Vol.2020-HPC-177 No.5
2020/12/21

クラウドオブジェクトストレージを活用したメタスケジューラ

滝澤 真一朗[1,a)]　高野 了成[1]　清水 正明[2]　松葉 浩也[2]　中田 秀基[1]　小川 宏高[1]

概要：計算能力の増強を理由にオンプレミスシステムに加えてクラウドを利用することや，処理対象とするデータの機密性に応じてオンプレミスシステムとクラウドを使い分ける利用形態がある．この時，複数計算機資源間での実行環境の共通化と，計算内容とデータのセキュアな共有，これら 2 点を実現するシステムの管理コスト最小化が課題となる．この課題を解決することを目的に，計算実行環境にコンテナイメージを活用し，クラウドオブジェクトストレージに計算内容とデータを格納するメタスケジューリングシステムを提案する．システムを統一認証基盤を必要としない，利用者権限で実行できる設計とすることで管理コストの課題も解決する．本システムのプロトタイプを実装し，クラウドに ABCI を，クラウドオブジェクトストレージとして ABCI クラウドストレージを用いる試験環境を構築し，利用者端末での同時ジョブ投入時の応答性能，連続ジョブ投入時のクラウドでの受付までの待ち時間の評価を行った．

1. はじめに

画像分類処理の大幅な精度向上を皮切りに，機械学習，特に深層学習技術は産業界からも注目され，大規模分散学習の研究への投資や [1, 2]，サイバーフィジカルシステム実現のための要素技術として深層学習の活用が始まっている [3]．深層学習の計算はデータからモデルを構築する学習フェーズと，構築したモデルに新規データを当てはめて結果を導出する推論フェーズからなる．学習フェーズにおける特徴として，行列演算を多数行うことから，GPU が広く利用されていることが挙げられる．しかしながら，GPU を用いることで CPU のみの場合よりも計算時間は短縮されるものの，ネットワークサイズや入力データ量に応じて長い計算時間を要すこと，高精度達成にはハイパーパラメータのチューニングが必要であり，ハイパーパラメータを変えた多数の実行が必要である．そのため，1 企業や研究室がオンプレミスで所有できる GPU クラスタでは，規模の問題から逐次に計算せざるを得ず，成果創出に時間を要すことがあるため，Amazon Web Services（AWS）や産総研の AI 橋渡しクラウド (AI Bridging Cloud Infrastructure, ABCI) [4–6] 等の産業・学術クラウドの併用が重要になってきている．

ここでの課題として，(1) オンプレミスシステムとクラウドの研究開発環境の共通化と，(2) クラウドへの計算内容とデータのセキュアな転送，がある．(1) に関しては，使い勝手の異なる複数システムを切り替えて使うことは利用者にとっての負担になるため，既存システムで利用していたソフトウェア開発・実行環境を可能な限り他システムでも利用可能とすることが重要である．(2) に関しては，企業の研究開発で使用するデータ等，組織の定める秘匿性の高いデータをクラウドで使用する際には，組織の定めるセキュリティ要件を満たす転送・蓄積手段を採用しなければならない．複数のクラスタ，あるいはクラスタとクラウドを連携させる既存技術としてメタスケジューラやクラウドバースティングが提案されており，それらは上記課題を解決する可能性を有している．しかしながら，既存技術では統一認証基盤の配備や資源のコアロケーション，データ転送の効率化などの優れた機能を有する反面，(3) 導入・管理が複雑となる課題を抱えている．

本研究では，上記 3 課題を解決する軽量なメタジョブスケジューリングシステムを提案する．深層学習などの長時間かつ複数回の実行を必要とする計算実行を目的とし，共にバッチジョブ管理されているオンプレミスの GPU クラスタと，GPU 計算資源を提供するクラウドを対象とした，一般利用者権限で動作するシステムである．オンプレミスとクラウドの研究開発環境の共通化を促進するために，デファクトスタンダードとして普及しているフレームワークや，開発ツールをパッケージ化したコンテナイメージを積極的に活用する．また，多様な組織のデータ管理セキュリティ要件を満たすために，多様なセキュリティオプション

[1] 国立研究開発法人産業技術総合研究所
[2] 株式会社日立製作所
[a)] shinichiro.takizawa@aist.go.jp

を提供するクラウドオブジェクトストレージを介したジョブデータ転送を行う．提案システムのプロトタイプを実装し，クラウドオブジェクトストレージに ABCI クラウドストレージ [7] を，クラウドに ABCI を使用する試験環境を配備した．評価の結果，同時投入ジョブ数 16 まではクライアントの応答性能に低下なく処理できることを確認した．また，本システムに対して連続してジョブ投入した場合，70%のジョブは ABCI クラウドストレージ監視間隔の 2 倍の時間以内に ABCI に投入されることを確認した．

2. 要件定義

本研究では，オンプレミスの GPU クラスタと GPU 計算資源を提供するクラウドを連携させて，環境の違いを意識せずに機械学習計算を実行できるメタスケジューリングシステムの実現を目指す．以下にその設計要件を述べる．

