Vol.2012-ARC-202 No.33
Vol.2012-HPC-137 No.33
2012/12/14

SDN を用いたジョブ管理システムの提案

渡場 康弘[1]　伊達 進[1]　阿部 洋丈[1]　市川 昊平[2]　山中 広明[3]　河合 栄治[3]　竹村 治雄[1]

概要：近年，高性能計算資源はクラスタシステムとして構成されていることが主流で，より高性能を求めてますます大規模化する傾向にある．複数の計算ノードがインターコネクトによって相互接続されたクラスタシステムでは，ユーザがより高い実行性能を得るには，できるだけ多数の計算資源を高効率に利用する分散並列処理を行う必要がある．とりわけ，計算ノード間の通信が密な計算では，割り当てられた計算ノード間のネットワーク性能が計算性能に多大な影響を与える．しかし，今日利用できる，計算ノードを管理しユーザジョブの計算要求に応じて，効率的に計算ノードの計算資源を割り当てる役割を担うジョブ管理システムの多くは，計算ノード間のネットワーク，すなわちインターコネクトの効率的利用を考慮した仕組みを有していない．われわれは，そのような視点から，ネットワーク資源も含めた計算資源の適切な管理・割当を行うことができるジョブ管理システムの実現を目的とした研究開発に着手した．本稿では，計算ノード間のインターコネクトを Software-Defined Network (SDN) で制御することにより，計算性能の向上を目的とするジョブ管理システムを提案し，提案システムの有用性についてケーススタディを用いて検討する．

A proposal of Network-aware Job Management System leveraging SDN

Yasuhiro Watashiba[1]　Susumu Date[1]　Hirotake Abe[1]　Kohei Ichikawa[2]　Hiroaki Yamanaka[3]
Eiji Kawai[3]　Haruo Takemura[1]

Abstract: High performance computing has been becoming more large-scale and distributed. Today, since the current dominant trend of high-performance computing is cluster system, it tends to be built with more computers for more performance improvement and acceleration. Therefore, the performance of interconnect among computing resources of the cluster plays more important role due to a fact that parallel computation performed on distributed multiple resources requires communication. However, no current Job Management System (JMS) is designed to consider how network resources, that is, interconnects are used in submitting jobs to computing resources of a cluster system. We have started the research and development of network-aware job management system that allocates appropriate set of computing and network resources in an on-demand fashion, based on user-provided resource requirement information. The key feature of our proposed JMS is the integration of Software-Defined Network (SDN) concept into traditional JMS.

1. はじめに

近年，高性能計算資源はますます大規模化・分散化する傾向にある．今日の計算資源の主流は複数台の計算ノードをネットワークで接続したクラスタシステムとして構成されるため，計算性能を向上させるためにより多くの計算ノードでクラスタシステムを構築する傾向にある．さらに，計算ノードの CPU 自体のマルチコア化による性能向上の結果，高性能計算資源は膨大な数のプロセッサコアで構成されたシステムとなりつつある．このことは，TOP500 [1] のアーキテクチャ別システムシェアにおいて，2007 年頃よりクラスタシステムが約 80 %を占めていることに現れている．また，TOP500 に登録された高性能計算システム全体に占めるクラスタシステムの数はそれほど変わっていないのに，クラスタシステムの総コア数は年々増加してい

[1] 大阪大学
Osaka University
[2] 奈良先端科学技術大学院大学
Nara Institute of Science and Technology
[3] 独立行政法人情報通信研究機構
National Institute of Information and Communications Technology

る．現在は百万個以上のコア数で構成されるシステムも出現している．

このようなクラスタシステムにおいて，一般的により高い演算性能を得るためには，できるだけ多くの計算資源を利用した並列計算を効率よく行うことが重要である．一般的に並列化による性能向上は，並列数の増加による演算時間の短縮と並列化によるオーバーヘッドなどによって決定される．この並列化の性能向上を阻害する要因の中でも，処理間の通信コストによるオーバーヘッドが並列計算に与える影響は大きい．特にクラスタシステムでは，多くの分散した計算資源を並列に利用するため，この影響がより顕著となる．そのため，計算資源間で発生する通信コストを抑えることが並列計算の性能向上において重要な要因となる．

