TSUBAMEシリーズを使いやすくするためのソフトウェア運用

Tokyo Tech

東京工業大学 学術国際情報センター
マネジメント准教授　野村 哲弘
nomura@gsic.titech.ac.jp

2024年2月5日（月）
PCCC AI/HPC OSS活用ワークショップ

この講演の概要

- 東工大のスパコンシリーズ TSUBAME の運用を裏から支えてきたシステムソフトウェアに着目し、どのような機能を実現するためにどのようにソフトウェアを選択してきたかを報告する
  - ということで、ユーザが持ち込むOSSのような話はしません
  - 空気読まずにOSS以外の話もします
- 事例1: スケジューラで資源分割
- 事例2: 初心者向けインタラクティブ利用

東工大のスパコン TSUBAMEシリーズの紹介(超簡易版)

- TSUBAME3.0
  - 2017/8~2024/3
  - 540ノード
  - 2x 14コアXeon(Broadwell-EP)
  - 4x P100 (SXM, HBM2 16GB)
  - 大岡山キャンパスに設置

- TSUBAME4.0
  - 2024/4~2030/3(予定)
  - 240ノード
  - 2x 96コア EPYC 9654(Genoa)
  - 4x H100 (SXM, HBM2e 94GB)
  - すずかけ台キャンパスで設置中
  - 今年PCCCイベントで見学会予定

スパコンセンターにとってのOSS・プロプラ選択基準(私見)

	OSS	作りこみ	プロプライエタリ
新しいものへのキャッチアップ	◎ 予算を立てなくても お試しで入れられる	○ 簡単に作れるなら やる価値はある	× 運用中に新たに 予算を割くのは困難 →数年に1回
サポート (何か起こったときに泣きつく先があるか)	△ 外部のサポートを買うと プロプラと同レベル支出	○ 自分で作ったらなら 何か起こっても対応できる? (対応できない規模のを作らない)	◎ 導入業者としては 訊く先があるのは安心
ソフトウェア資産として残るか?	◎ 他所のスパコンなどにも 持ち出すことができる	×～○ 作り方次第 特定のシステムべったりだと 他所では使えない	×? 他所で同じことを やろうとしたら 要車輪の再発明?
手元での再現	◎ 研究室マシンで利用可	△ 研究室マシンで要再構成	× ライセンスとか別途必要

実際はきれいに分かれるわけではなく、○○**+作りこみということは往々にして起こる**

- 結論: 適材適所
  - 止まると全員が困るもの(スケジューラとかOS本体・ストレージ)は**サポートがあるもの**が必須
  - 止まっても全員が困るわけではないものはOSS活用で**新技術の恩恵**を
  - システムに近いもの(コンテナランタイムとか)を入れる場合、いずれにせよそのソフトが何をやっているのかは分かってないと入れられない

事例1
スケジューラで資源分割 (プロプラ+作りこみ)

スケジューラはUniva(Altair) Grid Engine

大規模な作りこみでTSUBAMEの資源分割を実現

TSUBAMEにおける資源分割の必要性

- TSUBAMEの計算ノードは比較的大きく、必ずしもユーザが1ノードを使い倒すことができるわけではない
  - マルチGPUプログラミング無理
  - GPUを使えないユーザ
  - GPUだけは使うけど、CPUコアそんなに要らない
- TSUBAMEは世代を経るごとにノード数が減少、同時実行可能なジョブ数を稼ぎたい
  - TSUBAME2: 1408Nodes, 2CPU(6 Core/Socket)+3GPU
  - TSUBAME3: 540Nodes, 2CPU(14 Core/Socket)+4GPU
  - TSUBAME4: 240Nodes, 2CPU(96 Core/Socket)+4GPU

