AI Computing Broker:
AI コンピューティング効率の最適化

著者: Matthias Loipersberger[1]*, Shinichi Awamoto[2], Godai Takashina[2] and Yusuke Nagasaka[2§]

概要

AI の能力が急速に拡大するに伴い、企業やテクノロジー各社は GPU インフラに多額の投資を行っている。しかし、これらの投資にもかかわらず GPU 利用率の低さは依然大きな課題であり、75%以上の組織が最適な利用率レベルを下回って運用されている。この非効率性は、インフラコストの増大を招くだけでなく、AI 開発とイノベーションの阻害要因にもなっている。

AI computing broker (ACB) は、これらの課題に対し、既存のスケジューラーを置き換えるのではなく、補完することで解決策を提供する。ACB はスタンドアロンのソリューションとしても機能し、既存環境への変更や置き換えを強制することなく、シームレスに統合が可能である。特に、物理的に利用可能な GPU 数を超える要求があるオーバーサブスクリプションシナリオや、利用率の最大化と GPU のアイドル状態の削減を目指す環境において、ACB は効果を発揮する。動的な GPU 割り当て、フルメモリアクセス、そしてリアルタイムのワークロード監視を通じて、ACB はスループットと運用効率の向上が見込まれる。 ACB は Slurm にネイティブ対応しており、Kubernetes サポートもまもなく提供予定である。これにより、コンテナベースのクラスター環境においても、ACB による GPU 動的割当ておよび監視機能を拡張して利用可能となる。

本ホワイトペーパーでは、技術リーダーに向けて、ACB のアーキテクチャ、導入戦略、そして具体的な活用例を詳細に解説し、ACB が AI インフラの効率性をどのように変革し、競争優位性をもたらすのかを提示する。

[1] Fujitsu Technology Strategy Unit
[2] Fujitsu Research Computing Laboratory
お問合せ先:* mloipersberger@fujitsu.com §nagasaka.yusuke@fujitsu.com

目次

1.はじめに

AI の急速な拡大を支援するために、様々な企業が、GPU インフラに歴史的な投資を行っている。ハイパースケーラーは、**2025 年に AI インフラに合計 3000 億ドル以上を費やす**と予測されており、企業のリソース配分が計算量の多いシステムに地殻変動的にシフトしていることを示している。さらに、大規模な AI インフラの消費電力は、2030 年までに世界全体の電力消費量の 10%に達すると予測されている。このような投資は、実験段階の AI からミッションクリティカルなワークロードへの移行を反映し、高性能計算の戦略的重要性を浮き彫りにしている。

大規模な GPU 投資を行っているにもかかわらず、企業は利用率の低さに苦慮しており、これにより ROI が直接的に損なわれてきている。The State of AI Infrastructure at Scale 2024 レポートでは、75%以上の組織がピーク時の GPU 利用率が 70%未満であることが明らかになっている。この非効率性は、インフラコストの肥大化、モデル開発のボトルネック、次世代 AI ワークロードの実験能力の制約を引き起こしている。競争力を維持し、AI 投資から最大限の価値を引き出すために、企業は GPU 利用率のギャップを早急に埋める必要がある。GPU 利用率の低下の要因には、静的なジョブ割り当て、異種コンピューティングプロファイル、非効率的なスケジューリングなど、いくつかの要因があげられる。静的な割り当ては、ジョブ期間全体にわたって GPU を拘束するため、CPU 負荷の高いフェーズでは GPU アイドル時間が長くなっている。

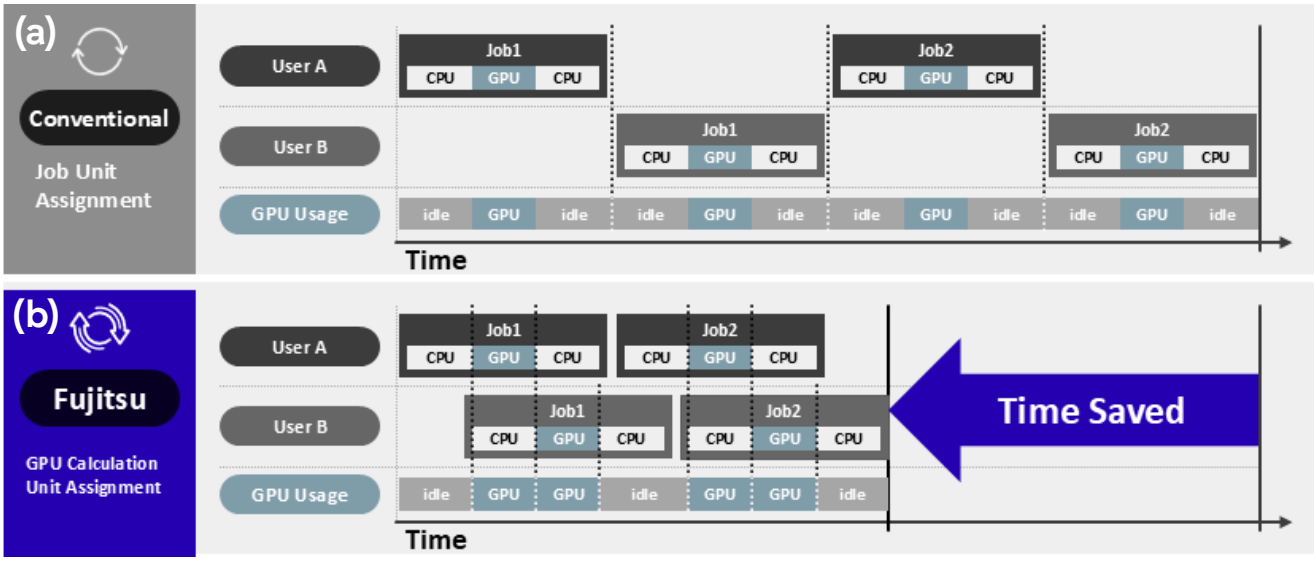
AI computing broker (ACB) は、実行時認識型の GPU 割り当て、フルメモリアクセス、高度なスケジューリングアルゴリズムによって、これらの課題解決に取り組んでいる。ACB は、フレームワークレベルのアクティビティを監視し、GPU を動的に割り当て、バックフィル技術を適用してスループットを最大化し、アクティブなジョブが必要なメモリを確保する。これらの機能により、GPU 利用率の向上、インフラコストの削減、AI 開発の加速が実現可能となる。

このホワイトペーパーは、技術リーダー向けに ACB の明確な概要を提供し、そのアーキテクチャ、一般的な導入パターン、実践的なユースケースを解説する。ACB が一般的な AI スタックに統合される方法と、より少ない GPU でより多くのワークロードを実行するための活用方法を説明する。構成は次の通り:

