ネットワーク資源を動的割当可能な
ジョブ管理システムに関する研究

2015年1月

渡場　康弘

ネットワーク資源を動的割当可能な
ジョブ管理システムに関する研究

提出先　　大阪大学大学院情報科学研究科

提出年月　2015年1月

渡場　康弘

研究業績

1. 学術論文誌発表論文

(1) Yasuhiro Watashiba, Susumu Date, Hirotake Abe, Yoshiyuki Kido, Kohei Ichikawa, Hiroaki Yamanaka, Eiji Kawai, Shinji Shimojo, and Haruo Takemura, “Efficacy Analysis of a SDN-enhanced Resource Management System through NAS Parallel Benchmarks”, The Review of Socionetwork Strategies, vol. 8, no. 2, pp. 69–84, December 2014.

(2) 渡場康弘, 木戸善之, 伊達進, 阿部洋丈, 市川昊平, 山中広明, 河合栄治, 竹村治雄, “計算資源とネットワーク資源を考慮した割当ポリシを配備可能とするジョブ管理フレームワーク”, 電子情報通信学会論文誌, vol. J97-D, no. 6, pp. 1082–1093, 2014 年 6 月.

2. 研究会等発表論文（査読付）

(1) Yasuhiro Watashiba, Susumu Date, Hirotake Abe, Yoshiyuki Kido, Kohei Ichikawa, Hiroaki Yamanaka, Eiji Kawai, Shinji Shimojo, and Haruo Takemura, “Performance Characteristics of an SDN-enhanced Job Management System for Cluster Systems with Fat-tree Interconnect”, Emerging Issues in Cloud (EIC) Workshop, The 6th IEEE International Conference on Cloud Computing Technology and Science (CloudCom 2014), pp. 781–786, December 2014.

(2) Yasuhiro Watashiba, Yoshiyuki Kido, Susumu Date, Hirotake Abe, Kohei Ichikawa, Hiroaki Yamanaka, Eiji Kawai and Haruo Takemura, “Prototyping and Evaluation of a Network-aware Job Management System on a Cluster System Leveraging OpenFlow”, The 19th IEEE International Conference on Networks (ICON 2013), pp. 1–6, December 2013. (DOI:10.1109/ICON.2013.6781934)

(3) Yasuhiro Watashiba, Susumu Date, Hirotake Abe, Kohei Ichikawa, Hiroaki

Yamanaka, Eiji Kawai, Haruo Takemura, “An Architectural Design of a Job Management System Leveraging Software Defined Network”, 2013 IEEE 37th Annual Computer Software and Applications Conference Workshops (COMPSACW 2013), pp. 724–729, July 2013. (DOI:10.1109/COMPSACW.2013.108)

3. その他の研究会等発表論文

(1) Yasuhiro Watashiba, Susumu Date, Hirotake Abe, Yoshiyuki Kido, Kohei Ichikawa, Hiroaki Yamanaka, Eiji Kawai and Haruo Takemura, “An Architecture of SDN-enhanced Job Management System Capable of Managing Virtual Computational Resources and QoS Control”, Pacific Rim Applications and Grid Middleware Assembly (PRAGMA) 27 Workshop, October 2014.

(2) Yasuhiro Watashiba, Yoshiyuki Kido, Susumu Date, Hirotake Abe, Kohei Ichikawa, Hiroaki Yamanaka, Eiji Kawai, and Haruo Takemura, “Development of QoS Control Framework on the SDN-based Job Management System”, Pacific Rim Applications and Grid Middleware Assembly (PRAGMA) 26 Workshop, April 2014.

(3) 渡場康弘, 木戸善之, 伊達進, 阿部洋丈, 市川昊平, 山中広明, 河合栄治, 竹村治雄, “OpenFlow 連携ジョブ管理システムの実装と評価”, 情報処理学会研究報告 vol. 2013-HPC-140 no. 11 (SWoPP 2013), pp. 1–6, 2013 年 8 月.

(4) 渡場康弘, 伊達進, 阿部洋丈, 市川昊平, 山中広明, 河合栄治, 竹村治雄, “SDN を用いたジョブ管理システムの提案”, 情報処理学会研究報告 vol. 2012-HPC-137 no. 33 (HOKKE-20), pp. 1–6, 2012 年 12 月.

(5) Yasuhiro Watashiba, Susumu Date, Hirotake Abe, Kohei Ichikawa, Hiroaki Yamanaka, Eiji Kawai, and Haruo Takemura, “A proposal of Network-aware Job Management System Leveraging SDN”, Pacific Rim Applications and Grid Middleware Assembly (PRAGMA) 23 Workshop, October 2012.

内容梗概

本論文は，筆者が 2012 年から現在までに，大阪大学大学院情報科学研究科情報システム工学専攻メディア統合環境講座在学中に行った研究の成果をまとめたものである．

高性能計算環境における資源管理は，個々の計算の実行性能および計算環境の効率的な運用に影響する要因の 1 つである．近年の高性能計算環境は，多数の計算ノードをインターコネクトと呼ばれるネットワークで相互接続したクラスタシステムとして構成されており，その規模は計算環境の性能向上を目指し大規模化する傾向にある．一般的に，高性能計算環境を必要とする計算要求は高い実行性能を得るため，できるだけ多くの計算ノードを同時に利用した分散並列計算として実行される．そのため，その計算に割り当てられた計算ノード間の通信性能は計算の実行性能に多大な影響を与える．また，高性能計算環境は，複数のユーザから出された多数の計算要求に対して資源割当が可能な範囲で複数の計算を同時に実行する共用環境として提供されるのが一般的である．そのため，個々の計算に対して高い実行性能を提供するためには，各計算における資源利用の状況を考慮した効率的な資源割当が必要である．

このような高性能計算環境において，ユーザの計算要求をジョブとして受け付け，それらに対して効率的に資源を割り当てるジョブ管理システム（Job Management System, JMS）が広く採用されている．ジョブ管理システムは高性能計算環境から適切な資源をジョブに対して提供するため，必要となる資源情報や利用状況を把握する機能を有する．しかし，今日利用可能なジョブ管理システムの多くは，Central Processing Unit（CPU）やメモリなどの計算資源のみ管理しており，計算ノード間のインターコネクトをネットワーク資源として管理するための機能を有していない．このことは，インターコネクトが通信要求を十分に満たす性能を常に有するよう構築されているとの仮定に起因すると考えられるが，大規模化傾向にある今日のクラスタシステムの状況から，そのようなインターコネクトの構築は困難になっていくと考える．以上のような背景から，本研究では，クラスタシステムにおけるインターコネクトに着目し，計算資源と同様にネットワーク資源を動的に管理するジョブ管理システムに関する研究開発を行う．

本論文では，まず従来のジョブ管理システムによる資源管理，およびクラスタシステム

において高いネットワーク性能を得るためのアプローチについて調査し，計算資源と同様にネットワーク資源であるインターコネクトを動的に管理・割当を行うための課題について分析を行う．分析結果より，本目的を実現するための技術課題として，(1) ネットワーク資源を動的に管理・制御する機能を備えたジョブ管理システムの実現可能性の検証，(2) 提案ジョブ管理システムへのネットワーク資源を制御する資源割当ポリシの配備による有用性・実用性の評価が必要であることを示す．

1 点目の課題であるネットワーク資源を動的に管理・制御する機能を備えたジョブ管理システムの実現可能性の検証に対しては，新しいネットワークアーキテクチャのコンセプトである Software-Defined Networking（SDN）に着目し，従来のジョブ管理システムに対して SDN を利用したネットワーク制御機能を拡張した SDN-enhanced JMS フレームワークを提案する．SDN-enhanced JMS フレームワークは，SDN の一実装である OpenFlow を統合した外部モジュールとして実装する．本モジュールと従来のジョブ管理システムを連携させることにより，計算資源とネットワーク資源をともに管理することが可能となる．Flat-tree インターコネクトを持つクラスタシステムにおいて，本システムと従来のジョブ管理システムに対して数種類のジョブセットを投入することにより，インターコネクトの利用状況を考慮した計算資源の割り当て，およびジョブの実行性能低下の抑制が可能であることを確認する．

2 点目の課題である提案ジョブ管理システムへのネットワーク資源を制御する資源割当ポリシの配備による有用性・実用性の評価に対して，実際的な冗長経路を有するクラスタシステムを対象とし，インターコネクトのトポロジ，各リンクの利用状況を考慮した資源割当ポリシを提案・実装し，インターコネクトの利用状況を考慮した計算資源およびネットワーク資源の効率的な割当がなされているかを検証する．具体的には，多くの高性能計算環境で採用されている Fat-tree インターコネクトを有するクラスタシステムをターゲットとした資源割当ポリシを提案し，SDN-enhanced JMS フレームワーク上で実際に実装・配備を行えることを確認する．その上で，提案する資源割当ポリシを組み込んだ SDN-enhanced JMS と従来のジョブ管理システムに対して，NAS Parallel Benchmarks などの数種のジョブセットを用いて，通信の衝突によるジョブの実行性能低下が抑制されることを確認する．これにより，提案手法の有用性・実用性を示す．

最後に，本研究の成果についてまとめ，今後の展望を述べて本論文を締めくくる．

目次

第 1 章

序論

1.1 研究背景

近年の高性能計算環境は，インターコネクト（interconnect，相互結合網）と呼ばれるネットワークで多数の計算ノードを相互接続したクラスタシステムとして構成される．その構成規模は性能向上を目的とし，ますます大規模化する傾向にある．実際，世界の高性能計算機システムの上位 500 までを年 2 回ランキングしている TOP500 Supercomputer Sites によれば，現在登録されている計算機システムの 85.8% がクラスタシステムであり [1]，数千台以上の計算ノードで構成されたクラスタシステムも登場しつつある．例えば，2014 年 11 月時点で世界第 1 位である中国人民解放軍国防科学技術大学の Tianhe-2（天河二号）は 16,000 計算ノード，日本第 1 位である理化学研究所計算科学研究機構の K computer（京）は 88,128 計算ノード構成である．

一方，クラスタシステムの大規模化に伴い，計算ノードのインターコネクトの大規模化・複雑化が進んでいる．一般的に，インターコネクトには，計算ノード間の高い通信性能を得るために，低遅延・広帯域なネットワークが求められる．これを実現するため，通信効率の良いネットワークトポロジに関する研究，専用デバイスを用いたインターコネクト技術など，様々な研究開発が活発に行われている．例えば，低遅延のインターコネクト技術として Myrinet [2, 3] や Infiniband [4] はその代表例としてあげることができる．しかし，今日のクラスタシステムの大規模化に伴い，大規模化・複雑化傾向にあるインターコネクト上で高い通信性能を得るためには，高性能なネットワークデバイスを駆使し，複雑なネットワークトポロジを構成していく必要がある．結果，その構築・運用管理コストは顕著な問題となりつつある．

大規模化するクラスタシステム上で高い計算性能を得るためには，できるだけ多くの計算ノードを同時に利用した分散並列計算を効率よく行うことが必要となる．一般的に，分散並列化による性能は，並列数の増加による計算時間の短縮と，分散並列化したことに

よって生じるオーバーヘッドで決定される．この分散並列化の性能向上を阻害する要因の中でも，とりわけ，計算ノード間の通信によるオーバーヘッドが分散並列計算に与える影響は大きい．また，非効率な通信により計算対象のデータの到着が遅れると，その間 CPU が待機状態になるため，時間とエネルギーを無駄に消費することになる．クラスタシステムでは，インターコネクトで接続された計算ノードを複数同時に利用することから，インターコネクトを高効率に利用して通信コストを最小限に抑えることが高性能計算を行う上で重要である．

また，高性能計算環境の運用は，複数のユーザからの計算要求に対して資源の割当を行い，複数のユーザの計算を同時に実行させるのが一般的である．その目的は高性能計算環境の利用効率を最大化することにある．その際，各計算要求に対してどのように計算環境の資源を配分するのかが，個々の計算の実行時間の短縮および計算環境全体の効率的な資源の配分を実現する上で重要となる．

今日，様々な科学研究分野において解くべき問題や計算対象のデータが大規模化・大容量化し，その結果，ますます高性能計算環境を必要とする計算が増加している．さらに，高性能計算環境に対する資源要求は，様々な研究分野の計算要請に基づき多様化傾向にある．例えば，社会科学分野におけるソーシャル・シミュレーションでは，より多くの Agent を利用した Multi-Agent シミュレーションを実行するため，CPU だけでなく Graphics Processing Unit（GPU）も含めたより多くの計算資源の利用手法が研究開発されている [5-12]．このような計算要求多様化の観点からも，高性能計算環境には，それを構成する多様な計算資源を管理し，高効率に配分する資源管理の重要性が急速に高まっている．多様な計算要求が増加している現状から，高性能計算環境における資源の管理・割当は，個々の計算の実行性能や利便性，および計算環境全体の効率的な運用に影響する重要な課題となっている．

今日では，高性能計算環境における資源管理システムとして，ユーザの計算要求をジョブとして受け付け，それらに対して効率的に資源を割り当てるジョブ管理システム（Job Management System, JMS）が広く採用されている．ジョブ管理システムは，主に高性能計算（High-Performance Computing，HPC）に代表される科学技術計算環境で利用されている資源管理システムであり，現在利用されている主要なジョブ管理システムとして，Network Queuing System（NQS）[13]，Portable Batch System（PBS）[14], Platform Load Sharing Facility（LSF）[15], Condor [16, 17]，Open Grid Scheduler/Grid Engine（OGS/GE）[18] をあげることができる．ジョブ管理システムは，ゲートウェイ，資源管理，利用統計の機能を有する．ゲートウェイはユーザが計算要求を出すための窓口であり，ユーザが高性能計算環境の構成や利用状況を意識せずに，実行したい計算と必要な資源量を計算環境に要求することができるインタフェースを提供する．資源管理機能は，高性能計算環境における資源の利用状況を把握し，ジョブに対して効率的に資源を割り当てる．利用統計

は，ユーザごとの資源利用履歴を管理し，計算環境の運用状況の把握やユーザへの課金のために使用される．

しかし，今日利用可能なジョブ管理システムは，各計算ノードの CPU やメモリなどの計算資源の利用状況とジョブに対するユーザの資源要求にのみ基づいて行う．つまり，分散並列計算の実行性能に影響を与えるクラスタシステムのネットワーク資源であるインターコネクトは，ジョブ管理システムでは資源として管理されておらず，インターコネクトをネットワーク資源として制御するための機能をジョブ管理システムは有していない．その理由の 1 つとして，ジョブにどのように計算ノードが割り当てられても，計算ノード間の通信は常に十分な性能が得られるようインターコネクトは設計・構築されているとの仮定に基づきジョブ管理システムが設計されていることに起因すると考えられる．しかし，今日の高性能計算環境の主流であるクラスタシステムは，構成する計算ノードの数が増加し大規模化する傾向にある．そのため，計算ノード間を接続するインターコネクトも大規模化する傾向にある．さらに，クラスタシステム上で行われる分散並列計算の並列数の増加や，個々の計算ノードの性能向上による処理の高速化や取り扱うデータサイズの大規模化などにより，インターコネクトはますます高い通信性能が求められつつある．しかし，上述の仮定を満たすインターコネクトを実現するためには，専用のネットワークハードウェアや多数のネットワーク機器を必要とし，また，構築・運用におけるコストを鑑みると，今後このような大規模なインターコネクトを構築していくことはますます困難になると考えられる．また，GPU に代表されるアクセラレータ資源や，クラウドサービスなどで提供されている仮想化資源のような新たな資源についても現在のジョブ管理システムでは制御することができない．計算資源だけに基づいて資源管理を行うジョブ管理システムでは，様々な資源を有する現在の高性能計算環境を，柔軟かつ効率的に管理することはできない．

1.2 本研究の目的

1.1 節で記したように，今日の高性能計算環境で高い計算性能を得るためには，ますます多様化するユーザからの計算要求に対し，その構成資源の利用状況を考慮し，構成資源を動的かつ効率的に割り当てることのできる柔軟な資源管理の仕組みが必要不可欠となりつつある．そのような視点から，本研究では，クラスタシステムを構成する資源のうち，特に，ネットワーク資源であるインターコネクトに着眼し，計算資源だけでなくネットワーク資源を動的な資源と捉え，ユーザの計算要求に対して適切な資源割当を行うことのできるジョブ管理システムの実現を目指す．本目的を実現するために，本研究では，まず，ネットワーク資源を動的に制御することによる資源管理手法のプロトタイプ化をはかり，本研究で構想するジョブ管理システムの実現可能性を検証する．その後，今日のクラ

スタシステムで採用される実際的なインターコネクトをもつクラスタシステムを対象とし，計算資源とネットワーク資源をともに扱う資源割当ポリシの提案・実装を行い，その性能評価を行うことで，その有用性，実用性を検証するアプローチをとる．これにより，ユーザからの多様な計算要求に基づき GPU や仮想計算機資源にも対応できる新しいジョブ管理システムの一形態を示すこともまた本研究の目的の一つである．

1.3 論文構成

本論文の構成は以下のとおりである．

2 章では，1.2 節で記した本研究の目的達成のための技術課題を抽出する．そのために，今日利用できる従来のジョブ管理システムによる資源管理手法，ならびに，クラスタシステム上で高い通信性能を得ることを目的とした関連研究について整理する．その際，特に，計算資源と同様にネットワーク資源であるインターコネクトを動的に管理・割当を行う手法，技術について着目する．その後，本研究で達成すべき 2 点の技術課題を明確にする．

3 章では，2 章で導出された 1 点目の課題に基づき，ネットワーク資源を考慮したジョブ管理システムフレームワークとして，近年急速に関心と着目が集まっている新しいネットワークアーキテクチャ概念である Software-Defined Networking（SDN）を利用した SDN-enhanced JMS フレームワークを提案する [19-23]．SDN では，対象となるネットワークの振る舞いをコントローラで中央集権的かつ動的に制御可能とする．本章では，この SDN の一実装である OpenFlow を応用して従来のジョブ管理システムにネットワークのプログラム機能を拡張することで，計算資源だけでなくネットワーク資源を動的な資源と捉え管理・制御するジョブ管理システムフレームワークのプロトタイプ開発を行う．これにより，本研究の提案の実現可能性を検証する．

4 章では，2 章で導出された 2 点目の課題に基づき，計算資源およびネットワーク資源の利用状況を考慮して計算資源およびインターコネクトの通信経路の割当を実現する資源割当ポリシを提案・実装する [24, 25]．通常，冗長経路を持つインターコネクトでの経路選択は各ネットワークデバイスが持つ機能により実現され，ジョブ管理システムの計算資源割当とは独立して行われる．この制御を SDN-enhanced JMS 上で実装し，計算資源および通信経路の割当を実現する資源割当ポリシについて Fat-tree トポロジインターコネクトを持つクラスタシステム上で提案・評価する．これにより，本研究で提案するジョブ管理システムが，実用的なインターコネクトを持つクラスタシステム上で計算資源およびネットワーク資源を効率的に管理することで，ジョブに高い実行性能を提供できることを確認し，その実用性を検証する．

5 章では，本研究にて得られた成果についてまとめ，今後の課題を述べる．

第 2 章

クラスタシステムにおける資源管理の現状と課題

2.1 はじめに

ユーザからの計算要求に応じた効率的な資源割当を行うことは，高性能計算環境の運用における主要課題の 1 つである．効率的な資源割当を実現するため，様々なサービスや計算環境のアーキテクチャを対象とした資源割当を実現する資源割当手法や資源管理システムに関する研究開発が行われている．その資源管理の対象である高性能計算環境の多くは大規模なクラスタシステムとして構築されている．クラスタシステムにおける資源は，各計算ノードが持つ CPU やメモリなどの計算資源と，計算ノード間を繋ぐインターコネクトにおけるネットワーク資源に大別される．また，近年のクラスタシステムによってはGPU などのアクセラレータ資源や，計算資源やネットワーク資源が仮想化されて提供される仮想化資源など，資源管理システムで扱うべき対象は多様化してきている．それゆえ，クラスタシステムを構成する資源をどのように管理し，どのように効率的にユーザの計算要求に割り当てるかが，資源管理システムを設計・開発する上で重要である．ジョブ管理システムは，HPC 分野の高性能計算環境において広く採用されている資源管理システムであり，計算処理の効率的な負荷分散や耐障害性を実現するために導入される．しかし，今日のジョブ管理システムの多くは，クラスタシステムの計算資源のみを対象とし，インターコネクトなどのネットワーク資源については考慮していない．ネットワーク資源を考慮しない資源割当では，今日のクラスタシステムにおける主な計算要求である分散並列計算に対し，得られる通信性能を考慮せずに割り当てる計算資源を決定するため，高い実行性能を得られる計算資源をジョブに提供できている保証は無い．

本章では，クラスタシステムの現状，一般的なジョブ管理システムにおける資源管理手法および技術についての整理を行うとともに，クラスタシステムにおける効率的なネット

ワーク利用に関する関連研究の調査を行う．その際，計算資源と同様にインターコネクトをネットワーク資源として管理・割当を行う手法および技術に着眼する．これにより，1 章で記した本研究の目的を実現するための技術課題を明確にする．

以下に本章の構成を示す．2.2 節では，クラスタシステムの現状について調査を行い，インターコネクトの観点から整理を行う．2.3 節では，一般的なジョブ管理システムにおける資源管理の仕組みについて確認し，その問題点を明らかにする．2.4 節では，クラスタシステムにおいて高いネットワーク性能を得るための関連研究について調査し，そのアプローチを整理する．2.4 節では，2.2 節，2.3 節，2.4 節を踏まえて，計算資源と同様にインターコネクトをネットワーク資源として管理・割当を行うための課題について分析を行い，本研究で解決すべき技術課題を明確にする．

2.2　クラスタシステムの現状

本節では，今日の高性能計算環境の主流であるクラスタシステムにおける資源管理の問題点を明確にするため，クラスタシステムが大規模化する背景およびインターコネクトの構成技術と現状について説明する．

2.2.1　クラスタシステムの構成

クラスタシステムは，多数の計算ノードをインターコネクトで接続し，全体を一つの計算環境とすることで高性能計算に対応した分散並列用計算機システムである．クラスタシステムの性能を表す際，理論ピーク性能と実効性能が用いられることが一般的である．理論ピーク性能は，ベンダや計算センターなどが提示する高性能計算環境の性能公称値として用いられる．クラスタシステムの場合，理論ピーク性能は（個々の計算ノードが持つ演算性能）×（計算ノード数）で表される．より高い性能を持つクラスタシステムを構築するとき，個々の計算ノードの演算性能自体は向上しているが，その向上分だけでシステムの目標性能を達成できる場合は少ない．そこで，計算ノード数を増加させることで性能向上を図るため，結果として高性能計算環境は大規模化していく．

一方，実効性能とは，分散並列計算を実行した際に得られる実質的な性能である．分散並列計算は通常，複数の処理への分け方や分割された処理間での通信に起因するオーバーヘッドが生じるため，理論ピーク性能値を実行性能値として得ることは極めて困難である．上述のオーバーヘッドは分散並列計算の処理内容に依存するため，実効性能は理論ピーク性能と異なり一意には表せない．そこで，高性能計算環境の性能を比較する場合には，特定のアプリケーションにおける実行効率（実効性能／理論ピーク性能）を算出する．例えば，TOP500 Supercomputer Sites でのランキングでは，連立一次方程式の解を求める

プログラムでシステムの浮動小数点演算性能を評価する LINPACK ベンチマーク [26-28] が用いられている [1]．高い実効性能を出せる高性能計算環境を構築するには，分散並列計算における並列化のオーバーヘッドの削減が必要不可欠である．特に，オーバーヘッドの要因の中でも，計算ノード間の通信コストは影響が大きいため，高い性能が得られるインターコネクトが高性能計算環境には求められる．

2.2.2 クラスタシステムのインターコネクト

一般的に，インターコネクトの性能指標には，遅延と帯域幅が用いられる．遅延とは送信したデータが宛先で受信されるまでにかかる時間であり，帯域幅とは単位時間あたりに転送できるデータ量である．高性能計算環境のインターコネクトでは，分散並列計算のオーバーヘッドを抑制するために低遅延・広帯域なネットワークが求められている．

インターコネクトの性能に関連する主要素は，大別するとネットワークデバイスと，ネットワークトポロジに分類できる．計算ノードが扱える帯域幅のピーク性能は，使用するネットワークデバイスによって決定される．高性能計算環境のインターコネクトで利用されている主なネットワークデバイスとして，一般的な Ethernet，Myrinet [2, 3]，Infiniband [4] があげられる．Ehternet によるインターコネクトは，その汎用性から高コストパフォーマンスかつ高拡張性を有するインターコネクトを構築できるが性能面で劣る．一方，Myrinet や Infiniband は低遅延な通信を提供できるネットワークデバイスであり，高性能なクラスタシステムのインターコネクトに適しているが，その導入コストは Ethernet によるインターコネクトより非常に高くなる．上述のネットワークデバイス以外にも，ベンダ固有の専用ハードウェアによるインターコネクトも提供されている．

本研究では，クラスタシステムのインターコネクトをネットワーク資源として動的に制御して計算要求に対して適切に資源割当を行うことで，クラスタシステムの資源を効率的に運用するジョブ管理システムの実現を目的としている．その効果は 1.1 節で述べたインターコネクトに求められる性能を実現できない環境において有効である．そこで，本論文では，非効率な資源割当により性能面に大きな影響が現れると考えられる Ethernet によるインターコネクトを対象とする．

一方，インターコネクトのトポロジは，通信性能だけでなく，インターコネクトの冗長性や拡張性にも影響する重要な要素である．インターコネクトのトポロジはその構成によって直接網と間接網に分類される．直接網のインターコネクトでは，各計算ノードが複数のネットワークインタフェースを持ち，それらを相互接続してリンクを張り，インターコネクトを形成する．直接網のインターコネクトにおけるルーティングは計算ノード上で行われる．代表的な直接網のトポロジとして，メッシュ（Mesh），トーラス（Torus），ハイパーキューブ（Hypercube）などがあげられる．一方，間接網では，計算ノードが持つ

ネットワークインタフェースは基本的に 1 つであり，計算ノードはスイッチを介することでインターコネクトを形成する．そのため，間接網におけるリンクは，計算ノードとスイッチ，およびスイッチ同士の間に形成される．なお，間接網でのルーティングは基本的にスイッチで制御される．間接網のトポロジの例として，ツリー（Flat-tree，Fat-tree）やクロスバ（Crossbar）などがあげられる．上述のネットワークデバイスとネットワークトポロジに対し，高性能計算環境を構成する計算ノード数，インターコネクへの要求性能，導入コストなどの条件から判断し，インターコネクトを構築する．

本研究では上述したトポロジのうち，間接網で構成されたインターコネクトを対象とする．

2.2.3　システム大規模化により生じる問題

クラスタシステムにおける計算ノード数の増加に伴い，ネットワーク通信性能の視点からインターコネクトはその構成を大規模化・複雑化せざるを得ない状況にある．大規模クラスタシステムでのインターコネクトでは，多くのリンクやスイッチを用いた複雑なトポロジ設計や新たなネットワークデバイスが必要となり，システム規模の大規模化に伴うインターコネクトの構築コストもますます大きくなりつつある．

例えば，8 万計算ノード以上で構成される理化学研究所の K computer では，多数の計算ノードを高速に接続するため，Torus fusion（Tofu）インターコネクトと呼ばれるベンダ固有のネットワークハードウェアを利用している [29]．Tofu インターコネクトでは，10 万計算ノードの拡張性を実現し，かつ，高性能・高信頼性・高可用性を備えるため，6 次元メッシュ/トーラスの直接網のトポロジ構成を，ネットワークインタフェースやルーティング制御のため専用に開発したインターコネクトコントローラを各計算ノードに採用することで実現している．

また，クラスタシステムのインターコネクトを k-ary Fat-tree トポロジ [30] で構成した場合，計算ノードの増加に伴い多数のネットワークスイッチや機器間のリンクのためのケーブルが必要となる．図 2.1 に 3 層構造で $k = 4$ の場合のクラスタシステムの構成例を示す．3 層 k-ary Fat-tree トポロジでインターコネクトを構築する場合，k 個のポートを持つネットワークスイッチを $5k^2/4$ 台用いることにより，$k^3/4$ 台の計算ノードで構成されたクラスタシステムを構築できる．その際のインターコネクトにおけるネットワークリンクの数は $3k^3/4$ 本となる．図 2.1 の例のように，4-ary Fat-tree トポロジのインターコネクトであれば，20 台の 4 ポート・ネットワークスイッチと 48 本のリンクを用いて 16 台の計算ノードで構成されたクラスタシステムを構築できるが，32-ary Fat-tree トポロジでインターコネクトを構成した場合，8,192 台の計算ノードで構成されたクラスタシステムを構築するために，32 個のポートを持つネットワークスイッチを 1,280 台，計算ノードや

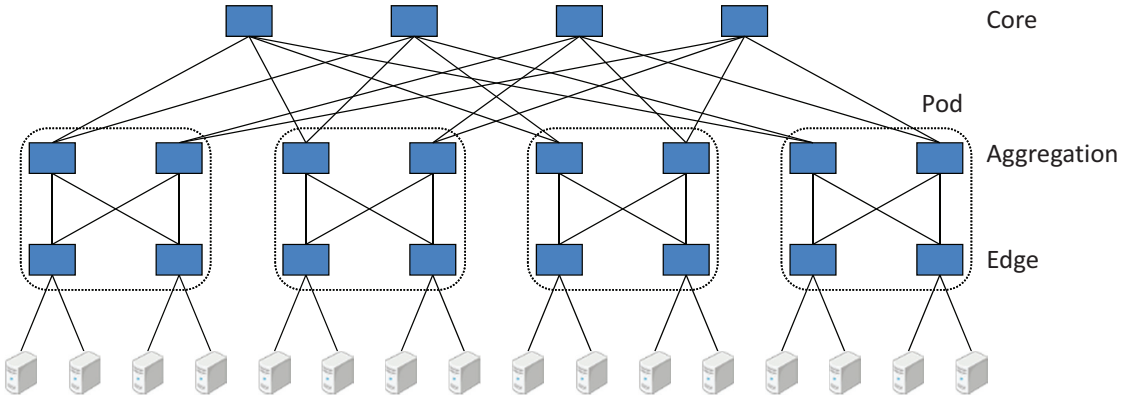


図 2.1　k-ary Fat-tree インターコネクト（k=4）.

