大島 聡史（名古屋大学 情報基盤センター 准教授）

「不老」の特徴的な機能とベンチマーク結果の紹介　＋　利用状況の報告

名古屋大学
NAGOYA UNIVERSITY

発表の目次

- 主要ベンチマークプログラムによる性能評価
- この二ヶ月の稼働状況・利用状況
- 「不老」の特徴的な機能・便利な機能・利用上の注意
  - クラウドシステム
  - 会話型バッチ（インタラクティブバッチジョブ実行）
  - ストレージ（コールドストレージ、Type IIのローカルSSD）
  - コンテナ対応
- まとめ

発表の目次

- 主要ベンチマークプログラムによる性能評価
- この二ヶ月の稼働状況・利用状況
- 「不老」の特徴的な機能・便利な機能・利用上の注意
  - クラウドシステム
  - 会話型バッチ（インタラクティブバッチジョブ実行）
  - ストレージ（コールドストレージ、Type IIのローカルSSD）
  - コンテナ対応
- まとめ

主要ベンチマークプログラムによる性能評価

- FX1000、Cascade Lake、V100それぞれの性能自体は既知の部分が多いため、同じ問題を各サブシステムで実行した場合の性能など、「不老」ならではの比較結果を紹介する

- CPUとメモリ
  - Stream
  - HPL
  - HPCG
- 通信
  - OSU Micro-Benchmarks
- ストレージ
  - IOR

コンパイラなど（特に断りのない限り）
- Type Iサブシステム：TCS 1.2.26
- Type IIサブシステム：Intel 2019.5.281, CUDA 10.2, PGI 20.4
- クラウドシステム：Intel 2019.5.281

Stream Triad (N=20,000,000)

- Type Iサブシステム
  - 1ノード：826 GB/s
    - コンパイル時オプション -Kfast,openmp,zfill
    - 実行時環境変数 XOS_MMM_L_PAGING_POLICY=demand:demand:demand
    - 特にzfillとdemand設定が大きく影響、CとFortranの差はなし

- Type IIサブシステム
  - 1ノード 2CPU：202 GB/s　(-xHost -qopenmp -qopt-zmm-usage=high)
    - -qopt-streaming-stores は変更しても向上せず（デフォルトはauto）
  - 1ノード 1GPU：777 GB/s　※streamベンチコードにOpenACC指示文を入れて測定

- クラウドシステム
  - 1ノード 4CPU：338 GB/s　（〃）

- A64FXのHBMの速さが良くわかる結果
- 4CPUのクラウドシステムは流石に2CPUのType IIの倍は出ない（もう少し出てくれると嬉しいが……）

HPL

- Type Iサブシステム
  - 1ノード：2.4849 TFLOPS： 2.4849/3.3792=73.5%
  - 全系：6.6178 PFLOPS： 6.6178/7.782=85.0%
    - TOP500 2020年6月版で世界36位・国内5位（「富岳」、ABCI、OFP、TSUBAME3.0の次）
    - メモリサイズが小さめ＝大きな問題を解けないため1ノードの効率は控えめ、大規模では高効率
- Type IIサブシステム（GPU利用）
  - 1ノード：24.440 TFLOPS： 24.440/33.888=72.1%
  - 全系：4.880 PFLOPS（測定時期の都合でTOP500未登録、〃51位相当）： 4.880/7.489=65.2%
- クラウドシステム
  - 1ノード：3.25 TFLOPS：3.250/5.376=60.5%

- GPUの性能の高さと
  A64FXの効率の良さがよくわかる結果

  - Type I, II は富士通による最適化・測定
  - クラウドはIntelコンパイラ2019.5.281（のMKL）に付属のmp_linpack

HPCG

- Type Iサブシステム
  - 1ノード：105.956 GFLOPS： 105.956/3379.2=3.14%
  - 全系：230.594 TFLOPS： 230.594/7782=2.96%
    - HPCG 2020年6月版で世界16位・国内4位、「不老」以上の性能で2%以上は「富岳」の2.6%のみ
    - 富士通による最適化・測定
- Type IIサブシステム（GPU利用）
  - 1ノード：550.6 GFLOPS： 550.6/33888=1.62%
  - 100ノード：48.2544 TFLOPS： 48.2544/33888=1.42%
    - 全系では100 TFLOPS程度か？
    - HPCG Webサイトにて配布されているHPCG3.1バイナリ(2019.12.05版)で測定、60秒試行
- クラウドシステム
  - 1ノード 1CPU（1MPIプロセス）：16.6079 GFLOPS： 16.6079/1344=1.24%
  - 1ノード 4CPU（4MPIプロセス）：61.7766 GFLOPS： 61.7766/5376=1.15%
    - Intelコンパイラ2019.5.281（のMKL）に付属のベンチマークにて測定、60秒試行