機械学習の豊富なエコシステムを使用　深層学習の開発環境は，TensorFlow や PyTorch などのフレームワークや，それらと関連ツールをパッケージ化した Docker コンテナによるエコシステムの普及により，開発環境の容易な再現や再利用が進みつつある．この特徴を活かすことにより，オンプレミスとクラウドの開発環境の共通化の実現を目指す．

計算実行環境の差異を吸収　AWS が提供する IaaS 型サービスのように，利用者がクラウド上に任意の環境を構成できる場合はオンプレミスと同じ構成の環境をクラウドに構築できるが，ABCI や一般的なスーパーコンピュータでは利用者が行える操作には制限があり，不可能である．利用者にとって，使い勝手の異なる複数の計算機を使い分けることは負荷が大きいため，透過的に使い分けることができるインターフェースが求められる．

計算・データ転送のセキュリティ保証　オンプレミスの計算・データをクラウドで実行するためには，それらをクラウドに転送し保存しなければならない．そのためには事前に，組織の定めるデータ管理セキュリティ要件を考慮し，クラウドに転送可能な計算・データとそうでないものの区別をつける必要がある．転送可能な計算・データは，クラウド上での転送・蓄積・利用の 3 点について，セキュリティ要件を満たしつつクラウドで管理する必要がある．

容易な導入・管理　目標とするシステムを実現するには，オンプレミスとクラウド計算機の管理だけでなく，それらを連携させるメタスケジューリングシステムの管理も必要になるため，クラウド側およびメタスケジューリングシステムの導入・管理は容易であることが好ましい．また，IaaS 型以外のクラウドサービスを利用する場合にもシステムを導入できるように，クラウド側へのシステム導入には特権利用者権限を必要としないことが好ましい．

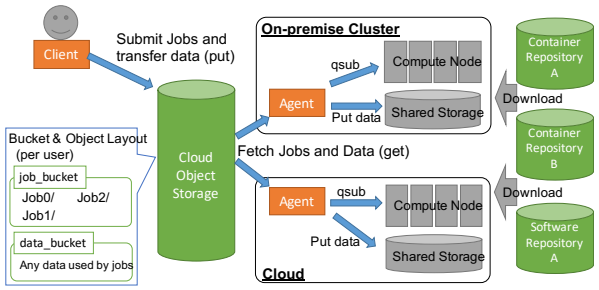

図 1: 提案メタスケジューリングシステムのアーキテクチャ

3. 設計

3.1 設計の概要

2 章で挙げた要件を満たすメタスケジューリングシステムの設計を行なった．オンプレミスもクラウドも，バッチジョブスケジューラによりジョブ管理されていることを前提とする．理由は以下の通りである．

- 計算の目的はサイバーフィジカルシステム実現のための，深層学習計算とシミュレーションである．どちらも長時間・複数回実行を要するため，バッチジョブとしての実行に適合する．
- クラウドサービスとして，AWS Batch [8] などバッチジョブ実行を提供するものもある．また，IaaS 型クラウドでも AWS ParallelCluster [9] などを用いることで，バッチジョブ実行環境を整備できる．
- クラウドとしてスーパーコンピュータを使用する場合，バッチジョブ実行での利用形態に限定される．

以降，オンプレミス，クラウドの計算資源を総称して「クラスタ」と呼ぶ．各クラスタ内のジョブスケジューラを「ローカルスケジューラ」と呼ぶ．Linux コンテナのエコシステムを活用するため，各クラスタでは Docker や Singularity などのコンテナランタイムが利用可能であることとする．

設計概要を図 1 に示す．提案システムでは，外部のコンテナ・ソフトウェアリポジトリにあるコンテナイメージやアプリケーションを除き，ジョブ情報をクラウドオブジェクトストレージに保存する．ジョブ情報は，計算内容を定義するジョブスクリプト，ジョブの実行状態，入出力データから構成されるものとする．クラウドオブジェクトストレージには，Amazon S3 互換のプロトコルを有するオブジェクトストレージを想定する．利用者はクラウドオブジェクトストレージ上に，ジョブ情報を格納するバケットを作成し，所属組織が定めるジョブ管理のセキュリティ要件を満たすよう，そのバケットに適切なセキュリティ設定を施す．

クラスタごとに，以下を行うエージェントを配備する．

- クラウドオブジェクトストレージからのジョブ情報取得

- ローカルスケジューラへのジョブ投入，実行監視
- 出力結果のクラウドオブジェクトストレージへの転送

エージェントは，各クラスタの一般利用者権限で，利用者ごとに動作するプロセスである．エージェントはクラスタのローカルスケジューラとファイルシステムを利用できるサーバで動作する．

利用者が投入したジョブは 1 クラスタ上で実行され，複数クラスタを同時に使用するジョブの実行はサポートしない．次の 2 種類のジョブスクリプトを記述できる．

ローカルジョブスクリプト　特定のクラスタ固有の機能を使用でき，そこでのみ実行できるジョブスクリプト．ローカルスケジューラが解釈できるジョブスクリプトをそのまま使用できる．