- **セクション 2** では、現代の AI ワークロードにおいて GPU 利用における非効率性が生じる根本原因を探る
- **セクション 3** では、AI Computing Broker (ACB) の概要と、その核となるコンポーネントおよび技術設計について詳述する
- **セクション 4** では、導入を成功させるためのベストプラクティスを概説する
- **セクション 5** で、ACB のシステムアーキテクチャについて解説する
- **セクション 6** では、次の技術的なユースケースを紹介する
    - 異種コンピューティングプロファイルでの AlphaFold2 の実行
    - vLLM と連携した ACB による複数の LLM（大規模言語モデル）の同時ホスティング
- **セクション 7** では、Docker、Slurm、Kubernetes を活用し、既存の MLOps パイプラインへ ACB を統合する方法を示す

2.GPU 利用効率低下の根本原因

GPU の利用率の低さは、静的な割り当て、スケジュールの非効率性、構造的な摩擦に起因する。AI パイプラインでは、GPU を集中的に使用するフェーズと CPU を使用するフェーズを交互に繰り返すことが多く、従来のスケジューラーはジョブ全体に GPU を完全に割り当てたままにするため、GPU 需要の低いフェーズでハードウェアがアイドル状態になる。このシナリオを 図 *1 (a)* に示す。多くのパイプラインは異種混合型（セクション 6 を参照）であるため、固定割り当てモデルは効果的ではない。The State of AI Infrastructure at Scale 2024 レポートによると、**74%の企業が現在のスケジューリングツールに不満を抱えており**、ジョブキューを可視化できているのはわずか **19%**である。関連するもう 1 つの問題は、GPU を共有する際の **GPU メモリの分割**である。GPU 仮想化やマルチインスタンス GPU などの従来のアプローチでは、GPU のメモリが複数のジョブに分割される。これにより各ジョブが利用できるメモリが減少するため、各ジョブが処理できるモデルやバッチのサイズが制限される。つまり、仮想化では並列使用を可能にするが、各ジョブに割り当てられるメモリが制限されるため、パフォーマンスの低下や大規模モデルが実行できない可能性がある。これらの要因により、**75%以上**の企業がピーク時の GPU 利用率 70%未満で運用している。



【図 *1*】異種コンピューティングプロファイルを持つ *2* つの並列 *AI* ワークロードにおける *GPU* スケジューリング動作 *(*上段と中段*)*;下段は、時間経過に伴う *GPU* 利用率の合計を示している。*(a) ACB* を使用しない*:* 静的 *GPU* 割り当てにより、*CPU* を使用するフェーズで *GPU* の未使用が発生している。*(b) ACB* あり*: GPU* のアイドル時間を有効活用し、動的に再割り当てすることで、*GPU* 利用率が向上する。

3.AI Computing Broker の概要

ACB は動的で時間依存型の共有アプローチを通じて、複数の AI アプリケーションに効率的な GPU 共有を提供する。GPU メモリを静的に分割する代わりに、ACB はリアルタイムなアクティビティに基づいて GPU-CPU メモリ間でモデルを戦略的にスワップし、単一 GPU の容量を超える大規模な言語モデル（LLM）でも同時実行を可能にする。このランタイム対応の割り当ては、2 つの重要なコンポーネントによって管理される。中央スケジューラーである GPU assigner と、クライアントサイドのライブラリである Adaptive GPU allocator である。ACB を使用した並列 AI ワークロードの実行を図 *1(b)* に示す。ACB のサポート環境については付録 A1、競合他社との比較については付録 B を参照してほしい。

GPU assigner は、バックフィルメカニズムを含むインテリジェントなスケジューリングポリシーを使用して GPU 割り当てを調整する。これにより、大規模なタスクがキューに入れられている（順番待ちの）場合でも、小規模なタスクが利用可能な GPU リソースを活用でき、スループットを最大化し、小規模なジョブがブロックされるのを防ぐことができる。Adaptive GPU allocator は、自動モードと手動モードの両方を提供する。自動モードは PyTorch API 呼び出しをインターセプトし、検出されたアクティビティに基づいて GPU 割り当てを暗黙的に管理する。手動モードでは、割り当てと解放の明示的な制御が可能である。この柔軟なアプローチにより、統合が簡素化され、実際のアプリケーションの動作に基づいてリソース利用率が最適化される。

重要な点として、ACB はモデルのスワップ中にアプリケーションの状態を保持することで、チェックポイント処理のオーバーヘッドと複雑さを回避することである。これにより、複数の並列実験またはモデルサービングインスタンスの実行が簡略化される。GPU の動的割り当て、タスクの効率的なスケジューリング、リソース管理の柔軟な制御により、ACB はユーザーが GPU 利用率を最大化し、インフラコストを削減し、AI ワークロードのスループットを最大化できるよう支援する。

ACB パフォーマンスオーバーヘッド

ACB システムは、ほとんどの AI ワークロードに最小限のパフォーマンスオーバーヘッドを招く。ACB の主要な設計目標は、効率的な GPU 共有を可能にしながら、アプリケーションパフォーマンスへの影響を最小限に抑えることである。当社の測定結果によると、例えば PyTorch API 呼び出しのインターセプトコストは、AlphaFold2 のコンテキストでは約 5%に過ぎなかった。

ACB のオーバーヘッドは、主に 2 つの要因から発生する。1 つ目は GPU メモリとの間でモデルをスワップするときのホストとデバイス間のデータ転送、2 つ目は gRPC を介した中央の GPU assigner との通信、である。gRPC 通信のオーバーヘッドは無視できる程度である。フックのオーバーヘッドはモデルサイズに依存するが、大規模なモデル（上記の AlphaFold2 に関する見積参照）では無視できる程度である。PCIe を介した大規模なモデル転送では、顕著な遅延が発生し、個々のジョブの完了時間に影響を与える可能性があるが、このコストは一般に、ACB によって提供されるシステム全体のスループットと GPU 利用率の向上によって相殺される。アイドル状態の GPU を保留中のジョブに効率的に割り当てることで、ACB はリソース利用率を最大化し、動的共有のないシステムと比較して各期間中に完了するジョブ数を増加させる。これは、データ転送による個々のジョブ遅延の潜在的な増加にもかかわらず、クラスターレベルでの大幅なパフォーマンス向上につながる。このトレードオフは、動的 GPU 共有の本質的な特性であり、全体的なリソース効率の大きなメリットを考慮すると、合理的な妥協点といえる。

4.利用ガイドライン

ACB は、**異種の GPU ワークロード**と**複数の同時実行ジョブ**が存在する環境において、最高の価値を提供する。以下のベスト・プラクティスにより、最適なパフォーマンスが確保できる。