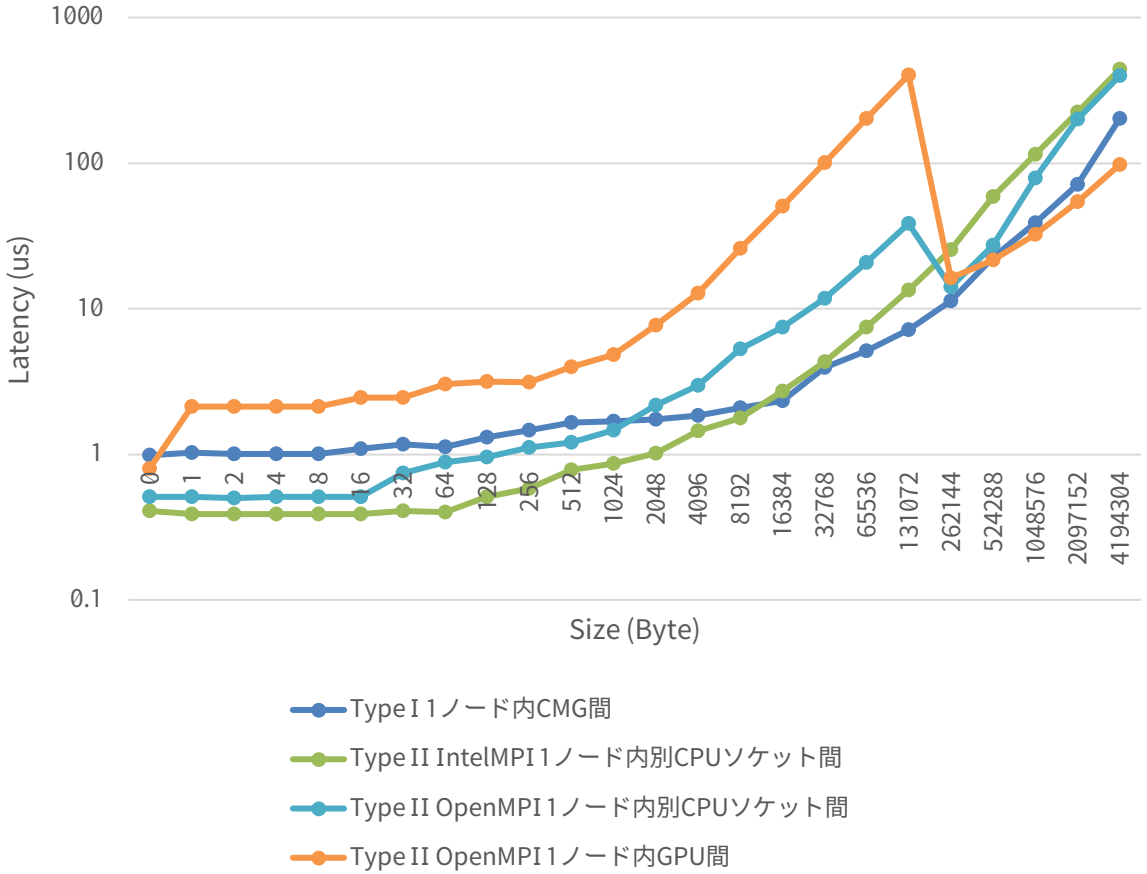
- HPL同様に、GPUの性能の高さと A64FXの効率の良さがよくわかる結果

OSU Micro-Benchmarks

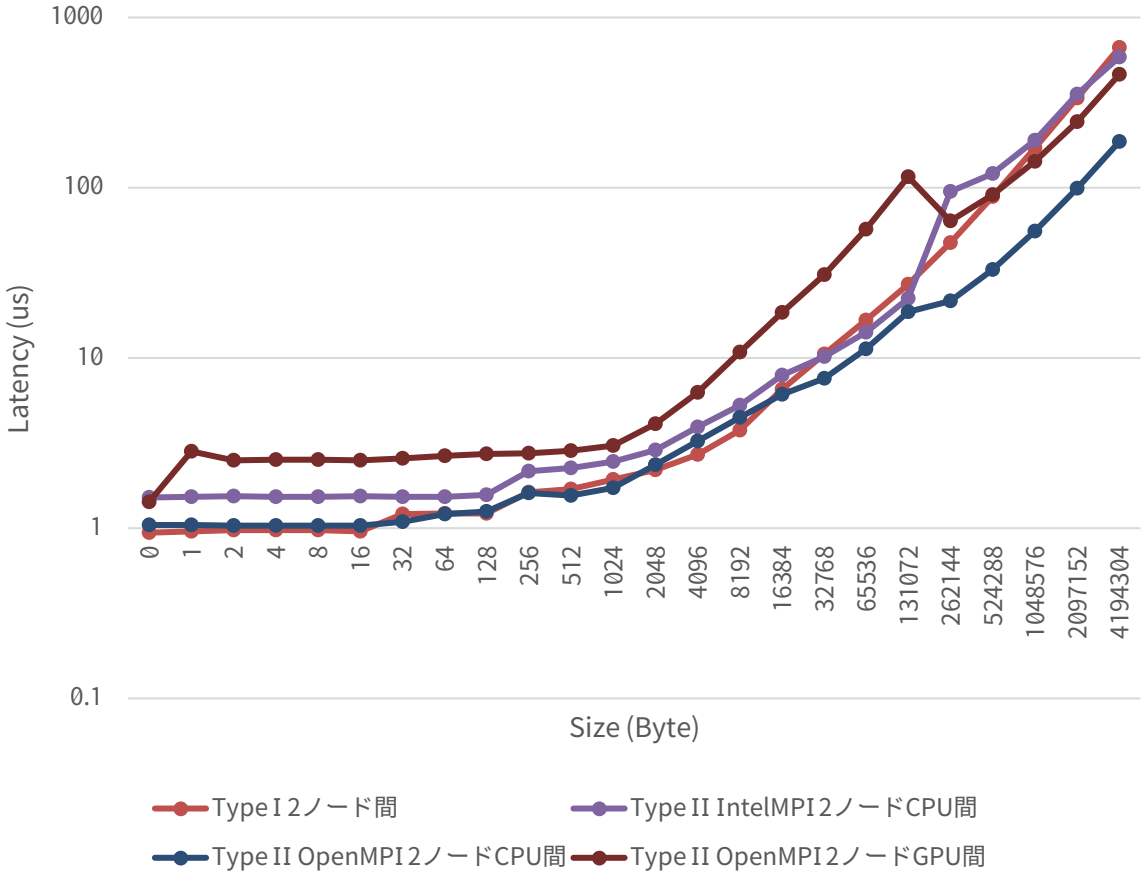
- 測定項目
  - Type IとType IIの通信性能を比較
    - latency
    - allreduce, alltoall