メタジョブスクリプト　任意のクラスタで実行できるジョブスクリプト．全クラスタが提供する機能のスーパーセットの機能の利用に限定されるが，実行先を指定せずとも任意のクラスタで実行される．

3.2　認証情報管理

クラウドオブジェクトストレージと複数のクラスタを使用するため，それらを使用するための $1 + N$ 個の認証情報の管理が必要である．本システムでは，システム導入管理の簡易化を目的に，シングルユーザ利用かつ統一認証基盤を導入しない設計とした．

一方で，利用者自らが複数の認証情報を管理する必要がある．個々の資源を独立に使う場合は認証情報を切り替えて使わなければならないが，本システムとして利用する場合，認証情報を管理する主体，各主体で管理すべき認証情報は以下の通りであり，少ない．

クライアント端末　クラウドオブジェクトストレージの認証情報のみを管理する．ジョブ実行管理のためのデータ転送に使用する．

エージェント　対象とする 1 クラスタとクラウドオブジェクトストレージの認証情報を管理する．オブジェクトストレージとのジョブ情報の通信と，クラスタのローカルスケジューラへのジョブ投入とファイルシステムアクセスのために使用する．

使用するクラスタを増やした場合も，そのクラスタを使用するエージェントでの設定のみを行えば良い．

3.3　ジョブ情報の一貫性管理

Amazon S3 では，以下に示すデータ一貫性モデル [10] を有すること，ロック機構を備えていないこと，といった特徴がある．

- 新規オブジェクトの PUTS に対して read-after-write consistency
- オブジェクトの上書き PUTS と DELETES に対して eventual consistency

表 1: ジョブの実行状態

状態	説明
INIT	本システムへのジョブ投入直後の状態
READY	クラスタへのジョブ投入直後の状態
RUN	実行中の状態
DONE	実行完了，または，キャンセル完了の状態
DELETING	本システムへジョブキャンセル要求された状態
ERROR	エラーとなった状態

- オブジェクトの操作はアトミックである

これにより，複数クライアントが同一バケット・オブジェクトを参照する際には，タイミングによって古い情報が取得されることがある．本研究で使用するクラウドオブジェクトストレージでも同様な制約があるものとする．このとき，クライアント端末とエージェントから見えるジョブ情報が異なる可能性がある．

ジョブ情報のクラウドオブジェクトストレージへの I/O パターンを以下に示す．

- ジョブスクリプトと入力データは，クライアント端末がジョブ投入時に 1 度だけ書き込む．
- 出力データは，エージェントがジョブ終了時に 1 度だけ書き込む．
- 実行状態は，利用者のアクションによる遷移と，クラスタ上での実行状況変化による遷移により変化する．クライアント端末とエージェントの双方が実行状態を監視し，遷移後の状態の書き込みを行う．

ジョブ情報を破壊しないためには，この I/O パターンをクラウドオブジェクトストレージの一貫性モデルに当てはめて実現する必要がある．

本研究では，ジョブ情報の Staleness（最新でないこと）を許可する，ジョブ情報の一貫性管理を提案する．以下に，ジョブ情報を構成する各要素における対応を説明する．

3.3.1　ジョブスクリプトと入出力データ

これらは Write-Once の情報であるため，繰り返し参照を行えば，後続処理が遅延される以外の影響はない．

3.3.2　実行状態

ジョブは表 1 に示す 6 状態を取るものとし，その状態遷移は図 2 の通りとする．キャンセル要求（`DELETING` への遷移）を除き，ジョブ投入後の実行状態変化はエージェントによるものであり，クライアント端末では古い情報が参照される可能性はあるが，実行状態の一貫性は保証される．次の 2 点の課題がある．1 点目は，キャンセル要求の場合に，クライアント端末とエージェントの書き込み競合が発生し，一方の意図しない状態が記録される可能性があること．2 点目は，エージェントにおいてもクラウドオブジェクトストレージから古い実行状態を取得する可能性があること．これら問題を回避するために，次の規則を設けた．

- クライアント端末による状態遷移を，エージェントに

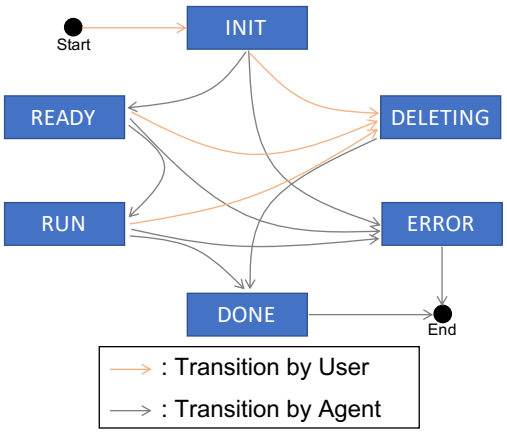

図 2: ジョブの状態遷移図

よる状態遷移よりも優先度の高いものとする．

- エージェントは，クライアント端末により遷移された状態を除き，ローカルスケジューラから取得できるジョブの実行状態で，クラウドオブジェクトストレージ上の実行状態を上書きする．