4.1 ACB の効果を最大化するためのベストプラクティス

ACB から最大限の価値を引き出すためには、その動的 GPU スケジューリングが最も効果を発揮するワークロードと構成の種類を理解することが重要である。このセクションは、実践的なヒントとガイドラインを含む主要なパターンについて概説する。**主要な質問は Infobox 1 にまとめられている**。詳細については、A~F を参照。

Infobox 1: ACB の適合性を評価するための重要な質問

1. ワークロードには、CPU と GPU タスクのフェーズが交互に含まれているか？
2. 順番待ち（キュー）またはバッチ処理が必要な複数の GPU 対応ジョブがあるか？
3. ワークロードは、GPU メモリ全体を断続的に使用できるか?
4. 共有 GPU インフラ上で、需要が不均一な複数のドメイン固有の LLM をホストする必要は？

A. Multi-Phase Workloads with Mixed CPU/GPU Profiles

ACB は、GPU 処理と意味のある CPU バウンド処理の作業を交互に行うジョブで最も効果を発揮する。これには、AlphaFold2 のような推論パイプラインだけでなく、明確な前処理と後処理段階を有する学習のワークロードも含まれる。

重要: CPU と GPU メモリ間でのタスクコンテキストの転送は低速である。このオーバーヘッドを正当化するには、GPU ハンドオフが価値あるものとなるよう十分な長さの CPU フェーズを確保する必要がある。

推奨事項: ジョブは、30%以上の CPU を使用するタスクで構成する必要がある。

ヒント: プロファイリングツール（例: nsys、nvprof）を使用して、少なくとも数百ミリ秒の CPU のみ期間が頻繁に発生しているか確認が必要である。プロファイリングのスクリーンショットは付録 図 *5* を参照。

B. GPU プールを埋めるために複数のジョブを実行

ACB は、GPU を使用する待機中のジョブから選択できる場合に最適に機能する。1 つのジョブのみが実行されている場合、アイドル時間を回収できない。

ヒント: GPU 対応ジョブのキューまたはバッチを保持して、ACB が常にバックフィルできるようにする。

C. 類似ジョブのグループ化

ACB は現在、プリエンプションや優先順位付けをサポートしていないため、類似した優先度のジョブを同時にスケジュールすることをお勧めする。ジョブがあまりにも異なる場合、一方のジョブがリソースをブロックしたり、他のジョブに遅延を招く可能性がある。

例: 2 つの AlphaFold2 実行をペアにするのが理想的である。大規模な LLM ジョブと軽量モデルを混ぜると、バランスが悪くなる可能性がある。

D. GPU メモリを最大限活用

ACB は、MIG や vGPU による分割割り当てではなく、タスクごとに完全なメモリアクセスを可能にする。このメリットを活かすためには、タスクはフル GPU メモリを要求できる必要がある。

目安: GPU メモリの 80～100%を使用するが、計算サイクルを断続的に解放するワークロードは、ACB のメモリ保持とタスク多重化機能から恩恵を受けやすい。

E. 継続的な GPU バインドジョブの回避

中断のない GPU 利用率が非常に高いジョブでは、中断のないモデルトレーニングや高密度の推論ワークロードなど、ACB が最適化する余地がない。このような場合、メモリ転送とスケジューリングによるオーバーヘッドによってパフォーマンスが低下することさえある。

ガイドライン: ジョブが 90%以上の GPU アクティブで、CPU 側の処理が最小限の場合、ACB は適さない。

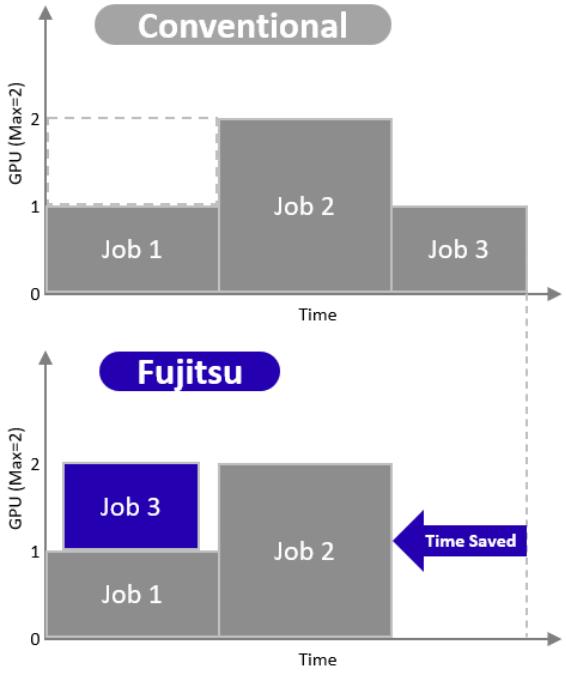
F. シングルノード vs マルチノード導入戦略

シングルノード導入: ローカルテストやパイロットワークロードに最適である。複数の AI ジョブを実行するマシンに ACB パッケージをインストールし、モニタリングツールで利用効率の改善を確認できる。

マルチノード導入: 本番環境に適している。ACB は共有スケジューラーフックとランタイムインターセプターを介してノード間の GPU 割り当てを調整する。ただし、ノード間の通信と永続ストレージの適切な設定が必要である。ドキュメントのクラスター導入ガイドラインを参照してほしい。

4.2 適切なスケジューラーの選択:

ACB は、異なる AI ワークロード向けに最適化された複数のスケジューリングポリシーを提供する。図 *2* は、適切なスケジューラーを選択することで、GPU の効率とモデルのスループットを向上させる方法を示している。



【図 2】バックフィルスケジューリングが *GPU* ジョブの効率に与える影響の模式図。上部のタイムラインはバックフィルなしの標準的なスケジューリングを示している: *ジョブ 1 (1 GPU)* 、*ジョブ 2 (2 GPU)* 、*ジョブ 3 (1 GPU)* が順次キューに追加され、*GPU* のアイドル時間が発生している。下部のタイムラインはバックフィルを適用した場合で、*ジョブ 3* が優先的に実行され、利用可能な *GPU* を効率的に活用しています。