OSU Micro-Benchmarks：latency

- サイズ小
  - Type IIのCPUソケット間、特にOpenMPIが速い
- サイズ大
  - Type IのCMG間が速い、GPU間も速い

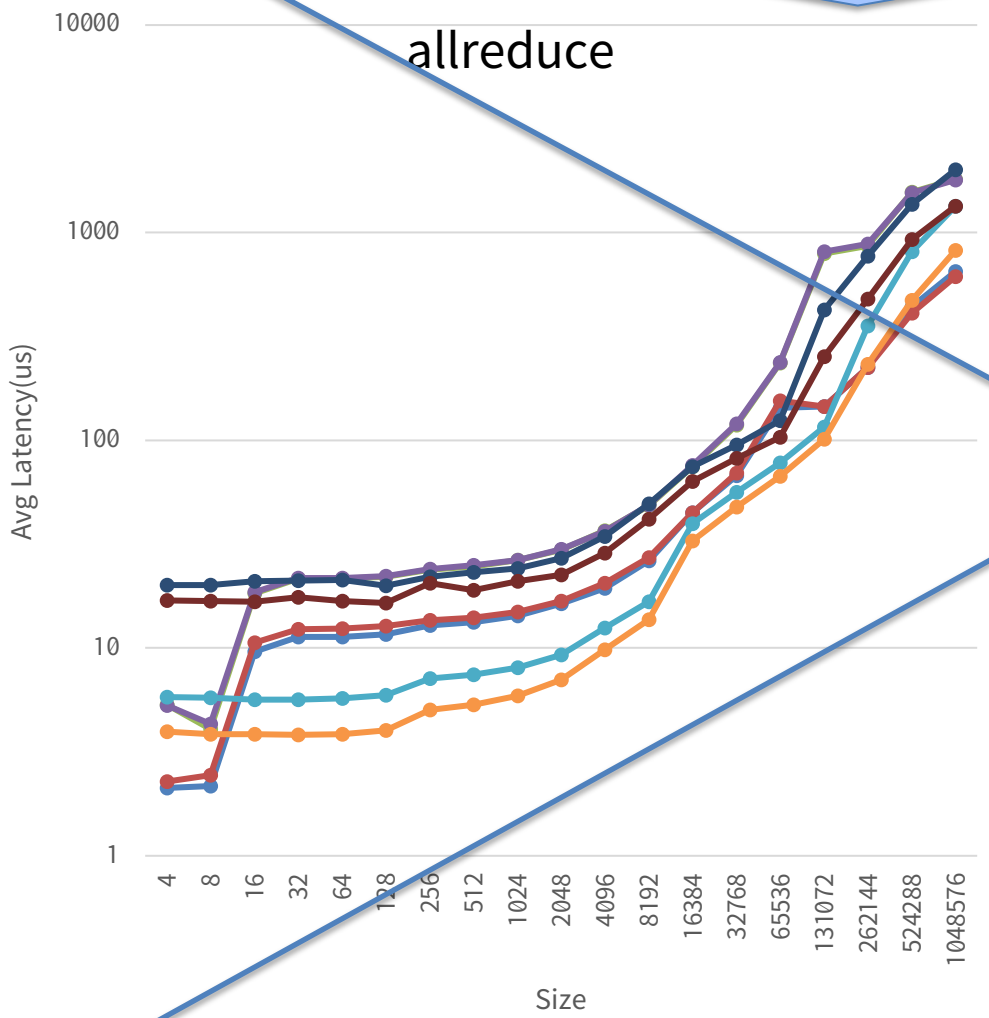