スイッチ間のリンクを 24,576 本必要となり，インターコネクトは大規模化する．

上述の例のように専用ネットワークハードウェアや多数のネットワーク機器を採用してクラスタシステムを構築できる場合，規模に応じた性能を持つインターコネクトを構築することは可能である．しかし，一般的なクラスタシステムでは，上述のようなインターコネクトは構築および構築後の運用で多大なコストを必要とするため，その実現が困難になりつつある．

2.3　ジョブ管理システムにおける問題

高性能計算環境の運用において，ジョブ管理システムの資源管理機能は提供資源における実行性能および資源提供サービスの品質を決定する重要な課題である．特に，提供した資源量に基づき課金することで運用されている高性能計算環境では，サービス品質の低下によりユーザが離れることは深刻な問題となる．それゆえ，今日までにシステム構成やサービス形態に応じたさまざまな割当資源決定アルゴリズムやジョブ管理システムが数多く提案されてきた [31]．本節では，一般的なジョブ管理システムにおける構成および各機能について説明し，現在のジョブ管理システムにおける問題を分析する．

2.3.1　ジョブ管理システムの構成

ジョブ管理システムに対して求められる役割は，ユーザの計算要求を受け付けるゲートウェイとしてのインタフェース，さまざまな計算要求に対して高い性能を得られる資源を効率的に割り当てることができる資源管理，ユーザに提供してきた資源量を管理する利用統計があげられ，それぞれに対応した機能を持つ．図 2.2 に一般的なジョブ管理システム

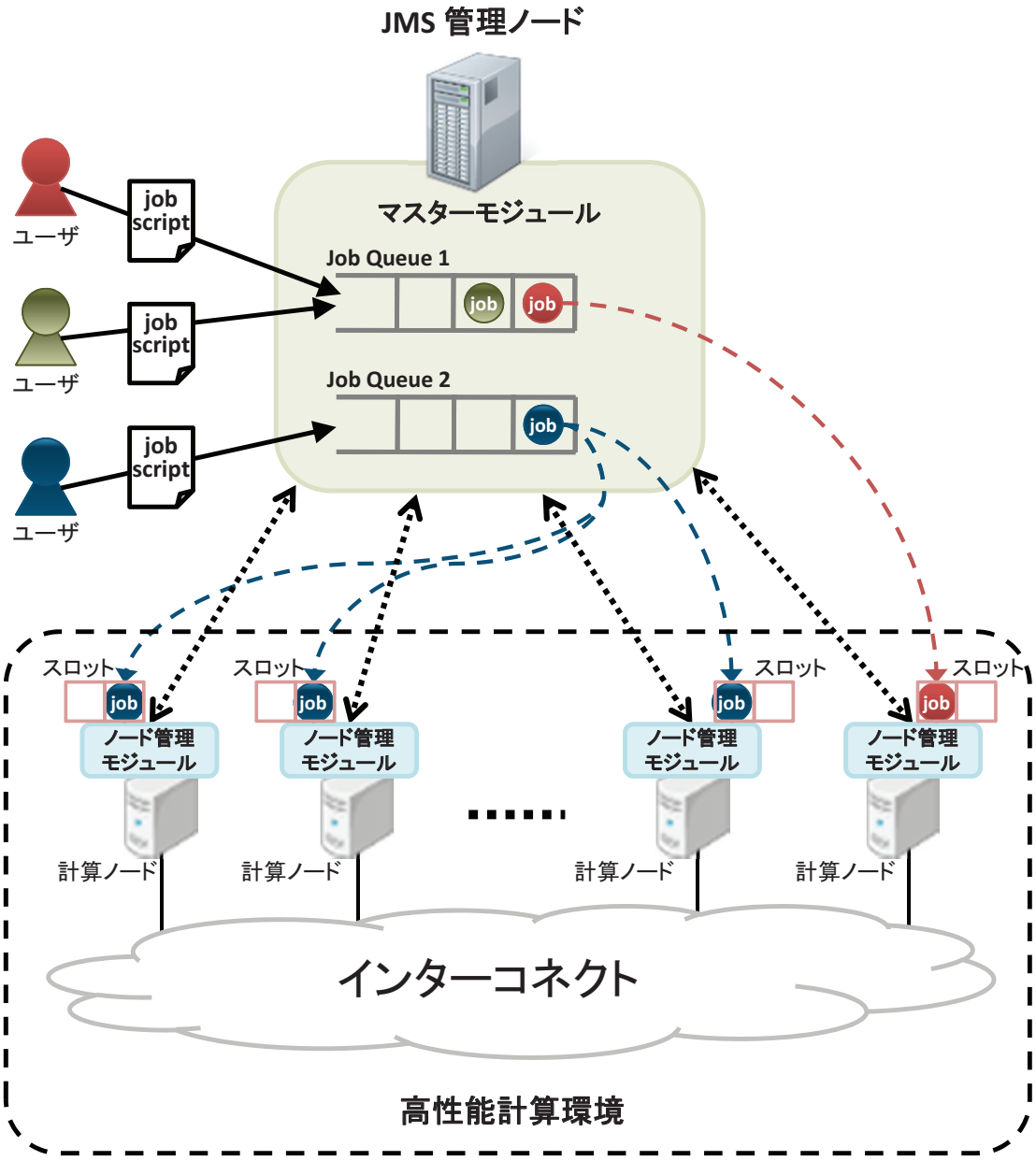


図 2.2　ジョブ管理システムの一般的な構成.

におけるシステム構成を示す．ジョブ管理システムでは，提供資源である高性能計算環境と，それを制御するための管理ノードから構成される．管理対象となる高性能計算環境は 1 つとは限らず，1 台の管理ノードで複数の計算環境を制御する場合もある．管理ノードのジョブ管理システムのマスターモジュール，および，各計算ノード上のノード管理モジュールが連携することで資源制御が実現される

2.3.1.1 ゲートウェイ機能

ユーザが高性能計算環境に対して計算要求を出すには，まず管理ノードのマスターモジュールに設定されたジョブキュー（Job Queue）を選択する必要がある．ジョブキューには資源量や計算実行可能時間の上限，ユーザ制限，実行優先度などが設定されている．ジョブキューは高性能計算環境の運用ポリシに従い管理者によって設計されており，複数のジョブキューが用意されるのが一般的である．各種設定を行ったジョブキューをユーザに提示することは，高性能計算環境におけるサービス一覧を示すことになる．この機能により，管理者の観点ではユーザの計算要求をその内容ごとに分類・整理することができ，ユーザの観点では高性能計算環境の構成などの詳細な情報を意識することなく，計算要求に適した資源提供サービスを選択することができる．

選択したジョブキューにジョブを投入するために，ユーザは利用するジョブキューや計算に必要な資源量を記述したジョブスクリプトを作成し，ジョブ管理システムのマスターモジュールに投入する．ユーザから投入されたジョブは指定されたジョブキューに格納され，資源割当処理が行われるのを待つ．なお，ジョブキューに設定された条件に反するジョブは拒否される．

一般的なジョブスクリプトの記述例として，OGS/GE [18] におけるジョブスクリプトの例を図 2.3 に示す．通常，ジョブスクリプトは，資源要求などの条件を記述する条件記述部（図 2.3 の例では#$ で始まる行）と，実行する計算について記述した実行記述部から構成される．ジョブスクリプトの条件記述部において，今日のジョブ管理システムの多くは CPU やメモリといった計算資源に対する要求を記述可能である．図 2.3 の例の場合，3 行目の “#$ -pe ompi 32” で 32 並列の分散並列計算を行うための CPU を要求しており，4 行目の “#$ -l h_vmem=8gb” では 1 並列あたりの使用メモリの上限を指定している．他の要求条件として，分散並列計算の環境設定，計算結果の出力形式，ジョブの実行開始時間，計算実行時の環境変数などが指定できる．実行記述部には計算を実行するためのコマンドを記述する．ジョブスクリプトは実質，一般的なシェルスクリプトであるため，実行記述部は柔軟な処理の記述が可能である．例えば，単純な連続計算，計算に対する前処理や後処理，計算結果に従った次の計算の選択や実行プログラムの生成，新たなジョブの生成などが可能である．

2.3.1.2 資源管理機能

ジョブキューに資源の割当を待つジョブに対し，ジョブの資源要求を満たし，かつ，システム全体の効率的利用の観点にも適した資源をジョブに割り当てることがジョブ管理システムの資源管理機能に求められる．計算環境の利用状況を考慮してジョブに資源を割り当てるためには，資源の利用状態を監視する必要がある．ジョブ管理システムの資源管理

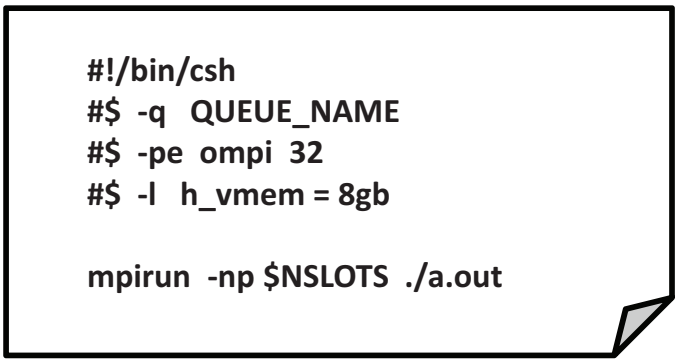


図 2.3　一般的なジョブ管理システムにおけるジョブスクリプトの記述例.

における役割は，高性能計算環境の各資源における利用状況の把握と，ジョブに対する割当資源の決定である．

高性能計算環境の資源における利用状況の把握は，管理ノードのマスターモジュールから各計算ノードのノード管理モジュールに定期的に計算資源の利用状況を問い合わせ，計算ノードのノード管理モジュールが持つ利用状況の情報を収集する．収集される情報として，各計算ノードの CPU ロードアベレージやメモリの空き容量などがある．集められた各計算ノードの利用状況情報は，割当資源決定処理で利用される．

ジョブへの資源割当処理は 2 つのフェーズに分けられる．まず，最初のフェーズでは，ジョブキューに投入されているジョブの中から，資源割当を実行するジョブを選択する．ジョブの選択は基本的にはジョブごとに算出された優先度の値に従い決定される．ジョブの優先度は，ユーザおよび投入されたジョブキューに設定された優先度，要求している資源量，ユーザのこれまでの利用資源量，ジョブキューに投入されてからの時間などのパラメータと，各パラメータに対する重みや優先度制御ポリシなどの管理者による設定に基づき，ジョブ管理システムが持つ計算式から算出される．優先度の順に並べ替えられたジョブに対し，優先度の高いジョブから順に資源割当処理を実行する．

次のフェーズでは，選択されたジョブに対して実際に資源割当を行う．割当資源の決定処理では，まず割当対象となる計算ノードで構成された割当候補リストに対し，ジョブの資源要求やジョブキューに設定された資源に対する条件を満たさないものを除外する．次に，収集した計算ノードの利用状況に基づき割当候補リストにある計算ノードに優先度を設定してソートする．多くのジョブ管理システムでは，優先度の設定には各計算ノードの CPU ロードアベレージが用いられる．最後に，ソートされた割当候補リストに基づき，ジョブに割り当てる計算ノードを決定する．なお，今日のマルチコア CPU を有する計算ノードのように，1 つの計算ノードにジョブプロセスを割り当てるための複数のスロットが設定されている場合には，管理者が設定したスロットの割当ポリシに従って計算ノード

を選択することになる．スロットへのジョブプロセスの割当に関する基本的なポリシとしては，選択した計算ノードのスロットをすべて消費して割り当てる方式と，割当候補リストの計算ノードに対して 1 スロットずつジョブプロセスを割り当てていく方式がある．

2.3.1.3　統計管理機能

ジョブ管理システムの利用統計機能は，資源割当を行ったジョブに関する情報を記録し，ユーザごとに管理する機能である．管理される情報としては一般的に，ジョブ ID，ジョブの投入・実行開始・実行終了の日時，割り当てた資源などがある．収集した情報は，高性能計算環境の運用におけるユーザへの課金や，ユーザに対する情報提供で使用される．

2.3.2　ジョブ管理システムによる資源管理の問題

2.2 節で述べたように，ネットワークの管理・制御の重要性が高まりつつあるにも関わらず，今日利用可能なジョブ管理システムでは，ネットワークなど他の資源を制御するための機能は従来のジョブ管理システムには備えられておらず，資源の利用状況の把握およびジョブへの資源割当など，資源制御に関する処理は計算資源に対してのみ行われる．ジョブ管理システムがネットワーク資源を管理対象としていない理由として，以下の 2 点が考えられる．1 点目の理由として，インターコネクトはハードウェア性能やネットワーク構成によって分散並列計算で生じる通信要求を常に許容できる性能を備えているとの仮定の下でジョブ管理システムが設計されていることが考えられる．2 点目の理由として，ネットワークは利用状況やユーザの要求に応じて動的に制御することができない静的な資源であるとの仮定に起因すると考える．しかし，システム構成が大規模化・分散化していく傾向にある高性能計算環境の現状を鑑みれば，前述の 2 つの仮定を実現した複数のユーザが同時にジョブを実行するのに十分なネットワーク性能を有するクラスタシステムの構築はますますコスト的に困難になっていくと考えられる．それゆえ，計算要求に適した資源を動的かつ効率的に配分することで高い実行性能を提供可能な新たなアプローチが必要であると考える．

2.4　分散並列計算のためのネットワークの効率的利用の関連研究

分散並列計算における通信性能による実効性能の低下は，高い実行性能を得るための重要な課題とされており，高性能計算環境上でネットワークを効率的に利用することで分散並列計算の実行性能を向上させることを目的とした研究が多数報告されている．これらの

研究は，大別すると，本研究の目的と同様の資源割当やシステム制御など管理者側からのアプローチと，アプリケーションからネットワークを効率的に利用することを目的としたユーザ側からのアプローチに分類できる．

管理者側からのアプローチの 1 つとして，ネットワーク資源も制御するリソース管理システムに関する研究が上げられる．このようなネットワーク資源管理技術に関する研究は，特に，グリッドコンピューティング [32, 33] の分野で多く行われている．高性能計算を目的としたグリッド環境は，複数拠点に分散している高性能計算環境を広域ネットワークで集約することで，さらに大規模な分散並列計算を実行可能な高性能計算環境として構築される．その性質上，広域ネットワークにおける通信性能が分散並列計算処理のボトルネックとなるため，ネットワークをどのように扱うかが重要な課題とされている．

GARA（General-purpose Architecture for Reservation and Allocation）では，DiffServ によるパケットスケジューリングを利用することによって，ユーザにネットワーク資源の QoS（Quality of Service）を事前予約するためのインタフェースを提供する [34]. G-lambda プロジェクトで開発されたグリッドスーパースケジューラ [35] や DRAGON Project [36] では，各拠点間を接続する光パスネットワーク資源に対し，ジョブに割り当てる光パスネットワークの経路設定を事前予約するネットワーク資源管理機能を GMPLS（Generalized Multi-Protocol Label Switching）によって実現している．Tomás らは，Exponential Smoothing（ES）アルゴリズムによってグリッド環境における計算資源及びネットワーク資源の利用状態を予測し，その結果に基づいてジョブに割り当てる資源を各拠点のジョブ管理システムに事前予約機能を実現したネットワークを考慮したグリッドメタスケジューラ [37] を提案している．

これらのグリッド環境に対する資源管理システムは，一般的に各拠点で運用されているジョブ管理システムに対して，資源確保のための事前予約の調整を行うスーパースケジューラとして実現されている．すなわち，各拠点における資源管理は従来のジョブ管理システムにより制御されるため，拠点内のインターコネクトに対するネットワーク資源管理機能は備えていない．一方，本研究では拠点内のインターコネクトに対して，ジョブへのネットワーク資源割当を制御することで，高い実行性能をジョブに提供し，かつ，インターコネクトを効率的に運用することを目的とする．

ユーザによるネットワーク資源要求を実現するための手法としては，ネットワーク資源も含めた計算環境の資源を要求する手段として，グリッド環境上に仮想インフラを構築する記述言語 VXDL（Virtual Resources and Interconnection Networks Description Language） [38] が提案されている．VXDL によるネットワーク資源要求では，ネットワークトポロジなどのネットワーク情報の記述手法に主眼がある．この手法は本研究においてネットワーク情報を管理する上で参考となるが，計算資源とネットワーク資源をともに扱うジョブ管理手法そのものに主眼がある点が異なる．

Ethernet を用いた PC クラスタシステムにおいて，複数パスを持つ Fat-tree のようなネットワークトポロジを構成するための手法として，VLAN ルーティング法 [39] がある．この手法では，VLAN を用いることにより L2 Ethernet ネットワーク上に複数のパスを有する計算ノード間のインターコネクトを構成することを可能とする．これは PC クラスタシステムの構築時に VLAN の設定を行う手法であるため，基本的にはインターコネクトに対する静的な制御である．本研究では，各ジョブにおけるユーザの資源要求に応じてネットワークを動的に割り当てることを目的としているため主眼が異なる．

一方，ユーザ側からのアプローチでは，MPI（Message Passing Interface）による分散並列計算における通信を効率的に行うための研究が多数行われている．MPI はクラスタシステムにおける分散並列計算で広く利用されている並列プログラミングの規格であり，MPI ライブラリでは 1 対 1 通信やグループ通信を行うための関数が提供される．実行するアプリケーションの通信特性に基づいてジョブのプロセスを計算ノードに割り当てる手法に関する研究として，森江らが提案した MPI ランクの配置最適化技術 [40-44] があげられる．この最適化技術では，MPI の集合通信アルゴリズムにおける通信タイミングを考慮して MPI ランクの配置を決定することにより通信の衝突を回避し，計算処理における通信時間を削減する．この手法では 1 つのジョブ内における通信の最適化を目的としているが，他のジョブによる通信の影響は考慮されていない．本研究では計算環境で実行される複数のジョブに対して最適なネットワーク資源配分を目的とする点が異なる．

2.5　クラスタシステムにおける資源管理の問題

前述したように，大規模化している今日のクラスタシステムにおいて効率的に資源を割り当てるためには，様々な資源を柔軟に制御できる資源管理が重要である．しかし，今日利用されているジョブ管理システムで扱うことが可能な資源は，計算ノードの CPU やメモリのような計算資源だけである．そこで，本研究ではクラスタシステムの主な利用方法である分散並列計算の実行性能に大きな影響を与える計算ノード間のインターコネクトに着目し，計算資源とネットワーク資源をともに制御可能なジョブ管理システムの実現を目指す．本研究では，インターコネクトをネットワーク資源として動的に管理可能なジョブ管理システムを実現するための技術課題として，(1) ネットワーク資源を動的に管理・制御する機能を備えたジョブ管理システムの実現可能性の検証，(2) 提案ジョブ管理システムへのネットワーク資源を制御する資源割当ポリシの配備による有用性・実用性の評価，を設定した．本節では，上述の各課題について説明する．

2.5.1　ネットワーク資源を動的に管理・制御する機能

従来のインターコネクトにおける通信の制御は，個々のネットワークデバイスで設定されており，各デバイスの設定に基づいた制御が連携することで，インターコネクト全体の通信制御を実現している．そのため，インターコネクトをネットワーク資源として扱うためには，各ネットワークデバイスを動的に制御する仕組みをどのように実現するかが最大の課題となる．

次に，インターコネクトをネットワーク資源として扱うためには，上述のネットワーク制御機能を利用した資源管理の実現が必要となる．ジョブ管理システムにおける計算資源の管理・割当は，2.3.1.2 で述べたように，各計算ノードにおける計算資源の利用状況の把握，計算資源の利用状況とユーザの要求資源量に基づいた最適な割当資源の探索，ジョブへの資源割当の 3 つの手順で行われる．ジョブ管理システムで計算資源とネットワーク資源をともに効率的に扱うためには，ネットワーク資源の制御についても計算資源と同様の手順で行えることが望ましい．このような資源管理を実現するためには，インターコネクトに対するネットワーク構成や利用状況を把握する機能およびジョブに対してネットワーク資源を明示的に割り当てる機能を，ジョブ管理システムが備える必要がある．また，もう 1 つの手順であるジョブへの割当資源の決定についても，従来のジョブ管理システムの機能では計算資源しか扱えないため，計算資源とネットワーク資源の両資源に対応した資源割当ポリシを設定可能なフレームワークが必要となる．

2.5.2　ネットワーク資源を動的制御する資源割当ポリシ

今日のクラスタシステムでは，広帯域と耐障害性を実現するために，冗長経路を有するインターコネクトが採用されていることが多い．冗長経路を持つインターコネクトでは，計算ノードから発生する通信をどのように複数の通信経路に分散させるのかが，高い通信性能を得るために重要である．冗長経路における通信の負荷分散制御は，通常，スイッチの持つ機能によって行われている．従来のクラスタシステムにおける資源管理では，ジョブ管理システムは計算資源のジョブへの割当だけを制御しており，インターコネクトにおける通信の制御は，各ネットワークスイッチによって独立に行われる．その結果，ジョブに対してより高い性能を得られる資源の組み合わせが存在していたとしても，そのような資源割当を選択できない場合がある．それゆえ，計算資源とネットワーク資源の利用状況を鑑み，最適な資源の組み合わせを決定する資源割当ポリシの実現は，ジョブに高性能な資源を提供し，システム全体の資源を効率的に運用するための重要な課題である．

2.6 おわりに

本章では，クラスタシステムの現状およびジョブ管理システムによる資源管理について分析を行い，クラスタシステムから高い性能が得られる資源をユーザの計算要求に対して提供するためには，分散並列計算の実行性能に大きな影響を与えるインターコネクトをネットワーク資源として制御可能なジョブ管理システムが必要であることを示した．次に，クラスタシステムのインターコネクトから高いネットワーク性能を得ることを目的とした関連研究について調査を行い，本研究の位置づけを確認するとともに，計算資源とネットワーク資源をともに制御可能なジョブ管理システムを実現するための 2 点の課題を導出した．1 点目はジョブ管理システムからネットワーク資源であるインターコネクトを動的に管理・制御することが可能であるか検証することであり，2 点目は Fat-tree インターコネクトで通信衝突を回避する資源割当ポリシを提案・実装・評価することで，実際のクラスタシステムに対する提案ジョブ管理システムの有用性・実用性を確認することである．

インターコネクトをネットワーク資源として扱うための最大の課題は，ネットワークにおける通信の制御を動的かつ一元的に制御するための手段である．従来，ネットワークは静的な資源とみなされていたため，現在のジョブ管理システムはネットワークを制御するための機能を備えていない．しかし，ネットワークの動的制御が可能になれば，資源管理を行うために必要となる資源情報の収集機能とジョブへの資源割当機能が実現できる．また，割当資源を決定する機能を，ネットワーク資源に対応させる必要がある．3 章では，これらの機能を備えた Network-aware なジョブ管理システムフレームワークを提案する．

一方，冗長経路を持つインターコネクト環境において，リンク上における通信の衝突を抑制することは，計算ノード間で高い通信性能を得るために重要である．提案するジョブ管理システムは計算資源とネットワーク資源をともに扱うことが可能であるため，リンク上での通信の衝突を回避する資源割当を実現することが可能である．しかし，そのためには資源情報を考慮してジョブへの割当資源を決定する資源割当ポリシが必要である．4 章では，多くの計算環境で採用されている Fat-tree インターコネクトを持つクラスタシステムにおいて，計算資源とネットワーク資源の両資源を制御する資源割当ポリシを提案・評価し，提案するジョブ管理システムが実際的なクラスタシステム上で有用であることを確認する．

第 3 章

SDN を利用した Network-aware ジョブ管理システムフレームワーク

3.1 はじめに

クラスタシステムは多数の計算ノードをネットワークで相互接続した構成であるため，クラスタシステムの資源は大別して各計算ノードが持つ CPU やメモリなどの計算資源と，その計算ノード間を繋ぐネットワーク資源に分類される．クラスタシステム上で計算処理の大規模化および高速化を実現するためには，できるだけ多数の計算ノードを同時かつ効率的に利用した分散並列計算を行う必要がある．分散並列計算では計算ノード間で頻繁に通信を行うため，ジョブの実行時間はジョブに割り当てられた計算ノード間のネットワーク性能に依存するところが大きい．したがって，各ジョブの実行性能を最大化するためには，その処理に適したネットワーク資源と計算資源の組み合わせをジョブに割り当てられなければならない．また，クラスタシステムは，一般的に複数のユーザから多数の計算要求を受け付け，計算環境が許容する範囲で同時に実行されるため，各計算要求に対してどのように資源を割り当てるかが，個々のジョブの処理時間を短縮させる上で重要となる．

ジョブ管理システムは，クラスタシステムの利用状況を考慮し，資源をジョブに効率的に割り当てる役割を担う．しかし，2 章で述べたように今日利用可能なジョブ管理システムは，計算資源だけを対象に資源管理を行うのみである．このような資源管理では，他のジョブの通信との衝突などにより提供される資源における通信性能は保障されないため，効率的に分散並列計算を実行できる環境を提供できず，システム全体の利用効率も低下させる．これまでは，ハードウェアの性能やその構成によって，十分な通信性能を保証することが前提であった．しかし，大規模化・分散化傾向にあるクラスタシステムではハードウェアのみによって通信性能を保証することは困難になると考えられる．

そこで，本研究では，インターコネクトをネットワーク資源として扱うことができる

ネットワーク資源の動的な制御機能を実現する新しいアプローチで資源管理手法の実現を目指す．本章では，このような視点から，ネットワーク資源の動的な制御による資源管理手法の実現可能性を模索するため，ネットワーク資源を動的に制御することによる資源管理手法のプロトタイプ化を行う．

以下に本章の構成を示す．3.2 節では，ジョブ管理システムからインターコネクトをネットワーク資源として扱うための要件を分析する．3.3 節では，3.2 節の要件分析に基づき，インターコネクトに対するネットワーク制御機能を提案し，その機能を統合した Network-aware ジョブ管理システムフレームワークについて説明する．3.4 節では，提案するジョブ管理システムに対する評価実験について述べる．最後に，3.5 節で本章のまとめを行う．

3.2 ネットワーク資源管理の要件

本節では，ジョブ管理システムからインターコネクトをネットワーク資源として扱うために，まずネットワーク資源についての定義を行い，ネットワーク資源を管理するためのシステム要件を分析する．

3.2.1 ネットワーク資源

インターコネクトをネットワーク資源として管理するためには，その資源にあった資源管理指標を定める必要がある．計算資源と同様にネットワーク資源を扱うため，従来のジョブ管理システムにおける計算資源の管理割当について確認する．ジョブにおける CPU への資源要求は，一般的にジョブで実行する分散並列計算の並列数で行われる．それゆえ，資源の要求量は整数値であり，ジョブ管理システムはその要求数に従いジョブのプロセスを CPU に割り当てる．1 台の計算ノードに同時に割り当てることが可能なジョブプロセスの数はスロット数として管理者によってジョブ管理システムに設定されている．一般的にその値は計算ノードの CPU 数および CPU のコア数と同じであるが，CPU のタイムシェアリング処理によりそれ以上の値も設定可能である．メモリに対するジョブの資源要求は使用する容量で行うため，MB（メガバイト）や GB（ギガバイト）等の単位を指定する必要がある．以上より，ジョブ管理システムにおける計算資源の管理は計算ノード単位で行われており，各計算ノード上における資源量として CPU に対するスロット数やメモリ容量が用いられている．

前述の計算資源に対する資源管理を参考に，ジョブ管理システムが管理・割当を行うネットワーク資源について分析する．まず，計算資源における計算ノードに相当するインターコネクト上におけるネットワーク資源の管理単位を定義する必要がある．ジョブに割

り当てた資源におけるインターコネクトの利用は計算ノード間の通信となるため，ネットワーク資源の管理単位は割り当てた計算ノード間の通信経路であると考える．通信経路に対して要求および割当を行う資源量をインターコネクトにおける性能指標である遅延と帯域幅から検討する．インターコネクトにおける遅延の主な要因は，ネットワークデバイスの性能およびネットワークデバイスを経由するホップ数が考えられる．ネットワークデバイスの性能による遅延については，高性能計算環境の構築時に決定される要因であり，資源制御による抑制はできない．ネットワークデバイスを経由するホップ数については，計算ノード間の総リンク長はインターコネクトのトポロジ構造に基づいて算出が可能であるため，割り当てる計算ノードの配置に依存する．一方，各通信経路における利用可能帯域幅に関しては，クラスタシステムにおいて実行されている各ジョブの通信状況に依存する．帯域幅の測定箇所としては，計算ノード間または各リンクにおける使用帯域幅を測定する方法が考えられる．ただし，各計算ノードで使用帯域幅を測定した場合，冗長経路を持つインターコネクトでは，測定結果がどの通信経路における値であるのかを識別することが困難になると考える．

以上より，提案する Network-aware ジョブ管理システムフレームワークにおいて制御するネットワーク資源とは，ジョブに割り当てる計算ノード間の通信経路であり，その資源量として通信経路上における遅延および帯域幅とする．また，インターコネクトをネットワーク資源として扱うために必要な情報として，計算ノードの配置も含めたインターコネクトのトポロジ構造および各リンクの利用可能帯域幅を取得する必要があると考える．

3.2.2　システム要件

2.3 節で説明したように，今日の一般的なジョブ管理システムはインターコネクトは資源として管理・制御する機能を有していない．計算資源と同様に，インターコネクトをネットワーク資源として制御するジョブ管理システムを実現するためには，下記の 5 項目のシステム要件を満たす必要があると考える．以下，各要件について説明する．

（1）ネットワーク資源要求入力インタフェース

ユーザがジョブを投入する際，計算資源と同様にネットワーク資源を要求できなければならない．そのためには，ユーザが要求するネットワーク資源量を記載し，システムに伝達するための直感的なインタフェースが必要不可欠である．また，計算資源同様に要求できるよう一貫性のあるインタフェースでなければならない．

（2）ネットワーク資源情報取得機能

インターコネクトの利用状態を考慮してジョブへの資源割当を行うためには，計算資源と同様にクラスタシステムのネットワーク利用状況を監視し，利用情報を取得するための機能が必要である．

(3) ネットワーク資源割当管理機能

ジョブに計算資源を割り当てるのと同様に，ネットワーク資源を明示的に割り当てるための機能が必要である．また，実行中のジョブにどのようにネットワーク資源を割り当てたのかを管理する仕組みも必要となる．

(4) 割当資源決定機能

取得されたネットワーク資源情報と計算資源情報及びユーザの資源要求に対し，資源割当ポリシに基づいて最適な計算資源及びネットワーク資源の組み合わせを割り当てるためには，ジョブの計算特性やネットワーク利用特性を考慮して割り当てる資源を決定する機能が不可欠である．

さらに，上記の要件（4）に対しては，下記の要件（5）を満たすことが必要と考える．

(5) 任意の資源割当ポリシを配備できるシステム構成

一般的に，資源に対する管理・運用ポリシは，管理組織や資源構成などの要因により，高性能計算環境ごとに異なる．そのため，管理者が資源割当ポリシを柔軟に設計できるようにする必要がある．これにより，より実用性の高いジョブ管理システムの実現につながる．

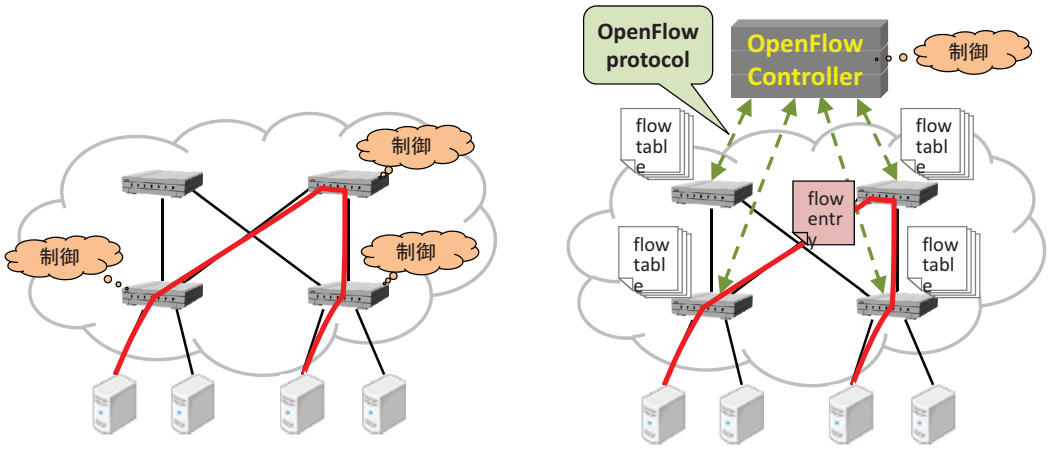
3.3 SDN-enhanced JMS フレームワーク

本節では，3.2 節でまとめた 5 つの要件を満たす，ネットワーク資源に対する動的な管理・割当機能を備えた Network-aware ジョブ管理システムフレームワークについて提案する．

3.3.1 SDN/OpenFlow

3.2 節で述べた 5 つの要件を実現するため，新しいネットワークアーキテクチャのコンセプトである Software Defined Networking（SDN）におけるネットワークのプログラミング性に着目した．SDN におけるネットワークの構成は，従来のネットワークにおいて個々のネットワークスイッチで行われていたネットワークパケットの制御機能と転送機能を分離し，制御機能を 1 つのコントローラに集約したアーキテクチャとなる．SDN では，ネットワーク環境における管理対象のスイッチを一元的にコントローラで管理する．そのコントローラはソフトウェアとして実装可能であるため，プログラムを書き換えることによってネットワークの制御を動的に変更可能である．

この SDN を実現する技術の 1 つとして OpenFlow [45] があり，OpenFlow コントローラをソフトウェアとして実装するため，多数の開発フレームワークが提供されている [46-51].



