Vol.2017-HPC-160 No.3
2017/7/26

インタークラウド環境における計算資源の動的再構成フレームワークの提案

丹生 智也[1,a)]　合田 憲人[1,b)]　竹房 あつ子[1,c)]　政谷 好伸[1,d)]　横山 重俊[1,2,e)]

概要：近年のクラウド計算基盤の高性能化に伴い，複数のクラウド基盤を連携して活用するインタークラウドへの関心が高まっている．インタークラウド上で効率よくアプリケーションを実行するには，実行状況に応じた計算資源の再構成が重要だが，既存システムの計算資源の割り当て方法は，あらかじめ確保された計算資源の利用効率を最大化することを目的としており，資源の動的な追加や削除が可能なクラウド環境を前提とした資源割り当て方法として最適とは言い難い．またアプリケーションごとに適切な再構成戦略は異なるが，既存の再構成システムはオートスケーリングなどの一般的な再構成手法のみしかサポートしていないものが多く，アプリケーションに特化した再構成の実現は容易ではない．これらの問題を解決するため，本論文では，計算資源の動的な確保および削除を行うことを前提とし，アプリケーションの要求に合わせて計算資源の再構成を行うフレームワークを提案する．本フレームワークでは，再構成に必要な計算資源が満たすべき要求が計算資源の性質に関する制約（資源要求制約）に帰着できることに着目して分割された，アプリケーションスケジューラおよびリソースアロケータの 2 つのサブシステムのインタラクションにより計算資源の再構成を行う．これにより，アプリケーションからは独立にリソースアロケータのモデル化や実装が可能になり，またアプリケーションスケジューラを実装することで，リソースアロケータに変更を加えることなく新たなアプリケーションの再構成が可能になる．最後にサブシステムの一つであるリソースアロケータのプロトタイプおよび，再構成の判断に不可欠な，アプリケーションメトリクスの収集機能についても述べる．

A Dynamic Reconfiguration Framework of Computing Resources for Inter-Cloud

Tanjo Tomoya[1,a)]　Aida Kento[1,b)]　Takefusa Atsuko[1,c)]　Masatani Yoshinobu[1,d)]
Yokoyama Shigetoshi[1,2,e)]

1. はじめに

近年のクラウド計算基盤の高性能化に伴い，ゲノム解析ワークフローなどの分散並列アプリケーションをクラウド基盤上に構築するための研究が活発に行われている [1], [2], [3]．またクラウド基盤ごとに提供する計算資源の性能（e.g. CPU のコア数やメモリ容量，ストレージ性能など）は異なるため，複数のクラウド基盤を連携して活用する**インタークラウド**への関心も高まっている [4]．

インタークラウド上でのアプリケーション環境の構築は，異なるクラウド基盤上の計算資源間接続などの困難があるため，手動で行うことは容易ではない．我々は先行研究において，この問題を解決するミドルウェアである Virtual Cloud Provider（VCP）アーキテクチャを提案し，VCP を用いてインタークラウド上でアプリケーション環境を自動的に構築できることを示した [5]．

[1] 国立情報学研究所
National Institute of Informatics, 2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101–8430, Japan
[2] 群馬大学
Gunma University, 4-2 Aramaki-machi, Maebashi, Gunma, 371–8510, Japan
[a)] tanjo@nii.ac.jp
[b)] aida@nii.ac.jp
[c)] takefusa@nii.ac.jp
[d)] ym@nii.ac.jp
[e)] yoko@nii.ac.jp

しかし効率的なアプリケーション環境の実現のためには，アプリケーション環境の構築技術だけでは不十分である．例えば，ウェブアプリケーションのように負荷がユーザーからのアクセス数によって決まるアプリケーションの場合には，サービス性能を維持するために，時系列ごとに変化するユーザー数に合わせて計算資源の性能や数を変化させる必要がある．またゲノム解析ワークフローでは，計算時間やメモリ使用量がデータサイズではなくデータの内容によって決まる場合があり，さらに未知のゲノムデータの解析なども行う必要があるため，ワークフローを実行前に適切な計算資源を決定できない場合がある．このため効率的なアプリケーション実行環境の実現のためには，実行状況に合わせた計算資源の再構成が重要である．

アプリケーションに計算資源を割り当てるシステムとして，Apache Mesos [6] や HTCondor [7] などのリソースマネージャ が広く用いられている．しかしこれらのシステムが採用している資源割り当てアルゴリズムは，あらかじめ確保された計算資源の利用効率を最大化することを目的としているため，管理する計算資源の動的な確保および削除が可能なクラウド環境を前提とした資源割り当て方法として最適とは言い難い．また再構成をいつ，どのように行うのかを決定する再構成戦略に関しても，CPU 使用率などの計算資源の状態に基づき，オートスケーリング等の一般的な再構成戦略を行うシステムやサービス [8], [9] が多く，アプリケーション固有のメトリクスや，アプリケーションに特化した再構成の実現は容易ではない．

これらの問題を解決するため，本論文では管理する計算資源の動的な確保および削除を行うことを前提とし，アプリケーションの要求に合わせてアプリケーション実行環境を構成する計算資源の再構成を行うフレームワークを提案する．提案するフレームワークは，再構成に必要な計算資源が満たすべき要求が計算資源の性質（e.g. 性能，リージョン，プロバイダ）に関する制約（**資源要求制約**）に帰着できることに着目し，アプリケーションの監視および資源要求制約を求める**アプリケーションスケジューラ**と，資源要求制約を満たす計算資源を求め，VCP を用いて各クラウド基盤から計算資源を確保して返す**リソースアロケータ**から構成される．アプリケーションスケジューラは，ツールの実行速度などのアプリケーションに依存したメトリクスを資源要求制約に変換し，資源要求制約を満たす計算資源をリソースアロケータに要求する．そのため，アプリケーションからは独立にリソースアロケータのモデル化や実装が可能になる．またアプリケーションスケジューラを実装することで，リソースアロケータに変更を加えることなく新たなアプリケーションの再構成が可能になる．