1 点目により，エージェントは DELETING への遷移を観測した場合，必ずキャンセル処理を行える．書き込みが競合し，クライアント端末からの DELETING への遷移が上書きされた場合は，再度クライアント端末から要求を行うものとする．2 点目により，エージェントが古い実行状態を取得してしまった場合も，最新の状態を反映することができる．以上の通り，古い情報が参照できてしまう可能性があること，クライアント端末からの繰り返し要求を行う必要性があることを許容することの制約の下，状態の一貫性を担保する．

3.4 メタジョブスクリプト

メタジョブスクリプトは，利用者がクラスタ実行環境の差異を気にすることなくジョブスクリプトを記述できることを目的に導入された．ここでの実行環境の差異は，(1) ローカルスケジューラの種類，(2) ジョブスクリプト中で記述できるクラスタが提供するコマンドの種類，である．(1) については，ローカルスケジューラごとに異なるジョブ管理コマンドの利用方法を共通化する．(2) については，全てを考慮すると果てがないため，Linux コンテナ管理とクラウドオブジェクトストレージ利用のコマンドに限定し，利用方法を共通化する．

エージェントがローカルスケジューラのラッパーとして働くことで (1) を解決する．また，メタジョブスクリプトにおける記述では，(1) のための専用指示文，(2) のための専用コマンドを用意する．エージェントにてジョブ投入前に，メタジョブスクリプトからローカルスケジューラ向けのジョブスクリプトへの変換が行われる．図 3 にメタジョブスクリプトの例を示す．図中，接頭語 cloudq_のコマンドが専用コマンドである．クラウドオブジェクトストレージからのデータ転送，コンテナ実行を行っている．これを ABCI 向けに変換した例が図 4 である．ABCI で AWS CLI と Singularity を利用する環境設定がされ，専用コマンドが ABCI 上で用意されたコマンドに置換されている．

```
#$ run_on:          ANY
#$ project:         project01
#$ resource:        type1
#$ n_resource:      1
#$ walltime:        1:00:00
#$ shell:           bash
#$ container_image: img0=\
   docker://nvcr.io/nvidia/tensorflow:19.07-py3

wget https://script.is/here train.py
cloudq_cs_cp s3://myobjs/data ./data
cloudq_container_run $IMG0 \
                     python ./train.py data
```

図 3: メタジョブスクリプト例

```
#!/bin/bash
#$ -l rt_F=1
#$ -l h_rt=1:00:00
#$ -cwd

source /etc/profile
source /etc/profile.d/modules.sh
module load aws-cli/2.0 singularitypro/3.5

IMG0=_IMG0
singularity pull $IMG0 \
    docker://nvcr.io/nvidia/tensorflow:19.07-py3

wget https://script.is/here train.py
aws --endpoint-url https://s3.abci.ai \
    s3 cp --quiet s3://myobjs/data ./data
singularity exec --nv $IMG0 python train.py

rm $IMG0
```

図 4: メタジョブスクリプトから ABCI 向けに生成されたジョブスクリプト例

3.5 要件への対応

2 章で挙げた要件への対応を以下に示す．

機械学習の豊富なエコシステムを使用 コンテナイメージとして配布されている機械学習開発環境を利用することができる．

計算実行環境の差異を吸収 コンテナイメージによる実行環境の共通化を行う．メタジョブスクリプトを導入することで，ジョブスケジューラ利用方法を共通化する．

計算・データ転送のセキュリティ保証 In transit, At rest の暗号化をサポートするクラウドオブジェクトストレージを経由したジョブ情報の転送を行うことで，暗

号化に対するセキュリティ要件の大部分は解決できると見込める．表 2 に Amazon S3 と ABCI クラウドストレージのセキュリティ機能の一例を示す．クラウドオブジェクトストレージ毎に提供する機能は異なりうるが，利用者はこれらから組織のセキュリティ要件を満たすために必要な機能を選択して適用することができる．一方で，クラウド上で計算実行する際にはクラウドオブジェクトストレージからクラウド上のストレージにデータを転送する必要がある．当該クラウドにおけるデータ管理方針が組織のセキュリティ要件を満たしているかどうかの確認は必要である．

容易な導入・管理　統一認証基盤を有していないこと，一般利用者権限で導入できることより，導入管理コストは低いと考える．

4. 実装

提案システム CloudQ と名付け，Python 3.6 を用いて実装した．現実装ではクラスタとしては ABCI のみを対象とする．クラウドオブジェクトストレージとしては ABCI クラウドストレージを用い，ABCI クラウドストレージが提供する機能を用いて実装した [*1]．メタジョブスクリプトとメタスケジューリング機能は現在開発中である．