クラスタシステムの構築において，計算ノード間を繋ぐインターコネクトには，同時に発生するネットワーク通信性能を考慮した高帯域かつ低遅延なネットワークが求められる．これを実現するため，ネットワークトポロジや専用デバイスなどのさまざまなインターコネクト技術の研究開発が進められている．クラスタシステムで利用されているインターコネクトとしては，Myrinet [13] や Infiniband [14] などを用いたものと，一般的な Ehternet を用いたものがある．前者はスーパーコンピュータなどの並列計算システム用に開発されたインターコネクト技術をクラスタシステムに適用するため開発されており，高帯域・低遅延なネットワークを実現できるが，構築コストが高くなる．一方，後者は汎用的に利用されている Ehternet 技術であり，高コストパフォーマンスなシステムを構築できるが，Myrinet や InfiniBand のような並列計算システム用インターコネクトと比べると帯域や遅延といった性能面で劣る．

また，実際にクラスタシステムで並列計算を行う場合，割り当てられた計算資源によって処理性能に差が生じる場合がある．通常，ユーザは計算資源のすべてを利用するのではなく，その一部を利用して並列計算を行う．その際，割り当てられた計算資源間のインターコネクトのネットワークトポロジによっては，処理効率が落ちる場合がある．例えば，Fat Tree で構成されたインターコネクトの場合，帯域としてはフルバイセクションバンド幅を保証するネットワークであるため割り当てられた計算ノードによる性能差は生じないが，計算ノード間のホップ数による遅延の差が生じる可能性がある．このことは，クラスタシステムの規模が大きくなるほど顕著に表れてくる．

このように，大規模クラスタシステム型の計算資源における計算ノード間の通信性能は，ジョブの実効性能において非常に重要な要因となっている．この課題に対し，クラスタシステム自体での対応や利用における工夫などの様々な対応が行われている．システム側での対応としては，広帯域・低遅延な専用ハードウェアの開発や効率的なネットワークトポロジの設計が行われているが，これを実現するには構築コストが非常に大きくなる．一方，ユーザ側での対応としては，割り当てられたネットワーク資源をできるだけ有効活用するための手法などの研究が行われているが，割当資源の構成や他のジョブとの競合などの制限により実環境での利用は難しい．そこで，計算資源を割り当てる際に，ジョブ管理システムが計算資源だけでなくネットワーク資源についても制御するための方法について検討する．

本研究の構成は以下のようになる．第 2 節でネットワーク資源の観点からのジョブ管理システムにおける課題について整理し，第 3 節で提案手法について述べる．第 4 節でケーススタディを用いて本提案の有効性を検証し，第 5 節でまとめを行う．

2. 計算資源の管理・割当における問題

ユーザジョブへの高性能計算資源の割当は一般的にジョブ管理システムが行う．例えば，主なジョブ管理システムとして，NQS [2]，PBS [3]，LSF [4], Condor [5]，Grid Engine [6] などがあげられる．ジョブ管理システムの役割は，ジョブに対して計算資源をあらかじめ管理者が定めたスケジューリングポリシに従って効率的に配分することである．一般的に，高性能計算資源は複数のユーザで共有して利用されるものであり，その資源量には限りがある．そのため，限られた計算資源を効率よく利用するには，無駄な空き資源を作らないよう管理・制御するめの枠組みが必要である．ジョブ管理システムは，CPU やメモリなどの計算資源を管理し，ユーザからの要求に対してポリシーに基づき資源の割当を行う．一方，計算資源を共有するユーザや各ジョブに対しても，通常はジョブを実行するための権限や優先度が設定されている．