計算資源の再構成戦略の実現のためには，アプリケーションおよびアプリケーションを動かす計算資源の実行時の状態 (**メトリクス**) の監視および取得が必要である．本論文ではアプリケーションスケジューラを実装する準備段階として，ゲノム解析アプリケーション環境を対象にメトリクス収集機能の実装を行った．実装した収集機能を活用することで，ゲノム解析アプリケーションスケジューラの実装が可能となる．

またアプリケーションスケジューラが求める資源要求制約については，GPU を搭載する計算資源やスーパーコンピュータ等への対応のため，今後拡張することが考えられる．本論文では資源要求制約の拡張を容易にするため，制約ソルバに基づくリソースアロケータの提案を行う．提案するリソースアロケータは資源要求制約を受け取り，要求を満たす計算資源を求める問題を整数および浮動小数点数上の制約（**制約充足問題** [10]）に変換し，外部の制約ソルバを用いて求解を行う．メモリサイズや CPU のコア数等に関する資源要求制約を制約充足問題の制約に変換するルールはそれぞれ独立しているため，ルールの追加により新たな資源要求制約への対応が可能である．

本論文の構成は以下の通りである．第 2 節で，本フレームワークが前提としているミドルウェアの Virtual Cloud Provider について述べ，第 3 節で本フレームワークの概要について述べる．第 4 節で計算資源の再構成を行うために必要な，アプリケーションのメトリクス収集機能について述べる．第 5 節では制約ソルバに基づくリソースアロケータについて述べる．第 6 節で関連研究について述べた後，最後に第 7 節で結論を述べる．

2. Virtual Cloud Provider

Virtual Cloud Provider（VCP）[5] はインタークラウド環境にアプリケーション環境を自動的に構築するためのミドルウェアである．VCP は各クラウド基盤を SINET5 [11] L2VPLS で接続し，オンデマンドに仮想ネットワークの構築を行うことで，ネットワーク設定の自動化を実現する．また Docker [12] などの Linux コンテナ技術を活用することで，異なる計算資源上（ベアメタルおよび仮想マシン）で高速なソフトウェアのデプロイを実現する．

本論文では VCP を使用して，ベアメタルおよび仮想マシンという計算資源を動的に確保することで，アプリケーション環境の再構成を行うフレームワークの提案を行う．

3. 動的再構成フレームワーク

本フレームワークでは，以下のように計算資源の再構成が行われる（**図 1** 参照）．まず，アプリケーションごとに定義されたアプリケーションスケジューラがアプリケーションの状態を監視し，再構成の必要性を判断する．その後アプリケーションの状態やユーザーからの要求を元に，再構成に必要な計算資源が満たすべき条件を資源要求制約として表現し，資源要求制約を満たす計算資源をリソースアロケータに要求する．リソースアロケータはリソースア

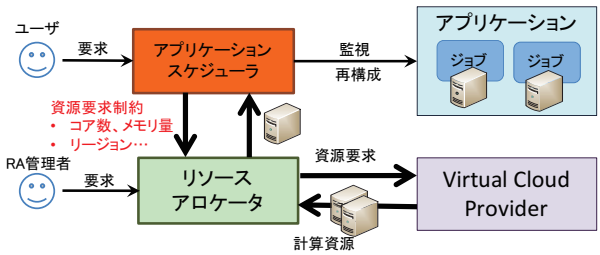

図 1 計算資源の動的再構成フレームワークの概要
Fig. 1 Overview of dynamic reconfiguration framework of computing resources

ロケータの管理者（図 1 では RA 管理者）からの要求および資源要求制約を満たす計算資源を求め，計算資源を VCP を用いて確保し，アプリケーションスケジューラに返す．最後に，アプリケーションスケジューラは受け取った計算資源を用いて，アプリケーションの再構成を行う．

本フレームワークでは，各サブシステムは計算資源要求と計算資源の授受以外の部分では独立している．そのため，アプリケーションスケジューラを変更することで，複数のアプリケーションの再構成を行うことが可能である．また VCP を活用することで，リソースアロケータは複数のクラウド基盤上の計算資源を動的に確保・削除する計算資源選択が可能になる．さらにアプリケーションの再構成と計算資源の確保戦略を別のサブシステムとして定義することで，例えばクラウドブローキングシステム [13] のようにアプリケーションのユーザと計算資源の提供者が異なるユースケースにも適用が可能になる．

以下では各システム間のインタラクションに着目して，各サブシステムの概要を述べる．**アプリケーションスケジューラ**はアプリケーションごとに定義された再構成戦略にもとづき計算資源の再構成を行うサブシステムで，以下を行う．

- アプリケーションおよびアプリケーションを動かす計算資源の実行状況を監視し，計算資源の再構成の必要性の判断を行う．
- 再構成を行うために必要な計算資源が満たすべき条件（**資源要求制約**）を求め，後述するリソースアロケーターに計算資源の確保を要求する．
- リソースアロケータから渡された計算資源を用いて，アプリケーションを動かす計算資源の再構成を行う．