4.1 ジョブ情報バケットの構成

クラウドオブジェクトストレージ上にジョブ情報を格納するジョブ情報バケットは，利用者ごとにバケットとして作成する．ジョブ情報を構成する，ジョブスクリプト，入出力データ，実行状態はそれぞれ個別のオブジェクトとして保存する．ジョブごとの区別をつけるため，これらには，ジョブ ID をプレフィックスに持つキーを付与する．

4.2 クライアントの実装

ジョブ管理のための以下の機能を実装した．

- ジョブ投入（ジョブ状態を `INIT` に設定）
- ジョブ状態確認
- ジョブキャンセル（ジョブ状態を `DELETING` に設定）
- 実行中ジョブの標準出力の取得
- 実行中・実行完了ジョブ一覧の取得
- 実行完了ジョブの出力のダウンロード

4.3 エージェントの実装

エージェントは，利用者権限で利用者ごとに ABCI のログインノード上で常駐するプロセスとして起動する．しかしながら，この設計には利用上の課題があり，6 章で議論する．ABCI を対象としたエージェントは，認証情報として，ABCI アカウント，ABCI グループ，ABCI クラウドストレージ利用アクセスキーを管理する必要がある．ABCI アカウントは ABCI 利用者に一意に割り当てられるアカウントであり，利用者権限でエージェントを実行することより，管理する ABCI アカウントは一意に定まる．ABCI グループは，ABCI へのジョブ投入時に計算機利用料を請求する対象として使用されるものである．本来ならば計算目的に応じて請求先 ABCI グループを切り替えるべきであるが，現実装では設定ファイルで 1 つ指定する．ABCI クラウドストレージ利用アクセスキーも同様に，設定ファイルで 1 つ指定する．

エージェントは定期的にジョブ情報バケットを監視し，実行状態に応じて以下の処理を行う．

INIT　ABCI にジョブを投入し，実行状態を `READY` に遷移

READY　ABCI 上のジョブが実行中の場合，実行状態を `RUN` に遷移

RUN　ABCI 上のジョブが完了している場合，実行状態を `DONE` に遷移

DELETING　上記 3 状態のジョブをキャンセルし，実行状態を `DONE` に推移

ジョブ実行中にエラーが生じた場合は，実行状態を `ERROR` に設定する．処理はジョブ毎に独立しているため，並列化して行われる．

5. 評価

エージェントを ABCI ログインノード 1 台に配備し，本システムの予備評価を行なった．

5.1 同時多数ジョブ投入時の応答性能

同時に多数のジョブ投入を行ったときの，クライアント端末における，ジョブ単位の応答性能を評価する．CloudQ ではジョブ状態管理に ABCI クラウドストレージを使用しているため，同時ジョブ投入時の応答性能は，ABCI クラウドストレージの並列処理性能に依存する．そこで，ダミージョブ（111 バイト）を同時投入数を変えて複数投入した時の，全ジョブの応答完了までの時間を計測し，1 ジョブの平均値を評価する．本評価では，クライアントには ABCI ログインノードを用いている．

結果を図 5 に示す．横軸は同時投入ジョブ数であり，縦軸はジョブ単位の応答時間を，同時投入ジョブ数が 1 の場合に対する相対値で示した値である．同時投入ジョブ数が 16 までは，ジョブ数に反比例して相対応答時間が減っているため，性能低下なく処理が行えている．それ以降，32，64，128 と増加させた場合は相対応答時間が増加していることから，同時投入ジョブ数 16 が性能限界と言える．ただ，同時投入ジョブ数が 1 の場合の応答時間が 0.59 秒程度であり，一方，ABCI のローカルスケジューラへの 1 ジョブ投入の応答時間が 0.3 秒程度であることから，クライア

[*1] ABCI クラウドストレージは Amazon S3 API のサブセットをサポートする．サポートする機能の範囲内で実装を行ったため，Amazon S3 を対象とする場合は異なる実装が可能である．

表 2: クラウドオブジェクトストレージのセキュリティ機能

機能	AWS S3	ABCI クラウドストレージ
暗号化転送	HTTPS (TLS) による暗号化をサポート	同左
サーバサイド暗号化	AES-256 による暗号化をサポート	同左
暗号鍵管理方式	S3 および利用者管理の鍵を使用可能	ABCI が管理する鍵を使用 (SSE-S3 相当)
利用者毎の操作の制限	IAM ポリシーにて設定可能	同左
バケットへのアクセス元 IP 制御	IAM ポリシー/バケットポリシーにて設定可能	IAM ポリシーにて設定可能
バケットへのアクセス許可アカウント制御	IAM ポリシー/バケットポリシーにて設定可能	IAM ポリシーにて設定可能
バケットへのアクセスログ	利用者権限でログの On/Off 設定可能	運用側で常時記録