その際，割り当てた計算資源，すなわち計算ノード間を接続するネットワークのトポロジや性能については，既存のジョブ管理システムでは考慮していないのが一般的である．つまり，計算資源をユーザジョブに割り当てる際に，そのジョブがどのようにインターコネクトを利用するのかを考慮した上で割当計算資源を決定する機能が現在のジョブ管理システムには備わっていない．ジョブ管理システムが導入された当初，計算資源は基本的に単一計算ノードまたは少数の計算ノードで構成されたシステムであったため，ノード間を接続するネットワークについては十分な性能要件を満たすことを前提としており，ジョブ管理システムがインターコネクトの有効利用については考慮する必要はなかった．しかし，現在のように計算資源が大規模化・分散化していくに伴い，高性能計算資源におけるインターコネクトの性能や利用の効率化が重要な課題となってきている．

この問題に対し，多くの高性能計算資源では，広帯域・低遅延なインターコネクトを用い，どのように割り当てて

も同様のネットワーク性能が保持される構成でシステムを構築することで，従来の運用と同様でも問題ない環境を実現している．しかし，そのようなインターコネクト環境を構築するには非常に大きな構築コストが必要となるため，どのようなクラスタシステムでも実現できることではない．また，システムが大規模化・分散化していく状況を考えれば，十分な性能を有するインターコネクトを構築するためのコストの問題は顕在化していくと考える．

また，一般的にネットワークの利用はベストエフォート形式であることが多く，ネットワークを専有的に利用するとなると，専用線のような特殊な環境を用意する必要がある．このことはインターコネクトの利用でも同様であり，システム構築時の設計などによって定められた性能の中でのベストエフォートとなる．そのため，割り当てられた計算資源は，ネットワークトポロジや他のジョブのネットワーク利用による影響で思うような並列計算性能が出せない場合がある．

割り当てられたネットワーク資源を効率的に利用するため，ネットワーク資源の情報を収集して処理を最適化するための様々な研究が行われてきている．特に，ネットワーク資源において重要な情報であるトポロジ情報に関しては，利用する目的によって必要とするトポロジ情報の要素や取得方法が異なるため，これまでの研究の多くはその目的に特化した手法となることが多い．利用目的にはクラスタシステムにおける分散並列計算もあり，分散並列計算の最適化を対象とした研究 [7] も行われている．一般的に分散並列計算では，必要とするトポロジ情報は計算ノード，スイッチなどの接続情報であり，得られた情報を基に実行するジョブを最適化する．そのため，もしユーザがアプリケーションに最適なネットワークトポロジをジョブ投入時に要求できるなら，得られたネットワークトポロジへのアプリケーションでの最適化は不要となる．また，トポロジ情報取得に要する時間や資源を無くすことができる．

そこで本稿では，従来のジョブ管理システムにネットワーク資源情報の取得・管理機能を追加することで，ネットワーク資源を考慮したジョブ管理システムの実現を目指し，必要となる技術的要件などの検討を行う．

3. 提案システム

3.1 提案システムの概要

ジョブに計算資源を割り当てる際に，ネットワーク資源も考慮して割当資源を決定するジョブ管理システムの実現を目指す．ネットワーク資源を考慮することで，非効率なネットワークトポロジ構成での計算資源の割当を無くし，かつユーザがネットワーク資源に対して要求がある場合はそれに基づいた構成での計算資源の割当を可能とする．このようなジョブ管理システムを実現するにあたり，本研究では既存のジョブ管理システムにネットワーク資源を扱うためのフレームワークを追加することを考える．

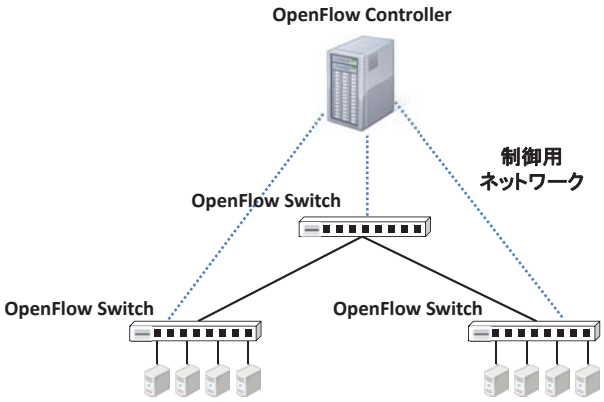


図 1 OpenFlow ネットワークの例.
Fig. 1 An example of OpenFlow network.