リソースアロケータは与えられた資源要求制約を満たす計算資源を確保してアプリケーションスケジューラに渡すサブシステムであり，以下を行う．

- アプリケーションスケジューラから受け取った資源要求制約および，リソースアロケータ管理者から受け取った計算資源に関する要求を満たす計算資源を求める．
- 資源要求制約を満たす計算資源を VCP を用いて動的に確保し，アプリケーションスケジューラに渡す．

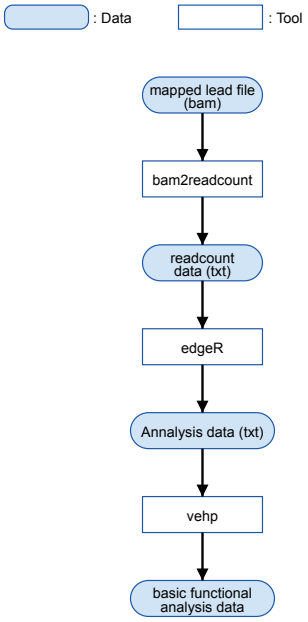

図 2 RNA シーケンス解析ワークフロー
Fig. 2 Workflow of RNA-Seq

以下にゲノム解析アプリケーションにおける再構成戦略と，アプリケーションのメトリクスを資源要求制約に変換する例を示す．

例 1 **図 2** に示すゲノム解析ワークフローの実行を考える．ワークフローは図のようなパイプライン処理であり，四角で囲まれているノードはツール，丸で囲まれているノードは入出力ファイルを表し，矢印は各データ間の依存関係を表している．ナイーブなゲノム解析アプリケーション用のアプリケーションスケジューラとして，以下の目的や戦略を持つものが考えられる．

- **目的**: 過去のツールの実行履歴（e.g. 割り当てられた計算資源の性能，処理時間や異常終了などの処理結果の情報）を元にコンテナ化された各ツールの実行に必要な資源要求制約を決定し，メモリ不足などによる異常終了率を最小化する．
- **再構成戦略**: 各ツールの実行直前に適切な計算資源を決定して確保を行う．

例えば図中のツール bam2readcount について，実行履歴から以下のことが推論できるとする．

- メモリ容量が 16GB 未満の計算資源に割り当てて実行すると，メモリ不足が原因で異常終了する．
- ストレージ容量が 16TB 未満の計算資源に割り当てて実行すると，ストレージ不足が原因で異常終了する．

この場合アプリケーションスケジューラは，「bam2readcount の異常終了率を最小化する」という再構成の目的を「メモリ容量 16GB 以上かつストレージ容量が 16TB 以上」という資源要求制約に変換し，この資源要

求制約を満たす計算資源をリソースアロケータに要求する．

3.1 資源要求制約

前節で述べたように，計算資源が満たすべき条件は資源要求制約として表現できる．資源要求制約として以下が考えられる．

- **計算資源の性能に関する制約**
  CPU のコア数やメモリ容量，ストレージ容量などの計算資源の性能に関する制約である．
- **計算資源のコストに関する制約**
  計算資源の時間あたりにかかるコストに関する制約である．
- **計算資源の配置場所に関する制約**
  この制約は，データ保護上の理由で，データの処理を特定の地域でしか行えない場合などに必要となる．
- **サービス品質（QoS）に関する制約**
  サービスの稼働率など，利用不可能になっては困る計算資源が満たすべき制約である．

アプリケーションスケジューラは，コスト最小化やツールの異常終了の最小化など，各アプリケーションスケジューラの目的を達成するために必要な条件のみを求めれば良く，指定されていない条件はリソースアロケータによって自動的に決定される．例えば例 1 のアプリケーションスケジューラの場合，メモリ容量およびストレージ容量はツールの失敗率に関係するため資源要求制約として指定されるが，その他の条件である CPU のコア数や計算資源の位置などは資源要求制約としては指定されない．

例 2 以下に資源要求制約の例を示す．

$$\begin{aligned} ncores(i1) \geq 3 \quad & \wedge \\ region(i2) = Tokyo \quad & \wedge \\ minimize(cost(i1) + cost(i2)) & \end{aligned}$$

制約中の $i1$，$i2$ は要求する計算資源を表しており，$ncores(i)$，$region(i)$，$cost(i)$ は計算資源 i の CPU コア数，リージョン，単位時間あたりのコストをそれぞれ表している．また $minimize(cost(i1) + cost(i2))$ は，アプリケーションスケジューラが単位時間あたりのコスト合計を最小化するような計算資源 $i1$ と $i2$ を要求していることを表している．

4. メトリクス収集機能

第 1 節で述べたように，計算資源の再構成には，アプリケーションおよびアプリケーションを動かす計算資源の実行状況の監視を行う必要がある．また適切な再構成戦略を実装したアプリケーションスケジューラを実現するには，アプリケーションを構成する各ツールの動作特性を知ることが不可欠である．そのため，アプリケーションスケジューラを実装するための準備段階として，ゲノム解析アプリケーション環境を対象としたメトリクス収集機能の実装を行った．