図 5: 同時ジョブ投入時の応答性能

ントと ABCI クラウドストレージがネットワーク的に近傍にある場合は，ローカルスケジューラへのジョブ投入と遜色ない性能であると考えている．

5.2 処理待ち遅延

CloudQ によるジョブ投入から実行開始までには，2 種類の待ち時間が発生する．

(1) CloudQ にジョブ投入後, エージェントが受信し, ABCI にジョブ投入するまでの時間
(2) ABCI にジョブ投入後，計算ノードが割り当てられて実行開始するまでの時間

CloudQ のみに依存する（1）について評価を行う．先の評価と同じジョブを 1 秒間隔で 100 個 ABCI に投入し，ジョブごとの (1) を計測した．

クライアントに，ABCI ログインノードを用いた場合の結果を図 6 に示す．エージェントにおける ABCI クラウドストレージ上のジョブ情報バケットの監視間隔と，並列処理するジョブ数を変化させた場合の，100 ジョブの平均処理待ち遅延を計測した．監視間隔が小さいほど，並列数が大きいほど遅延が小さくなることがわかる．しかしながら，並列数を 8 から 16 に増やした場合の効果は少ない．設定値により大きく遅延が変化するため，利用者はジョブ特製（数や即時応答性など）を考慮して設定する必要がある．

次に，監視間隔を 5 秒，並列数を 8 に固定してエージェントを動作させ，ABCI ログインノードと日立製作所内のサーバからジョブを投入した．それぞれ 100 ジョブずつ投入し，遅延の分布を，階級幅を 5 秒とした度数分布で示した表を表 3 に示す．ABCI 外部からジョブ転送する，日立

表 3: 監視間隔 5 秒、並列数 8 の時の処理待ち遅延の分布

遅延	クライアント@ABCI	クライアント@日立製作所
0-5	33	32
5-10	44	40
10-15	19	23
15-20	4	5

製作所内のサーバを用いた場合の方が遅延の大きいジョブが多いが，いずれの場合も 70%強のジョブは監視間隔の 2 倍以下の遅延に抑えられている．遅延のばらつきの原因は，(1) 利用者によるジョブ投入とエージェントによるジョブ情報バケットの監視タイミングの差と，(2) エージェントでの逐次処理による待ちの発生，にある．(1) について，投入タイミングによっては 5 秒の待ち時間は避けられない．(2) について，現在の実装では，エージェントにてジョブ情報バケット中の全ジョブの情報を毎回確認している．8 並列処理を行っているため，最大 13 ($\lceil 100 \div 8 \rceil$) ジョブは逐次に ABCI クラウドストレージへのジョブ情報問い合わせを行っているためである．エージェント側で完了したジョブを記録することで，不要な問い合わせを削減する予定である．

6. 考察

ジョブ情報をクラウドオブジェクトストレージに集約する設計には利点と欠点がある．利点としては，クラウドオブジェクトストレージが提供する高い耐久性，可用性，セキュリティを，管理の手間少なく利用できることである．欠点としては，3.3 章で述べたデータ管理の特性による，処理要求の紛失や処理遅延の発生がある．しかしながら，処理要求の紛失が起こり得るのは，ジョブ投入後の利用者による状態変更のみであること，処理遅延も実行状態確認の間隔次第であり，秒単位に抑えることは可能である．本研究で対象とするバッチジョブ管理においては，これら制約は許容されうると考えている．

ABCI のような共用システムにて，エージェントをログインノード上で常駐させ続けることは容易ではない．技術的には screen 等の仮想端末上で起動すれば良いが，ログインノードのような複数利用者が同時に使用するシステムでは，プロセスの生存期間が制限されている場合がある．

