Vol.2018-IOT-40 No.5
2018/3/5

学内向けオープンクラウドにおける計算資源の動的な増減への対応手法に関する検討

永井 陽太[1,a)]　松原 克弥[1,b)]

概要：情報系大学では，演習活動の高度化に伴って，学生 1 人に対して複数台のマシンが提供できるクラウドコンピューティング環境の要求が高まっている．しかし，設定される演習課題には，学内向けサービスとの連携を想定するものもあり，商用のパブリック・クラウドでは対応が困難な場合もある．一方で，大規模計算機管理を担う機関を持たない小規模大学では，多大な初期費用と管理コストを必要とするオンプレミス・クラウド導入に対する敷居が高い．本研究では，BYOD の普及による教室 PC の稼働率低下に着目し，教室 PC の余剰資源を活用した学内向けオープンクラウドの実現を目指している．教室 PC を活用したクラウド基盤では，授業利用や電源断により，クラウドに利用可能な計算資源が動的に増減する．本稿では，効率的なライブマイグレーション機構を実現することにより，積極的なインタンス再配置による計算資源の動的な増減に対応する手法を提案する．

キーワード：クラウドコンピューティング，ライブマイグレーション，コンテナ型仮想化

An Approach of Handling Unstable Computational Resources Toward an In-Campus Open Cloud Infrastructure

Abstract: Along with introducing advanced practical exercises especially in college of computer science, cloud computing environment, which can provide multiple machine instances for each student, becomes more expected. Unfortunately, the commercial public cloud service could not meet the expectation in case that collaboration with internal services may be required. Meanwhile, it could be hard for small colleges to carry initial and running cost of large-scale servers for on-premises private cloud infrastructure.
This research objective realizes an in-campus open cloud service with utilizing surplus resource of classroom PCs. The classroom PCs used as a cloud platform have issues that they may be down unexpectedly, or occupied for lesson. This paper describes implementation of an open cloud infrastructure on such the unstable platform with proposed mechanism of dynamic reallocation of instances by using OpenStack's alive monitoring and CRIU's checkpoint & restore function.

Keywords: Cloud Computing, Live Migration, Container type Virtialization

1. はじめに

文部科学省が平成 28 年度に公表した「平成 28 年度学術情報基盤実態調査 (概要)」によると，日本の 80.6%の大学がクラウドコンピューティング (以下，クラウド) を導入している [1]．また，近年の大学情報系学科では，PBL 等の実践的な開発演習の導入が進んでおり，学生 1 人に対して複数台のマシンが提供できるクラウドの需要が高まっている．クラウドを大学に導入する場合，商用のパブリック・クラウドを利用する方法と，大学内にプライベート・クラウド (以降，オンプレミス・クラウド) を構築する方法が存在する．商用のパブリック・クラウドを利用する場合，学内で展開されているサービスとの連携が困難になってしまうという課題が存在する．一方，情報基盤センタ等の計算機管理組織を持たない大学でオンプレミス・クラウドを構築するには，サーバの導入に伴う多大な初期費用と管理コ

[1] 公立はこだて未来大学 システム情報科学部
School of Systems Information Science, Future University Hakodate
[a)] b1014120@fun.ac.jp
[b)] matsu@fun.ac.jp

ストがかかるという課題が存在する．一方で，大学において BYOPC(Bring Your Own PC) が普及しつつあり，全学生がノートパソコン等の計算機を所有していることが仮定できる状態になりつつある．それに対応して，統一的な演習環境として導入・提供されている教室 PC が授業以外の時間に利用されなくなり，教室 PC の稼働率低下に伴う余剰な計算資源が増加している．

本研究は，教室 PC の余剰計算機資源を活用する学内オープン・クラウドを構築することを目的としている．オープンソースのクラウド構築ソフトウェアを活用しつつ，計算ノードとして教室 PC を利用できるクラウド基盤を実装することにより，大規模サーバ導入等のコスト負担を最小限にしつつ，学内サービスとも連携可能なオンプレミス・クラウドを実現する．本稿では，本研究の実装基盤として，OpenStack と Docker を用いたオンプレミス・クラウドの構築について紹介する．