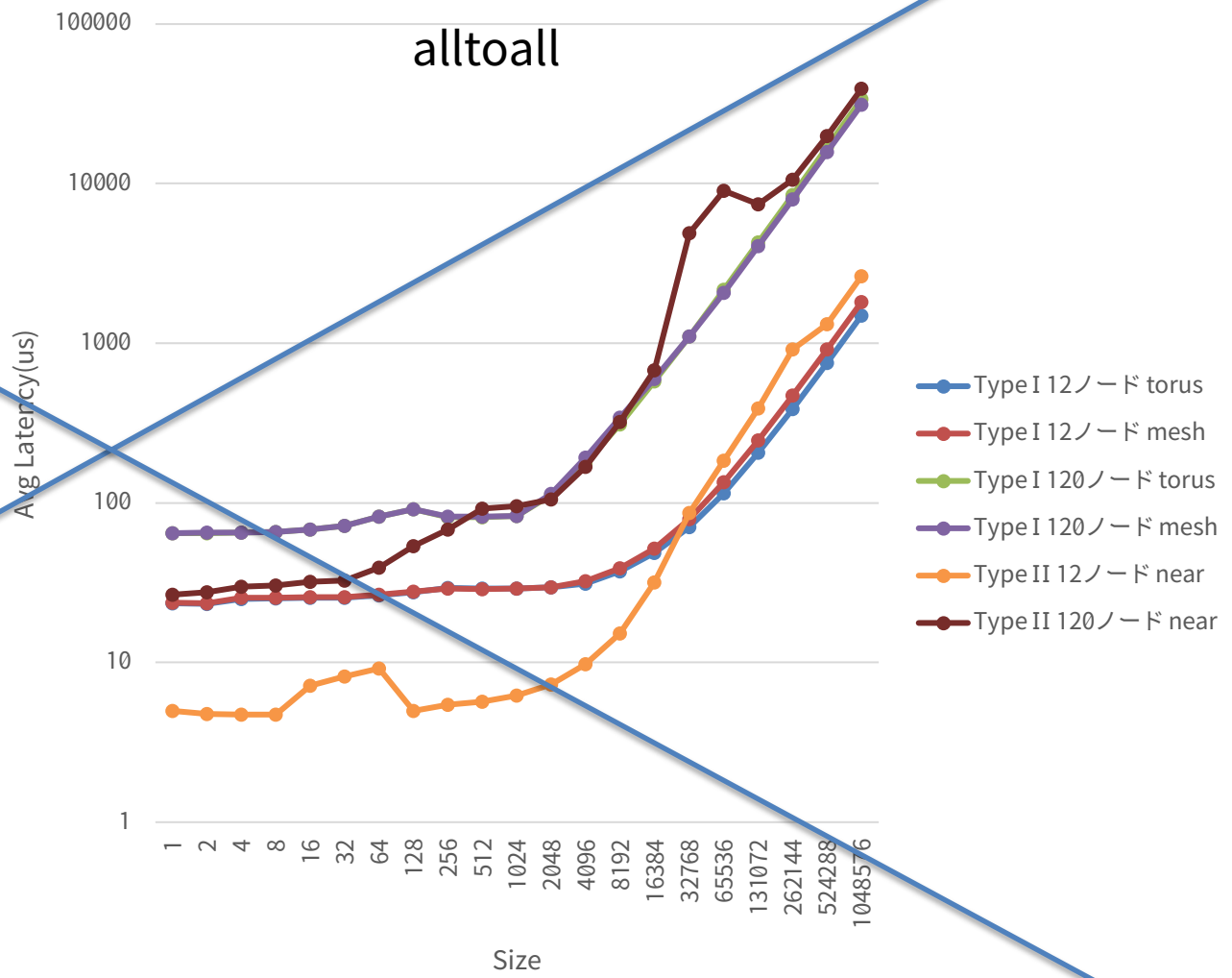
- Type IIのOpenMPIが速い
- Type Iもサイズ小では速いが、サイズ大はやや遅い
- GPU間は他より遅い