→ ノードを仮想的に分割して見かけ上の(=スケジュール対象とする)ノード数を増やす

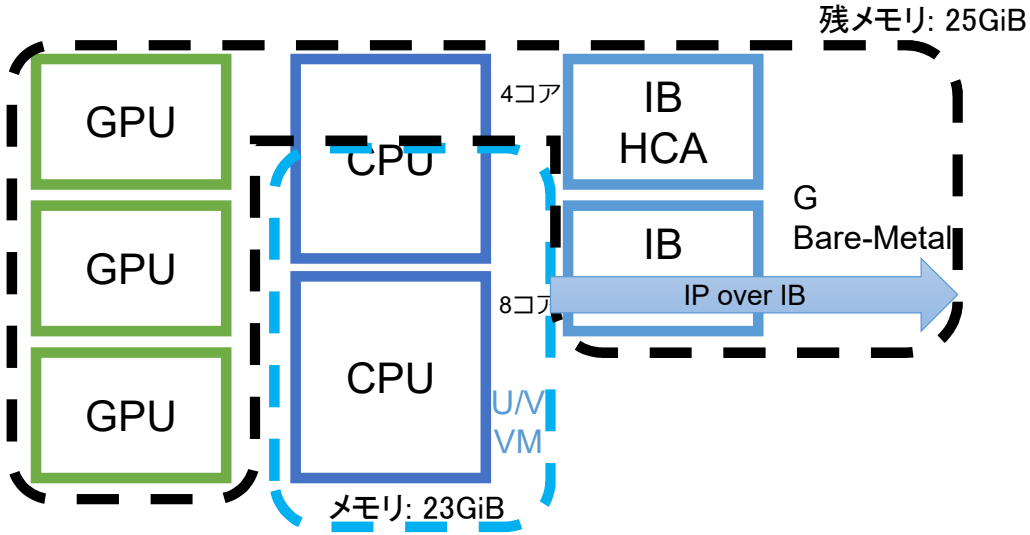
TSUBAMEでの資源分割まわりの要件 (As of 2017)

- Performance
  - GPUやInterconnectがリソース分割機構の所為で遅くならない
- Usability
  - リソース分割しても「今までできていたこと」ができる
    - 計算ノードにSSHしてのデバッグ・Xアプリケーション(Visualization)が使える
- Isolation
  - 計算ノードを共有する各ジョブが互いに相手を見ることができない
    - TSUBAMEには産業利用ユーザもいる
- Virtualization (Optional, extended goal)
  - ジョブによって最適なユーザランド(OSカーネル以外のシステムソフトウェア)を選択できる
    - ISV: ベンダーによる認証のある環境
    - AI研究者: 最新のライブラリが使える環境 (特に運用開始から時間が経った時)
  - セキュリティパッチが容易に当てられる

注: 以前は「仮想化」の語で、資源分割とユーザランド仮想化(コンテナ化)が一緒くたに語られていた (資源分割手段としてVM仮想化が使われてた名残?)

TSUBAME2.0のノード分割

- TSUBAME2のThinノード(1408台)のうち一部(435台)に、CPUジョブ用のVMを常駐させ、別マシンとして扱う
  - 当時GPU仮想化はできなかったので、GPU側はベアメタル
  - VM側のネットワーク性能(RDMA使用不可)は諦め
    - MPIもIB前提の物とは別にMPICH/TCPを用意



TSUBAME3における「仮想化」: cgroup + Docker

2017年ぐらいのスライド
(TSUBAME3構築期)より
この頃はまだ資源分割とユーザランド
仮想化がごっちゃになってた

- 2017年時点でのリソース分割の選択肢
  - VMを用いた仮想化
  - コンテナ技術(Docker, Shifter, Singularity等)を用いた資源分割
- VMを使う場合
  - 7年前(T2構築時)と比べ、PCI pass throughなど、VM仮想化でも性能を損なわない技術が出てきている
  - ネットワークが分割できない: TSUBAME3のOmniPathはSR-IOVをサポートしていない → VM仮想化だとネットワーク性能が出ない
- コンテナ技術を使う場合
  - 性能面での懸念はない: OSカーネル(GPU/通信)はホストで動作
  - SSH利用、リソース分離、セキュリティは要検証
- TSUBAME3.0ではDockerを導入
  - 運用開始時は試験的サービスとしてテスト
  - 将来的には全てのジョブでDocker利用を強制
  - Dockerイメージの持ち込みは当面許可しない (セキュリティの問題が未解決)