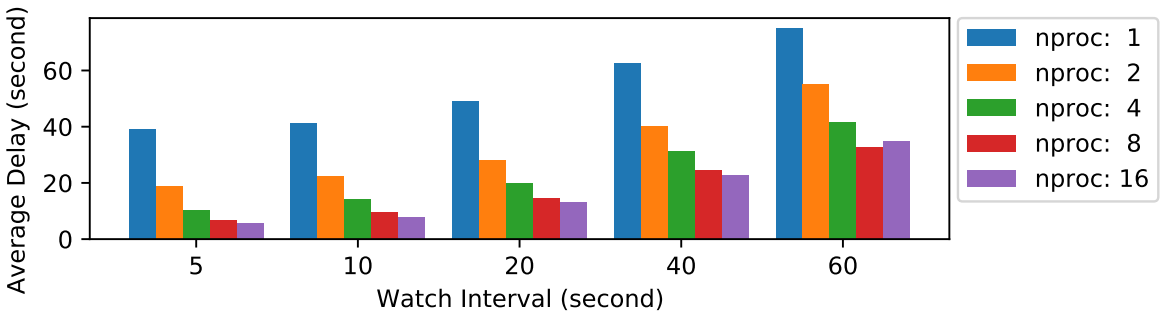

図 6: 平均処理待ち遅延

CRON で実行できるエージェント，プロセス生存期限の制限のないホストから SSH 経由でリモート接続するエージェント等，他手段のエージェント実装を検討する．

開発環境の共通化手段として，今回はコンテナイメージによる共通化を行なったが，他の抽象度による共通化手段もある．1 手段としてソフトウェアパッケージレベルでの共通化手段がある．例えば，Python ではライブラリリポジトリとそこからのライブラリ取得手段が提供されており，異なる環境への同一ライブラリ環境を容易に構築できる．Spack [11] は高性能計算分野のソフトウェアパッケージ管理システムであり，異なるクラスタへの同一ソフトウェア環境の配備を容易にする．しかしながらこれらは OS やシステムソフトウェアの差異は吸収できない．別手段として，コンテナイメージの定義レベルでの共通化手段もある．HPC Container Maker [12] は Python スクリプトで定義したレシピを基に，Docker，および Singularity コンテナイメージを生成できる．必要とする共通化のレベルに応じて，利用者はこれら手段を使い分けることができる．

7. 関連研究

複数の計算機クラスタをまたぐジョブ管理の仕組みは古くから研究されている．2000 年代のグリッドコンピューティング技術の研究開発が盛んだった頃は，管理主体の異なる複数組織が管理するクラスタを連携させた計算実行やデータ共有を実現するための，認証技術や資源管理システムの研究が行われていた．それらを基盤技術として用いて，GridWay [13]，CSF [14]，Condor [15]，gLite WMS [16]，NAREGI Super Scheduler [17]，GridARS [18] などの複数クラスタへのメタスケジューリングを行うシステムが開発された．これらは計算資源だけでなく，ネットワーク資源のコアロケーションなど，様々な機能を提供するが，反面，共通認証基盤や各クラスタへの専用サービスの導入が必要であり，管理の手間が大きい．より軽量に複数のクラスタを連携させて利用するための技術として，クラスタのインターフェースを共通化する研究もある．NEWT [19] はクラスタの機能を RESTful API で提供するシステムであり，認証やジョブ実行管理等の共通 API を提供する．Xcrypt [20] は，複数のジョブスケジューラを対象とするジョブ実行管理を行うスクリプトを記述する DSL を提供する．これらはジョブスケジューラの差異を吸収するが，各クラスタの実行環境の差異を吸収するものではない．本研究の成果物の利用拡大を目的に，これらをエージェント内部で利用することを検討している．

オンプレミスとクラウドを連携して使用するハイブリッドクラウドや，オンプレミスの資源があふれた場合に計算をクラウドにオフロードするクラウドバースティングが提案されている [21]．Guo らは，クラウド利用コストを最小化するプログラム判別し，オンプレミスからクラウドに移行する Seagull を提案している [22]．仮想マシン単位でのクラウドへの移行を行なっており，仮想マシンの負荷や使用頻度に応じた，クラウドへの仮想マシンの事前コピーを行うことで，移行時間の短縮を図っている．Bicer らは，オンプレミスとクラウドのハイブリッド環境上で，双方の資源を用いた MapReduce モデルによるデータインテンシブアプリケーションの実行環境を提案している [23]．本研究もクラウドバースティングの 1 つと言えるが，データ転送にクラウドオブジェクトストレージを介することで効率よりもセキュリティを優先すること，バッチジョブ実行をサポートするクラウドであれば管理権限不要であることが他と異なる．

8. まとめ

オンプレミスのシステムやクラウドなど複数の計算機システム上に，利用者が透過的に使えるサイバーフィジカルシステムの研究開発環境を構築することを目的に，ジョブ情報をクラウドオブジェクトストレージに管理するメタスケジューリングシステム CloudQ を提案した．プロトタイプを ABCI に配備し，同時ジョブ投入時のクライアントの応答性能，連続ジョブ上投入時の ABCI での受付性能の評価を行った．

今後の課題として，まずはメタジョブスクリプトを実現する．その後，日立製作所社内クラスタに対応するエー

ジェント等，複数のエージェントを開発し，それらへのメタスケジューリング機能を実現する．利便性の観点から，アレイジョブや依存関係のあるジョブなどのワークフロー実行支援機能を実現し，サイバーフィジカルシステムの研究開発者と協力し，実アプリケーションを用いた評価を行うことを計画している．