(a) 従来のネットワーク.　(b) OpenFlow ネットワーク.

図 3.1　従来のネットワークと OpenFlow ネットワークの例.

図 3.1 に従来のネットワークおよび OpenFlow ネットワークの例を示す．通常，OpenFlow ネットワークは，上述した SDN に基づき，パケット転送を行う複数の OpenFlow スイッチと，パケットの転送経路を制御する 1 つの OpenFlow コントローラで構成される．OpenFlow ネットワークにおいて，パケットの転送経路は各 OpenFlow スイッチが持つ *flow table* に基づいて決定される．*flow table* は，対象となるパケットごとに処理を定義した *flow entry* の集合である．各 *flow entry* は，パケットに対する判定条件及び処理方法と，その *flow entry* で処理されたパケット数などを記録したフロー統計情報の 3 項目で構成される．

OpenFlow コントローラは，各 OpenFlow スイッチの *flow table* に *flow entry* を OpenFlow プロトコルを通して追加及び削除することができる．例えば，OpenFlow ネットワークにおいて，該当する *flow entry* が *flow table* に存在しないパケットが届いた場合，OpenFlow スイッチはそのパケットを OpenFlow コントローラに転送し，どのように処理するのかを OpenFlow コントローラに問い合わせる．OpenFlow スイッチからの問い合わせに対し，OpenFlow コントローラはそのパケットに対応した *flow entry* を生成し，OpenFlow スイッチの *flow table* に追加する．この OpenFlow コントローラによる *flow entry* の制御により，従来のネットワーク環境では実施が困難であったネットワーク全体に対する動的な制御を行うことが可能となる．

3.3.2　システム概要

本節では，計算資源とネットワーク資源をともに管理・割当できる Network-aware ジョブ管理システムとして，計算資源に対する管理・割当を行う従来のジョブ管理システム

に対し，3.2 節のシステム要件に基づき設計したネットワーク資源動的制御機能を拡張した SDN-enhanced JMS フレームワークを提案する．提案する SDN-enhanced JMS フレームワークの構成を 図 3.2 に示す．上述したように，3.3.1 節で述べた OpenFlow を中核技術として採用し，SDN-enhanced JMS フレームワークを設計する．SDN-enhanced JMS フレームワークでは，従来のジョブ管理システムにネットワーク資源に対する機能を備えた Network Management Module（NMM）と呼ぶ外部モジュールを連携させる．さらに，NMM は，Brain コンポーネントと OpenFlow コントローラを内包した Network Control コンポーネントの 2 つのモジュールとデータベースで構成する設計とする．このようなモジュール構成により，従来のジョブ管理システムへの改良が最小限に抑えた実装を行うことができる．また，従来のジョブ管理システムのマスターモジュールが実行されている管理ノード上に NMM が導入できない場合や，異なる管理ノードで既存の OpenFlow コントローラが実行されている場合のように，単一管理ノードによる構築ができない環境が想定される．そのような環境でも SDN-enhanced JMS フレームワークを導入可能とするため，ネットワークを介して従来のジョブ管理システムおよび NMM の各コンポーネントが通信を行う設計とする．

Brain コンポーネントは，従来のジョブ管理システムにおいて計算資源情報に基づいて生成された計算ノードの割当候補リストを，Network Control コンポーネントからはネットワーク資源の利用状況を取得し，ジョブに割り当てるべき計算資源とネットワーク資源を決定する要件（4）を実現するコンポーネントである．また，要件（5）を実現するため，ジョブに割り当てる資源を決定するためのポリシの定義を，資源割当ポリシクラスモジュールを介して行う設計とした．資源割当ポリシクラスモジュールは，システム管理者が任意の資源割当ポリシを自由に設計でき，資源割当ポリシを容易に切り替えて適用できるようにするため，スクリプト言語を使用する設計とした．

一方，Network Control コンポーネントは，OpenFlow を利用して要件（2），（3）を実現するためのコンポーネントである．要件（2）のネットワーク資源情報取得機能は，Network Control コンポーネントが OpenFlow コントローラおよび各 *flow entry* が持つ統計情報を介してネットワーク資源の情報を取得する．要件（3）のネットワーク資源割当管理機能は，計算ノード間の通信経路を OpenFlow コントローラによる *flow entry* を用いたネットワーク制御によりネットワークフローとして扱うことで実現する．ジョブに割り当てる資源における *flow entry* 情報は，ネットワーク資源の割当を決定する Brain コンポーネントで生成された情報から取得する．

なお，要件（1）は，従来のジョブ管理システムにおける計算資源の要求方法を拡張することで実現する．これは，SDN-enhanced JMS フレームワークにおいても，従来のジョブ管理システムと同じ手順でユーザの資源要求及びジョブ投入を可能にするためである．ネットワーク資源に対する資源要求方法として，SDN-enhanced JMS フレームワークでは

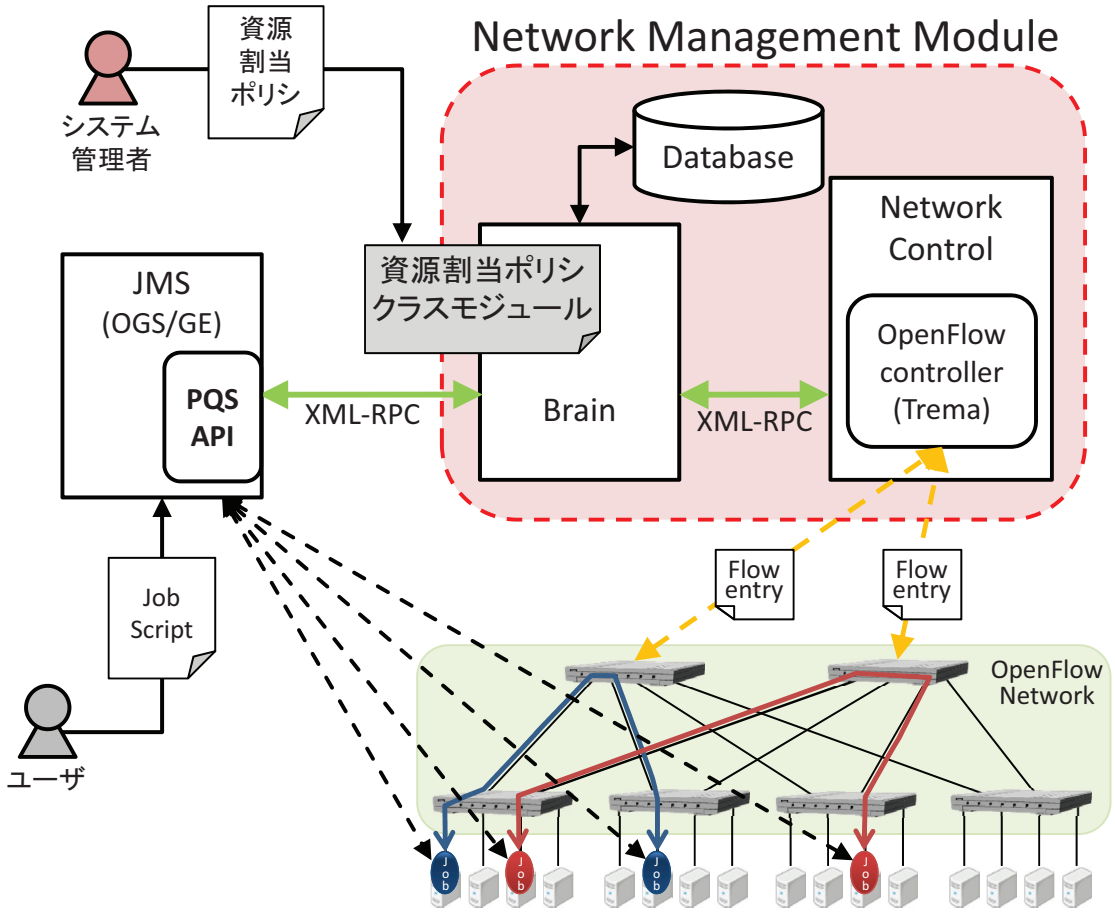


図 3.2　SDN-enhanced JMS フレームワークの構成.

資源割当ポリシクラスモジュールで定義する資源割当ポリシの名前を指定することによって要求できる設計とした.

3.3.3　システム実装と詳細

本節では，提案する SDN-enhanced JMS フレームワークのプロトタイプ実装の構成，資源割当の処理の流れ，資源割当ポリシの構成について述べる.

3.3.3.1　SDN-enhanced JMS フレームワークの構成

本節では提案する SDN-enhanced JMS フレームワークのプロトタイプ実装について説明する．提案する SDN-enhanced JMS フレームワークは，3.3.2 節で述べたように，従来のジョブ管理システムに NMM を連携させることで実現する．本章における SDN-enhanced

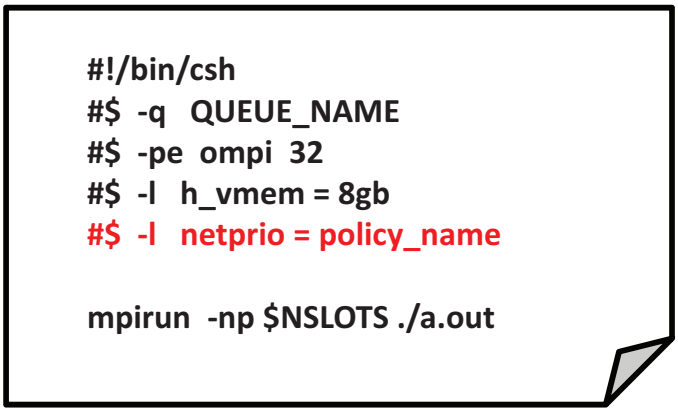


図 3.3 SDN-enhanced JMS フレームワークにおけるジョブスクリプトの記述例.

JMS フレームワークのプロトタイプ実装では，従来のジョブ管理システムとして Open Grid Scheduler/Grid Engine (OGS/GE) [18] を，NMM に組み込む OpenFlow コントローラには前述の OpenFlow コントローラ開発フレームワークの中から Trema [46] を採用した．Trema は Ruby または C 言語でコントローラの実装が可能なフレームワークであるため，コントローラ設計段階でのスクリプト言語である Ruby による変更の容易さと，実用段階での C 言語による高性能化が期待できる点から，提案する SDN-enhanced JMS フレームワークのプロトタイプ実装に Trema を採用した．なお，本プロトタイプ実装で用いた OpenFlow 仕様のバージョンは 1.0 である．また，ジョブ管理システムのマスターモジュールと Brain コンポーネント間及び Brain コンポーネントと Network Control コンポーネント間の通信には XML-RPC [52, 53] を利用する．XML-RPC は Remote Procedure Call (RPC) プロトコルの一種であり，非常にシンプルな仕様のため広く利用されている．提案するジョブ管理システムフレームワークでは，従来のジョブ管理システムや OpenFlow コントローラを同一管理ノードに集約できない場合も想定し，汎用性の高い XML-RPC を用いて通信を行う．以下，3.2 節の要件がどのように実装されたかについて説明する．

ネットワーク資源要求入力インタフェース

提案する SDN-enhanced JMS フレームワークのプロトタイプ実装における従来のジョブ管理システムである OGS/GE の計算資源の要求方法である *qsub* コマンドに対し，ネットワーク資源を要求するための資源要求オプションを拡張することで実現する．*qsub* コマンドは，ユーザが OGS/GE のマスターモジュールにジョブを投入するために提供されているコマンドである．通常，ユーザはコンパイルなどを行うために提供されている作業用ノードでジョブスクリプトを引数として *qsub* コマンドを実行することでジョブの投入を行う．提案する SDN-enhanced JMS フレームワークにおけるネットワーク資源の要求は，ユーザの計算資源要求と同様に，ジョブスクリプトにネットワーク資源要求を記述する

（図 3.3 参照）．提案する SDN-enhanced JMS フレームワークのプロトタイプ実装では，このネットワーク資源に対する拡張資源要求オプションを “*netprio*” という名前で定義しており，本オプションの値に後述する資源割当ポリシの名前を指定することでネットワーク資源の割当手法を要求できる．すなわち，本プロトタイプ実装におけるネットワーク資源への要求はシステム管理者が定義したポリシに基づいて行われ，ユーザが自由に計算ノード間のトポロジや通信経路における資源量を要求する仕組みは実装していない．

ネットワーク資源情報取得機能

本機能は OpenFlow コントローラを介して実行するため，Network Control コンポーネントに実装する．本ジョブ管理システムフレームワークの実装では，計算環境におけるネットワークトポロジと，ネットワークの各リンクにおける帯域幅と遅延をネットワーク資源情報として取得する．計算環境のネットワークトポロジ情報は，OpenFlow コントローラによる OpenFlow ネットワークの管理においても必要であり，一般的に Link Layer Discovery Protocol（LLDP） [54] を用いて取得している．提案するジョブ管理システムフレームワークにおいても，Network Control コンポーネントが OpenFlow コントローラを介して LLDP を実行することによってトポロジ情報を取得するよう実装した．

各リンクにおける帯域幅についての情報は，各 *flow entry* の統計情報にそのネットワークフローにおける転送バイト数を保持しているため，これらの情報からリンクにおける使用帯域幅を取得する．各リンクにおける使用帯域幅は，上記の転送バイト数と本ジョブ管理システムフレームワークの設定ファイルに記載する最大帯域幅から算出する．具体的には，対象とするリンクに対して，そのリンクを利用する各ネットワークフローの転送パケット量を *flow entry* の *byte_count* フィールドから取得し，前回取得時からの差分を算出する．なお，1 つのリンクには複数のネットワークフローが存在する場合があるため，転送パケット量は各 *flow entry* の *byte_count* フィールドの値の合計値となる．ある時刻 t における *flow entry* i の *byte_count* フィールドの値を $byte\_count_i(t)$ とし，ネットワーク資源情報の収集間隔を Δt とした場合，転送パケット量は下記の式で算出される．

$$(\text{転送パケット量}) = \sum_i byte\_count_i(t) - byte\_count_i(t - \Delta t)$$

この転送パケット量と情報取得間隔の時間を基にその間の使用帯域幅を算出する．

$$(\text{使用帯域幅}) = (\text{転送パケット量}) \times 8/\Delta t$$

この使用可能帯域幅と最大帯域幅との差分を利用可能帯域幅とする．なお，本プロトタイプ実装において，Network control コンポーネントによるネットワーク資源の利用状況の情報収集間隔 Δt は 10 秒としている．遅延については，その資源量を本提案 SDN-enhanced JMS フレームワークでは計算ノード間の通信経路におけるホップ数としているため，その通信経路と取得したインターコネクトのトポロジ情報に基づき算出する．それゆえ，遅

延の資源量については本機能により定期的に収集されるネットワーク資源情報に含まれない.

ネットワーク資源割当管理機能

本機能は OpenFlow コントローラの機能を利用して実現するため，Network Control コンポーネント内に統合することとした．本機能は，提案するジョブ管理システムフレームワークが決定したジョブへのネットワーク資源割当に従い，そのジョブが使用するネットワーク経路の制御を行う．具体的には，Brain コンポーネントが決定したジョブに割り当てるネットワーク経路に応じて生成された *flow entry* を *flow table* に書き込むことで実現する．そのため，Network Control コンポーネントには，XML-RPC を介して Brain コンポーネントからジョブに割り当てた *flow entry* 情報を取得する機能を実装した.

割当資源決定機能

本機能は Brain コンポーネントに実装した．計算資源情報に基づいたジョブに割り当てる計算ノードの選定処理は，従来のジョブ管理システムが有する機能であるため，Brain コンポーネントでは行わない．Brain コンポーネントでは，従来のジョブ管理システムが計算ノード選定処理で算出した優先度に従い生成した計算ノードの割当候補リストに対し，インターコネクトの利用状況を反映させた新たな優先度を算出し，その優先度に従い割当候補リストの順序を変更することによって両資源を考慮した資源割当を実現する．このネットワーク資源情報を考慮した新たな優先度の算出方法は後述する資源割当ポリシで定義される.

Brain コンポーネントが従来のジョブ管理システムの計算ノード割当候補リストに対して変更を行うため，提案する SDN-enhanced JMS フレームワークでは OGS/GE が持つ Parallel Environment Queue Sort (PQS) API を利用する．PQS API は，OGS/GE の計算ノード選定処理に対して，システム管理者が定義した規則を反映させることができる．この選定規則の定義は動的ライブラリとして実装する必要がある．本ジョブ管理システムフレームワークでは，PQS API の動的ライブラリを XML-RPC で通信可能な実装とすることで，外部モジュールである NMM の Brain コンポーネントからジョブに割り当てる計算ノードの割当候補リストを制御可能としている.

任意の資源割当ポリシを配備できるシステム構成

本要件は，Brain コンポーネント上に実装した資源割当ポリシクラスモジュールで実現する．資源割当ポリシクラスモジュールは，管理者が資源割当ポリシを定義するスクリプトを記述するためのインタフェースを提供する．資源割当ポリシを記述するスクリプト言語に，SDN-enhanced JMS フレームワークでは Ruby を採用する．その理由は，OpenFlow コントローラとして採用した Trema において Ruby を使用しているため，NMM 内での親和性が高いためである．資源割当ポリシは Ruby で記述したスクリプトであるため，システム管理者は各資源に対する処理を自由に記述できる設計となっている．また，Brain コ

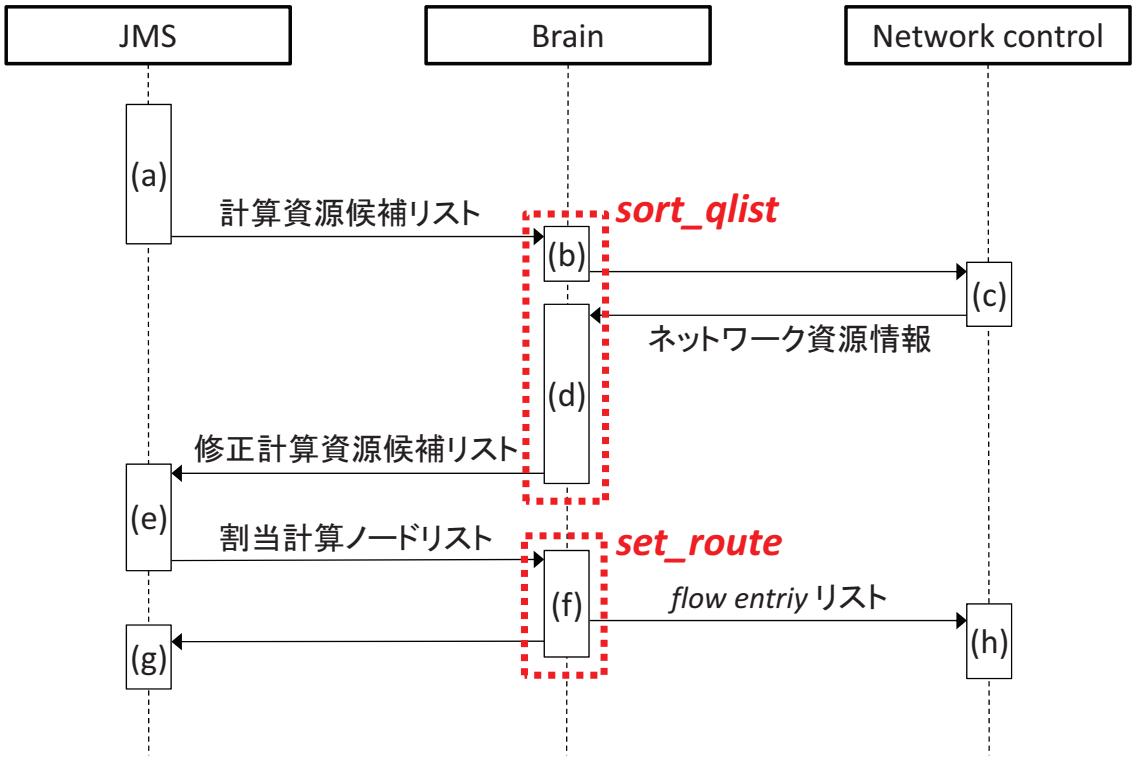


図 3.4　SDN-enhanced JMS フレームワークにおける処理の流れ.

ンポーネントが 3.2 節の要件（4）を実現するために取得する，従来の ジョブ管理システムからの計算資源情報や，Network Control コンポーネントが収集したネットワーク資源情報を参照するための API を，資源割当ポリシクラスモジュールで提供する．資源割当ポリシクラスモジュールでは，3.2 節の要件（1）による拡張資源要求オプション “*netprio*” の値ごとに処理を記述することによって，システム管理者は複数の資源割当ポリシを定義することができる.

3.3.3.2　資源割当の処理フロー

本節では，提案する SDN-enhanced JMS フレームワークのプロトタイプ実装における計算資源およびネットワーク資源の割当について説明する．図 3.4 は SDN-enhanced JMS フレームワークにおける資源割当処理を，システムを構成する JMS，Brain，Network control に切り分けて処理の流れを示す.

まず，図 3.4 の（a）では，2.3.1.2 節で述べたように，JMS はジョブキューに格納されているジョブから資源割当を行うジョブを選択し，そのジョブのジョブスクリプトに記載されたユーザの資源要求について確認する．選択されたジョブに対し，提案する

SDN-enhanced JMS フレームワークのプロトタイプ実装において従来のジョブ管理システムとして採用した OGS/GE の場合，クラスタシステムの計算ノードの中から，ユーザの資源要求を満たす計算ノードを抽出し，その計算ノード名を並べたジョブへの割当候補リストを生成する．その際，割当可能なスロットを持たない計算ノードや，メモリに対する資源要求を満たさない計算ノードは割当候補リストから除外される．次に，生成された計算ノードの割当候補リストに対し，各計算ノードにおける利用状況に基づき，資源利用の少ない計算ノードからジョブに割り当てるよう並べ替えを行う．通常，OGS/GE におけるこの並べ替え処理は，各計算ノードの利用状況として CPU ロードアベレージを用いる．CPU ロードアベレージの値は計算ノードで CPU が利用されているほど大きくなるため，その値が小さいほど高い優先度を持つ計算ノードとして割当候補リストの上位になるよう並べ替える．OGS/GE では，この割当候補リストに対して上位の計算ノードから順にジョブに割り当てる計算ノードを決定し，計算資源の割当処理を完了する．一方，提案する SDN-enhanced JMS フレームワークでは，割当候補リスト，各計算ノードが持つ割当候補リストの並べ替えに使用した優先度や割当可能なスロット数やメモリ容量，ジョブに関する情報とともに上述の PQS API および XML-RPC を介して Brain コンポーネントに送る．ジョブに関する情報にはジョブ ID，ジョブスクリプトに記述されたスロット数やメモリ容量などのユーザが要求する資源量があげられる．

計算ノードの割当候補リストを受け取った Brain コンポーネントは，（b），（c）の処理で，Network Control コンポーネントからインターコネクトのトポロジ情報および各リンクにおいて使用されている帯域幅と最大帯域幅の値をネットワーク資源情報として取得する．インターコネクトのトポロジ情報は，計算ノードやネットワークスイッチの一覧で構成されるノード情報と，それらのノードがどのように接続されているかの対応をリストにしたリンク情報で構成される．（d）の処理で，割当候補リストの各計算ノードが持つ計算資源に関する優先度，取得したネットワーク資源情報，ユーザが要求する資源量に基づき，後述する資源割当ポリシにおける処理の定義に従って各計算ノードにおける新たな優先度を定める．新たに設定された優先度に従い，計算ノードの割当候補リストを並べ替え，その変更された割当候補リストを OGS/GE に送り返すことで，（e）において Brain コンポーネントで決定したジョブへの計算ノードの割当を OGS/GE に反映させることができる．（e）でジョブへの割当が確定した計算ノードリストを Brain コンポーネントに送り，Brain コンポーネントはその各計算ノード間の通信経路を *flow entry* として生成する．生成された *flow entry* リストは Network Control コンポーネントに送られ，（g）でジョブプロセスの計算ノードへの割当を，（h）で *flow entry* リストの OpenFlow スイッチへの設定を行うことで，計算資源，ネットワーク資源のそれぞれの資源割当を完了する．

提案する SDN-enhanced JMS フレームワークによる計算ノードの割当候補リストへの制御の例を図 3.5 に示す．図 3.5 ではジョブに 4 台の計算ノードを割り当てる状況を想

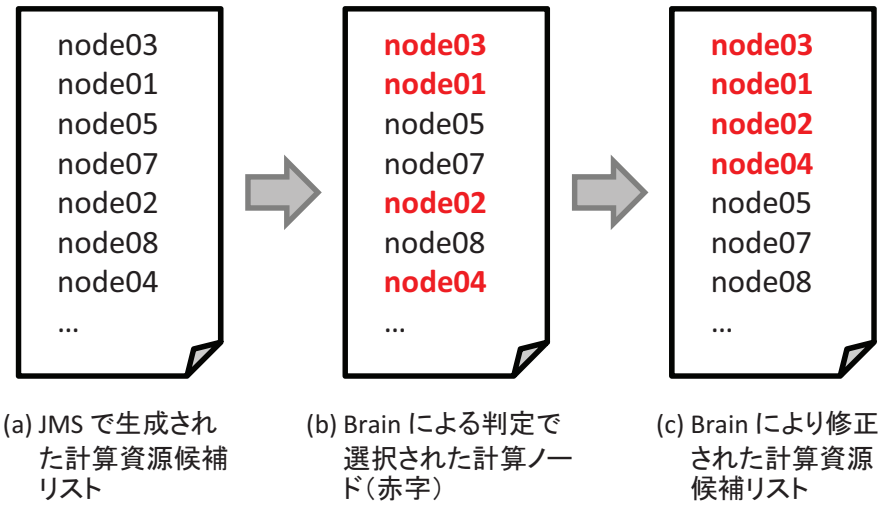


図 3.5 提案する SDN-enhanced JMS フレームワークにおける計算ノードの割当候補リストに対する処理の例（4 台の計算ノードをジョブに割り当てる場合）.