以下、さまざまな ACB モードの概要を示す

- **simple (default) :** この先入れ先出し（FIFO）スケジューラーは、各ジョブに 1 つの GPU を割り当てる。単一の GPU と単一のノードを使用するシナリオで、ジョブが GPU の共有を必要としない場合に適している。
- **gpu-sharing:** このスケジューラーは、複数のジョブ間で単一の GPU の空間共有を可能にする。個々の AI モデルが GPU メモリの半分以下しか消費しない場合に有益である。このスケジューラーは、同じ GPU 上で複数の小さなモデルを並列実行することで GPU 利用率を最大化する。モデルごとのメモリ要件が低く、実行するモデルが多数ある場合に gpu-sharing を検討してほしい。
- **gpu-affinity (multi-GPU Jobs) :** このスケジューラーはマルチ GPU ジョブ向けに設計され、効率的な GPU 使用を優先する。FIFO 方式で動作するが、GPU 利用率を向上させる「バックフィル」メカニズムが組み込まれている。バックフィルを使用すると、小さなジョブは待機中の大きなジョブよりも先に「列を飛ばして」、空き GPU リソースを利用できるようになる。例えば、2GPU ジョブがキューに並んでいるが 1GPU しか利用できない場合、1GPU の小さなジョブがスケジュールされ、残りの GPU を利用することでリソース利用を最大化し、全体的なジョブ完了時間を短縮する可能性がある。このスケジューラーには、GPU 移動を回避しコンテキストスイッチングのオーバーヘッドを削減する GPU アフィニティ機能も組み込まれている。

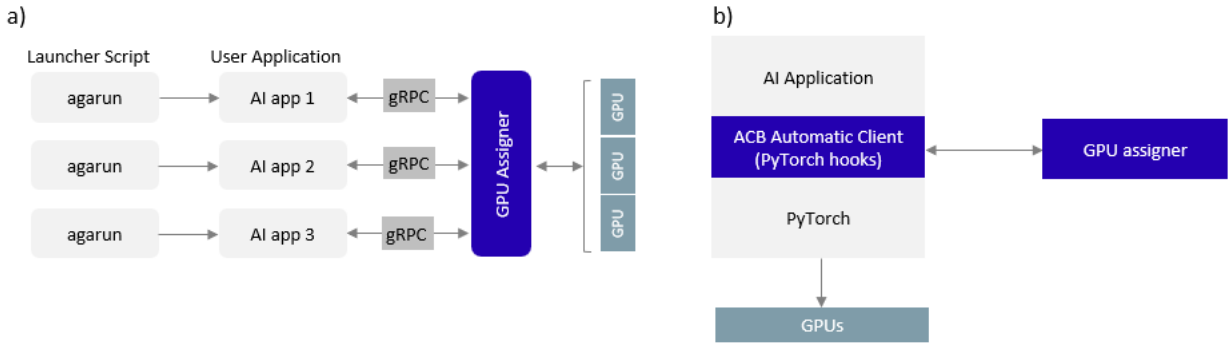
5.AI Computing Broker の技術アーキテクチャ

このセクションでは、モジュール式で 2 つのパートから構成される ACB のコアアーキテクチャの概要を説明する。

- **サーバー側:** GPU assigner デーモン
- **クライアント側:** クライアントライブラリとランチャースクリプト（*agarun*）を含む ACB client

GPU assigner は GPU サーバー上でデーモンとして動作し、利用可能な GPU を検出するとともに、クライアントアプリケーションへの割り当てを調整する。ACB client は AI アプリケーションプロセスに直接統合され、*agarun* ランチャースクリプトを通じて拡張機能を有効にする。ACB client と GPU assigner 間の通信は gRPC 経由で実現され、効率的かつ信頼性の高いデータ交換が保証される。

アプリケーションが GPU リソースを必要とする場合、GPU assigner に対して *AllocRequest()* gRPC 呼び出しを発行する。GPU の使用が完了すると、アプリケーションは *Release()* gRPC 呼び出しを送信し、GPU assigner がリソースを他のアプリケーションに再割り当てることができる。このリクエスト単位の割り当てモデルは、進行中のプロセスを中断することなく、動的かつ柔軟なスケジューリングをサポートしている。アーキテクチャは図 *3* に示している。



【図 *3*】アーキテクチャ
a) サーバー側*: GPU assigner* デーモンが *GPU* プールを管理し、*gRPC* 経由でアプリケーションにリソースを割り当て
b) クライアント側*: ACB client* は *PyTorch* の呼び出しをインターセプトし、*GPU* の使用パターンを監視および分析

マルチノードジョブの処理

ACB は、現在 *torchrun* に限定されているが、マルチノードジョブ処理に対応している。マルチノード構成では、各ノードに GPU assigner サービスが展開され、1 つのノードがコントローラーノードとして指定され、残りのノードが実行ノードとして機能する。コントローラーノードはすべてのノードからのリクエストを管理し、GPU リソースの全体プールを監視する。一方、実行ノードはローカル GPU の状態監視（メモリ使用量など）に焦点を当てている。内部の gRPC 呼び出しにより、コントローラーノードがこの情報を収集し、ネットワーク全体の包括的なリソース管理を保証する（詳細は、付録 図 *6* を参照）。

自動クライアント統合

この機能により、コードを変更することなく、ACB を既存の AI アプリケーションにシームレスに統合できる。自動クライアントを有効にして *agarun* を実行すると、ACB PyTorch API のフックが有効化され、GPU デバイスの利用状況を検出するとともに GPU assigner と通信することができる。自動クライアントは割り当てられた GPU デバイスとその上の計算を効率的に管理するため、手動でのテンソルデータ移動を不要にする。GPU 利用が終了すると、クライアントライブラリが GPU を自動

的に解放し、リソースの可用性を最適化し、アイドル時間を最小限に抑える。現在、自動クライアント機能は PyTorch アプリケーションのみ対応する。

API の手動使用

厳格なパフォーマンス最適化が必要なシナリオでは、アプリケーションコードに ACB client API 呼び出しを手動で組み込むことができる。この方法は PyTorch API フックをバイパスするため、オーバーヘッドが削減され、利用効率を向上させる。開発者は *on_device_begin ()* と *on_device_end ()* API を使用して GPU を利用するコードセクションを定義できるため、GPU リクエストとリリース、およびテンソルデータ管理を正確に制御できる。

フレームワークのサポート

ACB client は主に PyTorch フレームワークをサポートし、API の手動使用と自動クライアント統合の両方を提供する。自動クライアント機能は未サポートだが、TensorFlow の基本サポートも提供されている。さらに、ACB client を使用する場合、TensorFlow アプリケーションでプロセス再起動の可能性がある点に注意してほしい（サポートマトリクスは以下の表）

Framework	Manual Mode	Automatic Client
PyTorch	Supported	Supported
TensorFlow	Limited	Not Supported

Infobox 2: 専門的なアプリケーション領域において、開発者は ACB クライアント API を活用してカスタムクライアント統合を作成し、カスタマイズやパフォーマンス最適化を実現可能

6.AI Computing Broker の実際の使用例

ACB の実用的な価値を実証するために、2 つの一般的な AI インフラのボトルネックに焦点を当てる: 限られた GPU リソース下での[6.1]多段階の科学的推論、[6.2]マルチ LLM ホスティングである。これらの例は、ACB がどのようにして、コードの変更や最小限の統合で効率性を即座に向上させるかを示している。