このフレームワークにおいて必要とされる機能は以下の 4 つである．

(1) ネットワーク資源に関する情報収集
(2) 収集した資源情報や割当済み資源情報の管理
(3) 割当情報に基づくネットワーク資源の制御
(4) ネットワーク資源に対するユーザの要求手段

(1)，(2)，(4) は既存手法の活用や，ジョブ管理システムの改修で実現可能であると考える．(1) については既存研究手法を活用することで実現できると考える．また，情報収集を実行するのがジョブ管理システムであるため，管理者権限を要する手法も利用できることから，実現にあたっての制限は少ないと考える．(2) については (1) で得られた情報をデータベースなどでどのように管理するかを検討する必要がある．また，既存のジョブ管理システムが保持する計算資源情報との連携方法が課題になると考える．(4) についてはジョブ管理システムのジョブ投入機能の中でその手段を拡張すれば良い．しかし，(3) については既存の技術や手法では実現が困難である．

そこで，本研究では最近注目を集めている Software-Defined Network (SDN) に着目した．SDN ではネットワークに対する制御をプログラマブルに行うことができるため，従来のネットワーク環境では実施が困難であった動的な制御を可能とする．そこで，この SDN を実現する技術である OpenFlow を活用してネットワーク資源管理フレームワークを構築する．

3.2 SDN/OpenFlow 技術

SDN は，スイッチなどのネットワーク機器におけるデータ転送機能と制御機能を分離し，従来は個々の機器で行ってきた制御機能を集約することを可能とする新しいコンセプトである．従来のネットワークにおいて経路を変更しようとした場合，関係するルータやスイッチ等の設定を個別

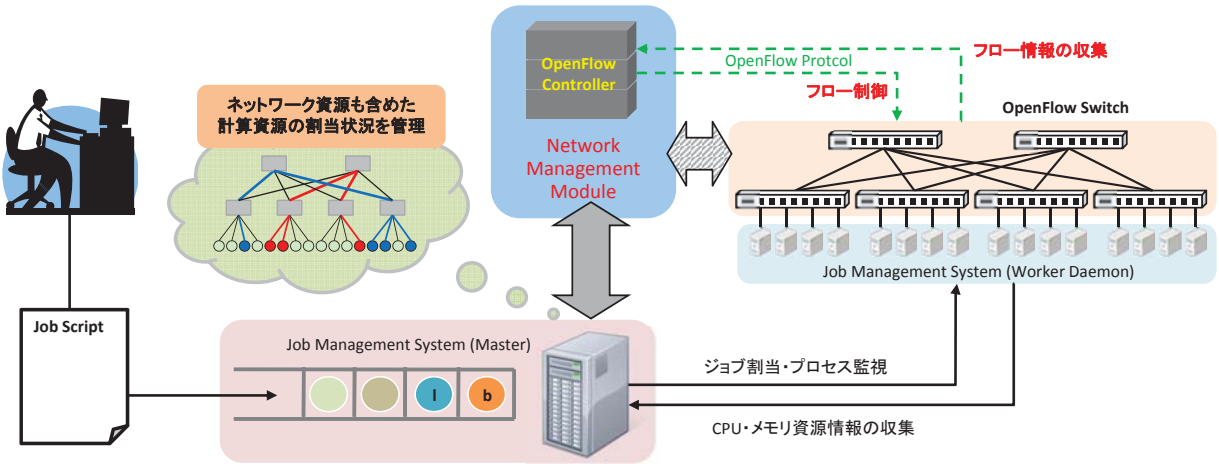


図 2 提案する SDN と連携したジョブ管理システムの概要.
Fig. 2 Job Management System leveraging SDN.