TSUBAME3における「仮想化」: cgroup + Docker

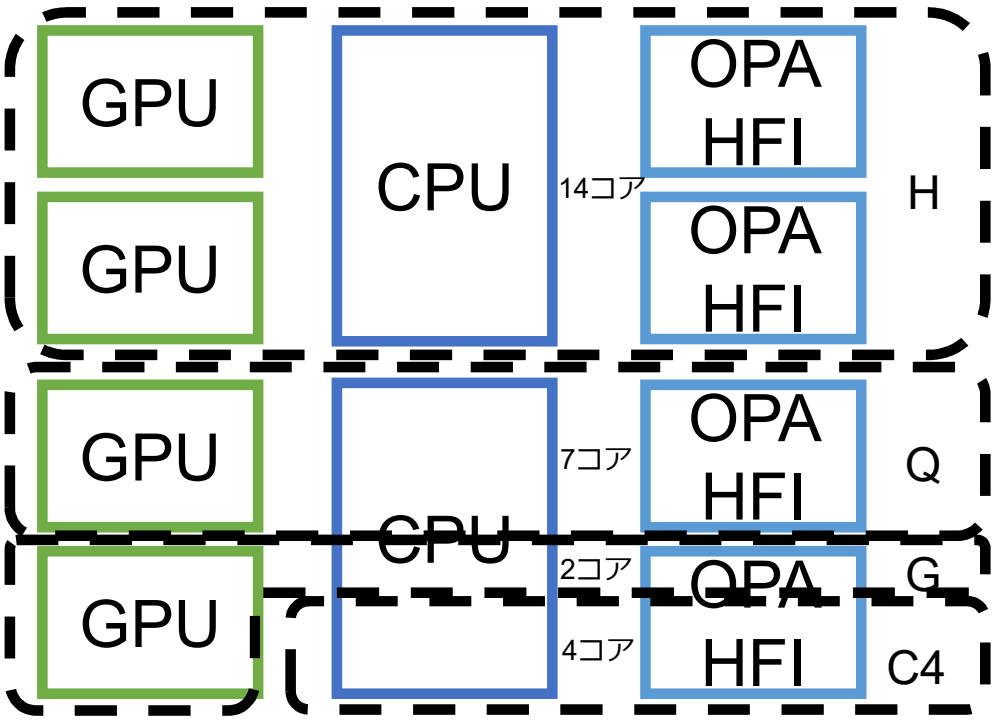
2017年ぐらいのスライド
(TSUBAME3構築期)より
この頃はまだ資源分割とユーザランド
仮想化がごっちゃになってた

- 2017年時点でのリソース分割の選択肢
  - VMを用いた仮想化
  - コンテナ技術(Docker, Shifter, Singularity等)を用いた資源分割
- VMを使う場合
  - 7年前(T2構築時)と比べ、PCI pass throughなど、VM仮想化でも性能を損なわない技術が出てきている
  - ネットワークが分割できない: TSUBAME3のOmniPathはSR-IOVをサポートしていない → VM仮想化だとネットワーク性能が出ない
- コンテナ技術を使う場合
  - 性能面での懸念はない: OSカーネル(GPU/通信)はホストで動作
  - SSH利用、リソース分離、セキュリティは要検証

Dockerイメージはまともに使われず、**忘れてもらうことになった** (TSUBAME4では不採用)

- **ユーザがイメージ持ち込めない**時点でユーザ利点0
- 過去のソフトウェア環境を再現するという妄想は実現せず
  - コンテナ外のドライバのバージョンにユーザランドが強く依存、環境仮想化など夢でしかない
- 後から入れたSingularity(Apptainer)がrootlessのため、ユーザ持ち込みイメージの実行が可能に
  - ABCI, 富岳などにも入りデファクトスタンダードに

TSUBAME3.0でのリソース分割メニュー



- F: 計算ノード1台をそのまま使用
- ノードを階層的に分割
- H: ½ノード
- Q: ¼ノード
- G: 1 GPU + 2 CPU Core
- C4: 4 CPU Core
- C1: 1 CPU Core