前述のクラウド基盤において，教室 PC を計算機資源として利用する際の最大の課題のひとつは，計算資源量が不定で予測が困難なことである．オープンクラウドを含む多くのクラウド基盤ソフトウェアでは，インスタンスの生成時に計算機資源の割り当てを行う．しかし，教室 PC を計算機資源として利用する場合，資源割り当て後にも計算機資源量が増減するため，インスタンス生成時の資源割り当てが最適でなくなる可能性がある．また，授業の利用や PC 教室に入室可能な学生等による電源断により，インスタンスに割り当てた計算機資源が消失することにも対応する必要がある．これらの課題への対処として，ライブマイグレーションによるインスタンスの再配置が有効であると考えるが，動作中のインスタンスのメモリ等の状態をファイル保存し，ネットワーク転送した後，移送先で実行状態を復元するためのオーバヘッドは少なくない．本稿では，ライブマイグレーションのオーバヘッドを軽減することを目的として，分身型マイグレーションを提案する．教室 PC 間でインスタンスのマイグレーションを実施した際に，移送元のインスタンスを削除せずに停止状態のまま保持し，移送先で実行再開後に更新されたメモリ領域を管理することで，電源断により再度マイグレーションが必要な際に，移送元の PC を優先的に再度移送先として選択する．再度のマイグレーションでは，移送後に変更された部分のみを移送元へコピーバックすることで，移送元での実行再開までのオーバヘッドを最小化する．また，授業等のための計画的なインスタンス再配置では，プレコピーにより実行状態を移送先にコピーした後にインスタンスを停止し，前述と同様にコピー中に変更された部分のみを追加移送することで，インスタンスの停止時間を最小化する．本提案の効率的なライブマイグレーションを活用することにより，積極的なインスタンス再配置を行う機構を OpenStack に追加実装し，計算機資源の動的な増減に対応するクラウド基盤を実現する．

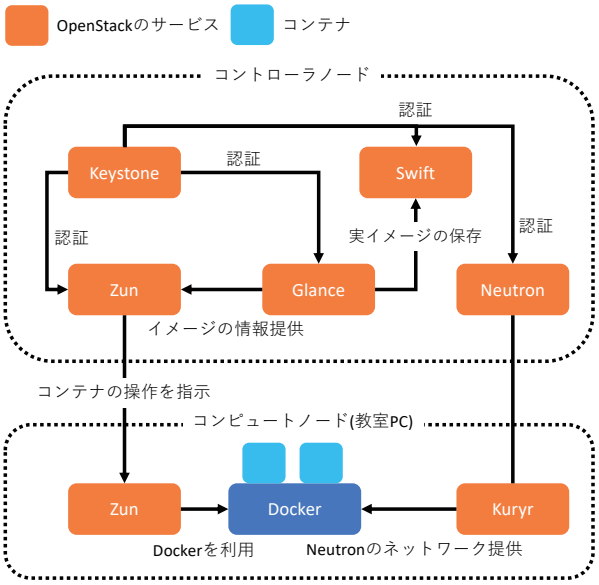

図 1　コントローラノードとコンピュートノードの構成

2. 学内オープンクラウドの構築

学内オープンクラウドの構築には，オープンソースなクラウド管理システムである OpenStack を利用する．また，インスタンスの軽量化のためにインスタンスはコンテナ型仮想化とする．

2.1 システムの構成

OpenStack を用いてクラウド環境を構築する場合，1 つのコントローラノードと複数のコンピュートノードを構築する．コントローラノードとは，クラウド環境全体の管理を行うノードである．コンピュートノードとは，実際のインスタンスを立ち上げるノードである．つまり，教室 PC がコンピュートノードとなる．

2.1.1 コントローラノードの構成

コントローラノードは専用のサーバを用意する．図 1 に示すように，コントローラノードに各サービスをインストールする．Keysotne は，コントローラノード上で各サービスの認証とエンドポイントの管理を行う [2]．クラウド環境の管理者や利用者からのあらゆるリクエストはこの Keystone で認証が行われる，Neutron は，クラウド環境上のネットワークの管理を行う [3]．ネットワーク上に仮想ルータや dhcp の機能を提供することでクラウド上のインスタンスにネットワークを提供する．また，Neutron はアベイラビリティゾーンという抽象化で，コンピュートノードをグルーピングする．この機能によってコンピュートノードを教室単位でグルーピングすることが可能である．Glance は，OS やコンテナイメージのメタ情報を管理