に変更する必要があった．そのため，ネットワークの規模が大きくなるほど管理者への負担は大きくなり，設定の変更は容易に実施できなかった．しかし，SDN では制御機能が集約されているため，容易に経路の変更を行うことができ，動的な対応も可能となる．また，従来は個々の制御機能が保持していた各機器の情報も，制御機能が集約されたことで情報を一元管理することができる．

この SDN のコンセプトを実現する技術として OpenFlow [8] 技術がある．OpenFlow ネットワークは，制御機能を統合した 1 つの OpenFlow コントローラと，データ転送機能を担う複数の OpenFlow スイッチで構成される．**図 1** に OpenFlow ネットワークの例を示す．OpenFlow ネットワークでは，各 OpenFlow スイッチは OpenFlow プロトコルを用いた OpenFlow コントローラからの指示に従いデータ転送を行う．この OpenFlow コントローラの指示をソフトウェアで制御することによって，ネットワークフローの動的な変更のような制御をプログラマブルに行うことができる．また，OpenFlow コントローラは各 OpenFlow スイッチのイベントや状態を収集することができるため，ネットワークの状態に基づいた動的なネットワークフローの切り替えのような制御も行うことができる．

3.3 提案システムの構成

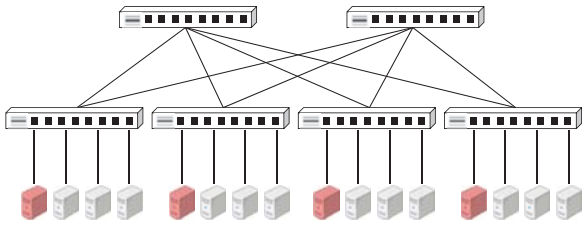
OpenFlow による SDN のコンセプトを活用したジョブ管理システムの構成を**図 2** に示す．図 2 において，従来のジョブ管理システムのフレームワークとネットワーク資源の間を繋ぐ役割を果たすモジュールとして，Network Management Module (NMM) を新たに拡張し，ネットワーク資源に対する機能 (1)〜(3) をこのモジュールで実現する．

機能 (3) は OpenFlow 技術によって実現するため，NMM は OpenFlow コントローラの機能を有する必要がある．そこで，ソフトウェアで実装された OpenFlow コントローラを NMM に組み込む形で実装する．主なソフトウェア実装の OpenFlow コントローラとして，trema [9] ，NOX [10] ，POX [11] などがある．ジョブに対するネットワーク資源の管理などは NMM が担当し，情報に基づき OpenFlow スイッチに指示を出す処理を OpenFlow コントローラが行う．また，機能 (1) の実装は OpenFlow コントローラが持つスイッチ情報の収集機能を活用して行う．しかし，OpenFlow コントローラから得られる情報だけでは資源管理を行うのに十分な情報を取得できない状況も考えられるため，NMM で他のプロトコルや手法を用いた情報収集機能を実装し，効率的な資源管理に必要な情報を収集する．

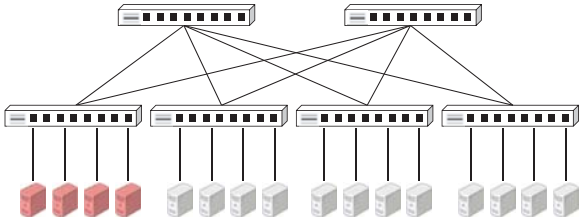
なお，ユーザがネットワーク資源に対して要求するフレームワークである機能 (4) は，ジョブ管理システムのジョブ投入コマンド (qsub) のオプションを拡張する形で設計する．システムの設計として，ネットワーク資源の要求を別のコマンドなどで行い，ネットワーク資源に関する操作は NMM で閉じて行う形態も考えたが，ユーザの利便性への影響の方が大きいと判断した．ユーザはジョブスクリプト内で CPU 数やメモリ量を指定するのと同様にネットワーク資源も要求が可能となる．実装する資源の要求方法は，当初は遅延抑制と帯域優先の 2 種類の大枠で用意し，検証を行いながら順次拡張していく予定である．

4. 機能評価

それぞれのネットワーク資源要求オプションを指定した場合における提案するジョブ管理システムによる計算資源割当と，従来システムでの動作をケーススタディを用いて



(a) 従来のジョブ管理システムでの計算資源割当の例.



(b) 提案ジョブ管理システムでの計算資源割当の例.