これをUGE(現AGE)の機能(+**相当な**作り込み)でノード単位で動的に切り替えれるようにした
(キューとしては単一で、資源タイプごとにResource Mapを定義して重ならないよう制御)

TSUBAME3でのResource Isolation

- Linux cgroupで許可されてない資源へのアクセスを制御
  - CPUのコア割り当て
  - メモリ割り当て
  - GPUの使用可否 (デバイスが見えなくなる)
- ローカルSSDはQuotaをかけたディレクトリを提供
- VMを介していないので、資源分割に起因する性能オーバヘッドはない
- ノードへのSSHはf_node(資源分割なし)以外は不可、かわりにqrshを使うことでXの利用を含めて対応できた
- 他人のプロセスが見える問題は、/procのマウントオプション hidepid で対処
  - 一時期Singularityが hidepid されている /proc に対応していないせいで不具合を起こしたことがある(直してもらった)

TSUBAME3でのユーザランド仮想化

- Dockerはコンテナ内のroot=ホストのroot
  - ユーザに任意のイメージを持ってきてもらうのは困難
  - (最近は Rootless Dockerもある)
- Singularity
  - HPC環境で動作することを前提に開発されたコンテナランタイム
  - 実行時に原則 root 不要
    - その分できることは限られる(すべきでないことができないことを担保)
  - TSUBAME3導入後、ユーザの希望を受けて整備
  - 現在は Singularity Pro/CE と Apptainer にフォーク

TSUBAME4では?

- ノード数が半減(540→240)したので、分割を倍にしたい
  - 1/4ノード(1GPU)をさらに半分にする → NVIDIA MIGでGPU分割
- CPUコアがいっぱい増えた(28→192)ので、CPUジョブはもっと分割できる
- 以下の資源タイプを設定 (緑:新規)
  - node_{f,h,q,o}: ノード単純分割系, oのみMIGで2分割
    - Full, Half, Quarter, One eighth とつけたけど、もう少し考えるべきだった?
  - gpu_{1,h}: GPU1個とCPUコア少数(8コア/GPU, 4コア/MIG)
  - cpu_{160,80,40,16,8,4}: CPUコアだけ (T3は4と1)
  - 資源タイプ名はTSUBAME3とあえて語順を変えた
    - 同じスクリプトがそのまま通らないようにして、見直しをさせるため
    - 同じ文字を複数の意味で使うのも避けた
      - T3: q_nodeは1/4ノード, q_coreは4コア
    - ↑の関係で数字が1文字目に来るようになるのを念のため避けたというのも