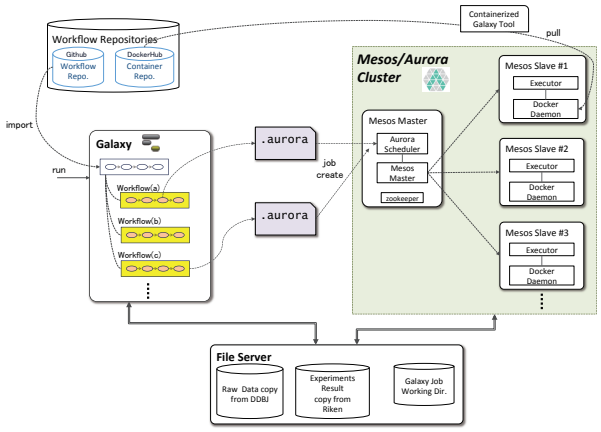

図 3 ゲノム解析アプリケーション環境
Fig. 3 Genome analysis application environment

4.1 アプリケーション環境

図 3 にメトリクス収集機能を実装するアプリケーション環境の概要を示す．このアプリケーション環境は，ゲノム解析に広く用いられているワークフローマネージャ Galaxy [14]，クラスタリソースマネージャ Apache Mesos [6]，クラスタジョブマネージャ Apache Aurora [15]，共有ファイルストレージおよびワークフローの各ツールを実行する複数の Mesos Slave から構成される．また Galaxy，Mesos Master，Mesos Slave，共有ファイルストレージはそれぞれコンテナ化されており，Master Slave 上で，コンテナ化されたワークフローの各ツールが docker in docker を用いて実行される．

本節では，ワークフローの各ツールの実行直前に適切な計算資源の決定および確保を行い，確保した資源を動的に Mesos Slave として追加・削除を行うアプリケーションスケジューラを実現するために実装したメトリクス収集機能について述べる．また収集するメトリクスは，今後アプリケーションスケジューラの再構成戦略の検討にも用いる予定のため，上記の再構成戦略では直接利用しないメトリクスについても収集できるようにした．

4.2 実装

図 4 に実装したメトリクス収集機能の概要を示す．図中で赤く囲まれているのはコンテナ内部で動くツール群，緑で囲まれているのはメトリクス収集に使われているツールおよび機能である．今回は各ツールの再現性への影響を抑えるため，コンテナ化されたツールそのものには変更を加えないメトリクスの収集機能を実装した．

各メトリクスは，収集経路によって以下のように分類できる．

- Mesos Slave から取得できるメトリクス（**表 1**）：

表 1 Mesos Slave から取得できる主要なメトリクス
Table 1 Metrics obtained from Mesos Slave

メトリクス
CPU 使用率
メモリ使用量
ディスク I/O の速度
単位時間あたりのスワップイン・スワップアウト量
ページフォルトの頻度
ネットワークの送受信量

表 2 Mesos Master から取得できる主要なメトリクス
Table 2 Metrics obtained from Mesos Master

メトリクス
稼働中計算ノード数
停止中計算ノード数
クラスタ全体使用率

収集機能は Mesos Slave 内で動くコンテナ化されたツールのメトリクスを cAdvisor [16] を用いて取得し，Fluentd [17] を用いて外部のメトリクス集積システムに送信する．この収集機能は再構成対象がコンテナ化されていれば実現可能であり，また表中のメトリクスは特定のアプリケーションに依存した項目ではないため，他のアプリケーションを対象にした場合でも，同様の方法が利用できると考えられる．

- Mesos Master から取得できるメトリクス（**表 2**）：
  収集機能はクラスタマネージャおよびジョブマネージャの利用状況を REST API を用いて収集し，Fluentd 経由でメトリクス集積システムに送信する．表中のメトリクスは，今回想定している再構成戦略では活用する予定はないが，今後分散実行を行うツールの再構成の際には活用できると考えている．またゲノム解析アプリケーション以外にも，負荷に応じてエージェントの数を増減させるクラスタマネージャを対象とするアプリケーションスケジューラに対しても，同様の方法が利用可能だと考えられる．
- Galaxy サーバから取得できるメトリクス（**表 3**）：
  これらは各ツールの実行前および実行後に取得可能な，現在実装予定のアプリケーションスケジューラで利用予定のメトリクスである．収集機能は Galaxy の内部データベースからこれらのメトリクスを取得し，Fluentd 経由で集積システムに送信する．この収集機能は Galaxy に依存しているが，取得できるメトリクスは Galaxy 以外のゲノム解析アプリケーション環境でも活用できると考えられる．

アプリケーションスケジューラの再構成戦略の決定アルゴリズムは，メトリクス集積システムから必要なメトリクスを取得することで，再構成の必要性の判断および資源要求制約の決定を行う．また表 1 および表 2 のメトリクスに関しては，一定時間ごとの値がメトリクス集積サーバーに送信されるため，今後ツールの実行途中に再構成を行うアプリケーションスケジューラを実装する際にも利用できると考えている．

表 3 Galaxy サーバから取得できる主要なメトリクス
Table 3 Metrics obtained from Galaxy server

メトリクス
ツールの実行時間
入力データのサイズ
ツールの実行成功 or 失敗
失敗原因

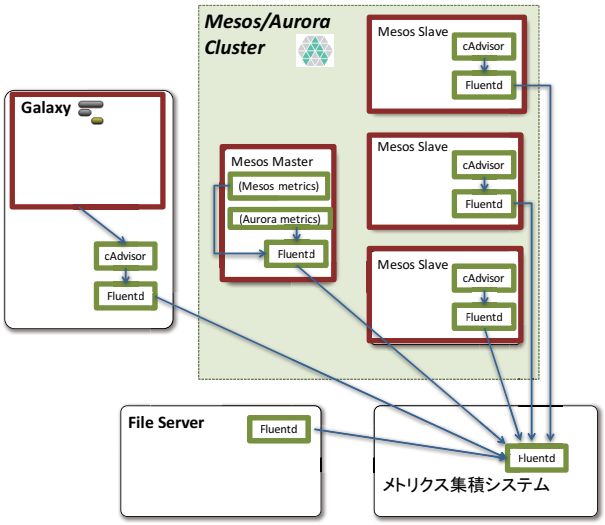

図 4 ゲノム解析アプリケーション環境におけるメトリクス収集機能の概要
Fig. 4 Overview of metrics collection scheme for genome analysis application environment