図 3 遅延を考慮した計算資源割当の例.
Fig. 3 An example of a latency based resource assignment.

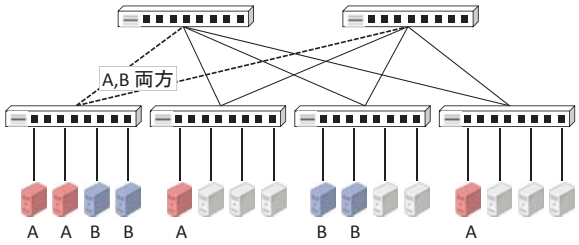
検証する．なお，この検証における資源の割当はノード単位で行い，1 つの計算ノードに複数のジョブが登録される状況は想定しない．

4.1 遅延抑制

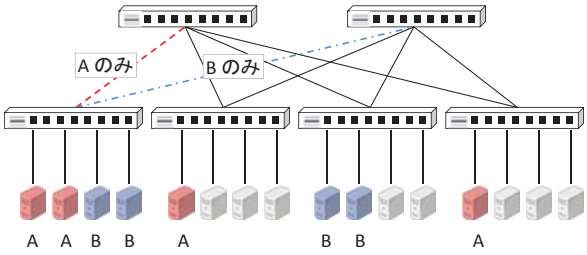
クラスタシステムにおける通信遅延は，計算ノード間のホップ数が増えるほど増加する．そのため，もっとも遅延を抑えた計算資源の割当は，計算ノード間の最大ホップ数が最小となる状態である．特に，割り当てる計算資源がすべて同一スイッチに接続されているときがもっとも遅延の小さい状態である．

遅延抑制オプションを指定した場合のケーススタディを**図 3** に示す．この例では，16 台の計算ノードを 2 段 Fat Tree で接続したクラスタシステムに対し，4 台の計算ノードがジョブに割り当てられている．図 3 (a) では，ジョブに割り当てられたすべての計算ノードが異なるスイッチに接続されており，遅延の大きい資源割当となっている．この資源の割り当て方は従来のジョブ管理システムではあり得る状況である．なぜなら，従来のジョブ管理システムは CPU のロードアベレージや空きメモリ量などで資源の割当を決定しており，ネットワーク資源に関する情報は持っていないため，この状況を確実に回避することはできない．

一方，図 3 (b) では，ジョブに割り当てられたすべての計算ノードが同じスイッチに接続されており，もっとも遅延の小さい割当となっている．提案システムでは，NMM がネットワークトポロジ情報を把握し，資源割当でこの情報も考慮するため，確実に集約して割り当てることが可能である．また，運用時の複数ジョブが投入される状況で，ネットワークトポロジを把握しながら各ジョブに計算資源



(a) 従来システムでの計算資源の利用例.



(b) 提案ジョブ管理システムでの計算資源の利用例.

図 4 帯域を考慮した計算資源割当の例.
Fig. 4 An example of a bandwidth based resource assignment.

を割り当てることで，1 台のスイッチに接続された計算資源を確保することも可能である．

4.2 帯域優先

帯域考慮オプションを指定した場合のケーススタディを**図 4** に示す．用いるクラスタシステムは，4.1 と同様，16 台の計算ノードを 2 段 Fat Tree 接続で構成されている．このクラスタシステムに A, B 2 つのジョブが投入されており，それぞれ 4 台の計算ノードが割り当てられている．なお，このジョブは両方とも広帯域を必要とするノード間通信を行うと仮定する．この場合，計算ノードが 4 台とも割り当てられている下段左端のスイッチから各上流スイッチへの経路がボトルネックとなりやすい．

図 4 (a) では，Fat Tree の冗長パスでのパケットの振り分けを，多くの場合，ECMP (Equal Cost Multi Path) [12] の機能で行っている．ECMP はスイッチの機能として実装されており，冗長経路でのパケット振り分けルールもハードウェアに依存する．パケットの振り分けルールは発信元／宛先の IP アドレスや Hash 値で決定される．クラスタシステムにおいては，割り当てる計算ノードの組み合わせや利用方法によって経路が変わるため，一般的には Hash 値を用いることが多い．冗長経路におけるパケットの分散性能は，全帯域を確保できるほど高くはなく，空いている経路があるのに混雑する経路に振り分ける場合がある．そのため，この例のような資源割当となった場合，A, B 両方のジョブに対して影響を与えてしまう場合がある．