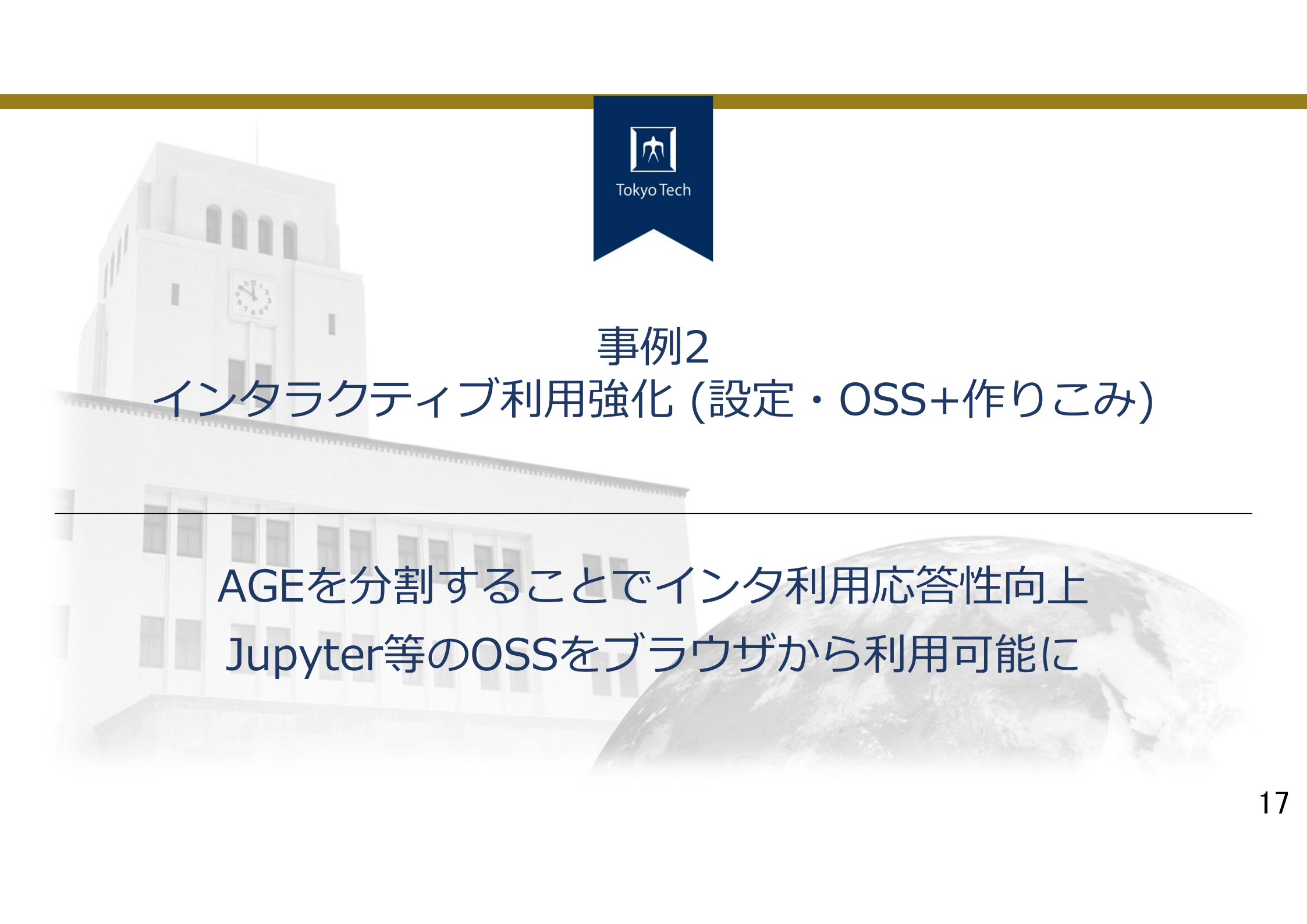
MIG(Multi Instance GPU)によるGPU分割

- NVIDIA A100からできるようになったGPU分割機能
  - タダでできるわけではなく、MIGモードにした瞬間に、GPUのコア数が7/8ぐらいに減少
    - → MIGやりたいからって常時MIGモードにすると性能減
  - MIGモードにした瞬間に、GPUのデバイス指定の書式が変わる
    - CUDA_VISIBLE_DEVICEとかに書く内容
    - 結果的にcgroupだけで管理して、CUDA_VISIBLE_DEVICEはいじらないことに
- 事前検証の結果(by HPE, Altair)などから、TSUBAME4ではMIGによるGPU2分割とMIGなしをGPU単位で切り替えることに
  - 普段はMIGで2分割しておく
  - スケジューラがMIGの両側のGPUをつかんだ際に、Job prologueとJob epilogueでMIG解除・MIG再設定を実施
  - cgroupで制限した状態でMIGの解除・設定をすると、触れるGPUだけのMIGモードが切り替わる
  - 4分割以下にしない理由: MIG切り替え時に対象物理GPU上にジョブがいないことを保証できると考慮すべき状態が減るから

TSUBAME4での実装

- AGE(旧UGE)でのTSUBAME3向け作りこみの再利用
  - MIG利用や分割の細分化でそれなりに課題は多かった模様
  - AGEの開発ロードマップの方向性とは多少相違
    - AGEのロードマップに従った再実装では想定の機能が実装できなかった
  - 将来的に必要な機能が実装できれば、移行したい
    - 激しい作りこみ → サポートされたプロプラソフトの機能 へ転化

  (この辺、かなり細かい話なうえ、**現在進行形で**実際に作りこみを行っているのはHPEとAltairなので、詳細に説明できません、ごめんなさい)
- 最終的に東工大以外のスパコンでこの辺が再利用される日が来るといいな