5. リソースアロケータ

第 3 節で述べたように，リソースアロケータはアプリケーションスケジューラから資源要求制約を受け取り，制約を満たす計算資源を決定する．現在想定している資源要求制約については第 3.1 節で述べたが，今後 GPU などの特殊なハードウェアを搭載した計算資源への対応や，スーパーコンピュータ等の特殊な使用条件がある計算資源への対応のために制約の拡張を行うことが考えられる．そのためリソースアローケータの資源選択アルゴリズムでは，資源要求制約が拡張されることを想定した設計が必要となる．

本節では今後の拡張を容易にしつつ第 3.1 節で述べた資源要求制約に対応するため，**図 5** に示す変換器と制約ソルバからなるリソースアロケータを提案する．提案するリソースアロケータでは，まず変換器はアプリケーションスケジューラからの資源要求制約，リソースアロケータが確保可能な各クラウド基盤の計算資源の情報およびリソースアロケータ管理者からの要求を受け取り，与えられた資源

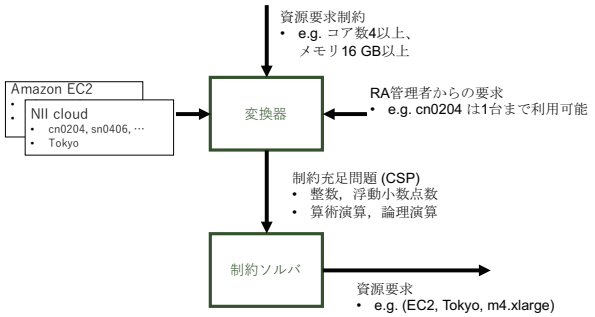

図 5 制約ソルバに基づくリソースアロケータ
Fig. 5 A resource allocator based on constraint solver

要求を満たす計算資源を求める問題を，整数および浮動小数点数上の制約（**制約充足問題** [10]）に変換する．その後制約充足問題を解く外部の制約ソルバを用いて，要求を満たす計算資源，すなわち（クラウド基盤，リージョン，フレーバー）の組を探索する．この手法では，変換器への変換ルールの追加によって資源要求制約の追加に対応できるため，今後の拡張が容易になると期待できる．

以下ではリソースアロケータの入力である計算資源情報およびリソースアロケータ管理者からの要求について述べた後，制約充足問題および制約ソルバの概要について説明する．最後に，リソースアロケータの入力を制約充足問題へ変換する方法について述べる．

5.1 計算資源情報

リソースアロケータは，クラウド基盤ごとの計算資源やリージョン情報を入力として受け取る．具体的には，以下の情報が考えられる．

- **サービス稼働率およびリージョン一覧**
  これらの情報は各クラウド基盤ごとに与えられる．
- **選択できるフレーバーおよび金銭コスト**
  Amazon EC2 [9] 等のクラウド基盤では，リージョンごとに配置できるフレーバーが異なっている．また同じフレーバーでも，配置するリージョンによって単位時間あたりの金銭コストは異なっているため，これらは各クラウド基盤のリージョンごとの情報が必要となる．
- **フレーバーごとの性能一覧**
  CPU コア数，メモリ容量，ストレージ容量等のマシン性能に関する情報はリージョンには依存せず，フレーバーによって決定される．

5.2 リソースアロケータ管理者からの要求

リソースアロケータの管理者は，オンプレミス環境の計算資源への対応等のために，リソースアロケータが確保できる計算資源に関して追加の要求を与える場合がある．リソースアロケータの管理者の要求には以下が考えられる．

- **特定のフレーバーの確保数制限**
  オンプレミス環境では確保できる計算資源の数が制限されている場合があるため，この要求が必要になる．
- **クラウド基盤の使用制限**
  メンテナンス等の理由で，特定のクラウド基盤が一時的に利用できない場合に対応するために，この要求が必要になる．

5.3 制約充足問題

制約充足問題（Constraint Satisfaction Problem; CSP）は変数の集合，各変数が取りうる値の範囲（ドメイン）および，変数上の制約の集合が与えられた時に，全ての制約を満たすような，各変数からドメイン内の値への割り当て（解）を探索する問題である[*1]．与えられた CSP を解くプログラムを**制約ソルバ**と呼ぶ．2008 年以降，制約ソルバの性能を競う MiniZinc Challenge [18] が開催されており，効率的な制約ソルバの研究・開発が活発に行われている．また MiniZinc Challenge に参加するソルバーは全て，MiniZinc 形式 [19] もしくは FlatZinc 形式で記述された CSP を入力として受け取ることができる．本論文で実装したプロトタイプでは，変換器の出力を MiniZinc 形式にすることで，リソースアロケータで用いる求解部分のアルゴリズムを，制約ソルバの変更により容易に行えるようにした．