一方，提案システムを利用した図 4 (b) の場合，各ジョブが利用するネットワークトポロジは NMM によって管理

されており，ジョブ実行時にネットワークフローを設定する．この機能により，図 4 (b) に示すように，一方の経路はジョブ A が使用し，もう一方はジョブ B に割り当てられ，それぞれのジョブがネットワーク資源を専有している状態を作ることも可能である．

また，ジョブ毎にネットワークフローが明確になっていることによって，障害発生時などの対応が容易となるという利点もある．例えば，この例において一番左端の経路で障害が発生した場合，提案システムでは影響を受けるのがジョブ A だと判明するため，ジョブ A にリスタート処理を行えば良いが，従来システムでは両方のジョブの通信が残った経路を利用し始めるので，両方のジョブに影響を与えることになる．

5. おわりに

本論文では，SDN 技術を活用することで，従来の CPU やメモリだけでなく，ネットワーク資源も考慮した計算資源の割当を実現するジョブ管理システムについて提案し，その有用性についてケーススタディを用いて検証した．今日のクラスタシステムにおける大規模化・分散化の傾向より，SDN 技術によってインターコネクトを動的に制御し，効率的な利用を実現することは，計算資源の運用とユーザの利用の両面で有用であると考える．

今後の課題として，ネットワーク資源の情報収集の手法について検討する必要がある．現在は OpenFlow 技術を活用することを目的としているため，本技術が有する情報収集機能を最大限活用する方向で検討している．しかし，効率的なネットワーク資源の管理を行うには，OpenFlow の機能だけでは実現できないと考えており，取得すべき情報およびその手段について調査・検討を行う予定である．

また, NMM の実装に関して, 既存ジョブ管理システムとの連携をどのように実現するかを検討する必要がある．例えば, 既存ジョブ管理システムには極力手を加えず, NMM を外部モジュールとすれば，開発コストが抑えられてかつ他の資源管理システムへの適用は容易となるが，機能面での制限や性能に影響があると考える．一方，ジョブ管理システムに組み込む形で実装すれば逆の状況となる．この点について，既存研究などにおける開発コストなどについて調査を行うとともに，プロトタイプの開発を行ってみて開発コストと性能・機能の検証を行う必要があると考える．

謝辞 本研究は, 独立行政法人情報通信研究機構（NICT）の委託研究「仮想分散コンピューティング・データ流通技術の研究」の支援に基づき推進している．

参考文献

[1] TOP 500 Supercomputer Sites：http://top500.org/ .
[2] B. Kingsbury, ”The Network Queuing System, Sterling Software”, Palo Alto, 1985.
[3] PBS Works - Enabling On-Demand Computing: http://www.pbsworks.com/ .
[4] Platform LSF: http://www.platform.com/workload-management/high-performance-computing .
[5] Condor Project Homepage：http://research.cs.wisc.edu/condor/ .
[6] The official Open Source Grid Engine：http://gridscheduler.sourceforge.net/ .
[7] 白井達也，斎藤秀雄，田浦健次朗，”高速なトポロジ推定：ネットワークを考慮した並列計算のための基盤として”，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.SIG13(ACS19)，pp.156-165，2007.
[8] Openflow - enable innovate in your network: http://www.openflow.org/ .
[9] Trema: Full-stack openflow framework for ruby/c: http://trema.github.com/trema/ .
[10] NOX: http://www.noxrepo.org/nox/about-nox/ .
[11] POX: http://www.noxrepo.org/pox/about-nox/ .
[12] 802.1Qbp - Equal Cost Multiple Paths: http://www.ieee802.org/1/pages/802.1bp.html .
[13] Myricom: http://www.myricom.com/ .
[14] InfiniBand Trade Association: http://www.infinibandta.org/ .