事例2
インタラクティブ利用強化 (設定・OSS+作りこみ)

AGEを分割することでインタ利用応答性向上

Jupyter等のOSSをブラウザから利用可能に

スパコンでのJupyter Labの利用

- Jupyter Lab
  - Webブラウザ上でPythonプログラムを対話的に実行
    - Mathematicaのノートブックと同様の操作感
  - 近年ではNotebook(Python)以外にも
    ファイル操作・編集やコンソールアクセスも可能に
  - AI分野での資料などにJupyter Notebookを使う例が多数
- 特にシステムの助けがない場合、以下のような起動手順
  - Jupyterのインストール (pip install --user jupyterlab)
  - 計算ノードの確保
  - Jupyterの起動 (を、**ジョブスクリプト内で**行う)
  - **ジョブが走り始めてから**、Jupyterが待ち受けする計算ノードまで
    SSHポートフォワード設定
  - 独力でTSUBAME上で使えるようにするのは**極めて面倒**

インタラクティブジョブ専用ノード/キュー

- TSUBAME3の運用開始から早々に、「京」停止に伴うHPCIユーザの流入があり、TSUBAMEが超混雑
- デバッグや可視化のための対話利用が困難に
  - TSUBAME2ではGPUつきノードをログインノード(20台)にして、対話的利用はそこでやってもらっていた (**インタラクティブノード**と呼んでいた)
  - TSUBAME3ではログインノードは2台、GPUもつけず、デバッグ利用等には利用不可
- TSUBAME3の4ノード(<1%)を切り出して、インタラクティブジョブ専用にした
  - ありものの改造で作ったので、基本はq_node(1/4ノード)
  - ただし同じcgroup領域に7ジョブまで同時着弾可能に
    - スケジューラ的には1コアジョブとして管理、同じcgroupに属する7コア分が結果的に混ざる
  - SSDの大半をswapfileとすることで、メモリ量の見せかけの制約を緩和
  - ノード共用、swapfileに見られるように「**性能度外視**」キュー
- のちに、インタラクティブジョブのみ別のスケジューラサーバに移動
  - ジョブ数が多くなりすぎ、qrsh(インタラクティブジョブ投入)の実行からジョブ開始に1分ぐらいかかるようになってしまっていたので、負荷を「逃がした」
  - TSUBAME2に較べサーバ1台の仕事量は5倍ぐらい
    - T2: 400ノード/サーバ → T3: 4(資源分割)x540ノード/サーバ