参考文献

[1] Mikami, H., Suganuma, H., U-chupala, P., Tanaka, Y. and Kageyama, Y.: Massively Distributed SGD: ImageNet/ResNet-50 Training in a Flash (2018).
[2] Yamazaki, M., Kasagi, A., Tabuchi, A., Honda, T., Miwa, M., Fukumoto, N., Tabaru, T., Ike, A. and Nakashima, K.: Yet Another Accelerated SGD: ResNet-50 Training on ImageNet in 74.7 seconds (2019).
[3] Liang, Y. C., Lu, X., Li, W. D. and Wang, S.: Cyber Physical System and Big Data enabled energy efficient machining optimisation, *Journal of Cleaner Production*, Vol. 187, pp. 46–62 (2018).
[4] 小川宏高，松岡　聡，佐藤　仁，高野了成，滝澤真一朗，谷村勇輔，三浦信一，関口智嗣：世界最大規模のオープン AI インフラストラクチャ AI 橋渡しクラウド（ABCI）の概要，第 165 回ハイパフォーマンスコンピューティング研究会 (2018).
[5] 佐藤　仁，溝手　竜，滝澤真一朗：AI 橋渡しクラウド ABCI の性能評価，第 166 回ハイパフォーマンスコンピューティング研究会 (2018).
[6] 滝澤真一朗，坂部昌久，谷村勇輔，小川宏高：ABCI 上でのジョブ実行履歴の分析による深層学習計算の傾向把握，第 176 回ハイパフォーマンスコンピューティング研究会 (2020).
[7] 谷村勇輔，滝澤真一朗，小川宏高，浜西貴宏：ABCI クラウドストレージサービスの構築と評価，第 172 回ハイパフォーマンスコンピューティング研究会 (2019).
[8] : AWS Batch, `https://aws.amazon.com/batch/`.
[9] : AWS ParallelCluster, `https://github.com/aws/aws-parallelcluster/`.
[10] : Amazon S3 data consistency model, `https://docs.aws.amazon.com/AmazonS3/latest/dev/Introduction.html#ConsistencyModel`.
[11] Gamblin, T., LeGendre, M., Collette, M. R., Lee, G. L., Moody, A., de Supinski, B. R. and Futral, S.: The Spack package manager: bringing order to HPC software chaos, *SC '15: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (2015).
[12] : HPC Container Maker, `https://github.com/NVIDIA/hpc-container-maker/`.
[13] : GridWay, `http://www.gridway.org/`.
[14] Xiaohui, W., Zhaohui, D., Shutao, Y., Chang, H. and Huizhen, L.: CSF4: A WSRF Compliant Meta-Scheduler, *In Proc. of World Congress in Computer Science Computer Engineering, and Applied Computing*, pp. 61–67 (2006).
[15] Frey, J., Tannenbaum, T., Foster, I., Livny, M. and Tuecke, S.: Condor-G: A Computation Management Agent for Multi-Institutional Grids, *Proceedings of the Tenth IEEE Symposium on High Performance Distributed Computing (HPDC)* (2001).
[16] Andreetto, P., Andreozzi, S., Avellino, G., Beco, S., Cavallini, A., Cecchi, M., Ciaschini, V., Dorise, A., Giacomini, F., Gianelle, A., Grandinetti, U., Guarise, A., Krop, A., Lops, R., Maraschini, A., Martelli, V., Marzolla, M., Mezzadri, M., Molinari, E., Monforte, S., Pacini, F., Pappalardo, M., Parrini, A., Patania, G., Petronzio, L., Piro, R., Porciani, M., Prelz, F., Rebatto, D., Ronchieri, E., Sgaravatto, M., Venturi, V. and Zangrando, L.: The gLite workload management system, *Journal of Physics: Conference Series*, Vol. 119, No. 6, p. 062007 (2008).
[17] Matsuoka, S., Hatanaka, M., Nakano, Y., Iguchi, Y., Ohno, T., Saga, K. and Nakada, H.: Design and Implementation of NAREGI SuperScheduler Based on the OGSA Architecture, *Journal of Computer Science and Technology*, Vol. 21, No. 4, pp. 521–528 (2006).
[18] Takefusa, A., Nakada, H., Kudoh, T., Tanaka, Y. and Sekiguchi, S.: GridARS: An Advance Reservation-Based Grid Co-allocation Framework for Distributed Computing and Network Resources, *Job Scheduling Strategies for Parallel Processing* (Frachtenberg, E. and Schwiegelshohn, U., eds.), pp. 152–168 (2007).
[19] Cholia, S., Skinner, D. and Boverhof, J.: NEWT: A RESTful service for building High Performance Computing web applications, pp. 1–11 (2010).
[20] Ueno, M., Hiraishi, T., Hibino, M., Iwashita, T. and Nakashima, H.: Multilingualization Based on RPC for Job-level Parallel Script Language, Xcrypt, 情報処理学会論文誌プログラミング（PRO），Vol. 6, No. 2, pp. 54–68 (2013).
[21] Nair, S. K., Porwal, S., Dimitrakos, T., Ferrer, A. J., Tordsson, J., Sharif, T., Sheridan, C., Rajarajan, M. and Khan, A. U.: Towards Secure Cloud Bursting, Brokerage and Aggregation, *2010 Eighth IEEE European Conference on Web Services*, pp. 189–196 (2010).
[22] Guo, T., Sharma, U., Wood, T., Sahu, S. and Shenoy, P.: Seagull: Intelligent Cloud Bursting for Enterprise Applications, *Presented as part of the 2012 USENIX Annual Technical Conference (USENIX ATC 12)*, pp. 361–366 (2012).
[23] Bicer, T., Chiu, D. and Agrawal, G.: A Framework for Data-Intensive Computing with Cloud Bursting, *2011 IEEE International Conference on Cluster Computing*, pp. 169–177 (2011).