定している．従来のジョブ管理システムである OGS/GE による資源割当では，図 3.5-(a) に従い，node03，node01，node05，node07 がジョブに割り当てられる．一方，提案する SDN-enhanced JMS フレームワークの図 3.4 の（d）の処理で，後述の資源割当ポリシにおける定義に従い新たに優先度を各計算ノードに設定した結果，図 3.5-(b) で示す 4 台の計算ノードが node03，node01，node02，node04 の順で高い優先度となったと仮定する．このとき，Brain コンポーネントは新たな優先度に従い，それらの計算ノードが割当候補リストの上位に来るよう並べ替えを行い，図 3.5-(c) に示す割当候補リストに修正して OGS/GE へと送り返すため，OGS/GE は図 3.4 の（e）の処理で上述の 4 台の計算ノードをジョブに割り当てる．

3.3.3.3 資源割当ポリシ

本節では，資源割当ポリシクラスモジュールを介してシステム管理者が定義できる資源割当ポリシについて述べる．資源割当ポリシクラスモジュールでは，3.3.3.1 節で述べたように，拡張資源要求オプション “*netprio*” で指定する資源割当ポリシ名ごとにジョブへの計算資源およびネットワーク資源の割当処理を定義できる．資源割当ポリシは，ジョブへの計算資源およびネットワーク資源の割当を制御するため，NMM の Brain コンポーネントにおける処理を 3 つのフェーズに分割し，各フェーズに対する処理を定義する設計としている．Brain コンポーネントにおける 3 つの処理フェーズは，（1）図 3.4 の（b）と（d）の処理である計算ノード割当候補リストの修正処理，（2）（f）の処理であるジョブに割り

当てる計算ノード間の通信経路の設定処理，（3）ジョブの実行終了時におけるネットワーク資源に対する後処理である．この各フェーズにおける処理を，（1）については *sort_qlist* 関数，（2）については *set_route* 関数，（3）については *del_route* 関数に定義する．

sort_qlist 関数は，PQS API から取得する計算ノードの割当候補リストに対し，Network Control コンポーネントから取得したネットワーク資源情報を計算ノードの優先度にどのように反映させて新たな優先度を算出するのかを定義する関数である．本関数での定義によりジョブへの資源割当が決定されるため，資源割当ポリシの設計において最も重要な関数となる．*set_route* 関数は，割当計算ノードリストの各計算ノード間における通信経路に対応した *flow entry* を生成し，それらの *flow entry* を OpenFlow スイッチの *flow table* に追加するよう Network Control コンポーネントに指示を出す関数である．そのため，この関数における処理は，基本的には，どの資源割当ポリシにおいても同じとなるが，計算ノード間の割当通信経路の再探索や通信経路の設定以外の OpenFlow ネットワークへの設定等も行えるよう自由に定義できる設計とした．*del_route* 関数は，ジョブが終了した際に，そのジョブに割り当てられた *flow entry* を OpenFlow スイッチの *flow table* から削除するための処理を定義する関数である．この関数における処理も *set_route* 関数と同様，基本的には共通の処理となるが，資源割当ポリシの独自の後処理が追加できるよう定義可能な設計とした．

資源割当ポリシを配備した資源割当ポリシクラスモジュールの擬似コードを図 3.6 に示す．図 3.6 は，2 つの資源割当ポリシ “*Policy1*” と “*Policy2*” を配備した場合の例であり，ユーザがジョブスクリプトの “*netprio*” でこれらの資源割当ポリシ名を指定することで，その資源割当ポリシ名で定義された処理に従い Brain コンポーネントがジョブへの割当資源を決定する．また，“*netprio*” の値が未指定，または，該当しない資源割当ポリシ名が指定された場合の例外処理として，OGS/GE による資源割当と同様の動作をする処理も定義されている．資源割当ポリシクラスモジュールは，3.3.3.1 節で述べたように Ruby を用いて資源割当ポリシを記述する設計となっているため，システム管理者は SDN-enhanced JMS フレームワークを停止することなく資源割当ポリシの追加・変更を行うことが可能である．

```
## Resource Assignment Policy
class Policy
  def sort_qlist ( 計算資源候補リスト )
    case netprio
    when Policy1 then
      ネットワーク資源情報の取得
      Policy1 における各計算ノードの新たな優先度を算出
      修正計算資源候補リスト = 算出した新たな優先度に従い
                              計算資源候補リストを修正
    when Policy2 then
      ネットワーク資源情報の取得
      Policy2 における各計算ノードの新たな優先度を算出
      修正計算資源候補リスト = 算出した新たな優先度に従い
                              計算資源候補リストを修正
    else
      修正計算資源候補リスト = 計算資源候補リスト
    end
    return 修正計算資源候補リスト
  end

  def set_route ( 割当計算ノードリスト )
    case netprio
    when Policy1 then
      flow entry リスト = Policy1 に基づいた計算ノード間の
                          割当通信経路のflow entry
    when Policy2 then
      flow entry リスト = Policy2 に基づいた計算ノード間の
                          割当通信経路のflow entry
    else
      flow entry リスト = 計算ノード間の通信経路のflow entry
    end
    return flow entry リスト
  end

  def del_route ( ジョブ情報 )
    case netprio
    when Policy1 then
      Policy1 固有の後処理
    when Policy2 then
      Policy2 固有の後処理
    end
    ジョブに割り当てたflow entry の削除
  end
end
```

図 3.6 資源割当ポリシを配備した資源割当ポリシクラスモジュールの擬似コード.

3.4 評価実験

3.3.2 節の要件に基づき提案・実装したジョブ管理システムフレームワークの実現可能性を検証するため，単純な 2 層 Flat-tree インターコネクトを有するクラスタシステムをターゲットとした資源割当ポリシを実装し，以下の評価実験を行った.

3.4.1 実験で用いる資源割当ポリシ

提案するジョブ管理システムフレームワークにおいて資源割当を行うためには，資源割当ポリシクラスモジュールを介して資源割当ポリシを定義する必要がある．そこで，本実験で用いる資源割当ポリシとして，“hop” と “bandwidth” の 2 種類の簡易的な資源割当ポリシを実装して資源割当ポリシクラスモジュールに配備する．“hop” は，割り当てる計算ノード間の遅延が最小となる計算ノードの組み合わせを求めてジョブに資源割当を行う遅延考慮ポリシである．“bandwidth” は帯域考慮ポリシで，ジョブに割り当てる計算ノード間の利用可能帯域が最大となる計算ノードの組み合わせを探索する.

この 2 つの資源割当ポリシにおける割当資源決定処理の流れは基本的に同様であり，下記の手順で行われる.

(1) ネットワーク資源に関する情報を取得する.
(2) 割当候補リストの計算ノード間の通信経路における対象とする資源量を算出し，その通信経路の優先度とする.
(3) 3 台以上の計算ノードが要求されている場合，要求台数の計算ノードの組み合わせにおける全通信経路の資源量からその組み合わせの優先度を算出する.
(4) 最も高い優先度を持つ組み合わせの計算ノードがジョブに割り当てられるよう割当候補リストの修正を行う.

（1）については資源割当ポリシクラスモジュールが提供する機能を用いて，インターコネクトのトポロジ情報と各リンクにおける利用可能帯域幅を取得する．（2）において，資源量の算出処理はダイクストラアルゴリズムによる経路探索処理を利用して実装した．具体的に，遅延考慮ポリシでは 2 台の計算ノード間の通信経路を探索する際のホップ数をカウントすることで資源量とし，帯域考慮ポリシでは通信経路を構成する各リンクにおける利用可能帯域幅の最小値をその通信経路の資源量とする．算出した資源量は，ホップ数については昇順で，利用可能帯域幅については降順で優先度として利用する．（3）では，（2）で求めた資源量を用い，要求台数の計算ノードの組み合わせにおける全通信経路の資源量の平均値を算出し，その計算ノードの組み合わせの優先度とする．（4）では，（3）の

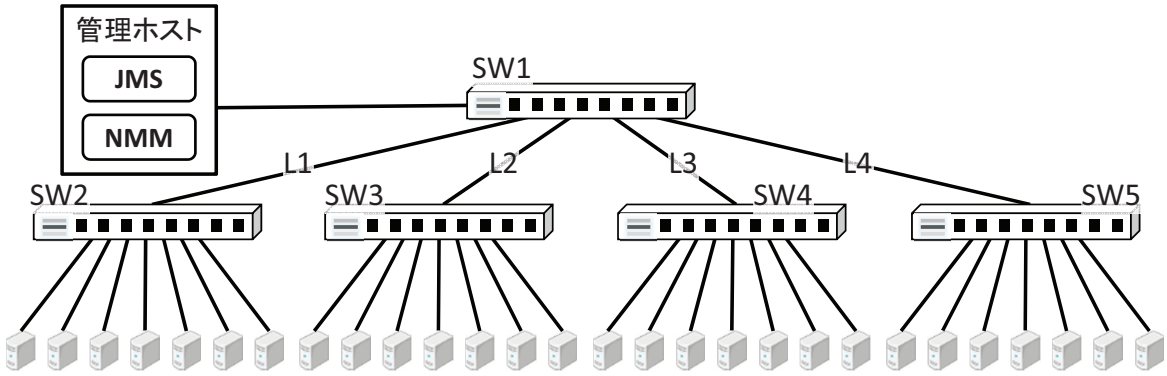


図 3.7　クラスタシステムの構成.

表 3.1　計算ノードの仕様.

OS	CentOS 6.2
CPU	Intel Xeon E5-2620(2.00GHz) x2
Memory	64GB
NIC	on board Intel I350 Gigabit Ethernet

結果から最も高い優先度を持つ組み合わせの計算ノードが割当候補リストの上位になるよう計算ノードの優先度を修正し，リストの並べ替えを行う．

3.4.2　実験環境

実験環境として，図 3.7 に示す構成の Flat-tree トポロジのインターコネクトを持つクラスタシステムを用いる．本クラスタシステムは，28 台の表 3.1 に示す計算ノードと，5 台の OpenFlow スイッチで構成されている．なお，使用した OpenFlow スイッチは 3 台の NEC UNIVERGE PF5240 であり，SW2 と SW3，SW4 と SW5 はそれぞれ同一スイッチを VLAN で 2 つに分割して使用している．それぞれ計算ノード及び OpenFlow スイッチを繋ぐネットワークはすべて Gigabit Ethernet である．管理ノードは 1 台であり，ジョブ管理システムのマスターモジュール，NMM，OpenFlow コントローラはすべてこのノードに集約されている．

3.4.3　SDN-enhanced JMS フレームワークの動作検証

本節では，提案する SDN-enhanced JMS フレームワークによるネットワーク資源の制御・割当を検証するため，ジョブにおける主な処理としてプロセス間での通信だけを行う

表 3.2 投入したジョブセットの並列数.

ジョブ番号	1	2	3	4	5	6	7	8	9	10
並列数	6	10	12	14	6	2	10	4	8	10
ジョブ番号	11	12	13	14	15	16	17	18	19	20
並列数	2	8	8	4	14	12	4	2	8	12
ジョブ番号	21	22	23	24	25	26	27	28	29	30
並列数	8	8	8	6	6	4	4	4	4	2

プログラムを用いて評価を行う.

3.4.3.1 実験の目的と方法

本実験の目的は，ネットワーク資源を考慮することによるジョブへの資源割当の効果を評価することにある．そこで，ジョブ管理システムが 1 つの計算ノードに割り当てるジョブのプロセスを 1 つに制限することで，ネットワーク資源を使用しないプロセス間通信を抑制する．また，本実験で使用するジョブはすべて，ジョブの実行時間が，基本的に割り当てられたネットワーク資源の性能に依存する Network インテンシブジョブを用いる．そのようなジョブで実行するプログラムとして，本実験で用いるジョブでは 100 MiB のデータをプロセス間で相互に交換するプログラムを採用する．具体的には，ジョブにおけるデータ交換処理として，本プログラムでは，ジョブの各プロセスが持つ 100MiB のデータに対して MPI_Alltoall を使用した全プロセス間でのデータ交換を 10 回実行する．MPI_Alltoall は分散並列計算で広く利用されている MPI ライブラリが提供する関数の 1 つであり，分散並列計算の全プロセスが自身の持つデータの該当部分を全プロセスに対して送信する処理である．本論文では，ジョブで実行する分散並列計算として，クラスタシステムで標準的に使用されている MPI を用いたプログラムを想定しているため，実験ではジョブのプロセス間における通信に MPI ライブラリを使用するプログラムを採用する．プログラム構築環境には，コンパイラに gcc-4.4.6，MPI ライブラリには OpenMPI 1.5.4 を使用した.

実験は OGS/GE 及び SDN-enhanced JMS フレームワークに対して任意の並列数を要求する，30 個のジョブで構成されたジョブセットを投入することで行う．並列数については，基本的に 2 個以上のジョブが同時に実行されるようにするため，2 並列から 14 並列までの同じ乱数列を用いる．ジョブセットにおける各ジョブの並列数を表 3.2 に示す．なお，各ジョブ管理システムへのジョブの投入はジョブ番号の順に実施するが，実行順序はジョブ管理システムのジョブ選択機能によって決定される．本実験では各ジョブ管理システムとも OGS/GE のジョブ選択機能を利用する.

3.4.3.2 遅延考慮ポリシでの性能評価

OGS/GE に対して 3.4.3.1 節で述べたジョブセットを投入した場合と，提案するジョブ管理システムフレームワークに対して遅延考慮ポリシ “hop” を要求してジョブセットを投入した場合について，各ジョブの実行時間の測定を行った．

各ジョブの実行時間の比較結果を図 3.8 に示す．図 3.8 におけるジョブ番号とは，ジョブセット内における各ジョブの並び順を示す番号であり，ジョブ管理システムにはこの番号順で投入される．なお，本評価実験では，ジョブセットを一度にジョブ管理システムに投入するためジョブの投入順序と実行順序は必ずしも一致しない．ただし，ジョブの実行順の決定は両ジョブ管理システムとも OGS/GE が持つジョブ選択機能によって行われるため，資源割当を行うジョブの選択条件は同じである．

図 3.8 より，提案するジョブ管理システムフレームワークによって資源を割り当てられたジョブの方が実行時間を縮減できたことが確認できた．実行時間の縮減は平均で 44.8% であり，その効果は通信量が多くなる並列数が大きいジョブほど顕著に表れた．この結果は，提案するジョブ管理システムフレームワークによる資源割当が，OGS/GE による資源割当よりも図 3.7 におけるリンク L1〜L4 での輻輳の発生を抑制したためと考える．

OGS/GE による資源割当では計算ノード間の遅延を考慮せずに計算資源をジョブに割り当てるため，異なる OpenFlow スイッチに接続された計算ノードの組み合わせになる場合が多く生じる．これにより，OpenFlow スイッチ SW1 を経由する通信が多くなり，リンク L1〜L4 で輻輳を発生させたと考える．一方，遅延考慮ポリシでは，計算ノード間のホップ数を小さくするため，同じ OpenFlow スイッチに接続された計算ノードを優先的に割り当てる．これにより，SW1 を経由した通信が減少するため，スイッチ間のリンクにおける輻輳の発生が抑えられたと考える．

また，遅延による影響が出やすい小さなメッセージサイズでデータ交換を行うジョブを用いて，遅延考慮ポリシの効果について評価を行った．ジョブで実行するプログラムには，前述の評価プログラムに対して取り扱うデータサイズを 1 KiB に，実行回数を 5,000 回に変更したプログラムを使用した．なお，実行プログラムにおける変更以外は，前述の 100 MiB のデータサイズにおける評価実験と同じ条件で実施する．その結果を図 3.9 に示す．

図 3.9 より，小さなメッセージサイズを扱うジョブにおいても，遅延考慮ポリシに基づいて資源を割り当てた場合の方が実行時間を縮減できることを確認できた．そのジョブの実行時間の縮減率は平均で 15.3% であった．なお，100 MiB のデータを用いた場合と比較するとジョブの実行時間の縮減率が落ちている理由としては，1 KiB のデータでの実験におけるジョブの通信処理の時間が 100 MiB の場合より短く，ジョブの実行時間に占める通信処理の時間の割合が少ないため，通信処理の時間の縮減によるジョブの実行時間へ

の影響が少なくなったためと考える．実際，30 ジョブにおけるジョブの実行時間に占める通信処理の時間の割合は，100 MiB の実験では多くのジョブで 90% 以上を占めていたのに対し，1 KiB の実験では 45% 〜60% 程度であった．

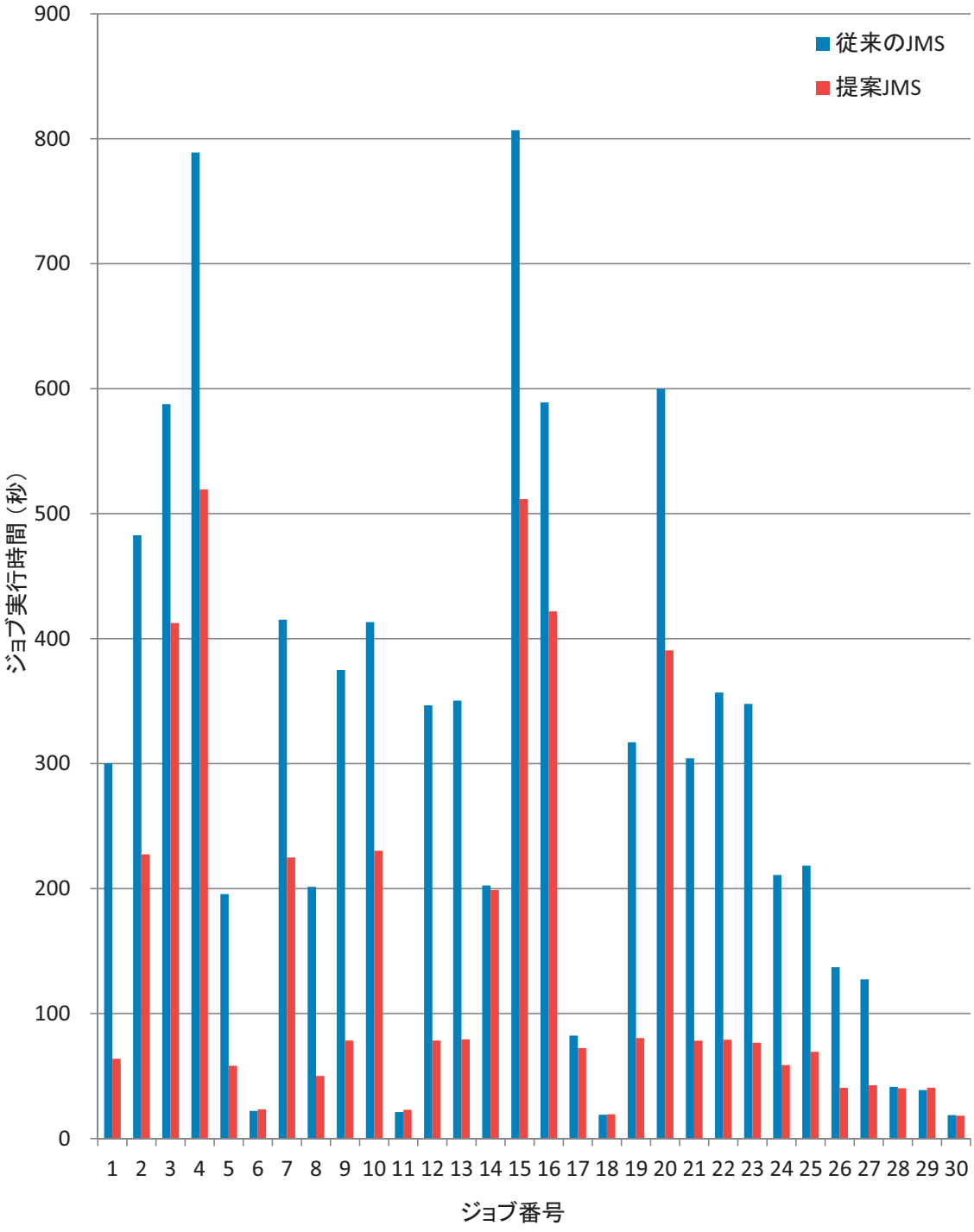


図 3.8　遅延考慮ポリシにおける各ジョブの実行時間の比較（100MiB）.

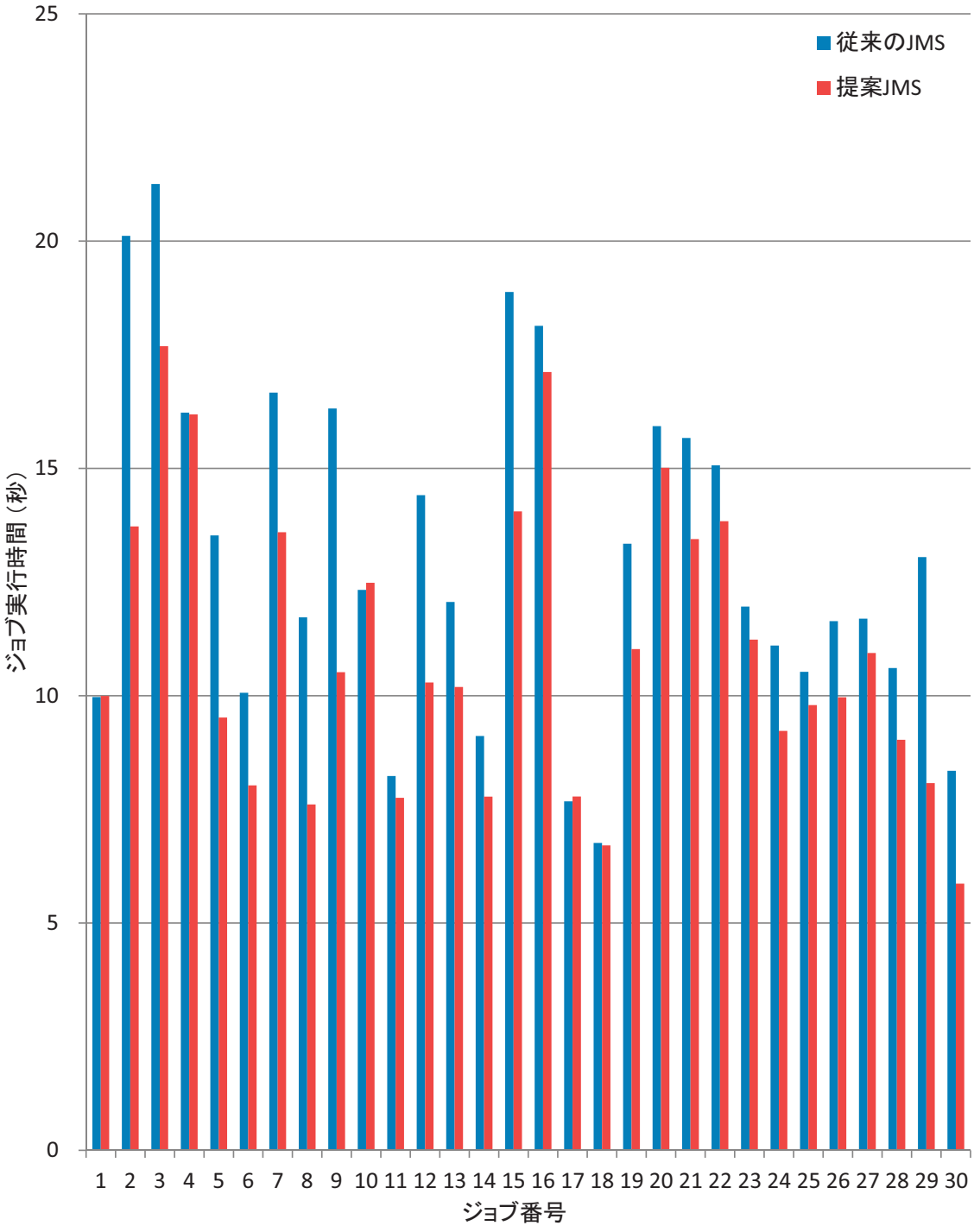


図 3.9　遅延考慮ポリシにおける各ジョブの実行時間の比較（1KiB）.

3.4.3.3 帯域考慮ポリシでの性能評価

次に, 提案するジョブ管理システムフレームワークに対して帯域考慮ポリシ “bandwidth” を要求した場合における各ジョブの実行時間を，従来のジョブ管理システムの場合と比較した．その結果を図 3.10 に示す．この資源割当ポリシを用いた場合も各ジョブにおける実行時間が縮減されていることが確認でき，その差は平均で 30.7% であった．その傾向は遅延考慮ポリシを適用した場合と類似している．これは，帯域考慮ポリシによる割当資源の決定においても，割り当てる計算ノードが同じ OpenFlow スイッチに集約されるよう選択されるためである．OpenFlow スイッチ間のリンク L1〜L4 は複数の通信で共有されるため，各リンクにおける通信量は大きい場合が多い．本資源割当ポリシでは利用可能帯域でジョブに割り当てる計算ノードを選択するため，OpenFlow スイッチ SW1 を経由する計算ノードの組み合わせをジョブに割り当てる可能性は低くなる．その結果，同じ OpenFlow スイッチに接続された計算ノードの組み合わせで資源割当が実行されるため，遅延を考慮した場合と似た結果になったと考える．

また，遅延考慮ポリシの実験と同様に，帯域考慮ポリシにおいてもデータサイズに 1 KiB を用いた小さなメッセージサイズでデータ交換を行うジョブによる実験を行った．実験の条件およびジョブで実行するプログラムは 3.4.3.2 節と同様である．その結果を図 3.11 に示す．

図 3.11 から，帯域考慮ポリシにおいても，提案する SDN-enhanced JMS によるジョブへの資源割当の方が実行時間を縮減できることを確認できた．そのジョブの実行時間の縮減率は平均で 6.0% であった．100 MiB のデータに比べて実行時間の縮減率が減少した理由は遅延考慮ポリシの場合と同様であると考える．また，遅延考慮ポリシによる実行時間の縮減率よりも低下した理由としては，帯域考慮ポリシは資源割当時の利用可能帯域幅の情報に基づきジョブに割り当てる計算ノードを決定しているため，ジョブが利用する帯域幅が小さい本実験では適切な通信経路を持つ計算ノードの組み合わせをジョブに割り当てることができていなかった可能性が考えられる．図 3.9 と比較した場合，他のジョブが実行中に資源割当が行われた後半のジョブにおける実行時間は，図 3.11 では遅延考慮ポリシの場合に比べてあまり縮減されていない．

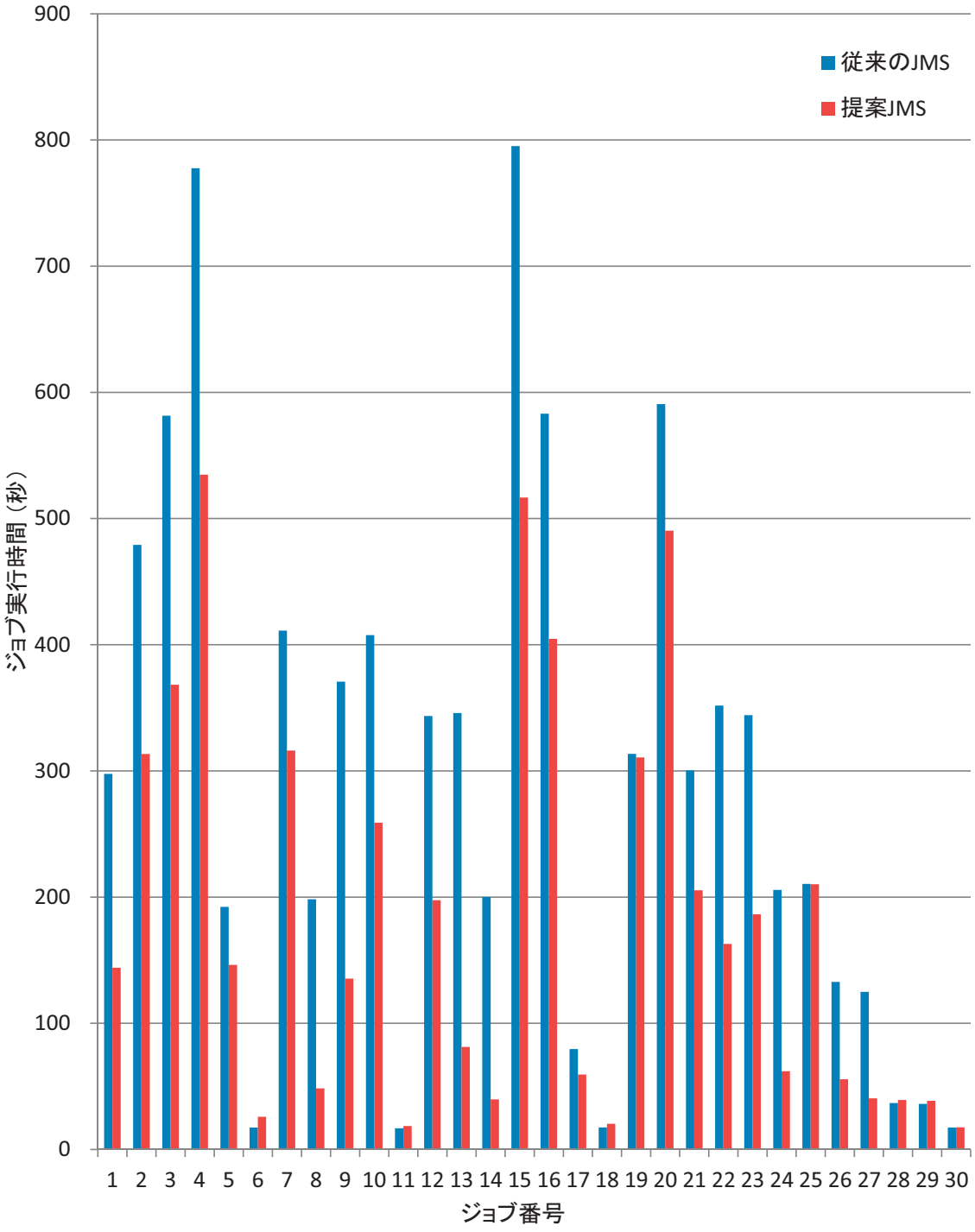


図 3.10　帯域考慮ポリシにおける各ジョブの実行時間の比較（100MiB）.