OSU Micro-Benc

発表時に古いグラフを表示してしまいました。
次のページが最新版です。

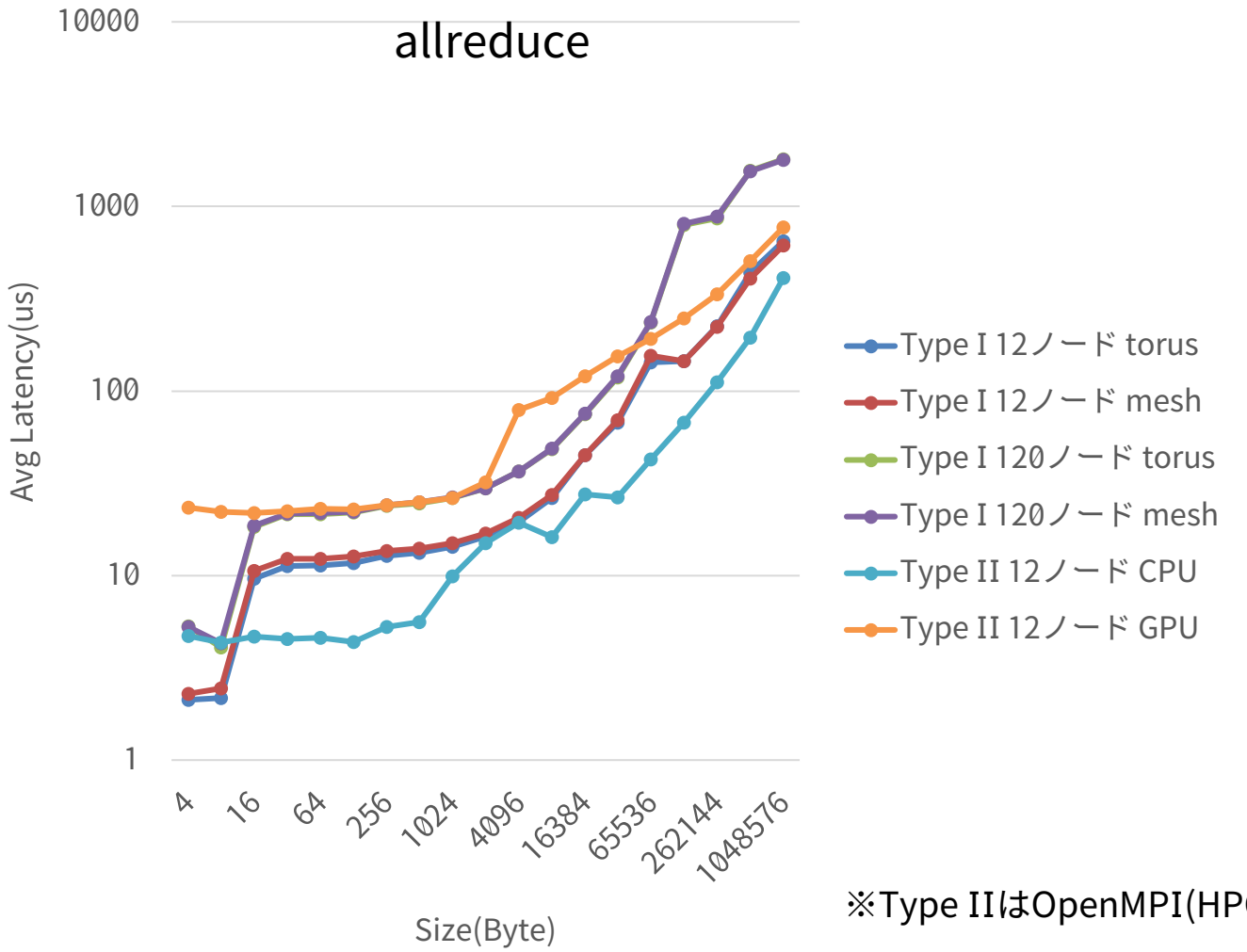