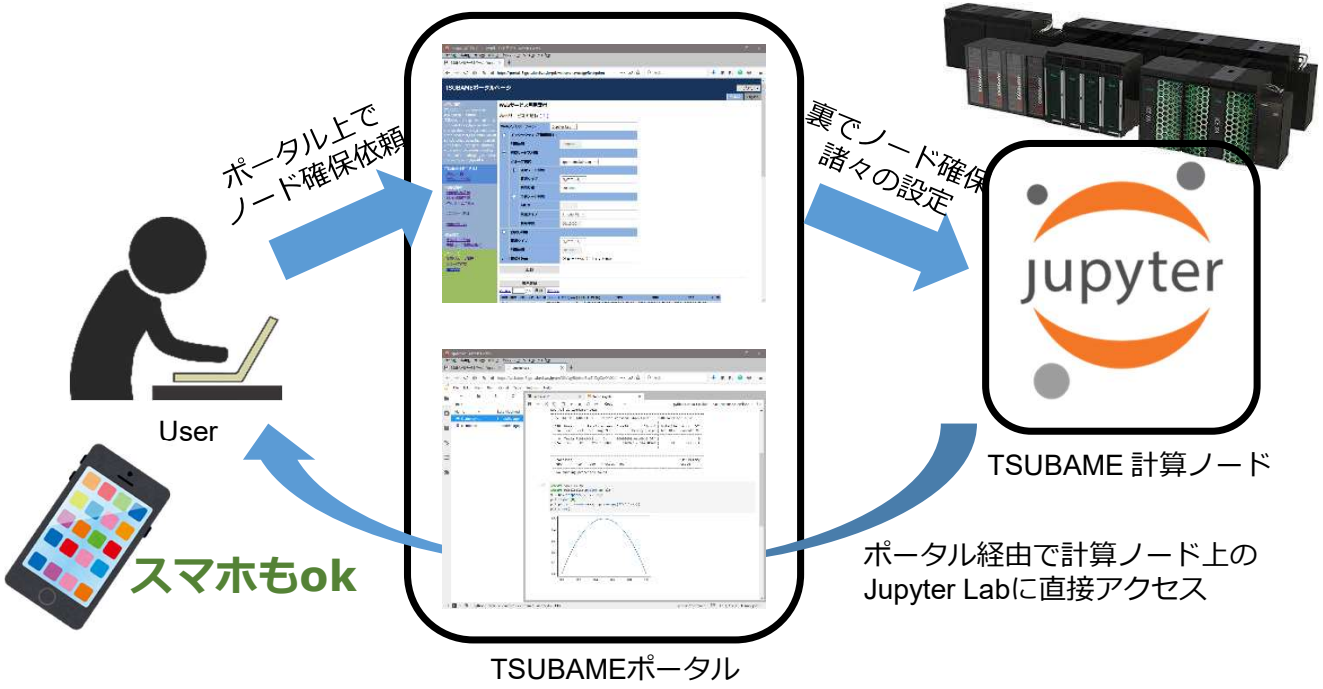
スパコン初学者がTSUBAMEを使うまで

- スパコンの計算ノードで何かをするために必要な知識
  - **SSHおよび公開鍵認証** ← 教育コスト超高 Zoomでトラブルシューティングできますか?
    - X転送・ポートフォワーディング (可視化・Jupyterなどを利用する場合)
  - Linuxの基本的なコマンド
  - ジョブスケジューラの概念・**ジョブスクリプトの書き方**
  - アクセラレータ(GPU)や並列処理(MPI, OpenMP, OpenACC)
  - ドメイン・実際に動かすアプリの知識
  - 本質的には、オレンジ色の箇所の知識しか要らないはず
    - あるには越したことないが、限られた時間で学ぶには余計
- **上記青色の知識がなくても、初心者がTSUBAMEの計算ノードを利用できるようになるといいな**

2020/4に導入したJupyterLab 環境 on TSUBAME3

ユーザは、東工大ポータル⇒TSUBAMEポータル(利用者サイト)からWebアプリケーション(Jupyter Lab)を直接起動

- ブラウザだけでTSUBAMEを直接利用できる
- さらに、東工大所属者は無料で1GPU、64GBメモリまで利用可能



認証は東工大ポータルに入る時の一回だけ

この図のどこにもSSH・公開鍵は出てこない→ 覚える必要がない

※学外者もTSUBAMEポータルへの直接ログイン(メール認証等)でok

さて、今の世の中は?

- Open OnDemand(OOD)では? と思った方がいるかも?
  - OOD自体の紹介は割愛
    - 誰かが話すだろうと思ってスライド用意してなかった
    - Ohioで開発、前ページみたいな機能を実現するOSS
  - 2020年時点で我々は存在を知らなかった
    - あとで調べるとSCのBoFとかで当時から宣伝されてた
  - その後、R-CCS中尾さん他の努力で知名度向上中?
- TSUBAME3で我々がやったことは、車輪の再発明
  - 車輪を作り直すことで理解できたことも多々あるけど
- TSUBAME4では、作りこみを捨ててOODに乗り換え
  - ただ、Fugaku OnDemandほどしっかりした作りこみはしていない
  - 最低限(Jupyter Lab, X)から始めて、余裕が出てきたら取り込みたい
    - 幸いにして、富岳の作りこみ部分はOSSとして公開してくれている