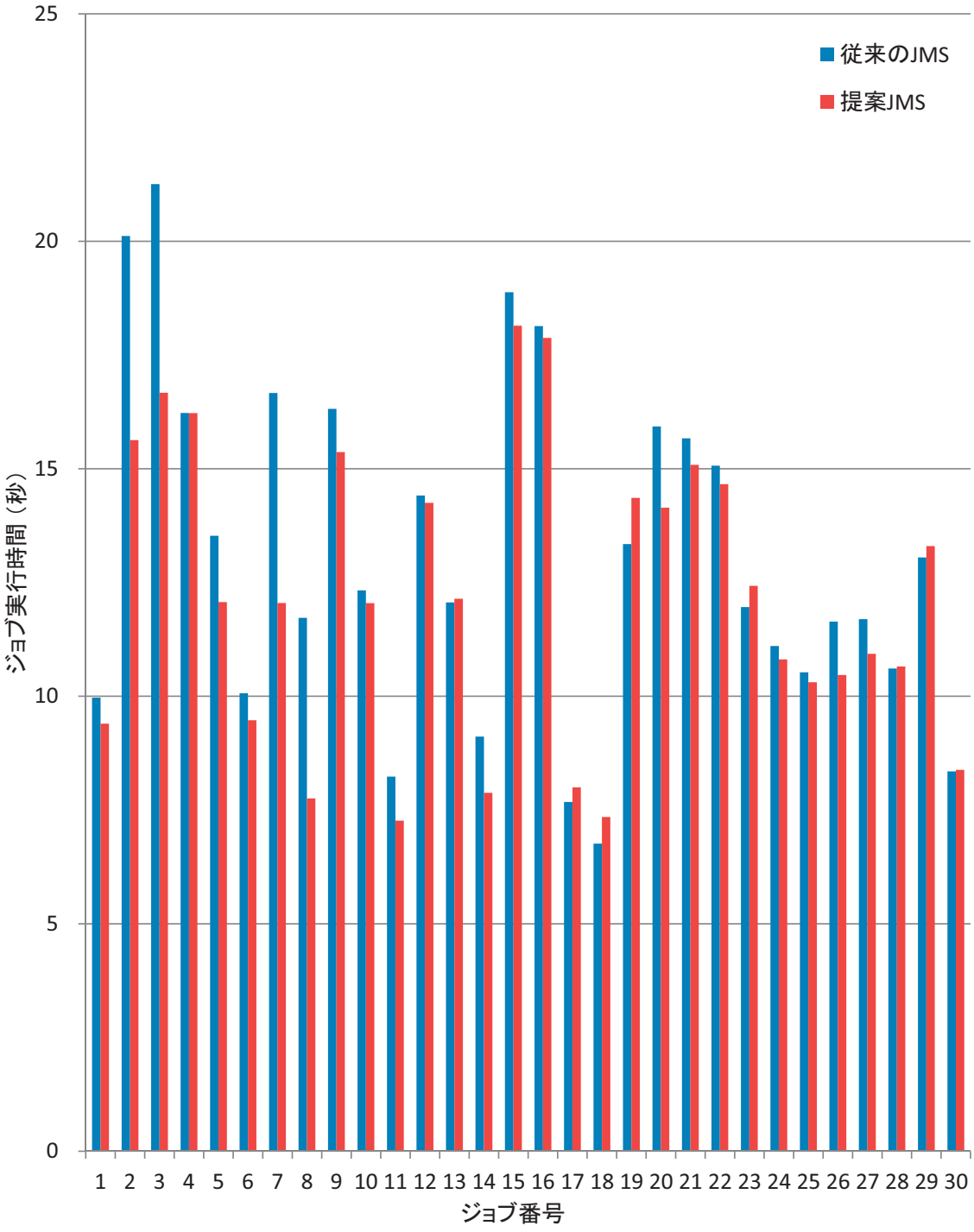


図 3.11　帯域考慮ポリシにおける各ジョブの実行時間の比較（1KiB）.

3.4.3.4 ジョブへの計算ノードの割当

本節では，提案する SDN-enhanced JMS フレームワークにより，インターコネクトの利用状況を考慮したジョブへの計算ノードの割当が行われていることを確認する．3.4.3.2 節及び 3.4.3.3 節の実験において，1 回のジョブセットの投入での各ジョブ管理システムによるジョブへの計算ノードの割当結果を図 3.12 〜図 3.14 に示す．図 3.12 は OGS/GE を，図 3.13 は遅延考慮ポリシを適用した場合，図 3.14 は帯域考慮ポリシを適用した場合のジョブへの計算ノードの割当結果である．図 3.12 〜図 3.14 では，縦軸は本実験で使用した PC クラスタシステムの各計算ノード番号を，横軸は時間を示す．なお，各グラフ中に描かれた破線は，同一 OpenFlow スイッチに接続された計算ノードの範囲を示す．

図 3.12 と比較すると，図 3.13 及び図 3.14 では，1 つのジョブが同一 OpenFlow スイッチに接続された計算ノードに集約して割り当てられる傾向にあることが確認できる．その結果，どちらの資源割当ポリシを適用した場合にも，従来のジョブ管理システムによる計算ノードの割当よりも図 3.7 の SW1 を経由した通信を行う頻度は低かったと考えられる．それゆえ，PC クラスタシステムのスイッチ間のリンクにおける輻輳の発生を抑制でき，3.4.3.2 節及び 3.4.3.3 節における各ジョブの実行時間の短縮が実現できたと考える．

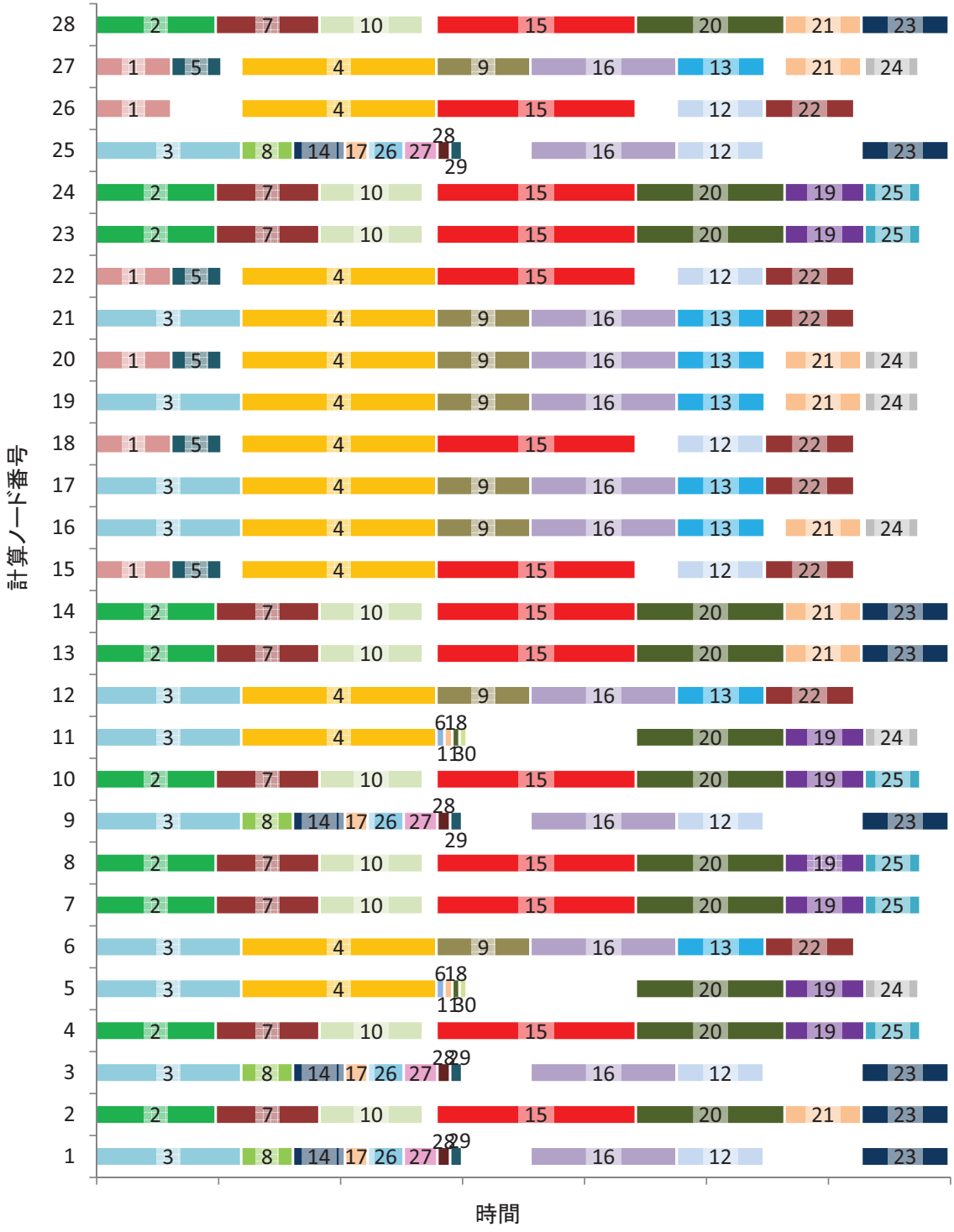


図 3.12　ジョブへの計算ノードの割当結果（OGS/GE）.

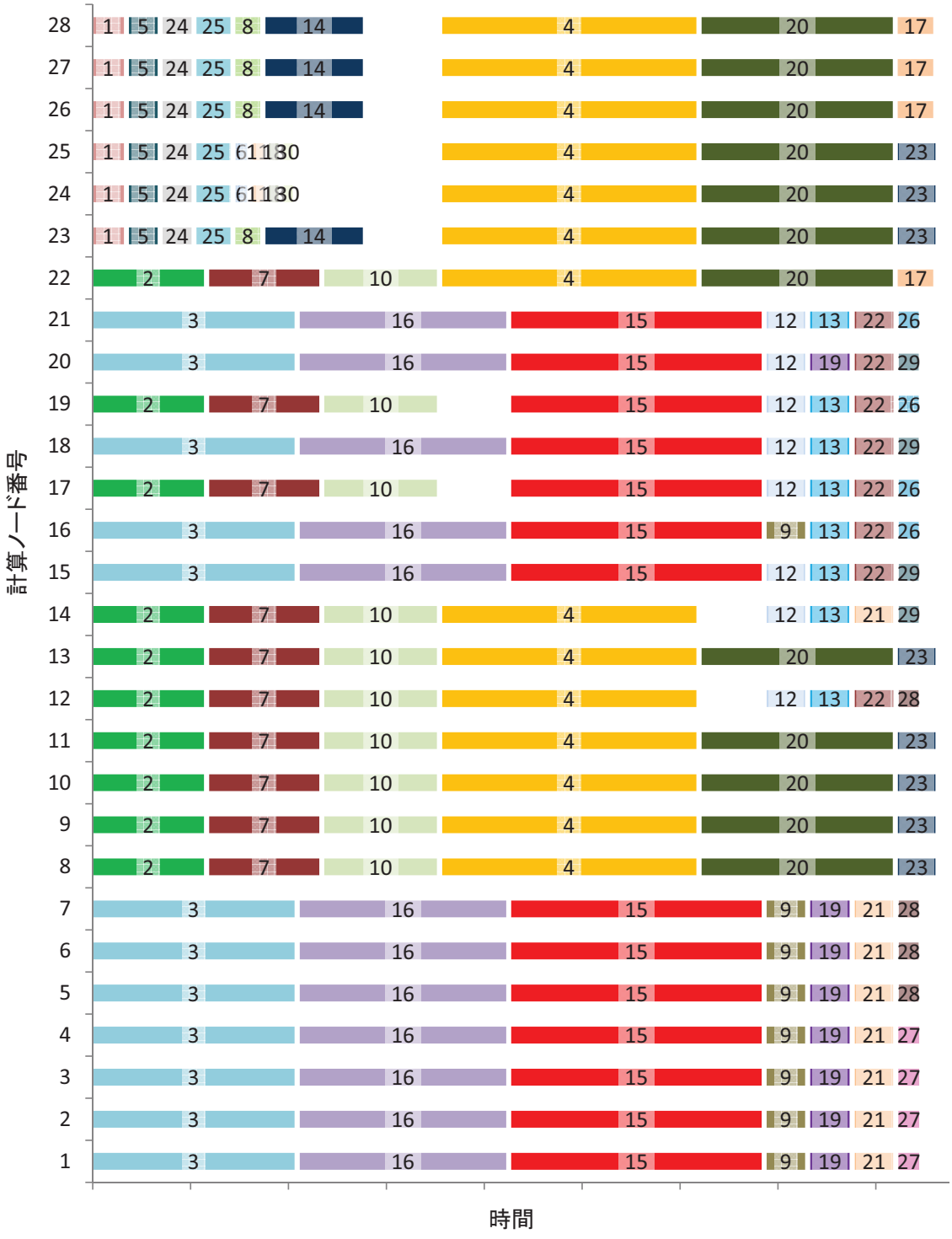


図 3.13　ジョブへの計算ノードの割当結果（遅延考慮ポリシ）.

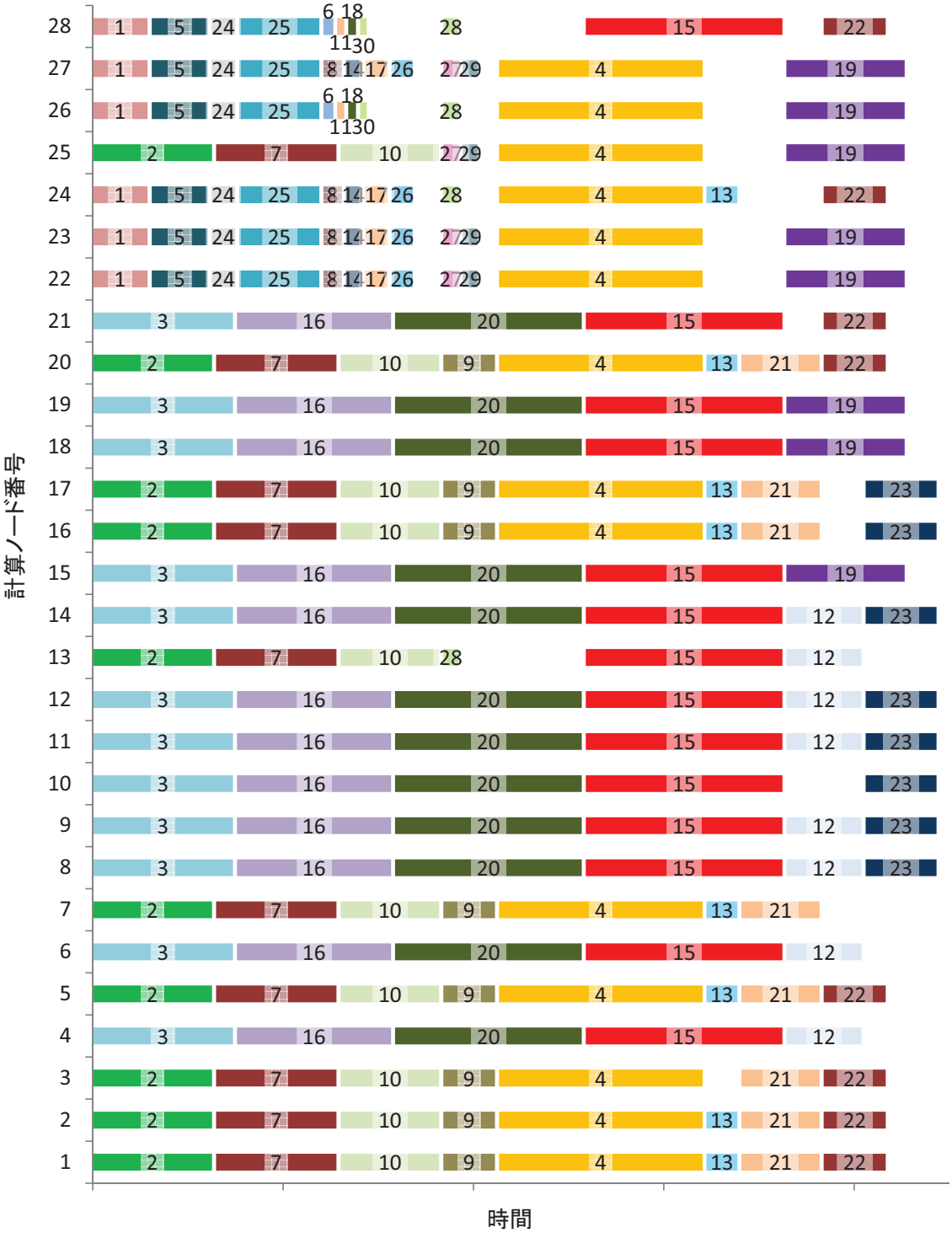


図 3.14　ジョブへの計算ノードの割当結果（帯域考慮ポリシ）.

3.4.4 NAS Parallel Benchmarks による性能分析

クラスタシステム上で実行される実際のアプリケーションへの SDN-enhanced JMS フレームワークによる資源割当の有効性を検証するため，基本的な分散並列計算を用いたベンチマークプログラムである NASA Advanced Supercomputing（NAS）Parallel Benchmarks [55, 56] を実行するジョブを用いて各ジョブ管理システムにおける実行性能を評価した．以下，NAS Parallel Benchmarks の概要について述べた後，実験結果について検証する．

3.4.4.1 NAS Parallel Benchmarks

NAS Parallel Benchmarks は，NASA（National Aeronautics Space Administration）Ames Research Center で開発された, HPC 分野でよく知られている分散並列計算のベンチマークスイートである．NAS Parallel Benchmarks は，5 種類の Parallel Kernel Benchmarks（EP，MG，CG，FT，IS）と，3 種類の Parallel CFD（Computational Fluid Dynamics）Application Benchmarks（LU，SP，BT）で構成されている．また，各ベンチマークは計算で使用する問題サイズを 5 つのクラス（A，B，C，W，S）から選択することができる．

NAS Parallel Benchmarks の各ベンチマークで行う計算の内容と特性について概説する．EP（Embarrassingly Parallel benchmark）は，ガウス分布に従う擬似乱数を用いて 2 次元統計情報を処理する並列プログラムである．このベンチマークプログラムの特性として，このベンチマークは並列計算のプロセス間での通信をほとんど行わないことがあげられる．MG（Multigrid benchmark）は 3 次元ポアソン方程式を V-Cycle マルチグリッド法を用いて解く並列プログラムである．MG の処理は 4 つのフェーズで構成されており，各フェーズの終了時にプロセス間で通信を行う．CG（Conjugate Gradient benchmark）は，大規模疎行列の最小固有値を共役勾配法を用いて算出する並列プログラムであり，行列ベクトル積の性能評価に用いられる．FT（3-D FFT PDE benchmark）は，3 次元 FFT（Fast Fourier Transform）を用いて偏微分方程式を解く並列プログラムである．このプログラムでは，各プロセスでの FFT 計算終了後に行われるデータ交換および 3 次元 FFT 後のチェックサム計算において通信が発生する．IS（Integer Sort benchmark）は，バケットソートアルゴリズムを用いて整数データのソートを行う並列プログラムである．このプログラムもデータ交換のために通信が発生する．Parallel CFD Application Benchmarks の LU（Lowerupper Diagonal），SP（Scalar Pentadiagonal），BT（block tridiagonal）は，一般的な流体力学計算のシミュレーションベンチマークであり，実行性能が通信性能に大きく影響されることが知られている．

これら 8 つのベンチマークにおいて，EP を除いた各ベンチマークには，実行でき

る並列数に制限がある．MG，CG，FT，IS，LU の 5 つのベンチマークでは並列数が 2^n $(n = 1, 2, 3...)$，SP と BT の 2 つのベンチマークでは n^2 $(n = 1, 2, 3...)$ である必要がある．

3.4.4.2 NAS Parallel Benchmarks による評価実験

本実験で使用した NAS Parallel Benchmarks は Ver.3.3.1 であり，問題サイズには Class B を採用した．なお，NAS Parallel Benchmarks を構成する 8 つのベンチマークには，3.4.4.1 節で述べたように，それぞれ実行できる並列数に制限がある．また，各ジョブ管理システムの資源割当によるジョブへの資源割当の効果を評価するため，複数のジョブが同時に実行される必要がある．そこで，本実験ではこれらの条件をすべて満たす 4 並列のジョブのみを用い，1 試行で投入するジョブセットは本評価環境で同時実行が可能である 7 ジョブで構成する．同じ並列数のジョブで構成されたジョブセットを投入した場合，初めに割り当てられた計算ノードの組み合わせのまま順次ジョブが実行される可能性が高い．そこで，様々な計算ノードの組み合わせで評価するためジョブセットは初期に割り当てられるジョブ数を 1 試行とし，これを繰り返して JMS の資源割当機能によるジョブの実行性能差を評価する．

各ジョブ管理システムにおけるベンチマークの平均実行時間を図 3.15 に示す．図 3.15 より，提案する SDN-enhanced JMS フレームワークによって資源を割り当てられたジョブの方が，いずれのベンチマークにおいてもその平均実行時間は同等，または短縮できたことが確認できた．特に，FT ベンチマークと BT ベンチマークで平均実行時間が大きく縮減されており，その縮減率は FT で約 18%，BT で約 10% であった．本実験において，遅延考慮ポリシと帯域考慮ポリシによる資源割当での平均実行時間の差は現れなかった．これは，本実験のジョブは 4 並列であることから，どちらの資源割当ポリシによるジョブへの計算ノードの割当においても，スイッチ間のリンク上で通信衝突が発生したためと考える．

EP ベンチマークについては，3.4.4.1 節で述べたように，プロセス間の通信をほとんど行わないため，各ジョブ管理システムにおける資源割当の違いに起因する実行時間の差は現れないはずである．しかし，本実験における結果では，提案する SDN-enhanced JMS フレームワークの両方の資源割当ポリシにおいて平均実行時間が短くなるとの結果となった．しかし，これらのジョブ平均実行時間の差はどちらの資源割当ポリシを用いた場合においても約 2 秒程度であり，これは平均実行時間の約 2% であることから，誤差の範囲であると考える．IS ベンチマークと SP ベンチマークについては，ジョブの平均実行時間が短いことから，その時間に占める通信時間も短く，提案する SDN-enhanced JMS フレームワークの資源割当による通信時間の削減効果が少なかったと考えられる．

以上より，提案する SDN-enhanced JMS フレームワークによるネットワーク資源を考

慮した計算資源割当は，実アプリケーションに近いプログラムにおいても有効であること
を確認できた.

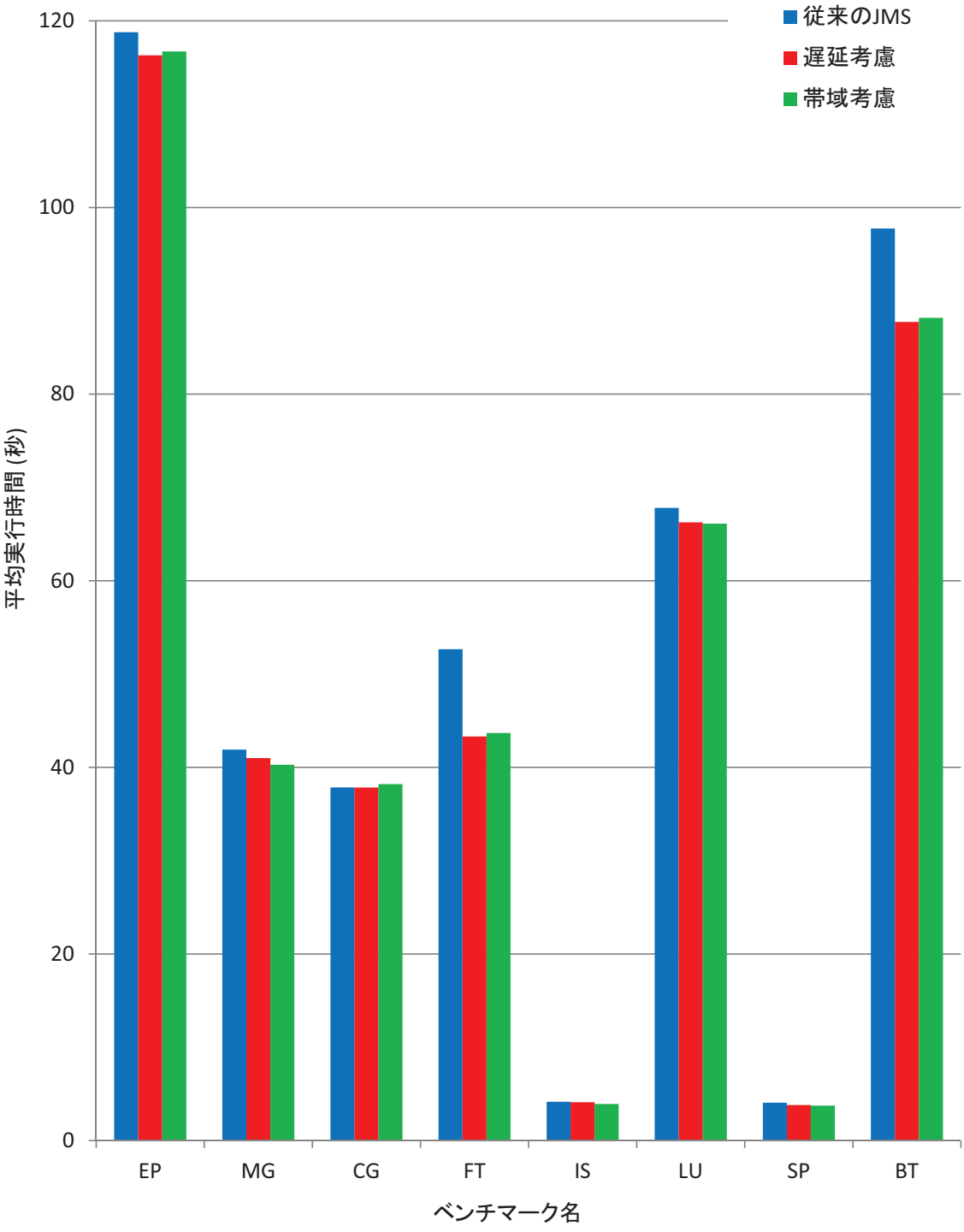


図 3.15　NAS Parallel Benchmarks による比較.