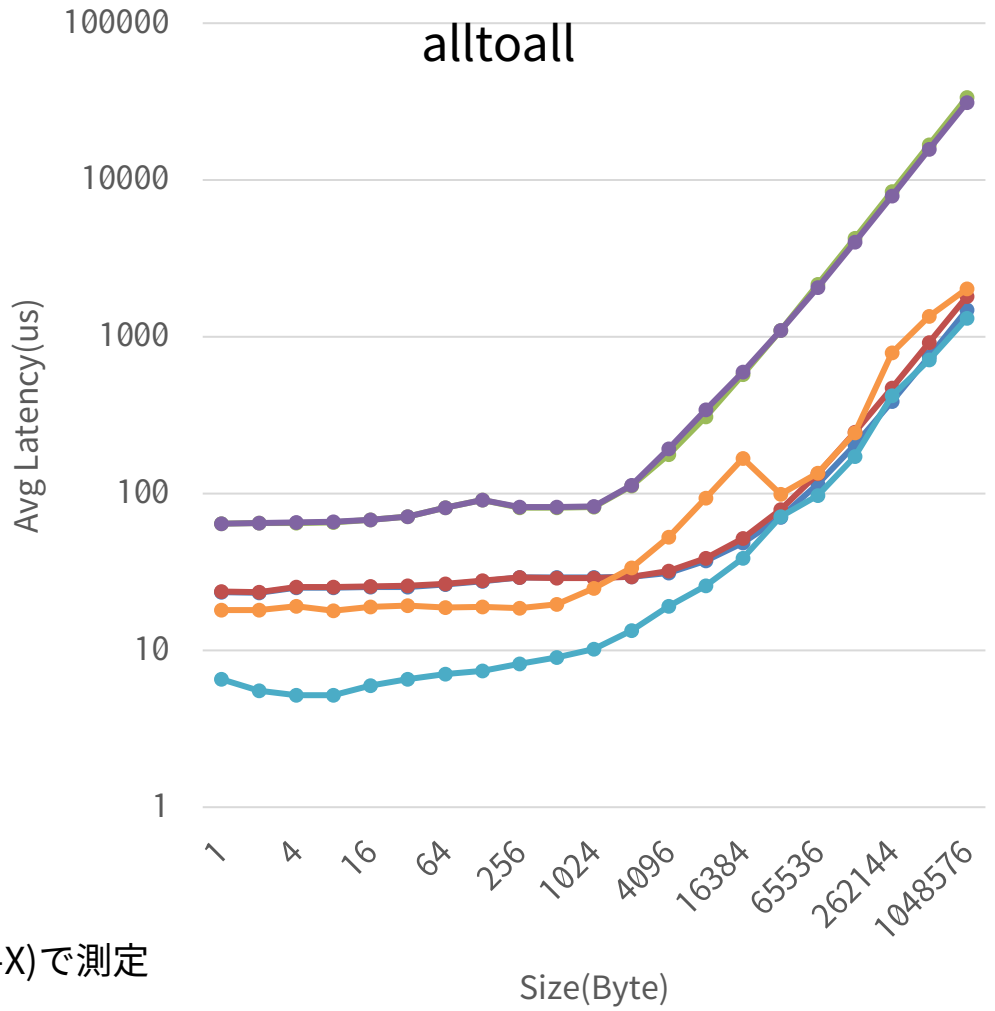
- 全体的にはType IIの方が低レイテンシの傾向
- Type Iについてはより多くのノードでも実験するべきであったか