例 3 **図 6** に MiniZinc 形式の CSP の例を示す．**var** $lb..ub : i;$ は，lb から ub の値を取る変数 i の定義を表しており，**constraint** で始まる行は各変数が満たすべき制約を表している．各制約は，加算や減算などの算術演算および算術比較，論理和 ($\vee$) や含意 ($\Longrightarrow$) などの論理演算，alldifferent (各変数の値が互いに異なる) や count (特定の値を取る変数の数を制限する) などの可変長個の変数上の制約である**グローバル制約**から構成される．**solve** で始まる行は，この CSP の目的が変数 $i1\_cost$ および $i2\_cost$ の和を最小化することを表している．この例の場合，解の一つとして値割り当て $i1\_flavor = 0, i2\_flavor = 0, i1\_ncores = 3, i2\_region = 0, i1\_cost = 0.0, i2\_cost = 0.0$ が得られる．

5.4 制約充足問題への変換

図 7 に変換器の概要を示す．変換器は資源要求制約で要求された計算資源と制約，リソースアロケータ管理者 (図中では RA 管理者) からの各要求に対応する変換ルールを適用していくことで，要求された計算資源を求める問題を CSP へと変換する．制約と要求に対する変換ルールはそれぞれ独立しているため，変換ルールを追加することで新

[*1] 厳密には制約を満たす解を探索する問題を制約充足問題，制約を満たす解のうち，与えられた目的関数を最小化（最大化）する問題を制約最適化問題（Constraint Optimization Problem; COP）と呼んで区別するが，本論文ではこれらまとめて制約充足問題と呼ぶ

```
var 0..3: i1_flavor;
var 0..3: i2_flavor;
var 1..128: i1_ncores;
var 0..2: i2_region;
var 0.0..20.0: i1_cost;
var 0.0..20.0: i2_cost;

constraint i1_ncores >= 3;
constraint i2_region = 0;

solve minimize i1_cost+i2_cost;
```

図 6 制約充足問題の例
Fig. 6 Example of a Constraint Satisfaction Problem

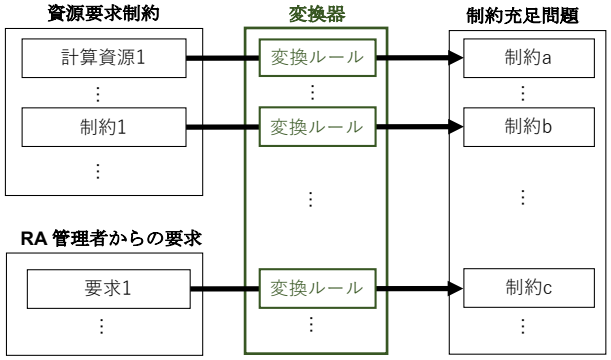

図 7 変換器の概要
Fig. 7 Overview of a translator for resource allocator

たな種類の資源要求制約やリソースアロケータ 管理者からの要求への対応が可能になる．

5.4.1 計算資源の CSP 表現

まず資源要求制約で要求された各計算資源について，フレーバー，配置リージョン，単位時間あたりのコスト，稼働率，CPU コア数等の性能に対応する CSP 変数を導入する．例えば例 2 の $i1$ に対しては，**図 8** に示すような CSP 変数を導入する．ここでフレーバーに対応する変数 $i1\_flavor$ のドメインの値は各フレーバーと対応しており，またリージョンに対応する変数 $i1\_region$ のドメインの値は各リージョンと対応している．例 2 のように同一のクラウド基盤のフレーバーを連続した整数値として表現することで，クラウド基盤を値の範囲として表現することができる．例えば $0 \leq i1\_flavor < 2$ の場合，$i1$ が Amazon EC2 の計算資源であることがわかる．またフレーバーの性能一覧は入力として与えられているため，CPU のコア数やメモリ搭載量を表す CSP 変数の上下限を計算することができる．

さらに CSP に各変数を関係付けるための制約を導入する．例えばリージョンごとに確保できる計算資源を制限するためには，**図 9** に示す制約を CSP に追加する．図中の “$->$” および “$\backslash/$” は，それぞれ含意と論理和を表している．CPU のコア数やメモリ搭載量なども同様の方法で，各

```
% フレーバー名を表す定数定義
int: ec2_m2_small  = 0; % EC2 のフレーバー1
int: ec2_m4_xlarge = 1; % EC2 のフレーバー2
int: nii_cn0204 = 2; % オンプレミスのフレーバー1
int: nii_sn0406 = 3; % オンプレミスのフレーバー2
% リージョン名を表す定数定義
int: Tokyo = 0;
int: Ohio = 1;
int: Oregon = 2;
% 計算資源ごとに導入される変数
var 0..3: i1_flavor;                    % フレーバー名
var 0..2: i1_region;                    % リージョン
var 0.0..0.4: i1_cost;                  % コスト
var 0.0..99.999: i1_availability;       % 稼働率
var 1..32: i1_ncores;                   % コア数
...
```

図 8 例 2 の $i1$ に対して導入される変数
Fig. 8 Variables that are introduced for $i1$ in Example 2

```
% 東京リージョンなら i1 はオンプレミスのフレーバーのいずれか
constraint i1_region = Tokyo ->
    (i1_flavor = nii_flavor1 \/
     i1_flavor = nii_flavor2);
% オハイオリージョンなら i1 は EC2 のフレーバーのいずれか
constraint i1_region = Ohio ->
    (i1_flavor = ec2_m2_small \/
     i1_flavor = ec2_m4_xlarge);
...
```