3.4.5 大規模クラスタシステムへの適用に向けた課題

本節では，提案した SDN-enhanced JMS フレームワークの大規模クラスタシステムへの適用について議論する．提案した SDN-enhanced JMS フレームワークを数千・数万の計算ノードで構成されるような大規模クラスタシステムに適用することを想定した場合，インターコネクトにおけるスイッチ数やリンク数は大規模かつ複雑になると考えられる．このような計算環境で実行されるジョブは，より多くの計算ノードを利用した分散並列計算になると考えられるため，ジョブの実行性能は割り当てられた計算ノード間における通信性能に大きく影響されると考えられる．それゆえ，割り当てるネットワーク資源を制御して高い通信性能を提供することによる効果はより高くなると考えられる．一方，要求に適したネットワーク資源の探索処理のオーバーヘッドはネットワーク資源の規模が大きくなるほど増加することが想定されるため，資源情報の管理や最適な割当資源の探索手法に関してさらなる工夫が必要になると考える．

本章で提案した SDN-enhanced JMS フレームワークの実装において，最適な割当資源の探探索処理のオーバーヘッド時間は，割当候補リストの計算ノード数，ジョブが要求する計算ノード数，計算ノード間の通信経路数やその経路におけるホップ数を決めるインターコネクトのトポロジ構造によって決定されると考えられる．本章で評価に使用した遅延考慮ポリシと帯域考慮ポリシの場合，割当候補リストの計算ノード数とジョブが要求する計算ノード数が割当資源の決定に要する時間に影響するパラメータとなる．3.4.3 節の実験において，連続してジョブが割り当てられた計算ノードにおける前のジョブの終了時間と次のジョブの開始時間の差を算出したところ，OGS/GE では平均 118.8 秒であったのに対し，SDN-enhanced JMS フレームワークでは 188.7 秒となり，その差は 69.9 秒で 58 % 増加している．その一方で，3.4.3.2 節の 100 MiB のデータを用いた実験では平均 44.8% のジョブの実行時間縮減ができているため，単純に計算した場合，156 秒以上の通信処理を行うジョブであれば，ジョブが計算ノードを専有する時間を削減できる．

3.5 おわりに

本章では，従来のジョブ管理システムとして採用した OGS/GE における計算資源管理に対して，SDN の一実装である OpenFlow を利用したネットワーク制御機能を拡張することによって，計算環境におけるネットワーク資源と計算資源をともに管理し，両資源の利用状況やユーザの要求を考慮した資源割当を実現する SDN-enhanced JMS フレームワークを提案・実装した．提案・実装した SDN-enhanced JMS フレームワークでは，システム管理者が資源割当ポリシクラスモジュールを編集することによって，ユーザの資源要

求に対する柔軟な資源割当制御を実現した．SDN-enhanced JMS フレームワークの評価では，シンプルなトポロジである Flat-tree インターコネクトを有するクラスタシステムをターゲットとして，資源割当ポリシに遅延及び帯域を考慮した 2 種類のポリシを定義し，従来のジョブ管理システムとの資源割当の違いによるジョブの実行性能の差を比較した．その結果，SDN-enhanced JMS フレームワークによるネットワーク資源を考慮したジョブへの資源割当は，従来のジョブ管理システムの場合より高い実行性能を得られる資源を提供することが確認できた．これにより，ネットワーク資源の動的な制御による資源管理手法の実現可能性を検証できたと考える．

第 4 章

Fat-tree インターコネクトを持つクラスタシステムのための資源割当ポリシ

4.1 はじめに

クラスタシステム上で分散並列計算を効率的に実行するためには，最適な計算資源とネットワーク資源がユーザの計算要求に割り当てられる必要がある．前章では，計算資源とネットワーク資源をともに管理・制御するための Network-aware ジョブ管理システムフレームワークとして SDN-enhanced JMS フレームワークを提案・実装し，単純な Flat-tree トポロジのインターコネクトを有するクラスタシステム上でその実現可能性を評価した．しかし，前章で提案した SDN-enhanced JMS フレームワークが現実的なインターコネクトで構成されるクラスタシステム上で，ユーザの計算要求に対して効率的に資源を割り当てられるかは検証されていない．このことは，ネットワーク資源を動的に制御するジョブ管理システムの有用性および実用性を評価する上で不可欠である．ネットワーク資源である計算ノード間の通信経路では，分散並列計算の実行性能に影響を与える遅延や帯域幅など複数の性能指標を考慮する必要があるため，計算資源の場合に比べてジョブに割り当てる資源の判定基準が複雑となる．そこで，本章では，冗長経路を持つインターコネクトとして，実際のクラスタシステムの多くが採用する Fat-tree インターコネクトを持つクラスタシステムを対象とし，高い性能を得られる計算資源およびネットワーク資源をジョブに割り当てる資源割当ポリシを設計・実装した上で，SDN-enhanced JMS フレームワークの有用性・実用性を評価する．

近年のクラスタシステムでは，インターコネクトにおける通信の負荷分散と広帯域化・耐障害性の観点から，冗長経路を持つトポロジで構築されたインターコネクトを採用して

いる．冗長経路を持つインターコネクトは，当然，単一経路のトポロジで構成されたインターコネクトよりも多くのリンクを持つため，トポロジ構造が複雑になる．その一方で，計算ノード間に複数の通信経路を有するため，各計算ノードからの通信を複数の通信経路に分散することで，広帯域の確保および通信衝突の回避ができるというメリットがある．

冗長経路を持つインターコネクト上では複数の通信経路への負荷分散を行う必要があるが，通常，その制御は各スイッチが持つ機能により実現される．そのため，スイッチにおける冗長経路への通信の負荷分散は，ジョブ管理システムにおける計算資源の割当とは独立して制御される．例えば，冗長経路への通信の負荷分散の制御が，各計算ノードが持つネットワーク機器の情報に基づいて静的に行われる場合，従来のジョブ管理システムでは，インターコネクト上の一部のリンクに通信が集中するような偏った計算資源にジョブを割り当てるような状況を引き起こす可能性がある．このとき，通信が集中したリンクを含む通信経路を利用する計算ノードが割り当てられたジョブは，そのリンク上で必要な帯域幅を得られず，他の通信との衝突も発生するため，高い通信性能を得ることができない．また，このように一部のリンクに通信が偏った状況は，インターコネクト全体における通信の負荷分散の観点からも非効率な状態となる．それゆえ，個々のジョブに高い実行性能が得られる資源を提供し，システム全体を効率的に運用するためには，通信衝突が発生するような状況を回避する必要がある．

本章では，冗長経路を持つ実際的なクラスタシステムにおいて，各計算ノード間の冗長経路およびインターコネクト全体に対して効率的な通信の負荷分散について述べる．負荷分散の実現方法として，SDN-enhanced JMS フレームワークによるジョブへの計算ノードおよび計算ノード間の通信経路の割当制御を行う．また，ネットワーク資源を計算資源と同様に動的に制御するジョブ管理システムの有用性および実用性を評価・検証する．冗長経路を有するインターコネクトとして，本章では今日多くのクラスタシステムで採用されている Fat-tree トポロジで構成されたインターコネクトを対象とする．

以下，本章の構成を示す．4.2 節では，本章で想定する Fat-tree トポロジ構成のインターコネクトを持つクラスタシステム上で回避すべき資源割当状況を示し，SDN-enhanced JMS フレームワークに配備する資源割当ポリシに求められる処理を分析する．4.3 節では，4.2 節の分析結果に基づき，資源の利用状況を考慮した計算ノードの割当および計算ノード間の通信経路の選択を行う資源割当ポリシを提案し，設計・実装について説明する．4.4 節では，提案する資源割当ポリシに対する評価実験を行い，得られた結果からその有用性および実用性について議論する．最後に，4.5 節で本章のまとめを行う．

4.2 問題分析

本節では, 冗長経路を有するインターコネクトの代表例として本章で対象とした Fat-tree トポロジのインターコネクトについて，特徴と回避すべき資源割当状況に関して分析する．分析した結果から，Fat-tree インターコネクトを対象とした資源割当ポリシに求められる機能要件を導く．

冗長経路を持つインターコネクトでは計算ノード間に複数の通信経路を持つため，計算ノード間の通信をどの経路を用いて行うのかを決定する必要がある．通常，この計算ノード間における複数の通信経路への分散制御は，経由する各スイッチに実装された機能を用いて行われる．それゆえ，利用可能な冗長経路への分散制御手法はインターコネクトの構築に使用するスイッチに依存するため，一般的なインターコネクトでは，同じ種類のスイッチで構成されている．

基本的な冗長経路への負荷分散制御手法として，通信の送信元および送信先となる計算ノードが持つ IP アドレスや MAC アドレスのようなネットワークの情報に基づいて使用する通信経路を決定する静的ルーティング手法や，ラウンドロビン方式による通信回数が均一になるように通信経路を振り分ける手法がある．しかし，これらの手法では必ずしも発生する通信を効率的に分散できる訳ではなく，複数の通信経路への振り分けが一方に偏り，計算ノード間における通信性能の低下を引き起こす場合がある．

このような通信性能の低下を回避し，計算ノード間の通信を効率的に分散させて高い通信性能を得ることを目的とし，これまでに様々な負荷分散アルゴリズムの研究が行われている．例えば，文献 [57] では，Fat-tree トポロジで構成されたインターコネクトにおいて，OpenFlow を用いた動的ルーティング技術を提案している．その負荷分散アルゴリズムでは，冗長経路内の各リンクの利用状況を OpenFlow により把握して，最適な通信経路を選択する．この手法では任意の計算ノード間における動的な負荷分散を実現しており，本研究における通信経路の選択手法の参考となる．しかし，本研究では，インターコネクトの利用状況を考慮した計算ノード間の冗長経路への負荷分散の制御だけでなく，通信を行う計算ノードの配置も併せて制御することでインターコネクト全体の通信を管理可能とするネットワーク資源を動的に制御するジョブ管理システムの実現を目的としている点が異なる．

冗長経路を持つインターコネクトのトポロジとして，Fat-tree は今日のクラスタシステムで広く採用されている．Fat-tree は間接網に分類されるツリー型トポロジの 1 つであり，木構造の葉から根に向けて広帯域にすることで，Flat-tree が持つ根側のリンクにおける帯域幅に関するボトルネックを改善する．この広帯域の確保は，より大きな帯域幅を得られるスイッチを根側に使用することでも実現できるが，複数のリンクやスイッチを用いて多

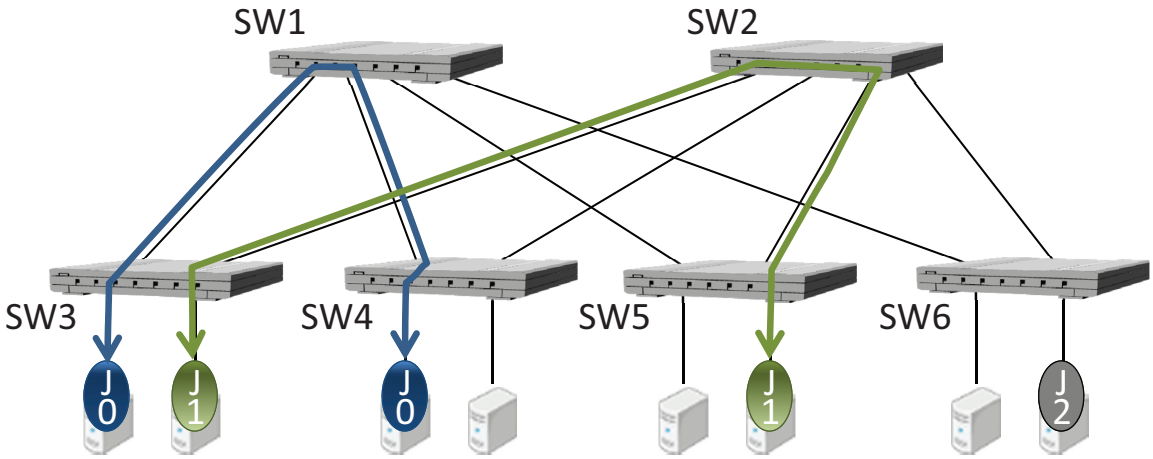
重化した構成にすることは，耐障害性の向上に繋がる．しかし，このように多重化した構成では，より多くのスイッチとケーブルが必要となるため，構築コストや設置スペースが増加する．

上述の Fat-tree トポロジで構成されたインターコネクトを持つクラスタシステムを例にして，計算資源の割当制御と冗長経路への分散制御が独立した状況で生じる問題について述べる．まず，3 つのジョブ（*J0*，*J1*，*J2*) が図 4.1(a) のように計算ノードに割り当てられている状況を想定する．ジョブ *J0* とジョブ *J1* は 2 つの計算ノードを使用し，広帯域を要求する Network インテンシブジョブであり，*J2* は 1 つの計算ノード内で処理が閉じているジョブである．このような資源利用状況において，新たに 2 つの計算ノードと広帯域の通信を要求する Network インテンシブジョブ（*J3*）が投入されたとする．ジョブ *J3* への資源割当の結果，図 4.1(b) に示すような計算ノードへのジョブの配置が行われた場合，ジョブ *J3* が行う通信をどのように冗長経路へ分散させても，他のジョブの通信との衝突が発生する．もし，冗長経路への分散制御アルゴリズムがジョブ *J3* の通信を，図 4.1(b) に示す *SW1* を通る *Path 1* に割り当てた場合，その通信は *SW1 – SW4* 間のリンクで，ジョブ *J0* で実行されている通信との衝突が起こる．もう一方の *SW2* を通る *Path 2* に通信経路を割り当てたとしても，今度は *SW2 – SW5* 間のリンクでジョブ *J1* の通信と衝突が発生する．そのため，ジョブ *J3* が使用する経路をどのように変えても，ジョブ *J3* と通信衝突が起こるジョブにおいて通信性能の低下が生じる．しかし，もしジョブ *J3* が図 4.1(c) に示すような計算ノードに割り当てられ，かつ通信経路に *SW2* を経由する経路が割り当てられるなら，ジョブ間での通信の衝突は発生しない．このような資源割当を実現するためには，ジョブに割り当てる計算ノードを決定する際に，計算ノードが使用する通信経路の状況も併せて評価し，計算ノードの選択に反映させる必要がある．

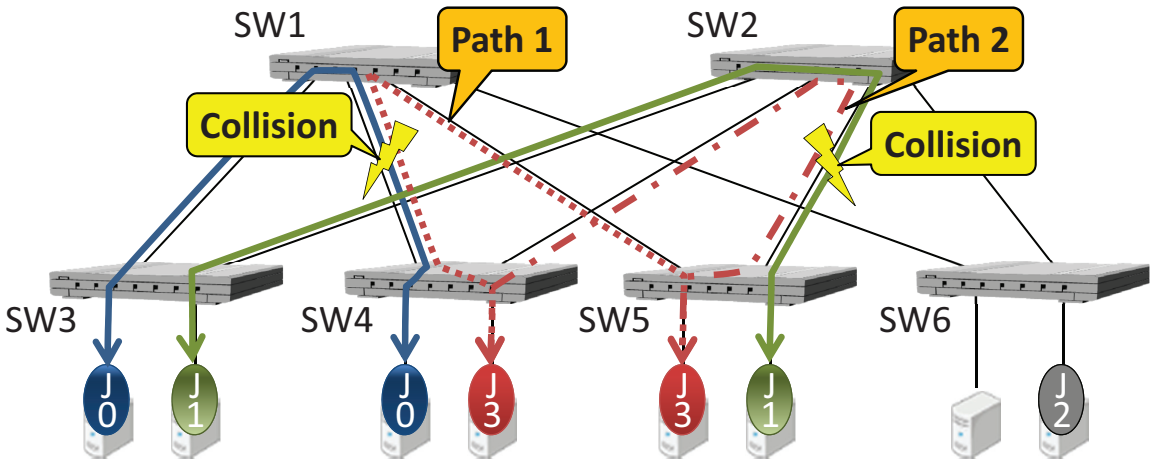
以上の考察より，本章の目的であるジョブへの効率的な計算ノードおよび計算ノード間の通信経路の割当を SDN-enhanced JMS フレームワークで行うための資源割当ポリシでは，他のジョブによる通信との衝突を抑制する計算ノードおよび計算ノード間の通信経路をジョブに割り当てる．そのような資源割当ポリシを実現するにあたり，提案する資源割当ポリシでは以下の要件を満たす必要があると考える．

(1) ネットワーク資源に関する情報の取得
(2) 計算ノードを接続されたスイッチごとに分類
(3) 同一スイッチに接続された計算ノードをできるだけ多く含む計算ノードの組み合わせの選択
(4) 選択した計算ノード間で使用する通信経路を決定

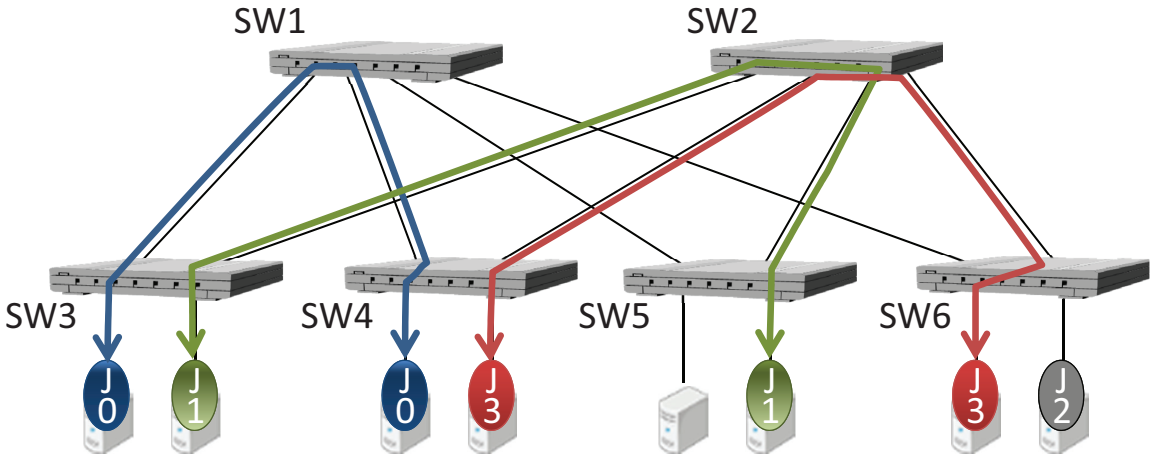
要件（1）は，ジョブに割り当てる計算ノードを決定する際に，割り当てる計算ノード間の通信経路についても考慮するため，必要となるインターコネクトのネットワーク資源



(a) 仮定する資源割当状況 (*J0*, *J1* and *J2*).



(b) 非効率なジョブ *J3* への資源割当.



(c) 非効率なジョブ *J3* への資源割当.

図 4.1 Fat-tree クラスタシステムにおける資源割当例.

情報を取得するための要件である．要件（2）では，取得した Fat-tree インターコネクトのトポロジ構造に基づき，同一スイッチに接続された計算ノードを事前に接続するスイッチごとに分類する．同一スイッチに接続された計算ノード間では通信経路が 1 つとなるため，複数の通信経路から割り当てる経路を選択する必要がない．要件（3）では，ジョブで要求された計算資源量を満たす計算ノードの組み合わせを決定する．その際，要件（2）の結果を利用して，同じスイッチに接続された計算ノードを多く含む構成とする．このような計算ノードの組み合わせにすることにより，他のジョブの通信と共有する可能性のあるスイッチ間のリンクを経由する通信経路の利用を抑制する．要件（4）では，要件（3）で生成した計算ノードの組み合わせに対し，各計算ノード間の通信経路を決定し，各通信経路が持つ資源量から計算ノードの組み合わせにおける優先度を算出する．

4.3　Fat-tree インターコネクトを考慮した資源割当ポリシ

本節では，4.2 節の要件を実現する資源割当ポリシの基本方針と実装について述べる．

4.3.1　基本方針と基本設計

4.2 節の各機能要件を満たす設計方針について述べる．提案する資源割当ポリシの目的は，この資源割当ポリシによるジョブへの計算ノードおよび計算ノード間の通信経路の割当により，各リンクにおける通信衝突の抑制とインターコネクト全体における効率的な通信の負荷分散を実現することである．この目的を達成するにあたり，本章で提案する資源割当ポリシでは，他のジョブの通信との衝突が発生する可能性があるスイッチ間のリンクの利用を抑制した計算ノードの割当を行う．そのような計算ノードの組み合わせとして，提案する資源割当ポリシではジョブに割り当てる計算ノードをできる限り同じスイッチに接続されるよう集約する．Fat-tree インターコネクトでは，2.2.2 節で述べたように計算ノードが持つネットワークインタフェースは 1 つであるため，計算ノードと接続するスイッチの間のリンクは 1 本である．それゆえ，同じスイッチに接続した計算ノードは，それぞれ 1 本のリンクでそのスイッチと接続しているため，その計算ノード間は単一の通信経路となる．また，ジョブ管理システムの設定で，1 台の計算ノードに対して異なるジョブの割当を許可しない限り，その通信経路に他のジョブの通信は発生しないため，他のジョブとの通信衝突を考慮する必要がなくなる．上述の設定は，計算ノードにおける計算資源制御の制限上，多くのクラスタシステムにおける運用で採用されているため，この設定を前提として資源割当ポリシを設計することは，実際的なクラスタシステムへの適用の妨げにならないと考える．以上の基本方針に基づき，インターコネクトの利用状況を考慮して通信経路を割り当てることにより，他のジョブとの通信衝突を回避可能な資源割当ポ

リシを実装する．

資源割当ポリシは，3.3.3.3 節で述べたように，資源割当ポリシクラスモジュール上で Brain コンポーネントにおける 3 つの処理フェーズに対して行う．Brain コンポーネントの各処理フェーズにおける処理内容を資源割当ポリシとして実装するため，資源割当ポリシクラスモジュールにおいて *sort_qlist* 関数，*set_route* 関数，*del_route* 関数の定義を行う必要がある．この 3 つの関数を定義することで，SDN-enhanced JMS フレームワークに資源割当ポリシを配備し，そのポリシに従った資源割当の制御を行うことができる．4.2 節で導出した資源割当ポリシの要件は，すべてジョブに割り当てる計算ノードを決定するための処理であるため，その実装は *sort_qlist* 関数において行う．したがって，ジョブに割り当てる計算ノード間の通信経路に対応した *flow entry* を生成する *set_route* 関数，およびジョブ終了後の後処理を定義する *del_route* 関数については提案する資源割当ポリシ固有の処理の追加は必要ない．

4.3.2 実装

本節では，4.2 節で導出した要件に基づき，インターコネクトの利用状況を考慮して計算ノードおよび割当計算ノード間の通信経路の割当を行う資源割当ポリシの実装について述べる．図 4.2 に提案する資源割当ポリシにおける *sort_qlist* のフローチャートを，図 4.3 に擬似コードを示す．なお，本資源割当ポリシの実装では，資源割当ポリシクラスモジュールに提案する資源割当ポリシのみを配備する．

図 4.2 に従い，提案する資源割当ポリシの実装について説明する．まず，提案する資源割当ポリシでは，ネットワーク資源情報の取得を行う．これは 4.2 節の要件（1）であり，図 4.3 の 2 行目に示す *get_net_info()* を用いる．*get_net_info()* は 3.3.3.3 節で述べた資源割当ポリシクラスモジュールの API で提供される機能であり，Network control コンポーネントからネットワーク資源情報を取得する．現在の SDN-enhanced JMS フレームワークの実装では，本機能で取得可能なネットワーク資源情報は，計算ノードも含めたクラスタシステム全体のトポロジ構造，インターコネクトの各リンクにおける使用帯域幅と最大帯域幅である．また，取得した各リンクにおける 2 つの帯域幅情報から，そのリンクにおける利用可能帯域幅を算出する．

$$(\text{利用可能帯域幅}) = (\text{最大帯域幅}) - (\text{使用帯域幅})$$

次に，OGS/GE から得た割当候補リストの全計算ノードに対し，その計算ノードと接続するスイッチを確認し，スイッチごとに計算ノードを分類する．これは要件（2）に対する実装であり，図 4.3 の 3〜5 行目が該当する．本処理における計算ノードと接続するスイッチの確認には，取得したネットワーク資源情報におけるトポロジ情報を利用する．

本実装では，スイッチごとに分類された計算ノードのリストを指定するための識別子として，取得したトポロジ情報に含まれる各 OpenFlow スイッチの *Datapath ID* を用いる．*Datapath ID* は，OpenFlow ネットワークにおいて，各 OpenFlow スイッチを識別するために設定された一意な値である．また，本処理の際に各スイッチにおける分類された計算ノード数，接続されている計算ノード数，該当スイッチから他のスイッチへのリンク数を取得する．

計算ノードの分類処理が終了した後，計算ノードと接続する各スイッチにおいて，スイッチの *Datapath ID* の値 *d-id* を識別子とした計算ノードのリストを用い，そのスイッチに接続された計算ノードが有する割当可能スロット数の合計値を算出する．

$$sw[d\text{-}id].slots = \sum_{i} sw[d\text{-}id].nodes[i].slot$$

スロットは，2.3.1.2 節で述べたように，各計算ノードに割り当てることができるジョブのプロセス数を定義している．このスロット数の合計値をスイッチごとに求めることにより，そのスイッチに割り当てることができるジョブのプロセス数を把握する．これは，要件（3）における計算ノードの組み合わせを求める処理を，計算ノードが接続されたスイッチ単位で行うためである．次に，計算ノードの組み合わせを，できる限り同じスイッチに接続された計算ノードで構成されるよう生成する．その際，できるだけ多くのジョブが同一スイッチに接続された計算ノードに割り当てられるようにするため，必要最小限のスロット数を持つスイッチに接続された計算ノードに優先的に割り当てる．例えば，図 4.4(a) に示すように，2 つのスイッチにおいて，それぞれのスロット数 1 の計算ノードが 2 台および 4 台が割当可能であると仮定する．このような状況において，2 並列を要求するジョブに割り当てる計算ノードを選択する場合を想定する．もし図 4.4(b) の緑色の計算ノードのように，計算ノードの選択を 4 台 から選択可能なスイッチで行った場合，この計算ノードの選択では，これ以降のジョブに対して 2 並列のジョブしか同一スイッチに接続された計算ノードを提供できない．しかし，図 4.4(c) の赤色の計算ノードのように，2 台の計算ノードが選択可能なスイッチを選択した場合，次のジョブでは 4 並列まで同一スイッチに接続された計算ノードを提供できる．なお，選択可能な計算ノードが各スイッチに 1 台だけとなった場合，同じスイッチに接続された計算ノードを選択することができないため，4.2 節で述べた用件（2）のメリットを得られないため，選択可能なすべての計算ノードの組み合わせを求める．上述の処理は図 4.3 の 8〜26 行目のように実装しており，これにより要件（3）のジョブで要求された計算資源量を満たす計算ノードの組み合わせの生成を実現する．

最後に，選択した計算ノードの組み合わせにおける通信経路の資源量を求め，各計算ノード間に割り当てる通信経路を決定する．本処理におけるネットワーク資源の資源量に

は，各リンクの利用可能帯域幅を用い，3.4.1 節で述べた帯域考慮ポリシと同様に，通信経路を構成する各リンクが持つ利用可能帯域幅の最小値をその通信経路の資源量とする．

$$resource\_value(path) = min(available\_bandwidth(link_i))(link_i \in path)$$

利用可能帯域幅を通信経路における資源量として利用する理由は 2 つある．1 点目は，Fat-tree インターコネクトにおいて，ある一対の計算ノードの組み合わせにおける計算ノード間の通信経路は常に同じホップ数となるためである．それゆえ，計算ノード間の通信経路における資源量にホップ数を利用した場合，冗長経路間の資源量の差が無い状態になるため，通信経路の選択を行うことができない．2 点目は，他のジョブの通信で利用されているリンクでは利用可能帯域幅が小さくなるため，そのようなリンクを経由する通信経路では上述の資源量も低くなる．上述の通信経路を構成する各リンクが持つ利用可能帯域幅の最小値を求める処理を実装するため，本資源割当ポリシではダイクストラアルゴリズムにおけるリンクの走査機能を利用した．ダイクストラアルゴリズムは，インターコネクトを各リンクにおける資源量を重みとする重み付きグラフとみなし，その枝の重みを用いて最短経路を探索する有名なアルゴリズムである．本実装では，枝の重みとして各リンクにおける利用可能帯域幅を用い，最短経路の走査処理で得る各リンクの利用可能帯域幅の最小値を保持することで，通信経路における資源量を算出する．

通信経路の決定について，同一スイッチに接続された計算ノード間では通信経路は 1 つであるため，経路選択処理を省略可能である．また，その単一経路における資源量は 4.3 節で述べた同一計算ノードへのジョブの割当の制限から，常にリンクの最大帯域幅となる．それゆえ，ジョブに割り当てる計算ノードがすべて同一スイッチに接続されている場合，通信経路の選択およびその経路における資源量の算出処理を行う必要がない．それ以外の通信経路における資源量の算出では，まず計算ノードの組み合わせに含まれる要件（2）のスイッチ間で各通信経路における資源量として各リンクにおける利用可能帯域幅の最小値を算出する．これは，計算ノードと接続されたスイッチ間のリンクは単一であり，その利用可能帯域幅は上述のとおり最大帯域幅であるため，計算ノードが接続するスイッチ間の通信経路における利用可能帯域幅と比較するだけで，それぞれのスイッチに接続された計算ノード間の通信経路における資源量を算出できる．この処理により，計算ノードごとに通信経路の資源量を算出するより計算量を削減することができる．

次に，異なるスイッチに接続された計算ノード間において，ジョブに割り当てる通信経路を決定する．各計算ノード間の通信経路の割当は，基本的には各通信経路における資源量である利用可能帯域幅の大きい経路を選択する．同じスイッチ間で複数の通信経路の割当を行う場合，まず割り当てた通信経路の資源量をスイッチ間の経路における資源量として算出した利用可能帯域幅から引く．その後，同様の処理を行っていき，すべての冗長経路において利用可能帯域幅を超える状態となった場合は，各通信経路に対してラウンドロ

ビン方式で割り当てる．

以上の処理により，ジョブに割り当てる計算ノードおよび計算ノード間の通信経路を決定し，割当候補リストの修正を行う．なお，要件（3）の実装において，選択可能な計算ノードが各スイッチに 1 台だけとなった場合は選択可能なすべての計算ノードの組み合わせを求めているため，3.4.1 節の資源割当ポリシと同様に，計算ノードの組み合わせにおける全通信経路の資源量の平均値を求めてその値をその計算ノードの組み合わせにおける優先度とする．この処理をすべての計算ノードの組で行い，最も高い値となった計算ノードの組み合わせを，ジョブに割り当てる計算ノードとして決定する．

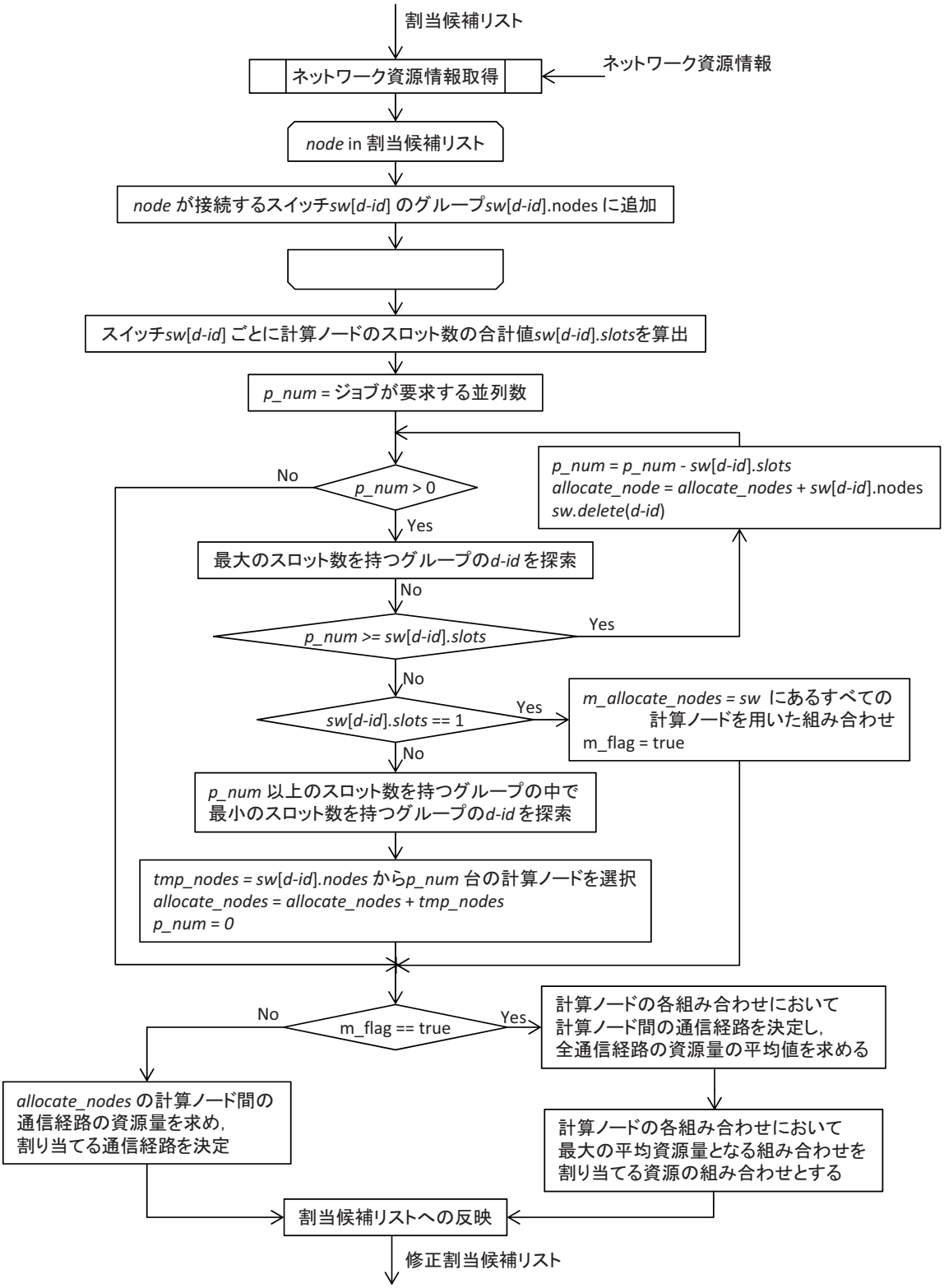
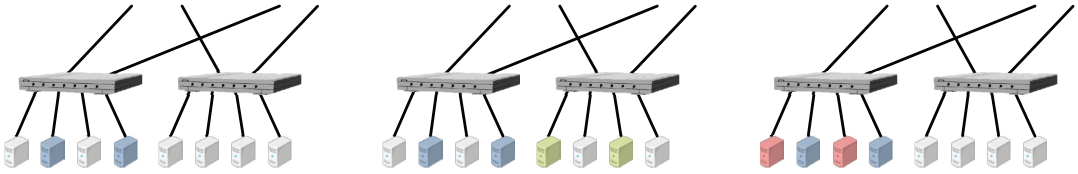


図 4.2 提案資源割当ポリシにおける *sort_qlist* のフローチャート.

```
1.  def  sort_qlist ( picklist ,  job_info )
2.    net_info  =  get_net_info ( )
3.    for node in picklist
4.      categorize_node ( node ,  net_info.topology ,  sw )
5.    end
6.    total_slot_num ( sw )
7.    p_num  =  job_info.process_num
8.    while  p_num  >  0
9.      d-id  =  max_slots ( sw )
10.     if p_num >= sw[d-id].slots && sw[d-id].slots > 1
11.       allocate_nodes = allocate_nodes + sw[d-id].nodes
12.       p_num  =  p_num - sw[d-id].slots
13.       sw.delete(d-id)
14.     elsif sw[d-id].slots == 1
15.       m_allocate_nodes = calc_node_set( allocate_nodes ,  sw )
16.       m_flag = true
17.       p_num  =  0
18.       break
19.     else
20.       d-id  =  search_allocation ( p_num ,  sw )
21.       tmp_nodes  =  select_nodes ( sw[d-id] ,  p_num )
22.       allocate_nodes = allocate_nodes +  tmp_nodes
23.       p_num  =  0
24.       break
25.     end
26.   end
27.   if m_flag
28.     calc_resource_mean ( topology ,  m_allocate_nodes )
29.     max_r_value ( m_allocate_nodes ,  allocate_nodes ,  allocate_path )
30.   else
31.     allocate_path  =  calc_path ( topology ,  allocate_nodes )
32.   end
33.   modify_picklist ( picklist ,  allocate_nodes )
34.   return  picklist
35. end
```

図 4.3　提案資源割当ポリシにおける *sort_qlist* の擬似コード.



