

[34] B. Urgaonkar, G. Pacifici, P. Shenoy, M. Spreitzer and A. Tantawi: Analytic Modeling of Multitier Internet Applications, *ACM Transactions on the Web*, Vol. 1, No. 1, Article 2 (2007).

[35] J. W. J. Xue, L. He and S. A. Jarvis: A Scheduling Algorithm for Revenue Maximisation for Cluster-based Internet Services, *Proceedings of the 3rd International Workshop on Scheduling and Resource Management for Parallel and Distributed Systems (SRMPDS'07)*, pp. 1-8 (2007).

著者発表論文

査読付き国際会議

1. K. Yoshinaga, S. Washizu, Y. Uratani, H. Koide, D. Cavendish and Y. Oie: Characterizing Transaction with Data Transfer on Multi-Server Systems, Proceedings of the 3rd International Workshop on Information Network Design (WIND2010), Thessaloniki, Greece (November 2010) Accepted.
(第 3 章の内容)

2. K. Yoshinaga, Y. Uratani and H. Koide: Utilizing Multi-Networks Task Scheduler for Streaming Applications, Proceedings of the Fourth International Workshop on Scheduling and Resource Management for Parallel and Distributed Systems (SRMPDS'08), pp. 25-30, Portland, Oregon, USA (September 2008).
(第 2 章の内容)

その他

1. 夏井宣匡, 吉永一美, 池永全志, 小出洋, 吉田香 : ユーザ-ネットワークインタラクションに基づくユビキタスネットワーク経路制御システムの開発, 電子情報通信学会技術研究報告, TM2007-60, Vol. 107, No. 545, pp. 43-48, 沖縄県石垣市, 2008 年 3 月.

2. 吉永一美, 小出洋 : 複数の特性の異なるネットワーク経路を持つ環境におけるストリーミングデータ処理のためのタスクスケジューリング, 情報処理学会研究報告, 2007-HPC-111, Vol. 2007, No. 80, pp. 49-54, 北海道旭川市, 2007 年 8 月.

3. 吉永一美, 小出洋 : ストリーミングデータ処理のためのタスクスケジューリング, 情報処理学会研究報告, 2006-HPC-103, Vol. 2006, No. 87, pp. 139-144, 高知県高知市, 2006 年 8 月.