図 9 リージョンごとのフレーバーを制限する制約
Fig. 9 Constraints to restrict flavors for each region

フレーバーごとに取りうる CPU のコア数等を制限する制約を導入すれば良い．

5.4.2 資源要求制約およびリソースアロケータ管理者からの要求の CSP 表現

資源要求制約は，CSP 変数を用いた算術比較として直接的に表現できる．例えば例 2 において計算資源 $i1$ の CPU コア数を 3 以上に制限する資源要求制約は，$i1\_ncores \geq 3$ という CSP 制約として表現できる．またリソースアロケータ管理者からの要求のうちクラウド基盤の制限は，フレーバーを表す変数が取りうる値の範囲を制限する CSP 制約として表現できる．特定フレーバーの確保数を制限する要求は，グローバル制約 count[*2]を使った CSP 制約として表現できる．例えば図 8 で示したオンプレミスのフレーバー $cn0204$ の確保数を 1 以下に制限する要求は，$\mathsf{count}(nii\_cn0204, [i1\_flavor, i2\_flavor], \leq, 1)$ という CSP 制約として表現できる．この制約は $i1\_flavor, i2\_flavor$ のうち nii_cn0204（$= 2$）になっている変数の数が 1 以下であることを表している．

[*2] FlatZinc では count_ge 制約として提供されている

6. 関連研究

6.1 リソースマネージャ

Krauter らは，計算資源を管理するリソースマネージャについてまとめている [20]．第 1 節で述べたように，これらのリソースマネジャの資源選択アルゴリズムは，あらかじめ確保した計算資源の利用効率を目的としたものであるため，本論文で提案するリソースアロケータの資源選択アルゴリズムには適していない．そのため，計算資源の動的な追加および削除を前提とする資源選択アルゴリズムを新たに開発する必要がある．

前節で利用した Apache Mesos [6] は，計算資源の管理を行う Mesos Master と，アプリケーションごとに必要な計算資源の条件を管理する Mesos Scheduler から構成される．Mesos Master は自身が管理する計算資源の利用効率を最大化するため，各 Mesos Scheduler に対して未使用の計算資源が必要かを問い合わせる（Resource Offer）ことで，計算資源の再構成を促す．また Mesos Master には，自動的に計算資源の追加・削除を行う機能は含まれていない．本論文で提案する再構成フレームワークは，アプリケーションに依存しない資源管理部とアプリケーションごとに定義される再構成部から構成される点では類似している．しかし，本フレームワークはリソースアロケータが管理する計算資源を動的に追加・削除を行うことを想定しており，Mesos とは資源管理部を実現するための前提条件が異なる．

インタークラウド上のアプリケーション環境構築およびマネジメントを行うシステムとして，CYCLON [4] プロジェクトの SlipStream が挙げられる．SlipStream は仮想マシンイメージを用いることでアプリケーション環境を構築し，また REST API を用いたオートスケーリングに対応している．本論文で提案するフレームワークでは，再構成戦略は各アプリケーション用に定義されるアプリケーションスケジューラによって決まる．そのため SlipStream が標準ではサポートしていない，例 1 のようにアプリケーションに特化した再構成方法も実現可能である．また提案するフレームワークは Linux コンテナを前提としているため，仮想マシンイメージを用いる場合よりも，実行時のオーバーヘッドが小さいと期待できる．

6.2 アプリケーションの再構成

Yu ら [21] はワークフローのスケジューリングや，ツールの実行が失敗した際の，計算資源の再割り当て戦略等についてまとめている．ゲノム解析アプリケーションを対象としたアプリケーションスケジューラの再構成アルゴリズムとして，これらの方法を用いることが可能である．

三浦ら [22], [23] は，三層ウェブアプリケーションやワークフローの各ツールの仕様を一階述語論理を用いて記述し，与えられた論理式をアプリケーションごとに定義された式変形ルールに基いて変形することにより，各ツールに適した計算資源を求める方法を提案している．式変形のルールを変更することで，計算資源そのものではなく資源要求制約を求めることも可能であるため，この方法を用いてアプリケーションスケジューラの再構成戦略を実装することも可能だと考えられる．

7. 結論

本論文では計算資源の動的な確保および削除を行うことを前提とし，アプリケーションの要求に合わせて計算資源の再構成を行うフレームワークを提案した．提案したフレームワークは再構成に必要な計算資源が満たすべき要求が資源要求制約に帰着できることに着目し，アプリケーションの再構成戦略に基づいて資源要求制約を求めるアプリケーションスケジューラと，アプリケーションスケジューラから与えられた資源要求制約を満たす計算資源を選択，確保を行うリソースアロケータから構成される．これにより，アプリケーションからは独立にリソースアロケータのモデル化や実装が可能になり，またアプリケーションスケジューラを実装することで，リソースアロケータに変更を加えることなく新たなアプリケーションの再構成が可能になる．またゲノム解析アプリケーションを対象としたメトリクス収集機能および制約ソルバに基づくリソースアロケータについても述べた．

今後は第 4 節で実装したメトリクス収集機能を利用し，Galaxy をベースにアプリケーションスケジューラのプロトタイプ実装を行い，本フレームワークの有効性の検証を行う予定である．また三層ウェブアプリケーションなどの他のアプリケーションや，GPU などの特殊なハードウェアを搭載した計算資源を用いる再構成シナリオについて必要な資源要求制約を議論し，本フレームワークの適用範囲を広げていく予定である．

謝辞 VCP および動的再構成フレームワークのメトリクス収集機能の実装の支援をして頂いた那須野 淳氏に感謝致します．本研究は，JST CREST の支援（グラント番号 JPMJCR1501）を受けたものである．

参考文献

[1] Juve, G., Deelman, E., Vahi, K., Mehta, G., Berriman, B., Berman, B. P. and Maechling, P.: Scientific workflow applications on Amazon EC2, *Proceedings of the 2009 5th IEEE International Conference on E-Science Workshops*, pp. 59–66 (online), DOI: 10.1109/ESCIW.2009.5408002 (2009).