6.1 ユースケース 1: ACB を活用した AlphaFold2 のスループット最大化

AlphaFold2 は、アミノ酸配列からタンパク質の 3 次元構造を予測することで構造生物学を変革し、長年の課題解決に貢献し、2024 年にノーベル賞を受賞した。しかし、そのパイプラインは CPU 負荷の高いステージ（例: MSA、テンプレート検索）と GPU 負荷の高いステージ（例: Evoformer、Structure Module）が混在しているため、GPU のアイドル時間が発生する。

通常、2 つの AlphaFold2 ジョブが 1 つの NVIDIA A100 GPU 上で独立して実行され、12 proteins/hour のスループットを達成している。静的割り当てのため、各ジョブは CPU 処理段階でも GPU への排他的なアクセスを維持し、大幅な GPU リソースの無駄が発生している。ライブシステムメトリクスは、妥当なスループットにもかかわらず GPU 利用率が低いことを確認している。

ACB を導入すると、複数の推論ジョブが単一の GPU を動的に共有することができる。ACB は実行中のアクティビティを監視し、アイドル状態の GPU フェーズを回収して他のジョブ（今回の例では 8 個）に割り当てることができる。これにより、AlphaFold2 スクリプトを変更せずにバックフィリングによる連続した実行が可能となり、32proteins/hour の処理速度を達成する。

インパクト

- 270%のスループット向上：ACB 非搭載の A100 GPU×2 から ACB 搭載の A100 GPU×1 へ切り替えて実現（8 つの並列ジョブ）
- 性能向上：処理能力が 1 時間あたり 12proteins から 32proteins に増加
- コード変更不要：ドロップイン導入可能
- ジョブの統合によりコストとエネルギー消費を削減

ACB はバイオインフォマティクスチームがより少ない GPU でより多くの AlphaFold2 ジョブを実行可能にする。大規模な分子ライブラリのスクリーニングにより、チームはより迅速で情報に基づいた意思決定を行うことができる。

6.2 ユースケース 2：ACB を活用した共有インフラストラクチャでの複数 LLM のホスティング

企業では、複数のドメイン特化型 LLM を導入するケースが増えている。通常、各 LLM は個別の vLLM サーバーを使用し、そのサーバーは特定の GPU に割り当てられている。そして、高速な応答を実現するために、モデルの重みは GPU のメモリに保持される。しかし、モデル間の需要は均一ではなく、静的な割り当ては GPU のアイドル状態と高コストを招く。

標準的な構成では、DeepSeek や Phi-4 のようなモデルは専用 GPU 上で個別の vLLM サーバーで提供される。低遅延は保証されるが、これはリソースの共有を妨げる。アイドル状態のモデルは GPU 全体を占有するが、需要の高いモデルはスケールできない。

ACB は共有 GPU 上で複数の vLLM インスタンスを管理することでこの課題を解決する。モデルごとにサーバーを割り当てる代わりに、ACB は実行コンテキストと GPU メモリを動的に処理する。アイドル状態の GPU メモリを回収し、インテリジェントにスケジューリングすることで、遅延や干渉なしに並列モデルサービングを実現できる。

2 モデルのデモを超えて、ACB は 8 つの H100 GPU 上で 10 以上の LLM にスケールできる。メモリの断片化を最適化し、GPU の割り当てを動的に調整することで、高密度なマルチモデルサービングを実現する。

インパクト

- より多くのモデルをより少ない GPU で実現
- モデル切替の遅延とコールドスタートを排除
- メモリ使用効率と計算リソースの飽和度向上
- 多様な LLM 間でスムーズな推論

戦略的影響

ACB はマルチ LLM ホスティングの経済性を変革する。チャットボット、多言語サポート、または内部アシスタントなどにおいて、ACB は柔軟な GPU 共有を可能にし、遅延や性能を犠牲にすることなくコストを削減できる。AI チームは、水平方向（より多くのモデル）と垂直方向（より高い GPU 密度）にスケーリング可能。

6.3 その他の検証済み展開例

これらの主要なユースケース以外にも、ACB は AI やクラウドプロバイダーとのトライアルでその存在を証明している。その結果、効率が最大 2.25 倍向上し、GPU アイドル時間が大幅に短縮された例もある。
詳細はプレスリリースを参照。

代表的なユースケース

- **外国為替（FX）リスクモデル**：複数の FX リスクモデルのトレーニングを同時に行うことができ、トレーニング全体の速度が向上
- **クラウドサービス**：クラウドサービスにおいてより多くのユーザーが GPU リソースを利用できるようになり、増加する AI 関連の需要に対応
- **小売業向け AI**：小売店で使用される AI カメラサービスの GPU 利用効率が向上

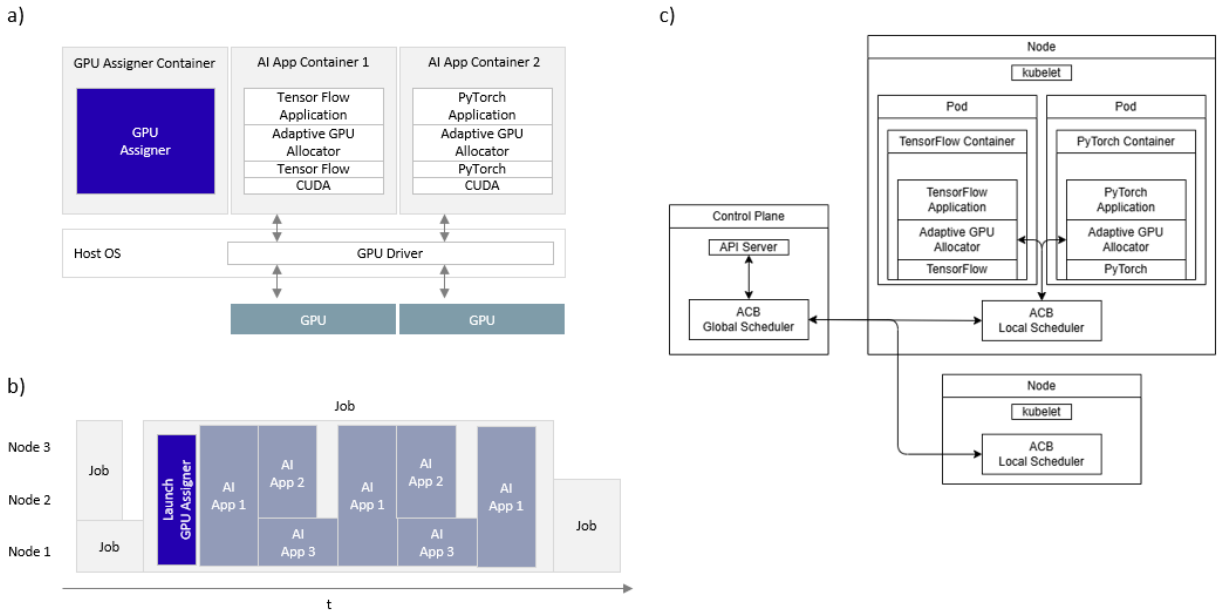
7.導入戦略

このセクションでは、ACB をローカルテスト環境から本番規模のクラスターまで、さまざまな環境でデプロイする方法について説明する。目標は、技術チームに対し、単一のマシン上で実行する場合でも、分散型コンピューティングインフラストラクチャ全体にデプロイする場合でも、明確な導入手順を提供することである。詳細な情報は、付録 A2 および A3 を参照。