(a) 各スイッチにおける計算ノードの前提状態.　(b) 資源割当ポリシの方針にあわない計算ノードの選択.　(c) 資源割当ポリシの方針にあった計算ノードの選択.

図 4.4　各スイッチに接続された計算ノードの選択例.

表 4.1 計算ノードの仕様.

CPU	Intel Xeon E5-2620(2.00GHz) × 2
Memory	64GB
NIC	on board Intel I350 GbE
OS	CentOS 6.2

4.4 評価実験

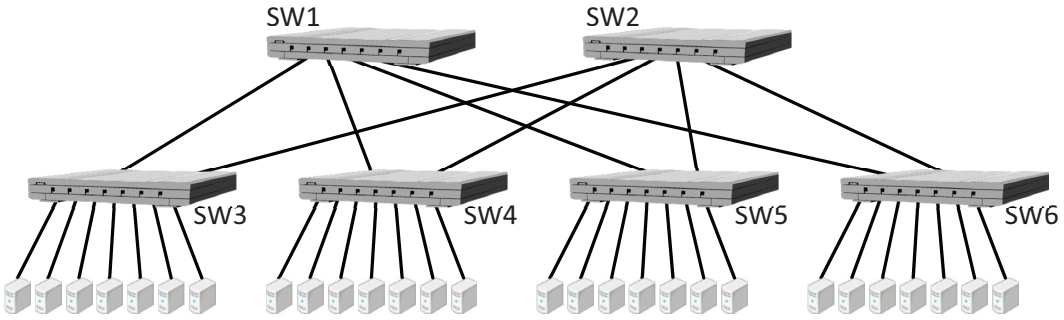
本節では，提案する資源割当ポリシを組み込んだ SDN-enhanced JMS が，Fat-tree インターコネクトを持つクラスタシステムにおいて有用であることを確認するため，以下の評価実験を行った.

4.4.1 実験環境

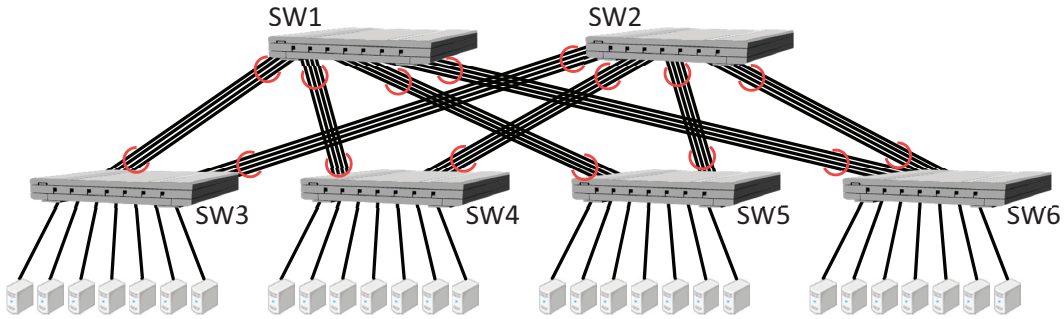
本実験で使用する環境は，図 4.5 に示す構成の Fat-tree トポロジのインターコネクトを持つクラスタシステムである．本クラスタシステムは，表 4.1 で示す仕様の 28 台の計算ノードと 1 台の SDN-enhanced JMS フレームワークが導入された管理ノードで構成される．インターコネクトは，6 台の OpenFlow スイッチ（NEC UNIVERGE PF5240）を用いて構築した 2 階層 Fat-tree トポロジで構成され，計算ノードと OpenFlow スイッチの接続は Gigabit Ethernet で接続する.

このクラスタシステムのインターコネクトにおいて，本実験ではスイッチ間の接続における帯域幅が異なる 2 種類のインターコネクト構成を用いる．図 4.5(a) に示すクラスタシステムでは，スイッチ間の接続をすべて Gigabit Ethernet で構成する．このシステム構成では，あるスイッチに接続された 7 台すべての計算ノードが，他のスイッチの計算ノード 7 台に対して一斉に通信を行った場合，スイッチが計算ノードから求められる最大帯域幅が 1Gbps × 7 の 7Gbps であるのに対し，スイッチ間で許容できる最大帯域幅が 1Gbps × 2 の 2 Gbps となり不足する．図 4.5(b) のクラスタシステムを，以下，非フルバイセクションクラスタとする．一方，図 4.5(b) に示すクラスタシステムでは，スイッチ間を 4 本の Gigabit Ethernet ケーブルで接続し，これらを IEEE 802.3ad Link Aggregation を用いて論理的に統合することで，4 Gbps の帯域幅を持つリンクとする構成とした．この場合，スイッチが許容できる最大帯域幅は 4Gbps × 2 の 8 Gbps となり，計算ノードの一斉通信を処理可能である．図 4.5(b) のクラスタシステムを，以下，フルバイセクションクラスタとする.

これらのクラスタシステムのインターコネクトは冗長経路を持つため，従来のジョブ管



(a) 非フルバイセクションクラスタの構成.



(b) フルバイセクションクラスタの構成.

図 4.5　クラスタシステムの構成.

理システムによる資源管理では，4.2 節で述べたようにスイッチの機能を利用した冗長経路への負荷分散の制御を各スイッチで設定する必要がある．そこで，本クラスタシステムでは，複数の通信経路への負荷分散制御に Open Shortest Path First（OSPF） [58] を採用した．OSPF を採用した理由は，クラスタシステムで使用した OpenFlow スイッチで，OpenFlow の機能を使わずに複数通信経路の制御を行うことができるのがこの手法だけであったためである．Fat-tree トポロジにおいて，ある対の計算ノード間における複数の通信経路は等コストとみなせるので，OSPF による通信経路への負荷分散には，ラウンドロビン方式による Equal Cost Multi Path（ECMP） [59, 60] が用いられる．

ジョブプロセスの計算ノードへの割当に関して，本実験では計算ノードに割り当てられ

るジョブのプロセスを 1 つ に制限するため，OGS/GE の計算ノード設定においてスロット数を 1 に設定した．これは計算ノード内でのプロセス間通信を発生させないための制限である．本実験の目的は，計算ノード間における通信経路の制御手法の違いによる実行性能への効果を評価することであるため，制御の対象外である計算ノード内のプロセス通信が発生する状況を制限する必要がある．また，この制限は現在の SDN-enhanced JMS フレームワークの実装に起因する制限でもある．もし異なるジョブのプロセスが同じ通信経路を持つ計算資源，すなわち 2 台の同じ計算ノードに割り当てられた場合，現在の SDN-enhanced JMS フレームワークはその経路で発生した通信がどちらのジョブのプロセスで行われているのかを区別できない．SDN-enhanced JMS フレームワークは，各ジョブに割り当てられる通信経路を NMM 上で *flow entry* として管理している．現在の実装では，*flow entry* は各計算ノードの MAC アドレスに基づいて定義される．それゆえ，Network control コンポーネントがその通信経路に対して生成する *flow entry* は同じとなり，Network control コンポーネントからは識別できなくなる．このことは，SDN-enhanced JMS フレームワークにおけるネットワーク統計情報の取得に影響を与えるため，今後の対応が求められる．

4.4.2 2 並列ジョブによる評価

本実験の目的は，提案する資源割当ポリシを組み込んだ SDN-enhanced JMS によってインターコネクトを効率的に利用できていることを確認することにある．そこで，本実験では 2 つのプロセスを生成する 2 並列 Network インテンシブジョブを用い，従来のジョブ管理システムと SDN-enhanced JMS の両方にジョブを投入して比較する評価実験を行う．4.4.1 節の制限によって 1 つの計算ノードに割り当てられるジョブのプロセスは常に 1 つである．これにより，2 並列ジョブは一対の計算ノードにプロセスが割り当てられるため，ジョブが利用した通信経路を確認しやすくする．

使用する 2 並列 Network インテンシブジョブで実行するプログラムには，任意のサイズでのデータ交換を 1,000 回繰り返す Ping-Pong プログラムを用いる．このデータ交換プログラムは MPI_Send と MPI_Recv を用いて実装する．本実験で利用するデータサイズは，1KiB，16KiB，32KiB，64KiB，128KiB，256KiB，512KiB，1MiB，16MiB，32MiB，64MiB の 11 種類である．本実験では，同じデータサイズを用いた 14 個のジョブで構成されたジョブセットを，ジョブ管理システムに投入することを 1 試行とし，各データサイズにつき 4 回ずつ行う．1 つのジョブセットを 14 個のジョブで構成したのは，図 4.5 に示すクラスタシステムで 2 並列ジョブを同時に実行できる上限が 14 個のためある．もし 15 個以上のジョブセットを投入した場合，上限数を超えたジョブはジョブ管理システムのキューで待ち，資源が空き次第計算ノードに割り当てられる．しかし，割り当てられる

計算ノードの組み合わせは，基本的に終了したジョブが使用していた組み合わせとなる．その結果，計算ノードの組み合わせは初期の割当に依存し，それ以降はほとんど変化しない．そこで，様々な資源割当の組み合わせで評価を行うため，1 試行で投入するジョブの数を上限数である 14 個とした．

以下，この実験で得られた結果に対し，資源割当およびジョブの実行性能についての評価について述べる．

4.4.2.1 ジョブへの資源割当についての評価

各ジョブに対する計算資源およびネットワーク資源の割当を評価するため，各ジョブプロセスがどのように計算ノードに割り当てられているかを調査した．

まず，各試行において異なるスイッチに接続した計算ノードに対してプロセスが割り当てられたジョブの数を調査した．このようなジョブで使用される通信経路はスイッチ間のリンクを経由するため，他のジョブによる通信と衝突が生じる可能性がある．各ジョブへの計算ノードの割当状況については，どちらのジョブ管理システムによる制御においてもOGS/GE のログファイルから確認できる．しかし，割り当てられた計算ノード間のどの通信経路を利用したのかの確認については，各ジョブ管理システムで状況が異なる．従来のジョブ管理システムによる実験では，計算ノード間における通信は各スイッチにおけるOSPF によるラウンドロビン方式での制御によりすべての通信経路に分散されるため，詳細な利用状況の確認は困難である．一方，SDN-enhanced JMS では使用する通信経路は，Netwrok control コンポーネントに組み込まれた OpenFlow コントローラによって制御されているため，ジョブが使用した通信経路は NMM のログに記録される．

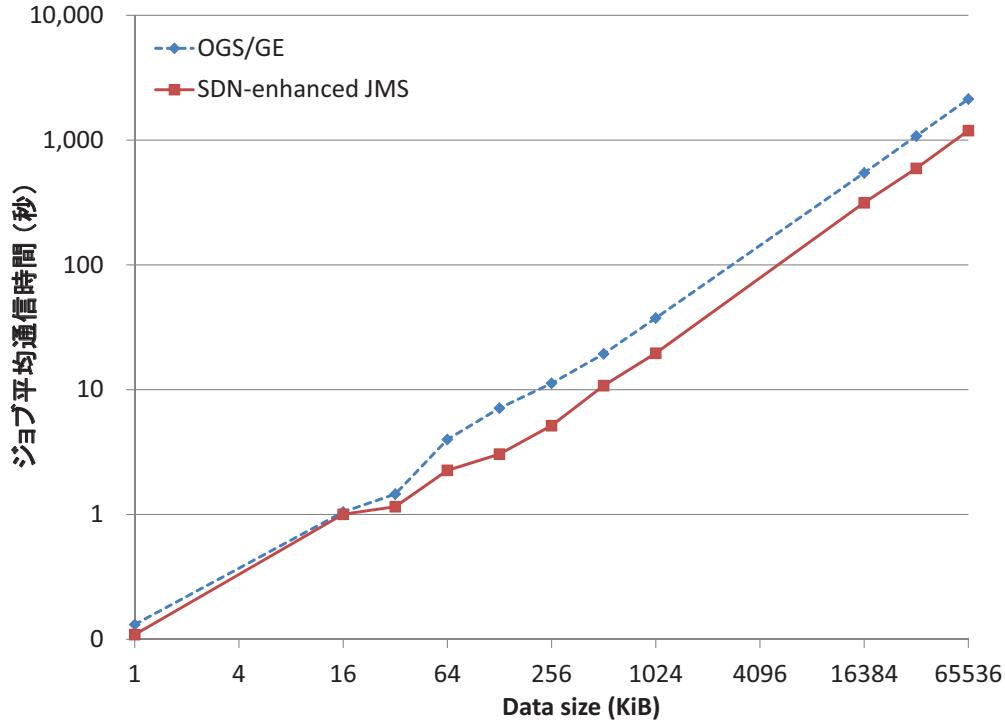
これらのログの解析結果より，従来のジョブ管理システムによる資源割当では，1 試行あたり平均 10.6 個のジョブが異なるスイッチに接続された計算ノードに割り当てられた．その結果，多くの通信が図 4.5 の *SW1* や *SW2* を経由して行われていたため，通信衝突の発生率は高くなっていたものと推測される．一方，SDN-enhanced JMS の場合，異なるスイッチに接続された計算ノードに割り当てられたジョブの数は平均 2.0 個であった．本実験で用いたクラスタシステムでは 1 つのスイッチあたり 7 台の計算ノードが接続されている構成のため，少なくとも 2 つのジョブが異なるスイッチに接続された計算ノードに割り当てられる．つまり，この結果は，本クラスタシステムで達成できる最小のジョブ数であり，提案資源割当ポリシはスイッチ間における通信衝突の発生を，計算ノードへのジョブの割当の観点から最大限抑制できた．また，NMM のログより，このような 2 つのジョブに割り当てられた通信経路に，同じスイッチ間のリンクが使用されているものが無かったことが確認できた．以上より，提案資源割当ポリシによる計算資源とネットワーク資源の割当は，通信衝突の発生を抑制できることが確認できた．

4.4.2.2 非フルバイセクションクラスタにおける性能評価

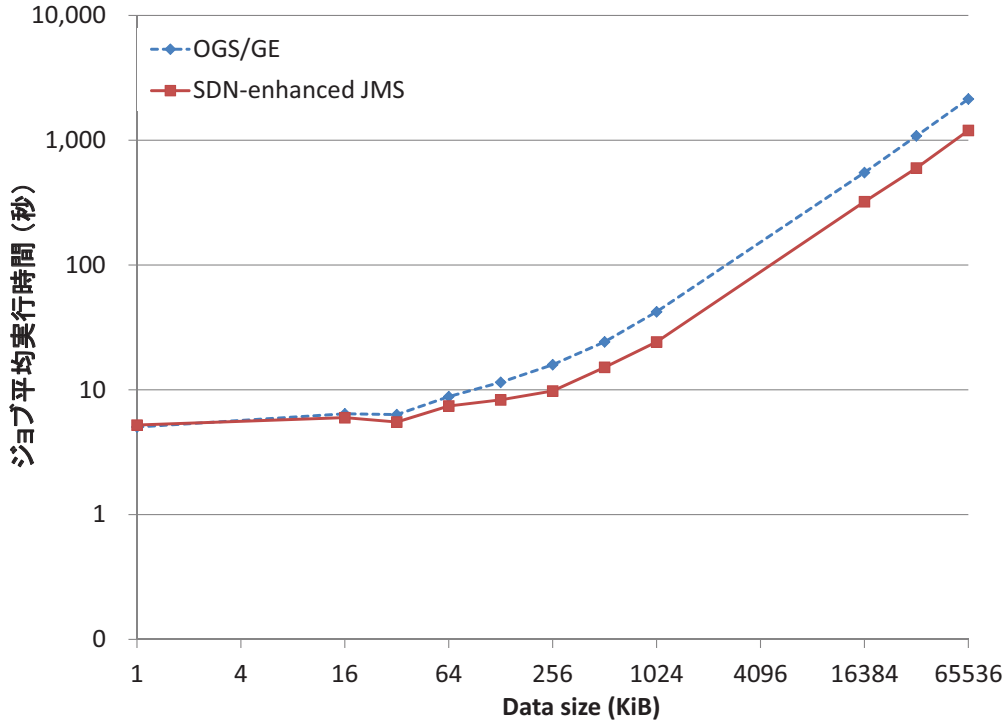
非フルバイセクションクラスタにおけるジョブの通信性能および実行性能について評価を行った．各データサイズにおけるジョブの平均通信時間および平均実行時間をまとめたグラフが図 4.6(a) および図 4.6(b) である．なお，このグラフにおいて両軸とも対数目盛である．

結果として，SDN-enhanced JMS による資源管理は，どのデータサイズにおいても平均

通信時間および平均実行時間の削減に成功した．特に，64 KiB 以上のデータサイズで大きな差が現れた．64 MiB のデータサイズでの実験では，平均通信時間の削減率は 44.1% であり，平均実行時間の削減率は 44.0% であった．



(a) 平均通信時間.

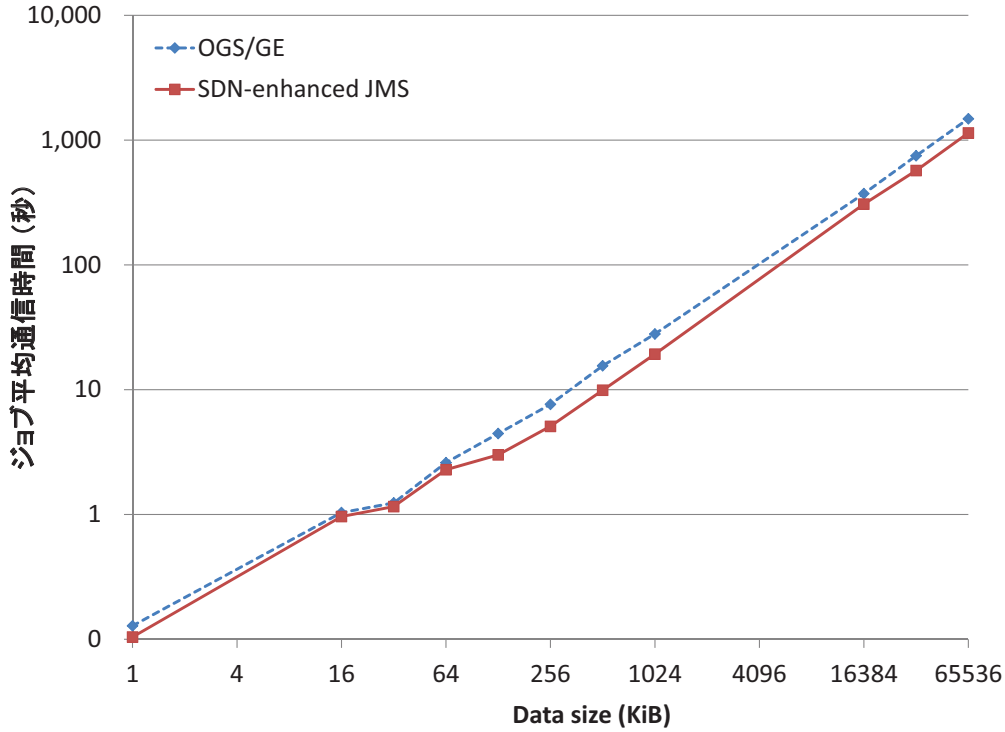


(b) 平均実行時間.

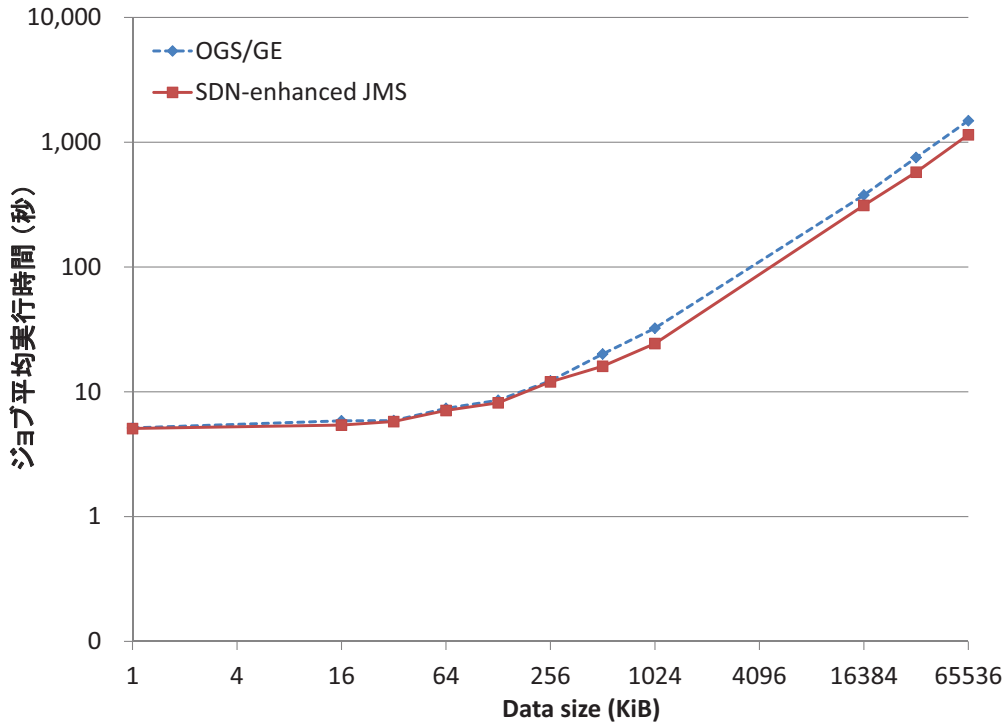
図 4.6 非フルバイセクションクラスタにおける 2 並列ジョブ実験の結果.

4.4.2.3　フルバイセクションクラスタにおける性能評価

フルバイセクションクラスタにおけるジョブの平均通信時間および平均実行時間をまとめたグラフを図 4.7(a) および図 4.7(b) に示す．スイッチ間の接続に十分な帯域を備えている本環境においても，提案する資源割当ポリシによるジョブへの割当資源の決定は，よりよいジョブの通信性能および実行性能を達成できた．図 4.7(a) や図 4.7(b) における結果の傾向は，4.4.2.2 における結果と似ている．しかし，ジョブの通信時間および実行時間の削減率は非フルバイセクションクラスタ環境での結果より少ない．実際，その削減率は，平均通信時間で 23.0%，平均実行時間で 22.8% であった．



(a) 平均通信時間.



(b) 平均実行時間.

図 4.7　フルバイセクションクラスタにおける 2 並列ジョブ実験の結果.

4.4.3　NAS Parallel Benchmarks による性能分析

本節では，提案する資源割当ポリシを適用した SDN-enhanced JMS による資源制御が，実際の並列計算アプリケーションでの利用に有効であることを確認するため，Fat-tree インターコネクトを持つクラスタシステムにおいて，基本的な分散並列計算で性能評価を行う NAS Parallel Benchmarks [56] を実行するジョブを用いて評価を行う．

本実験では，NAS Parallel Benchmarks を，3.4.3.5 節と同様の条件で使用する．すなわち，NAS Parallel Benchmarks のバージョンには ver.3.3.1 を使用し，評価に用いるベンチマークの問題サイズは Class B である．本ベンチマークを実行する際の並列数も 3.4.3.5 節と同様に，すべてのベンチマークを同じ並列数で実行でき，かつ実験環境のクラスタシステムで複数のジョブが同時に実行できる 4 並列とした．なお，両ジョブ管理システムに投入するジョブセットの構成は一部異なり，本実験用いるジョブセットのジョブ数は 30 ジョブとした．

4.4.3.1　非フルバイセクションクラスタにおける性能評価

非フルバイセクションクラスタにおいて，両ジョブ管理システムに対し，NAS Parallel Benchmarks の各ベンチマークを実行するジョブセットを投入した．その結果を図 4.8 に示す．図 4.8 の縦軸はジョブの平均実行時間比であり，（提案する資源割当ポリシを組み込んだ SDN-enhanced JMS でのジョブの平均実行時間）/（従来のジョブ管理システムを用いた場合のジョブの平均実行時間）で算出した．

その結果として，SDN-enhanced JMS による資源割当では，EP 以外のどのベンチマークにおいてもジョブの平均実行時間を削減できた．EP ベンチマークについては平均実行時間比がほぼ 1.0 となっているが，これは 3.4.3.5 節で述べたようにジョブのプロセス間でほとんど通信を行わないためである．それゆえ，提案資源割当ポリシによる通信経路の制御の効果が出ないことは妥当な結果である．

また，図 4.8 より，ジョブの平均実行時間比に特に大きく差が現れたのは，EP 以外の並列カーネルベンチマークを実行したジョブにおいてである．そのジョブの平均実行時間の削減率はそれぞれ，MG で 30.9%，CG で 28.8%，FT で 30.2%，IS で 20.4% であった．並列カーネルベンチマークは行列計算やデータソートなどの，様々なアプリケーション内で実行されている基礎的な計算を評価するベンチマークであるため，提案する資源割当ポリシを組み込んだ SDN-enhanced JMS フレームワークは，一般的な並列計算アプリケーションにより高い実行性能を得られる資源を提供できると考える．

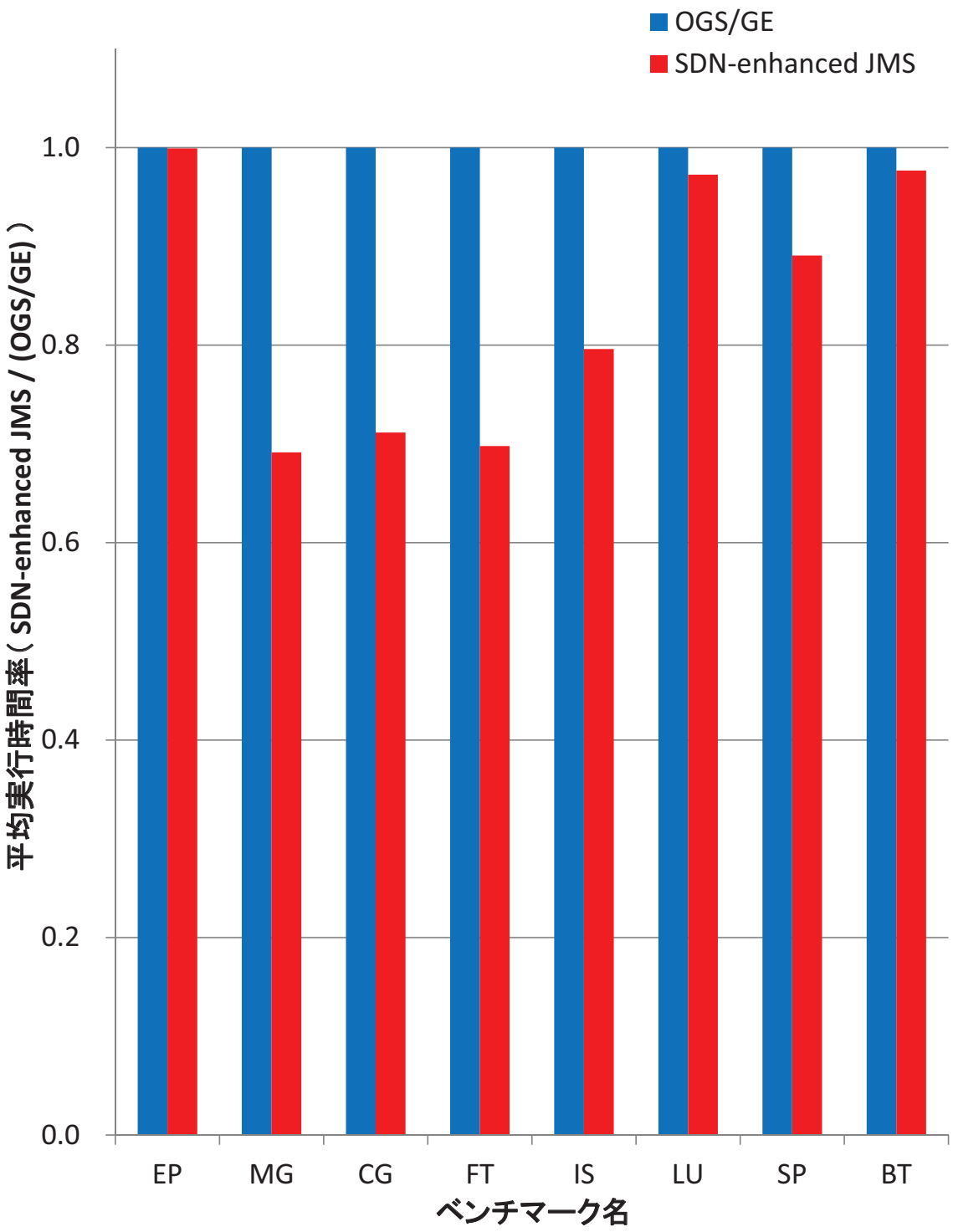


図 4.8　非フルバイセクションクラスタにおける NAS Parallel Benchmarks の結果.