する [4]．メタ情報には，名前やサイズ，最小限必要なディスク領域や実際のイメージの保存場所などが保存されている．Swfit は，クラウド上にオブジェクトストレージを提供するサービスである [5]．このオブジェクトストレージには，Glance によって管理されているイメージの実際のファイルが保存される．コントローラノード上の Zun は，コンテナの操作に関わる REST リクエストを受け付けて，コンピュートノード上の Zun にコンテナ操作の指示をだす．また，コンピュートノード上の Zun からコンピュートノードのリソースの状態を受け取りデータベースに登録している．各コンピュートノードのリソースの状態を取得することによって，コンテナ作成のリクエストを受け付けた際に，最適なコンピュートノードを決定するフィルタ機能を実現している．Zun のフィルタ機能には以下の 2 つのフィルタが存在する [6]．

- CPUFilter : CPU コアの使用率にもとづいたフィルタである．このフィルタはコンテナの作成に十分な CPU コア数のホストを通過させる
- RamFilter : RAM の使用率にもとづいたフィルタである．このフィルタは RAM にコンテナの作成に充分な余裕があるホストを通過させる

2.2 コンピュートノードの構成

コンピュートノードには，教室 PC を利用する．図 1 に示すように，コンピュートノードに各サービスをインストールする．Docker は，コンテナ操作のドライバとしてコンピュートノード上の Zun によって利用される．つまり，実際にコンテナの作成・削除などの操作を行うのは Docker である．Kuryr は，Docker のネットワークプラグインとして機能する．Docker は Kuryr を利用することで，Neutron が管理しているネットワークに接続することが可能になる．コンピュートノード上の Zun は，コントローラノード上の Zun からコンテナ操作の指示を受け取ると，内部で Docker を利用することでコンテナの操作を行う．

3. 動的増減する計算資源への対応手法

教室 PC を学内オープンクラウドの計算資源として利用する場合に考えられる技術的課題とその解決手法について述べる．

3.1 頻繁なコンピュートノードの増減

教室 PC は後述する 2 つの理由で，頻繁にコンピュートノードが増減するとかんがえられる．

1 つめの理由を述べる．教室 PC は本来，大学のリテラシー教育やプログラミング演習などの授業によって利用されることを目的に設置されている．そのため，学内オープンクラウドの計算資源として教室 PC が利用されることで，教室 PC に負荷がかかってしまい，教室 PC の本来の利用目的に支障をきたす可能性が存在する．また，インスタンスが動作している教室 PC が授業利用されることでインスタンスの処理性能にも悪影響をおよぼしてしまう．つまり，授業利用されている間はその教室 PC を計算資源から離脱させることが望ましい．しかし，教室 PC は授業利用される頻度が高いので，頻繁にコンピュートノードが離脱・参加を繰り返すことになるとかんがえられる．

2 つめの理由を述べる．教室 PC は，大学関係者であれば誰でも利用できるような環境に置かれているため，教室 PC が学内オープンクラウドの計算資源として利用されていることを知らずに，教室 PC をシャットダウンしてしまうことが考えれる．このような環境では，頻繁に教室 PC がシャットダウンされてしまい，コンピュートノードが減ることが考えられる．

上述した 2 つの理由によって，コンピュートノードが頻繁に増減を繰り返し，計算資源が不安定になることが考えられる．計算資源が不安定になると，インスタンスの性能にも悪影響を及ぼす．

3.2 解決手法：積極的インスタンスの再配置

現状の Zun では，インスタンスの生成時にフィルタ機能を用いて，最適なコンピュートノードを決定し，インスタンスを作成している．しかし，教室 PC を用いたクラウドでは，計算資源の状態が時間とともに常に変化する．そこで．計算資源の増減の際に，インスタンスの性能を高めるために，より最適なコンピュートノードへ積極的にインスタンスの再配置を行うことで，不安定な計算資源の上でもインスタンスの性能を最大限高めることを目指す．以下では，インスタンス再配置機構の設計について詳しく述べる．

3.2.1 インスタンス再配置の契機条件

インスタンスの再配置条件としては，コンピュートノードの増減を想定する．コンピュートノードの増加は，コンピュートノードから参加のリクエストを送信することで，増加とする．コンピュートノードの減少は，コンピュートノードのシャットダウン時や授業開始に合わせて離脱のリクエストを送信することで，減少とする．