デプロイモード

ACB は、研究チームと企業チームの双方のニーズに対応するため、柔軟な環境セットをサポートしている

- **ベアメタル**：サポートされている GPU ドライバーを搭載した Linux サーバーへ直接インストール
- **Docker コンテナ:** 迅速なデプロイのための事前構成済みコンテナーイメージ
- **ジョブスケジューラの統合:** Slurm やその他の一般的なスケジューラーと連携し、共有環境における AI ワークロードを管理
- **Kubernetes (計画中)** –将来のバージョンでは、コンテナ化されたクラスター全体での動的スケジューリングがサポートする予定。ステータスについては、リリースノートを参照



【図 4】*ACB* の統合 *(a) Docker: ACB* は *Docker* を使用して、*GPU assigner* と *Adaptive GPU allocator* を隔離し協調させ、リソースの効率的な利用を実現、 *(b) Slurm: ACB* は *Slurm* の *GPU* 共有機能を、*Slurm* 自体を変更せずにバッチスクリプトを通じて強化、 *(c) Kubernetes: ACB* はデュアルスケジューラーデザインにより、*Kubernetes* での細粒度な *GPU* 共有をサポート

7.1 Docker を使用した ACB の展開

ACB は Docker を活用し、コアコンポーネントである GPU assigner と Adaptive GPU allocator の展開と管理を効率化している。各コンポーネントは独立したコンテナで実行され、専用の Docker ネットワークを介して通信される。GPU assigner はスタンドアロンコンテナとして動作し、システムサービスのように機能し、クライアントアプリケーションからのリソース要求を待機する。ユーザーが作成した AI プログラムは個々のコンテナに展開され、Adaptive GPU allocator が自動的にプログラムをロードし、Docker ネットワーク経由で GPU assigner と通信する。

7.2 Slurm 統合

ACB は、Slurm で管理されるクラスター内に展開することで、Slurm 自体を変更することなく GPU リソース管理を強化できる。ユーザーは、Slurm のノード割り当てと ACB の効率的なノード内 GPU 共有の恩恵を受けることができる。デプロイには、Slurm のバッチスクリプトに GPU assigner と Adaptive GPU allocator の起動コマンドを追加する必要がある。ジョブが開始されると、Slurm がノードを割り当て、バッチスクリプトが ACB コンポーネントを起動する。GPU assigner は各ノードで利用可能な GPU を検出し、構成ファイルでノード間 GPU 管理用の IP アドレスを指定する。この設定により、Slurm ワークロード内で複数の AI アプリケーションが GPU を共有することができる。

7.3 Kubernetes Integration (Coming Soon)

ACB は、きめ細かい GPU スケジューリングを統合することで、Kubernetes における GPU リソース管理の課題に対処する。デーモンセットとして展開される Local scheduler は、GPU を搭載したノードに常駐し、ACB の Global scheduler と協調してリソースの効率的な配分を実現する。各 Pod 内では、ACB client が Local scheduler と対話して GPU リソースを管理し、アプリケーション間の時間共有を可能にする。この統合は NVIDIA GPU Operator を活用し、ハードウェアとのシームレスな相互作用を実現し、GPU 利用効率を向上させ、管理を簡素化する。

注意: ACB のバックフィル機能は Kubernetes 統合では利用不可。ジョブは FIFO でスケジュールされる。

要約

ACB は、企業の AI インフラにおける GPU の活用不足という普遍的な課題に対処するための革新的なアプローチである。GPU リソースへの歴史的な投資にもかかわらず、多くの組織は静的なリソース割り当てモデルと非効率的なスケジューリング手法のために、最適な利用率を達成できていない。これらの課題は、コストの増大を招き、AI ワークロードを効果的にスケールさせ、革新的な技術を生み出す能力を阻害している。

ACB は、リアルタイムのワークロード需要に基づいて GPU リソースをインテリジェントに割り当てる動的なオーケストレーションレイヤーを導入することで、これらの非効率性に対処している。このアプローチは GPU 利用率を最大化するだけでなく、インフラコストを削減し、AI 開発を加速する。本ホワイトペーパーでは、AlphaFold2 のスループット向上や共有インフラ上でのマルチ LLM ホスティング実現など、詳細なユースケースを通じて、ACB の効果を説明している。これらの例は、ACB がコード変更を必要とせずに大幅なパフォーマンス向上とコスト削減を実現できることを示している。

ACB は、ベアメタル、Docker、Slurm、そして将来的な Kubernetes 統合を含む柔軟な展開オプションを提供し、多様な運用環境に対応することで、シームレスな導入とスケーラビリティを確保する。AC を導入するで、組織は GPU インフラの潜在能力を最大限に引き出し、AI 開発の加速と ROI の向上を実現できる。

実行

AI ワークロードの規模拡大と GPU 供給の制約が続く中、**十分に活用されていないインフラは、イノベーションに対する隠れた税となっている。**ACB は、この非効率性を戦略的優位性に転換し、AI パイプライン全体で、スループットの向上、モデルあたりのコスト削減、そしてイテレーションの高速化を可能にする。

今こそ、AI リーダーは行動する時である。ACB を既存のスタックに統合し、その効果を検証し、既に所有しているハードウェアからパフォーマンスを取り戻すべきである。今日、GPU 利用率を最適化する組織こそが、明日の AI をより速く、より効率的に、そして時代の先駆けとして拡大する組織となるだろう。

謝辞

Kao I-Hsi 氏には、貴重な技術レビューと洞察をご提供いただいたことに深く感謝申し上げます。また、Iwata Ryuji 氏には、全体的なフォーマット、デザイン、および図表の洗練において多大なご支援を賜りました。ここに感謝の意を表します。