[2] Jackson, K. R., Ramakrishnan, L., Muriki, K., Canon, S., Cholia, S., Shalf, J., Wasserman, H. J. and Wright, N. J.: Performance Analysis of High Performance Computing Applications on the Amazon Web Services

Cloud, *Proceedings of the 2010 IEEE Second International Conference on Cloud Computing Technology and Science*, CLOUDCOM '10, Washington, DC, USA, IEEE Computer Society, pp. 159–168 (online), DOI: 10.1109/CloudCom.2010.69 (2010).

[3] Marathe, A., Harris, R., Lowenthal, D. K., de Supinski, B. R., Rountree, B., Schulz, M. and Yuan, X.: A Comparative Study of High-performance Computing on the Cloud, *Proceedings of the 22Nd International Symposium on High-performance Parallel and Distributed Computing*, HPDC '13, New York, NY, USA, ACM, pp. 239–250 (online), DOI: 10.1145/2462902.2462919 (2013).

[4] Gallico, D., Biancani, M., Blanchet, C., Bedri, M., Gibrat, J. F., Baranda, J. I. A., Hacker, D. and Kourkouli, M.: CYCLONE: A Multi-cloud Federation Platform for Complex Bioinformatics and Energy Applications (Short Paper), *Proceedings of the 2016 5th IEEE International Conference on Cloud Networking (Cloudnet)*, pp. 146–149 (online), DOI: 10.1109/CloudNet.2016.44 (2016).

[5] Yokoyama, S., Masatani, Y., Ohta, T., Ogasawara, O., Yoshioka, N., Liu, K. and Aida, K.: Reproducible Scientific Computing Environment with Overlay Cloud Architecture, *Proceedings of the 9th IEEE International Conference on Cloud Computing (IEEE Cloud 2016)* (2016).

[6] The Apache Software Foundation: Apache Mesos. http://mesos.apache.org/.

[7] Litzkow, M., Livny, M. and Mutka, M.: Condor - A Hunter of Idle Workstations, *Proceedings of the 8th International Conference of Distributed Computing Systems*, pp. 104–111 (1988).

[8] The Linux Foundation: Kubernetes. https://kubernetes.io/.

[9] Amazon Web Services, Inc.: Amazon EC2. http://aws.amazon.com/ec2/.

[10] Rossi, F., Beek, P. v. and Walsh, T.: *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*, Elsevier Science Inc., New York, NY, USA (2006).

[11] Urushidani, S., Abe, S., Yamanaka, K., Aida, K., Yokoyama, S., Yamada, H., Nakamura, M., Fukuda, K., Koibuchi, M. and Yamada, S.: New Directions for a Japanese Academic Backbone Network, *IEICE Transactions on Information and Systems*, Vol. E98.D, No. 3, pp. 546–556 (2015).

[12] Docker, Inc.: Docker. https://www.docker.com/.

[13] Petty, Christy and Meulen, Rob van der: Gartner Says Cloud Consumers Need Brokerages to Unlock the Potential of Cloud Services. http://www.gartner.com/newsroom/id/1064712.

[14] Galaxy: Data intensive biology for everyone. https://galaxyproject.org/.

[15] The Apache Software Foundation: Apache Aurora. http://aurora.apache.org/.

[16] Google, Inc.: cAdvisor. https://github.com/google/cadvisor.

[17] Treasure Data, Inc.: Fluentd. http://www.fluentd.org/.

[18] Stuckey, P. J., Feydy, T., Schutt, A., Tack, G. and Fischer, J.: The MiniZinc Challenge 2008-2013, *AI Magazine*, Vol. 35, No. 2, pp. 55–60 (2014).

[19] Nethercote, N., Stuckey, P. J., Becket, R., Brand, S., Duck, G. J. and Tack, G.: MiniZinc: Towards a Standard CP Modelling Language, *Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming*, CP'07, Berlin, Heidelberg, Springer-Verlag, pp. 529–543 (online), available from ⟨http://dl.acm.org/citation.cfm?id=1771668.1771709⟩ (2007).

[20] Krauter, K., Buyya, R. and Maheswaran, M.: A taxonomy and survey of grid resource management systems for distributed computing, *Software: Practice and Experience*, Vol. 32, No. 2, pp. 135–164 (online), DOI: 10.1002/spe.432 (2002).

[21] Yu, J. and Buyya, R.: A Taxonomy of Scientific Workflow Systems for Grid Computing, *SIGMOD Record*, Vol. 34, No. 3, pp. 44–49 (online), DOI: 10.1145/1084805.1084814 (2005).

[22] Miura, K. and Munetomo, M.: A Predicate Logic-Defined Specification Method for Systems Deployed by Intercloud Brokerages, *2016 IEEE International Conference on Cloud Engineering Workshop (IC2EW)*, pp. 172–177 (online), DOI: 10.1109/IC2EW.2016.47 (2016).

[23] Miura, K., Ohta, T., Powell, C. and Munetomo, M.: Intercloud brokerages based on PLS method for deploying infrastructures for big data analytics, *Proceedings of the 2016 IEEE International Conference on Big Data, Big-Data 2016*, pp. 2097–2102 (online), DOI: 10.1109/BigData.2016.7840836 (2016).