3.2.2 対象インスタンスおよび遷移先ノードの選択方針

インスタンス再配置の契機条件が満たされたとき，再配置するインスタンスや遷移先のコンピュートノードを選定する方針について詳しく説明する．前提として，インスタンスの再配置には必ず，マイグレーションのコストがかかってしまう．マイグレーションのコストとは，インスタンスのチェックポイントを作成する時間，インスタンスの情報を他のコンピュートノードに転送する時間，インスタンスをレストアする時間という 3 つの時間的コストのことである．積極的にインスタンスの再配置を行うために，再配置するインスタンスの決定方針と遷移先の決定方針はこの時間的コストをなるべく最小化する必要がある．そこ

で，本稿では，以下の条件を考えた．

再配置するインスタンスの決定方針と遷移先のコンピュートノード決定方針は，コンピュートノード参加時と離脱時とで異なる．まずは，参加時の方針から述べる．参加したコンピュートノードがクラウド上で最も計算資源に余裕があると考えられるので，遷移先は参加したコンピュートノードとする．参加時に再配置するインスタンスは，参加したコンピュートノードからの通信速度と負荷指標の積が最も高いコンピュートノード上の，最も CPU 利用率の低いインスタンスとする．ここでいう負荷指標とは，直近 5 分間の load_averagen の値を CPU 数で割ったものとし，負荷指標は 0 1 の値となる．参加したコンピュートノードからの通信速度と負荷指標の積が最も高いコンピュートノードを選択する理由は，インスタンス情報転送の時間を減らしながら，コンピュートノードの負荷を下げるためである．また，最も CPU 利用率が低いインスタンスを選択する理由は，インスタンスのチェックポイント作成の時間を減らすためである．次に，離脱時の方針を述べる．離脱時は離脱するコンピュートノード上のインスタンスを他のコンピュートノードへ退避させる必要があるので，再配置するインスタンスは離脱するコンピュートノード上のインスタンスとする．遷移先のコンピュートノードは，1 と負荷指標の差をとり，その差とネットワーク速度の積が最も高いコンピュートノードとする．なぜならば，負荷指標が低く，離脱するコンピュートノードからネットワーク速度が速いコンピュートノードにするためである．

3.2.3 分身型ライブマイグレーション

上述したように，ライブマイグレーションには時間的コストがかかる．そこで．その時間的をコストを削減するライブマイグレーション方法を検討する．一般的にインスタンスのライブマイグレーションを行うと，遷移元のインスタンスは削除される．しかし，遷移元のインスタンスを削除せず，過去の状態を保ったまま残しておく方法を提案する．そのメリットについて詳しく説明する．遷移元のコンテナを削除せずに状態を保ったまま残しておくことで，他のコンテナへ遷移したコンテナが，元のコンピュートノードへ遷移する際に，その差分のみを元のコンピュートノードへ転送するだけでレストアが可能になる．本稿では，この仕組を分身型ライブマイグレーションと呼称する．また，差分のみを転送しインスタンスをレストアすることを差分レストアと呼ぶ

3.3 再配置の流れ

ここでは，インスタンスの再配置が行われる際の流れを述べる．インスタンス作成時は，再配置などは考慮せず，Zun のフィルタ機能によって最適なコンピュートノードが決定され，インスタンスが作成される．計算ノード追加時，3.2.2 で述べた条件にもとづいて，分身型ライブマイグレーションを行う．計算ノードの離脱時，3.2.2 で述べた条件にもとづいて，もし差分レストアが可能であれば差分レストアを行い，不可能であれば分身型ライブマイグレーションを行う．

4. 実装

3.2 項で述べた，インスタンス再配置機構の実装の第 1 段階として，コンテナライブマイグレーション機構 Optima の開発を行った．Optima は Zun が管理しているインスタンスの再配置を行うように実装されている．本実装での再配置とは，あるコンピュートノード上で稼働しているインスタンス (コンテナ) のチェックポイントを作成し，そのチェックポイントをもとに，他のコンピュートノードでコンテナのレストアを行うことである．Docker によって管理されているコンテナのチェックポイント&レストアは，CRIU[7] によって実現されているので，その機能を利用した．

4.1 Optima のコンポーネント

Optima は以下 4 つのコンポーネントによって構成されている．これらのコンポーネントはすべて Go 言語で実装されている．また，離脱時と参加時にコンピュートノードで実行するシェルスクリプトも作成した．

4.1.1 optima-conductor

コントローラノード上に配置され，各コンピュートノードの負荷状態を確認する．コンピュートノードの離脱・参加をイベントとして，コンテナの再配置を行う．

4.1.2 optima-monitor

コンピュートノード上に配置される．毎分コンピュートノードの load-average を取得して，CPU 数で割った負荷指標を計算する．計算した値を optima-conductor へ送信する．