付録 A: システムの互換性と統合のシナリオ

ACB の円滑な導入を確実にするため、本付録では、現在のシステム互換性と高度な統合シナリオについて概説する。ACB ミドルウェアは、軽量かつ柔軟に設計されており、インフラストラクチャのオーバーヘッドを最小限に抑え、モデルコードの変更を一切必要としない。

A1 – ACB のサポート環境

コンポーネント	互換性
GPU ドライバー	NVIDIA CUDA 互換 (v11.x–12.x); CUDA 11.8+ および driver 525+ でテスト済
OS	Ubuntu 20.04 / 22.04
AI フレームワーク	PyTorch (2.1.2+): GPU 使用状況の検出にネイティブフックを使用 TensorFlow (2.15+): 手動変数モードにおいて限定的なサポートを提供
ハードウェアサポート	NVIDIA A100, H100, L40S, A10。MIG 対応および非対応の GPU をサポート (MIG 対応は近日公開)
推論ランタイム	vLLM との互換性
コンテナ化	Docker をサポート (Kubernetes への依存なし)
クラスター統合	シングルノード構成をサポート (マルチノードサポートは 2025 年 3Q まで限定的)
ライセンス/導入	Python パッケージとして提供 (API 経由でライセンス認証を実施)

A2: 統合に関する注意事項

- **モデルの透明性**: PyTorch を AI フレームワークとして使用する場合、ACB はユーザーモデルやトレーニングスクリプトの変更を一切必要としない。フレームワークプロファイリング API を介したバックエンドランタイムの計測を通じて GPU アクティビティを監視する。
- **メモリ管理**: ACB は、ランタイムアクティビティに基づいて GPU メモリとの間でモデル全体を一時的にスワップイン/スワップアウトする。これにより、すべてのモデルのメモリ使用量の合計が利用可能なメモリ容量を超えるオーバーサブスクリプションシナリオを可能にする。
- **実行コンテキスト**: ACB は、モデルのトレーニングまたは推論ジョブをラップし、GPU アイドル状態 (例: CPU 前処理) が検出された際に GPU を動的に解放する。アクティブな GPU 使用中は、フルメモリアクセスが許可される。

A3: 高度な互換性シナリオ

シナリオ 1: ACB + NVIDIA MIG (ドライバーレベルのパーティション化)

MIG 対応 GPU (例: A100 または H100) に展開された場合、ACB は MIG を補完し、ジョブに MIG インスタンスをリアルタイムで動的に割り当てる。MIG は隔離された GPU スライス (例: 10GB または 20GB のパーティション) を作成するが、ACB はこれらのスライスがアクティブな GPU フェーズの間のみジョブにバインドされることを保証する。これにより、複数の GPU バウンドジョブが、メモリ競合やリソースのアンダーユティライゼーションを最小限に抑えながら共存できる (MIG 対応は近日公開)

シナリオ 2: ACB + Triton Inference Server / vLLM (アプリケーション層)

Triton や vLLM などの推論サービングフレームワークは、通常、永続的な GPU バウンドワーカーを起動する。ACB と統合した場合、推論モデルはオンデマンドで GPU メモリにロードされ、推論後にアンロードされる。これにより、限られたメモリ容量でも**高スループットな LLM サービング**が可能になり、バースト的なトラフィックやマルチモデルデプロイメントシナリオに最適である (現在は vLLM のみをサポート)。

シナリオ 3: ACB + NVIDIA MIG + vLLM (フルスタック効率)

3 つの技術を組み合わせることで、強力な階層型システムが構築される。

- **MIG** は、単一の GPU を複数の独立したスライスに分割する。これは、モデルが GPU 全体を必要としない場合に最適である
- **ACB** は、各 MIG スライス内で動作し、複数の LLM モデルを動的にオーケストレーションする
- **vLLM** は、高速かつ効率的な LLM 推論を可能にする

このアーキテクチャは、**高密度マルチモデルホスティング**をサポートする。中小規模モデルは MIG スライスを共有することで、競合を回避しながら利用率を向上させることができる。

付録 B – 競争優位性: ACB の GPU 効率における真のイノベーション

富士通の ACB を、3 つの主要プラットフォーム（Run:ai、Exostellar.ai、Slurm）と比較し、機能マトリックスの有効性を検証するとともに、ACB が優れている点を明確にする。

機能比較表

機能	ACB	Run:ai	Exostellar.ai	Slurm
ランタイム対応スケジューリング	対応	非対応: コンテナ/ジョブ単位での時分割スケジューリング	非対応: リソース割り当てを監視し、動的に調整	非対応: ジョブ単位の静的割り当て、ジョブ内での再スケジューリングなし
メモリパーティショニング	対応:各アクティブフェーズで GPU メモリ全体を割り当て	対応: 細分化と自動化された MIG パーティション	対応: ソフトウェア定義された GPU 仮想化	一部対応: MIG およびソフトウェアによる"シャーディング"を使用
メモリのオーバーサブスクリプション	対応:GPU メモリ全体を時間ベースでトレード	一部対応: GPU メモリスワップ可、ただし Run:ai のタイムスライシングスケジューラと併用不可	対応: 仮想デバイスによる安全なオーバーコミットをサポート	非対応: サポートされていない。シャードは静的で、MIG はパーティション分割される
クラスターレベルのオーケストレーション	(計画中)	対応: Kubernetes の完全な統合	一部対応: Kubernetes を介した統合（移行とスケーリングを含む)	対応: HPC クラスター内のプラットフォーム;明確に定義されたキューとスケジューリング
プラグアンドプレイ統合	低: ミドルウェア、コード変更不要	中: K8s+Run:ai の展開が必要	中: 各ノードにシステムレベルのドライバー/Intel インストールが必要	高: 完全な HPC スケジューラーの設定が必要
主な差別化要因	タスクレベルの GPU キャプチャ	動的な GPU 分割とリソースクオータ管理	ソフトウェア GPU 仮想化+オーバーサブスクリプションコミット	高い信頼性とスケールを備えたバッチスケジューリング

競合技術のハイライト

ACB

- ランタイムの挙動から識別されるタスクの GPU フェーズ（例: 順伝播/逆伝播）に入る場合にのみ、**GPU メモリと計算リソース全体**を動的に割り当てる。これにより、ジョブのライフサイクル全体にわたるハードウェアの競合を最小限に抑える
- GPU メモリをタスク間で完全に再利用可能として扱うことで、**メモリオーバーサブスクリプション**を可能にする。タイムベースのスワッピングにより、40GB の GPU 上で 150GB の AI 処理を実現したユーザー事例がある

Run:ai

- GPU 分割および動的 MIG 機能により、ジョブの投入時に GPU メモリを動的に分割する
- プロジェクトおよび部門に基づいた高度なリソース割り当てを提供し、公平なスケジューリングと一時的なクオータ超過を可能にする
- Kubernetes クラスター環境に特化し、緊密な統合を提供する