おまけ: 運用の裏側で使ってるOSS

- システムに直結しておらず、ユーザが直接触らない場所ではOSSを活用
  - Redmine: 課題管理
    - ベンダーとの連絡も原則全部ここで
    - 昔は全部メールベース、忘れられた課題多数
      - Redmineになっても未解決チケットがなくなったわけではないけど
  - mkDocs: マニュアル管理 ← かなりおすすめ
    - Markdownで書いたものをいい感じにHTML/PDF化
      - cf) MS Wordで書いたマニュアルPDFからコピペしようとしたら…
      - PDF化はプラグインで対応 (mkdocs-with-pdf)
    - ABCIのドキュメントシステムがいい感じだったのでぱくりました
      - Wordベースのマニュアルを移植し、移行
      - Markdownのほうが表現力は劣るけど、その範囲で書けばいいだけ
    - Git管理にして、ベンダーと大学側両方がいじれるようにした
      - 「マニュアルの○○の場所直して」というより、差分をpushした方が速い
  - Drupal: Webサイト・問い合わせフォーム
  - Zabbix: システム監視
    - ベンダーの監視系と独立で簡易的な監視系を構築
    - そろそろPrometheusとかGrafanaとかで作り直す時期か?
- この辺は、「教員少数名が設計して教員が構築、教員が運用」...
  - こけても困るのはごく少数名なので、手厚いサポートは不要
  - **自分の仕事を減らすため**というモチベーション

スパコンセンターにとってのOSS・プロプラ選択基準(私見/再掲)

	OSS	作りこみ	プロプライエタリ
新しいものへのキャッチアップ	◎ 予算を立てなくても お試しで入れられる	○ 簡単に作れるなら やる価値はある	× 運用中に新たに 予算を割くのは困難 →数年に1回
サポート (何か起こったときに泣きつく先があるか)	△ 外部のサポートを買うと プロプラと同レベル支出	○ 自分で作ったらなら 何か起こっても対応できる? (対応できない規模のを作らない)	◎ 導入業者としては 訊く先があるのは安心
ソフトウェア資産として残るか?	◎ 他所のスパコンなどにも 持ち出すことができる	×～○ 作り方次第 特定のシステムべったりだと 他所では使えない	×? 他所で同じことを やろうとしたら 要車輪の再発明?
手元での再現	◎ 研究室マシンで利用可	△ 研究室マシンで要再構成	× ライセンスとか別途必要

実際はきれいに分かれるわけではなく、○○**+作りこみということは往々にして起こる**

- 結論: 適材適所
  - 止まると全員が困るもの(スケジューラとかOS本体・ストレージ)は**サポートがあるもの**が必須
  - 止まっても全員が困るわけではないものはOSS活用で**新技術の恩恵**を
  - システムに近いもの(コンテナランタイムとか)を入れる場合、
いずれにせよそのソフトが何をやっているのかは分かってないと入れられない

スパコンセンターにとってのOSS・プロプラ選択基準(私見/再掲)

	OSS	作りこみ	プロプライエタリ
新しいものへのキャッチアップ	◎ 予算を立てなくても お試しで入れられる	○ 簡単に作れるなら やる価値はある	× 運用中に新たに 予算を割くのは困難 →数年に1回
サポート (何か起こったときに泣きつく先があるか)	△ 外部のサポートを買うと プロプラと同レベル支出	○ 自分で作ったらなら 何か起こっても対応できる? (対応できない規模のを作らない)	◎ 導入業者としては 訊く先があるのは安心
ソフトウェア資産として残るか?	◎ 他所のスパコンなどに 持ち出すことができる	**作りこみ自体は 必要だけど 世の中に還元が必要 (コードに限らず)**	×? 他所で同じことを やろうとしたら 要車輪の再発明?
手元での再現	◎ 研究室マシンで利用可	△ 研究室マシンで要再構成	× ライセンスとか別途必要

実際はきれいに分かれるわけではなく、○○**+作りこみということは往々にして起こる**

- 結論: 適材適所
  - 止まると全員が困るもの(スケジューラとかOS本体・ストレージ)は**サポートがあるもの**が必須
  - 止まっても全員が困るわけではないものはOSS活用で**新技術の恩恵**を
  - システムに近いもの(コンテナランタイムとか)を入れる場合、いずれにせよそのソフトが何をやっているのかは分かってないと入れられない