4.4.3.2　フルバイセクションクラスタにおける性能評価

次に，フルバイセクションクラスタにおいて，4.4.3.1 と同様に両ジョブ管理システムに対する評価実験を行った．その結果を図 4.9 に示す．図 4.9 のグラフの縦軸は，図 4.8 と同様に算出したジョブの平均実行時間比である．

本実験における結果として，スイッチ間に十分な帯域幅を持つフルバイセクションクラスタにおいても，SDN-enhanced JMS による資源割当でジョブの実行時間を削減できた．なお，EP ベンチマークについては 4.4.3.1 と同様に効果が現れなかった．ジョブの平均実行時間の削減率はそれぞれ，MG で 20.2%，CG で 13.9%，FT で 9.2%，IS で 12.6% であった．

実験結果の傾向は非フルバイセクションクラスタ環境での実験結果に似ているが，ジョブの平均実行時間の削減率はフルバイセクションクラスタ環境の場合の方が小さかった．この結果は，スイッチ間のリンクに十分な帯域幅を持たせても，それを効率的にジョブに割り当てることができなかったため，実行性能は低下したと考えられる．実際，従来のジョブ管理システムによる資源管理で採用された OSPF による冗長経路への負荷分散は，ラウンドロビン方式に基づいたシンプルな手法であった．それゆえ，効率的な負荷分散とならない状況が発生し，ジョブにおける通信性能の低下を引き起こしたと考えられる．

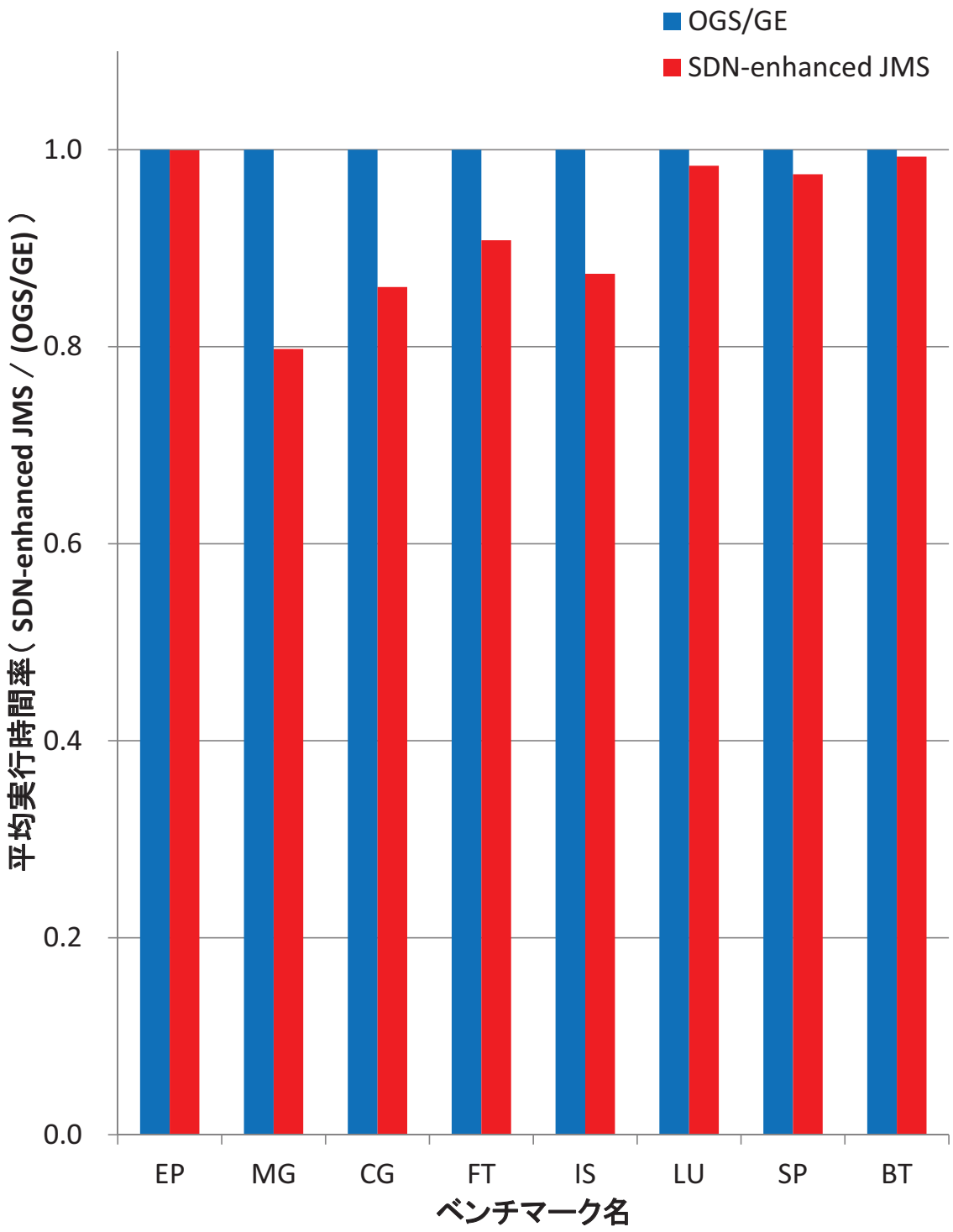


図 4.9　フルバイセクションクラスタにおける NAS Parallel Benchmarks の結果.

4.5　おわりに

本章では，今日のクラスタシステムで広く採用されている Fat-tree トポロジのインターコネクトで構成されたクラスタシステムを対象とし，計算資源の割当と冗長経路への負荷分散を連携して制御する資源割当ポリシを提案・実装し，前章で提案した SDN-enhanced JMS フレームワークによる資源割当の有用性，実用性を評価した．特に，冗長経路を効率的に利用することによる通信の衝突回避を実現する提案した資源割当ポリシでは，従来のスイッチによる負荷分散制御を利用せず，SDN-enhanced JMS フレームワークの機能を用いてインターコネクトの利用状況を把握し，それに基づいてジョブに割り当てる計算ノードと，それらの間の通信経路を明示的に指定する．

評価実験では，Fat-tree インターコネクトにおいて，スイッチ間のリンクに十分な帯域幅を持つ場合と持たない場合の 2 種類のクラスタシステムを用意した．また，従来のスイッチによる冗長経路への負荷分散手法として，OSPF が持つ制御機能を採用した．この 2 種類のクラスタシステムに対し，2 並列データ交換プログラムを用いた実験では，提案した資源割当ポリシが効率的な通信経路をジョブに割り当て，ジョブの平均実行時間を最大 44.0% 削減できることを確認した．また，NAS Parallel Benchmarks を用いた評価では，通信を伴うベンチマークにおいてジョブの平均実行時間を，スイッチ間のリンクに十分な帯域幅を持たないクラスタシステムにおいてジョブの平均実行時間を最大 30.9%，持つ場合でも最大 20.2% 削減できた．以上より，提案した資源割当ポリシを組み込んだ SDN-enhanced JMS フレームワークが，冗長経路を持つインターコネクトで構成された実際のクラスタシステム上で有用であることを確認できた．

今後の課題として，実際のアプリケーションを利用した性能評価が必要であることがあげられる．本章で行った NAS Parallel Benchmarks による評価では，あくまでアプリケーションの処理の一部に対する検証にしかならない．それゆえ，実際に利用されているシミュレーションなどで評価を行う必要がある．

第 5 章

結論

5.1　本論文のまとめ

本論文は，ますます多様化するユーザからの計算要求に対し，その構成資源を適切かつ効率的に割り当てることのできる柔軟な資源管理の仕組みの必要性に着目し，計算資源だけでなくネットワーク資源を動的な資源と捉え管理・制御することで，ユーザの計算要求に対して適切な資源割当を行うことのできるジョブ管理システムの実現を目的とした研究成果をまとめた.

1 章では，近年の高性能計算環境の現状と動向について整理し，今日の大規模クラスタシステムにおいて高い計算性能を得るためには，そのネットワーク資源であるインターコネクトの管理と制御が必要不可欠であることをまとめた．また，そのような視点から，クラスタシステムを構成するネットワーク資源であるインターコネクトに着眼し，計算資源だけでなくネットワーク資源を動的な資源と捉え管理・制御することで，ユーザの計算要求に対して適切な資源割当を行うことのできるジョブ管理システムの実現を本研究の目的とした.

2 章では，今日のクラスタシステムの現状，一般的なジョブ管理システムにおける資源管理手法および技術についての整理，ならびに，クラスタシステム上で高い通信性能を得ることを目的とした関連研究について整理した．これにより，1 章で示した本研究の目的達成のためには，(1) ネットワーク資源を動的に管理・制御する機能を備えたジョブ管理システムの実現可能性の検証，(2) 提案ジョブ管理システムへのネットワーク資源を制御する資源割当ポリシの配備による有用性・実用性の評価の 2 つの技術課題の解決が必要であることを述べた.

3 章では，2 章で導出した課題 (1) ネットワーク資源を動的に管理・制御する機能を備えたジョブ管理システムの実現可能性の検証に対して，インターコネクトをネットワーク資源として管理するためのネットワーク資源制御技術を組み込んだジョブ管理システム

フレームワークを実現し，その方向性を検証した．本研究では，ソフトウェアで実装可能なコントローラから動的かつ一元的なネットワークの制御を可能とする，新しいネットワークアーキテクチャ概念である SDN に着目し，SDN を従来のジョブ管理システムと統合させた SDN-enhanced JMS を提案・実装した．SDN-enhanced JMS によるネットワーク資源を考慮した資源割当の効果を評価するため，最も単純なインターコネクトである Flat-tree インターコネクトを有するクラスタシステムをターゲットとし，遅延または利用可能帯域幅を考慮してジョブに割り当てる計算ノードを決定する資源割当ポリシを定義し，Ping-Pong，MPI_Alltoall を用いた 2 種類の Network インテンシブジョブセットと NAS Parallel Benchmarks を実行するジョブセットを Flat-tree インターコネクトのクラスタシステムに投入することで評価を行い，インターコネクトを効率的に利用した資源割当を実現できていること，およびジョブの実行時間を短縮できていることを確認した．これにより，ネットワーク資源を動的に管理・制御する機能を備えたジョブ管理システムが実現可能であることを確認した．

4 章では，2 章で導出した課題（2）提案ジョブ管理システムへのネットワーク資源を制御する資源割当ポリシの配備による有用性・実用性の評価に対して，計算資源およびネットワーク資源の利用状況を考慮して計算資源および通信経路の割当を実現する資源割当ポリシに関する研究開発の成果をまとめた．本研究では，広帯域・耐障害性を実現するため多くのクラスタシステムで採用されている冗長経路を有するインターコネクトにおいて，従来のスイッチによる通信の負荷分散制御を用いず，実際の利用状況に応じて明示的に使用する通信経路をジョブに割り当てる資源割当ポリシを設計・実装し，本研究で提案するジョブ管理システムが現実的なインターコネクトを有するクラスタシステム上でより高い実行性能を持つ資源をジョブに割り当てることができることを確認した．提案した資源割当ポリシによって，クラスタシステムの利用状況に応じて，計算ノード間の冗長経路における負荷分散，およびインターコネクト全体における通信量の分散が可能となった．提案した資源割当ポリシによるネットワークの負荷分散およびジョブの実行性能への効果を確認するため，多くのクラスタシステムで採用されている Fat-tree トポロジのインターコネクト環境において，OSPF による冗長経路の負荷分散を利用したクラスタシステムにおける従来のジョブ管理システムの資源割当と比較実験を行った．実験は 2 種類の Network インテンシブ なジョブセットと NAS Parallel Benchmarks を実行するジョブセットで行った．その結果として，提案した資源割当ポリシにより決定された資源割当は，各ジョブに対して高い性能を得られる資源を割り当てることができることを確認した．

これにより，本研究の目的である計算資源だけでなくネットワーク資源を動的な資源と捉え，ユーザの計算要求に対して適切な資源割当を行うことのできるジョブ管理システムの実現性と有用性・実用性を示した．

5.2 今後の課題

提案した SDN-enhanced JMS では，インターコネクトをネットワーク資源として動的に制御する機能を実現し，計算資源とネットワーク資源の両資源を考慮した資源割当決定手法を設定できるフレームワークを提供した．しかし，現在の SDN-enhanced JMS では各リンクにおける利用可能な帯域幅のような割当資源量の制御は行えていない．そのため，インターコネクトのリンクは他のジョブによる通信が同時に行われることがあるため，提供できる通信性能はベストエフォートとなり，得られる性能を保障することができない．今後，ネットワークにおける QoS（Quality of Service）機能などの他の通信制御機能も，SDN-enhanced JMS の Network Management Module に取り入れていく必要がある．

提案した資源割当ポリシは，冗長経路を持つインターコネクトトポロジの 1 つである Fat-tree トポロジのインターコネクトを想定した実装となっている．それゆえ，他の冗長経路を持つインターコネクトに対して効率的な割当資源を決定できる資源割当ポリシの設計・実装が必要である．また，本資源割当ポリシはジョブの実行性能およびクラスタシステムにおける資源の効率的な割当に焦点を当てて実装している．しかし，実際の高性能計算環境では，資源割当の決定には運用ポリシなど他の条件も考慮する必要がある．これについても，それらの条件を反映させるために必要な機能などを分析していく必要がある．

謝辞

本研究の全過程を通じて，懇切なる御指導，御助言と格別なる御配慮を賜りました大阪大学サイバーメディアセンター情報メディア教育研究部門 竹村治雄教授に謹んで感謝の意を表します．

本論文をまとめるにあたり，貴重な時間を割いて頂き，懇切なる御指導と有益な御助言を賜りました大阪大学大学院情報科学研究科情報システム工学専攻 土屋達弘教授，大阪大学サイバーメディアセンター先端ネットワーク環境研究部門 長谷川剛准教授，大阪大学サイバーメディアセンター応用情報システム研究部門 伊達進准教授に心より感謝の意を表します．

本研究を推進するにあたり，多大なる御指導，御助言を頂きました大阪大学サイバーメディアセンター応用情報システム研究部門 下條真司教授，大阪大学サイバーメディアセンター応用情報システム研究部門 木戸善之講師，奈良先端科学技術大学院大学情報科学研究科ソフトウェア設計学研究室 市川昊平准教授，筑波大学システム情報系情報工学域 阿部洋丈准教授に深く感謝いたします．

本研究を進めるにあたり，多大なる御支援，御助言を頂きました独立行政法人情報通信研究機構（NICT）テストベッド研究開発推進センターテストベッド研究開発室室長 河合栄治博士，研究員 山中広明氏に深く感謝いたします．

大阪大学大学院情報科学研究科在学中において御指導，御助言を頂きました大阪大学サイバーメディアセンター情報メディア教育研究部門（竹村研究室）清川清准教授，黒田嘉宏准教授，間下以大講師，に深く感謝いたします．

本研究を進めるにあたり，多大なる御支援を頂きました大阪大学サイバーメディアセンター情報メディア教育研究部門（竹村研究室）研究員 石芳正氏に深く感謝いたします．

大阪大学大学院情報科学研究科在学中において御支援を頂きました，大阪大学サイバーメディアセンター情報メディア教育研究部門（竹村研究室）の秘書，学生の諸氏に深く感謝いたします．

参考文献

[1] “TOP 500 Supercomputer Sites.” [Online]. Available: http://top500.org/ .

[2] “Myricom.” [Online]. Available: http://www.myricom.com/ .

[3] N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. N. Seizovic, and W.-K. Su, “Myrinet: A Gigabit-per-Second Local Area Network,” IEEE micro, vol. 15, no. 1, pp. 29–36, 1995.

[4] “InfiniBand Trade Association.” [Online]. Available: http://www.infinibandta.org/ .

[5] N. Gilbert, “Agent-Based Social Simulation: Dealing with Complexity,” The Complex Systems Network of Excellence, vol. 9, no. 25, pp. 1–14, December 2004.

[6] X. Li, W. Mao, D. Zeng, and F.-Y. Wang, “Agent-Based Social Simulation and Modeling in Social Computing,” in Proceedings of the IEEE ISI 2008 PAISI, PACCF, and SOCO International Workshops on Intelligence and Security Informatics, pp. 401–412, June 2008.

[7] B. Varghese, G. McKee, and V. Alexandrov, “Intelligent Agents for Fault Tolerance: From Multi-Agent Simulation to Cluster-based Implementation,” in IEEE 24th International Conference on Advanced Information Networking and Applications Workshops (WAINA) 2010, pp. 985–990, April 2010.

[8] T. Chuang and M. Fukuda, “A Parallel Multi-agent Spatial Simulation Environment for Cluster Systems,” in IEEE 16th International Conference on Computational Science and Engineering (CSE) 2013, pp. 143–150, December 2013.

[9] R. Reuillon, F. Chuffart, M. Leclaire, T. Faure, N. Dumoulin, and D. Hill, “Declarative Task Delegation in OpenMOLE,” in 2010 international conference on High Performance Computing and Simulation (HPCS 2010), pp. 55–62, 2010.

[10] H. Arikawa and T. Murata, “Implementation Issues in a Grid-Based Multi-Agent Simulation System used for Increasing Labor Supply,” The Review of Socionetwork Strategies, vol. 1, no. 1, pp. 1–13, 2007.

[11] D. Chen, G. K. Theodoropoulos, S. J. Turner, W. Cai, R. Minson, and Y. Zhang, “Large

Scale Agent-based Simulation on the Grid," Future Generation Computer Systems, vol. 24, no. 7, pp. 658–671, 2008.

[12] Y. Sano and N. Fukuta, "A GPU-based Framework for Large-scale Multi-Agent Traffic Simulations," in 2013 IIAI International Conference on Advanced Applied Informatics (IIAIAAI), pp. 262–267, 2013.

[13] B. A. Kingsbury, The Network Queuing System, Sterling Software, Palo Alto, 1986.

[14] R. L. Henderson, "Job Scheduling under the Portable Batch System," in Job Scheduling Strategies for Parallel Processing, vol. 949, pp. 279–294, 1995.

[15] "Platform LSF." [Online]. Available: http://www.platform.com/ .

[16] "Condor Project Homepage." [Online]. Available: http://research.cs.wisc.edu/condor/ .

[17] J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke, "Condor-G: A Computation Management Agent for Multi-Institutional Grids," Cluster Computing, vol. 5, no. 3, pp. 237–246, July 2002.

[18] "Open Grid Scheduler: The official Open Source Grid Engine." [Online]. Available: http://gridscheduler.sourceforge.net/ .

[19] 渡場康弘, 伊達進, 阿部洋丈, 市川昊平, 山中広明, 河合栄治, 竹村治雄, "SDN を用いたジョブ管理システムの提案," 情報処理学会研究報告 vol. 2012-HPC-137 no. 33 (HOKKE-20), pp. 1–6, December 2013.

[20] Y. Watashiba, S. Date, H. Abe, K. Ichikawa, H. Yamanaka, E. Kawai, and H. Takemura, "An Architectural Design of a Job Management System Leveraging Software Defined Network," in The 4th IEEE International Workshop on High-Speed Network and Computing Environment (HSNCE 2013), pp. 724–729, July 2013.

[21] 渡場康弘, 木戸善之, 伊達進, 阿部洋丈, 市川昊平, 山中広明, 河合栄治, 竹村治雄, "OpenFlow 連携ジョブ管理システムの実装と評価," 情報処理学会研究報告 vol. 2013-HPC-140 no. 11 (SWoPP 2013), pp. 1–6, July 2013.

[22] Y. Watashiba, Y. Kido, S. Date, H. Abe, K. Ichikawa, H. Yamanaka, E. Kawai, and H. Takemura, "Prototyping and Evaluation of a Network-aware Job Management System on a Cluster System Leveraging OpenFlow," in The 19th IEEE International Conference On Networks (ICON 2013), pp. 1–6, December 2013.

[23] 渡場康弘, 木戸善之, 伊達進, 阿部洋丈, 市川昊平, 山中広明, 河合栄治, 竹村治雄, "計算資源とネットワーク資源を考慮した割当ポリシを配備可能とするジョブ管理フレームワーク," 電子情報通信学会論文誌, vol. J97-D, no. 6, pp. 1082–1093, June 2013.

[24] Y. Watashiba, S. Date, H. Abe, Y. Kido, K. Ichikawa, H. Yamanaka, E. Kawai, S. Shimojo, and H. Takemura, "Performance Characteristics of an SDN-enhanced Job Management System for Cluster Systems with Fat-tree Interconnect," in Emerging Issues in

Cloud (EIC) Workshop, The 6th IEEE International Conference on Cloud Computing Technology and Science (CloudCom 2014), pp. 781–786, December 2013.

[25] Y. Watashiba, S. Date, H. Abe, Y. Kido, K. Ichikawa, H. Yamanaka, E. Kawai, S. Shimojo, and H. Takemura, "Efficacy Analysis of a SDN-enhanced Resource Management System through NAS Parallel Benchmarks," The Review of Socionetwork Strategies, vol. 8, no. 2, pp. 69–84, December 2014.

[26] J. J. Dongarra, "The LINPACK Benchmark: An Explanation," in Supercomputing, pp. 456–474, 1988.

[27] J. Dongarra, P. Luszczek, and A. Petitet, "The LINPACK Benchmark: Past, Present and Future," Concurrency and Computation: Practice and Experience, vol. 15, no. 9, pp. 803–820, 2003.

[28] J. Dongarra and P. Luszczek, "LINPACK Benchmark," Encyclopedia of Parallel Computing, pp. 1033–1036, 2011.

[29] Y. Ajima, T. Inoue, S. Hiramoto, and T. Shimizu, "Tofu: Interconnect for the K computer," Fujitsu Science and Technical Journal, vol. 48, no. 3, pp. 280–285, 2012.

[30] M. Al-Fares, A. Loukissas, and A. Vahdat, "A Scalable, Commodity Data Center Network Architecture," in ACM SIGCOMM Computer Communication Review, vol. 38, no. 4, pp. 63–74, 2008.

[31] H. Hussain, S. U. R. Malik, A. Hameed, S. U. Khan, G. Bickler, N. Min-Allah, M. B. Qureshi, L. Zhang, Y. Wang, N. Ghani, J. Kolodziej, A. Y. Zomaya, C.-Z. Xu, P. Balaji, A. Vishnu, F. Pinel, J. E. Pecero, D. Kliazovich, P. Bouvry, H. Li, L. Wang, D. Chen, and A. Rayes, "A Survey on Resource Allocation in High Performance Distributed Computing Systems," Parallel Computing, vol. 39, no. 11, pp. 709–736, 2013.

[32] I. Foster and C. Kesselman, The Grid 2: Blueprint for a New Computing Infrastructure. Morgan Kaufmann Publishers Inc., 2003.

[33] I. Foster, "What is the Grid? A Three Point Checklist," GRIDtoday, July 2002.

[34] V. Sander and A. Roy, "GARA: A Uniform Quality of Service Architecture," Grid Resource Management, pp. 377–394, 2004.

[35] A. Takefusa, M. Hayashi, N. Nagatsu, H. Nakada, T. Kudoh, T. Miyamoto, T. Otani, H. Tanaka, M. Suzuki, Y. Sameshima, W. Imajuku, M. Jinno, Y. Takigawa, S. Okamoto, Y. Tanaka, and S. Sekiguchi, "G-lambda: Coordination of A Grid Scheduler and Lambda Path Service over GMPLS," Future Generation Computer Systems, vol. 22, no. 8, pp. 868–875, October 2006.

[36] X. Yang, T. Lehman, C. Tracy, J. Sobieski, S. Gong, P. Torab, and B. Jabbari, "Policy-Based Resource Management and Service Provisioning in GMPLS Networks," in The

26th Annual IEEE Conference on Computer Communications (IEEE INFOCOM 2007), 2007.

[37] L. Tomás, C. Carrión, B. Caminero, and A. C. Caminero, “Exponential Smoothing for Network-Aware Meta-scheduler in Advance in Grids,” in the 39th International Conference on Parallel Processing (ICPP Workshops 2010), pp. 323–330, 2010.

[38] G. P. Koslovski, P. V.-B. Primet, and A. S. Charão, “VXDL: Virtual Resources and Interconnection Networks Description Language,” in GridNets 2008, pp. 138–154, 2008.

[39] 工藤知宏, 松田元彦, 手塚宏史, 児玉祐悦, 建部修見, 関口智嗣, “VLAN を用いた複数パスを持つクラスタ向き L2 Ethernet ネットワーク (ネットワーク),” 情報処理学会論文誌. コンピューティングシステム, vol. 45, no. 6, pp. 35–44, May 2004.

[40] 森江善之, 末安直樹, 松本透, 南里豪志, 石畑宏明, 井上弘士, 村上和彰, “通信タイミングを考慮した衝突削減のための MPI ランク配置最適化技術,” 情報処理学会論文誌コンピューティングシステム, vol. 48, no. 13, pp. 192–202, August 2007.

[41] S. Girona, J. Labarta, and R. M. Badia, “Validation of Dimemas Communication Model for MPI Collective Operations,” in Recent Advances in Parallel Virtual Machine and Message Passing Interface, vol. 1908, pp. 39–46, August 2000.

[42] T. Kielmann, H. E. Bal, S. Gorlatch, K. Verstoep, and R. F. Hofman, “Network Performance-aware Collective Communication for Clustered Wide-Area Systems,” Parallel Computing, vol. 27, no. 11, pp. 1431–1456, 2001.

[43] L. A. Steffenel, “Modeling Network Contention Effects on All-to-All Operations,” in IEEE International Conference on Cluster Computing 2006, pp. 1–10, September 2006.

[44] A. Bhatele, T. Gamblin, S. H. Langer, P. Bremer, E. W. Draeger, B. Hamann, K. E. Isaacs, A. G. Landge, J. A. Levine, V. Pascucci et al., “Mapping Applications with Collectives over Sub-communicators on Torus Networks,” in International Conference for High Performance Computing, Networking, Storage and Analysis (SC) 2012, pp. 1–11, November 2012.

[45] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, “OpenFlow: Enabling Innovation in Campus Networks,” SIGCOMM Computer Communication Review, vol. 38, no. 2, pp. 69–74, March 2008.

[46] “Trema: Full-Stack OpenFlow Framework for Ruby/C.” [Online]. Available: http://trema.github.com/trema/ .

[47] “NOX.” [Online]. Available: http://www.noxrepo.org/nox/about-nox/ .

[48] “POX.” [Online]. Available: http://www.noxrepo.org/pox/about-nox/ .

[49] “Ryu SDN Framework.” [Online]. Available: http://osrg.github.io/ryu/ .

[50] “Floodlight OpenFlow Controller -Project Floodlight.” [Online].

Available: http://www.projectfloodlight.org/floodlight/ .

[51] D. Erickson, "The Beacon Openflow Controller," in Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, HotSDN '13, pp. 13–18, 2013.

[52] "XML-RPC.Com." [Online]. Available: http://xmlrpc.scripting.com/ .

[53] S. S. Laurent, E. Dumbill, and J. Johnston, Programming Web Services with XML-RPC. O'Reilly & Associates, Inc., 2001.

[54] IEEE 802.1AB-2005 IEEE Standard for Local and Metropolitan Area Networks Station and Media Access Control Connectivity Discovery, May 2005.

[55] "NAS Parallel Benchmarks." [Online].
Available: http://www.nas.nasa.gov/publications/npb.html .

[56] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, L. Dagum, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, R. S. Schreiber, H. D. Simon, V. Venkatakrishnan, and S. K. Weeratunga, "The NAS Parallel Benchmarks—Summary and Preliminary Results," in Proceedings of the 1991 ACM/IEEE Conference on Supercomputing, ser. Supercomputing '91, pp. 158–165, 1991.

[57] Y. Li and D. Pan, "OpenFlow based Load Balancing for Fat-Tree Networks with Multipath Support," IEEE International Conference on Communications 2013 (ICC 2013), pp. 1–5, June 2013.

[58] J. Moy, OSPF Version 2, RFC 2328, 1998.

[59] "802.1Qbp - Equal Cost Multiple Paths." [Online].
Available: http://www.ieee802.org/1/pages/802.1bp.html .

[60] C. Hopps, Analysis of an Equal-Cost Multi-Path Algorithm, RFC 2992, 2000.