- メモリオーバーサブスクリプションには対応していない。つまり、それぞれが GPU メモリ全体を必要とする 2 つのジョブを同時に実行することはできない（詳細はこちら）

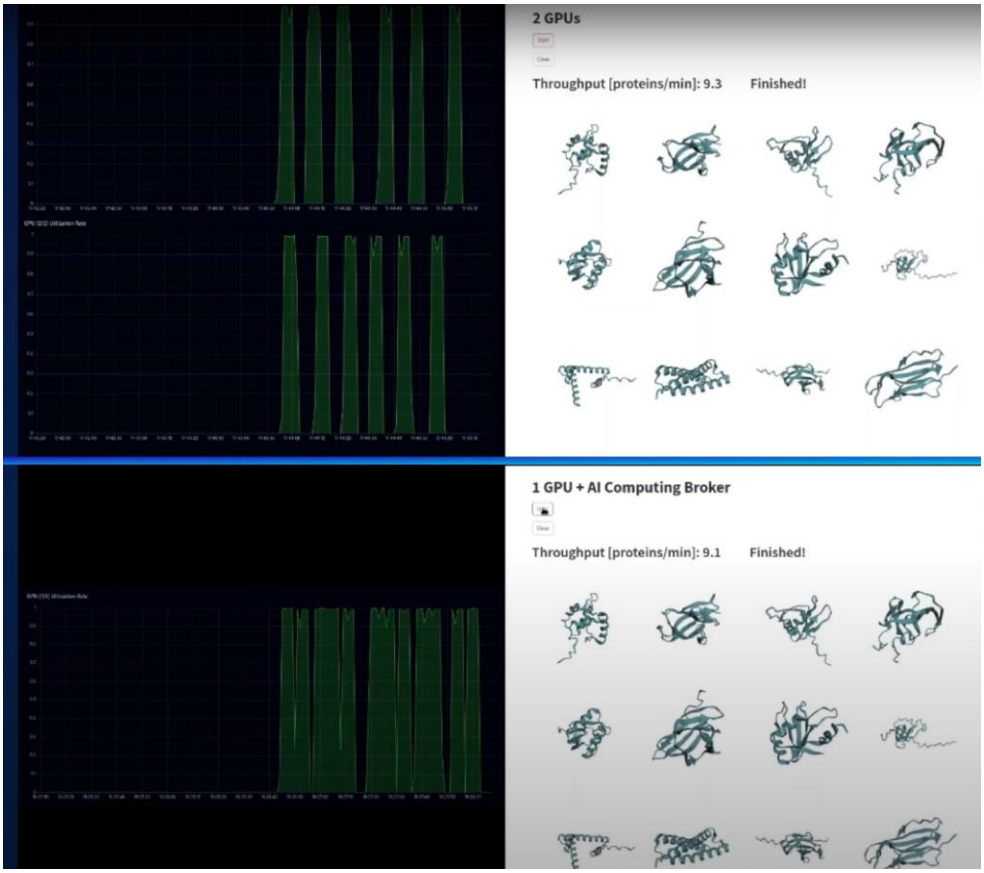
Exostellar.ai

- **ソフトウェア定義 GPU（SDG）**仮想化を採用：タスクは、構成可能なメモリ/計算リソースで分離された仮想 GPU を取得する
- タスクがピークリソースを同時に使用しない場合、ページングおよび仮想メモリのスワップを使用して安全な**メモリオーバーサブスクリプション**を可能にする
- テレメトリを介してリソースの使用状況を監視し、動的に調整する。Kubernetes 統合および移行と自動スケーリングのためのクラスタリングツールを含む（詳細こちら）

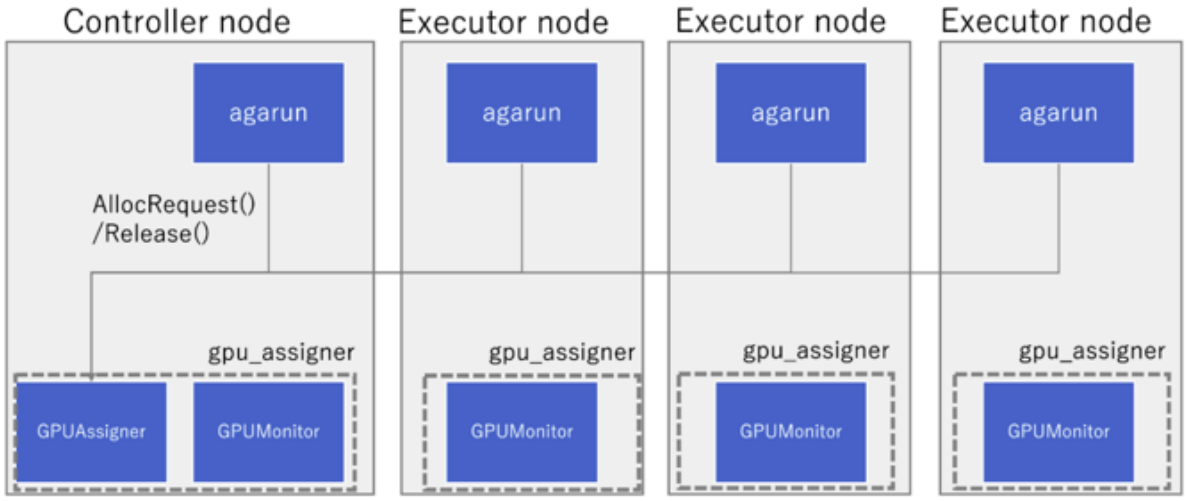
Slurm

- デファクトの **HPC スケジューラー**として、安定したポリシー駆動型のジョブキューイングと、パーティション分割された GPU 使用のための MIG/シャードサポートを提供する
- **ソフトウェア GPU シャーディング**（v22.05）を導入し、ジョブに部分的なリソース割り当てを可能にする。ただし、**メモリフェンシングは提供されない**ため、ユーザーは安全な協調スケジューリングを確保する必要がある
- **ランタイム GPU の再割り当てがない**。そのライフサイクル全体を通して GPU 割り当てを保持する

付録 C: 追加情報



【図 5】上段: *ACB* を使用せずに *AlphaFold2* を実行する *2* つの *GPU*。下段: 単一の *GPU* 上で同じ *2* つの *AlphaFold2* ジョブを実行。左側: *GPU* 利用状況、右側: *AlphaFold2* 推論の出力結果



【図 6】*ACB* マルチサーバ構成のアーキテクチャ: 主要な *GPU assigner* はコントロールノードに配置され、各実行ノード上の *GPU monitor* と連携して、グローバル *GPU* リソースプールを制御