4.1.3 optima-checkponter

コンピュートノード上に配置される．optima-conductor からコンテナのチェックポイント作成の指示を受け取ると，チェックポイントを作成し，チェックポイントをコントローラノードへ転送する．

4.1.4 optima-restorer

コンピュートノード上に配置される．optima-conductor からコンテナレストアの指示を受け取ると，チェックポイントをコントローラノードからダウンロードし，コンテナをレストアする．

4.1.5 leave_script.sh

離脱時にコンピュートノードで実行される．optima-conductor に対して，離脱のリクエストを送信する．

4.1.6 join_script.sh

参加時にコンピュートノードで実行される．optima-conductor に対して，参加のリクエストを送信する．もし，

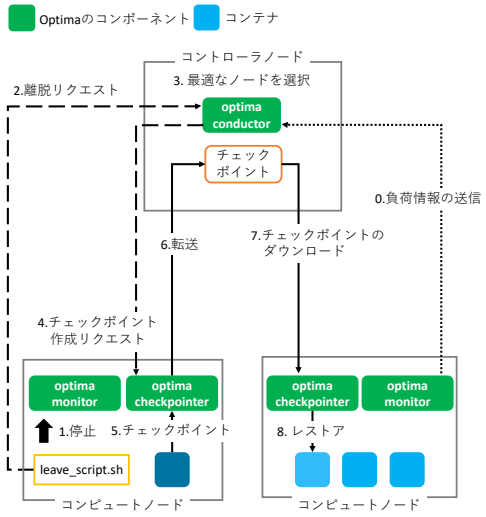

図 2　離脱時の処理の流れの概要

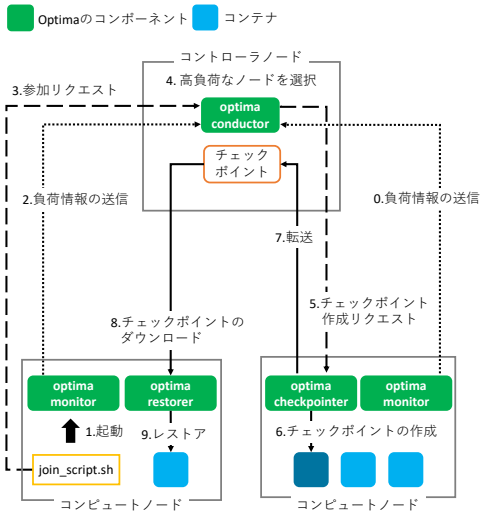

図 3　参加時の処理の流れの概要

初めての参加だった場合は，コントローラノードへの公開鍵の転送を行う．

4.2　離脱時のインスタンス再配置

コンピュートノードの離脱時は 2 に示すような手順でコンテナの再配置が行われる．詳しい手順を以下に述べる．

(1) 離脱するコンピュートノード上で，leave_script.sh が実行される
(2) optima-conducotr は，離脱するコンピュートノードから離脱リクエストをうけとると，そのコンピュートノードを計算資源から除外した後，そのコンピュートノードのインスタンスの情報をデータベースから取得する
(3) 1 つ以上インスタンスを情報が取得できた場合，それらのコンテナのチェックポイントの作成を，離脱するコンピュートノード上の optima-checkpointer にリクエストする
(4) optima-conductor からリクエストをうけた optima-checkpointer は，コンピュートノード上の全てのコンテナのチェックポイントを作成し，コントローラノード上に転送する
(5) optima-conductor は，レストア先のコンテナの作成を Zun にリクエストする
(6) Zun はフィルタ機能を利用して，最適なコンピュートノード上にコンテナを作成する
(7) optima-conductor は，レストア先のコンテナが作成されたことを確認すると，レストア先のコンテナが作成されたコンピュートノード上の optima-restorer にコンテナのレストアをリクエストする
(8) optima-conductor からリクエストをうけた optima-restorer は，チェックポイントをコントローラノードからダウンロードし，コンテナのレストアを行う

4.3　参加時のインスタンス再配置

コンピュートノードの参加時は 3 に示すような手順でコンテナの再配置が行われる，詳しい手順を以下に述べる．

(1) 参加するコンピュートノード上で，join_script.sh が実行される
(2) optima-conductor は，参加のリクエストを受け付けると，データベースから最も負荷指標の高いコンピュートノードの情報を取得する
(3) opitma-conductor は，最も負荷指標の高いコンピュートノード上からランダムに 1 つのコンテナを選択する
(4) optima-conductor は，選択したコンテナがホスティングされているコンピュートノード上の optima-checkpointer にコンテナのチェックポイント作成をリクエストする
(5) optima-conductor からリクエストをうけた optima-checkpointer は，コンピュートノード上の全てのコンテナのチェックポイントを作成し，コントローラノード上に転送する
(6) optima-conductor は，レストア先のコンテナの作成を Zun にリクエストする
(7) Zun はフィルタ機能を利用して，最適なコンピュートノード上にコンテナを作成する
(8) optima-conductor は，レストア先のコンテナが作成されたことを確認すると，レストア先のコンテナが作成されたコンピュートノード上の optima-restorer にコンテナのレストアをリクエストする
(9) optima-conductor からリクエストをうけた optima-restorer は，チェックポイントをコントローラノードからダウンロードし，コンテナのレストアを行う

表 1　実験で利用した PC のスペック

OS	macOS Sierra
メモリ	16GB
プロセッサ	2.3GHz Intel Core i5

表 2　実験で 2 つの VM のスペック

	コントローラノード	コンピュートノード
OS	CentOS7	
仮想プロセッサ数	2	
HDD	64GB	
メモリ	4GB	

表 3　各ステップ毎の平均所要時間

処理の種類	平均所要時間 (秒)
create	9.21
checkpoint	9.04
restore	12.23
delete	1.9
全体	32.39

5. 実験と評価

5.1 実験概要

第 4 章で述べたコンテナ再配置機構 Optima が，ライブマイグレーションにどれだけの時間がかかるのか計測するために，予備実験を行った．実験は，コントローラノードの VM とコンピュートノードの VM を 1 つずつ立ち上げ，以下の Optima を用いたライブマイグレーションを 100 回繰り返した．ライブマイグレーションの対象となるコンテナは 1 つである．ライブマイグレーション全体の所要時間と各ステップごとの所要時間を計測した．

(1) レストア先のコンテナ作成
(2) コンテナのチェックポイント作成とコントローラノードへのチェックポイント転送
(3) コンピュートノードへのチェックポイントのダウンロードとコンテナのレストア
(4) チェックポイントを作成したコンテナの削除

実験で利用した PC のスペックを表 1 に，各 VM の性能を表 2 に示す

5.2 実験結果

Optima を用いたコンテナライブマイグレーションを 100 回行った結果を表 3 に示す．ライブマイグレーション全体の平均所要時間は 32.39 秒となった．また，各ステップごとの所要時間はステップ順に，9.21 秒，9.04 秒，12.23 秒，1.9 秒となった．

5.3 評価・考察

コンテナ 1 つのライブマイグレーションの所要時間が 32.39 秒という結果から，コンテナのライブマイグレーションは時間的コストがかかるということが示された．

6. おわりに

6.1 まとめ

本稿では，教室 PC の余剰資源を活用した学内向けオープンクラウドの実現に向けて，考えられる技術的な課題とその解決手法について検討した．不安定な計算資源の上でもインスタンスの性能を最大限高める方法については，コンピュートノードの増減をイベントとし，クラウド上のインスタンスを最適なコンピュートノードへ再配置する機構を提案した．また，実装の第 1 段階として OpenStack 上でコンテナのライブマイグレーションを行う Optima の実装を行った．

Optima のライブマイグレーションにかかる時間を調査するための予備実験を行った．その結果，ライブマイグレーション全体の所要時間が 32.39 秒となり，コンテナのライブマイグレーションには時間的なコストがかかることが分かった．

6.2 今後の課題

今後の課題として，より高速なコンテナライブマイグレーションを実現するために，Optima にコンテナの差分を利用した差分レストアを追加実装していくことがあげられる．また，授業利用にあわせて計画的にライブマイグレーションを行う機構についても実装を行っていく．

参考文献

[1] 文部科学省. 平成 28 年度「学術情報基盤実態調査」について（概要）.
[2] OpenStack. Keystone, the openstack identity service.
[3] OpenStack. Welcome to neutron’s documentation!
[4] OpenStack. Image service overview.
[5] OpenStack. Object storage api overview.
[6] OpenStack. Filterscheduler.
[7] CRIU. Docker.