- GPU to GPUの通信性能はパラメタ等精査中……

OSU Micro-Benchmarks：allreduce, alltoall



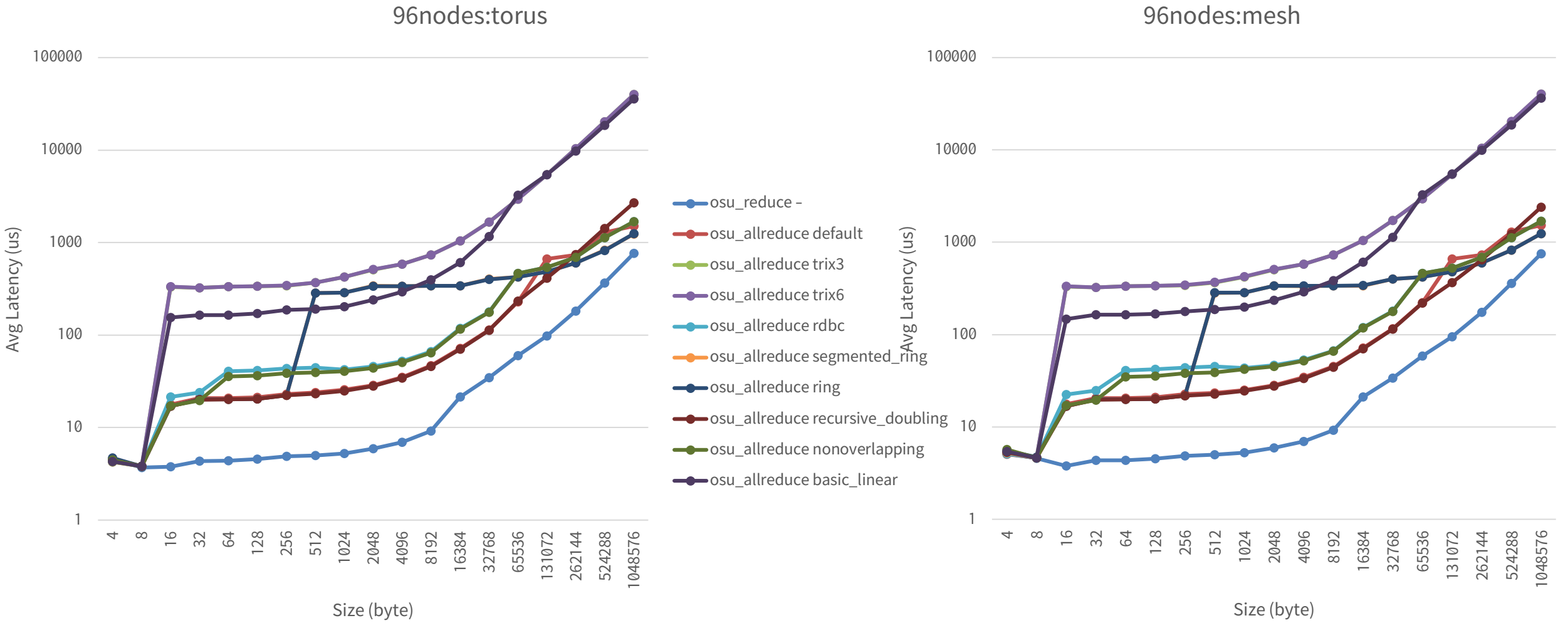
※Type IIはOpenMPI(HPC-X)で測定



- 全体的にはType IIの方が低レイテンシの傾向
- Type Iについてはより多くのノードでも実験するべきであったか

- GPU to GPUの通信性能はパラメタ等精査中……

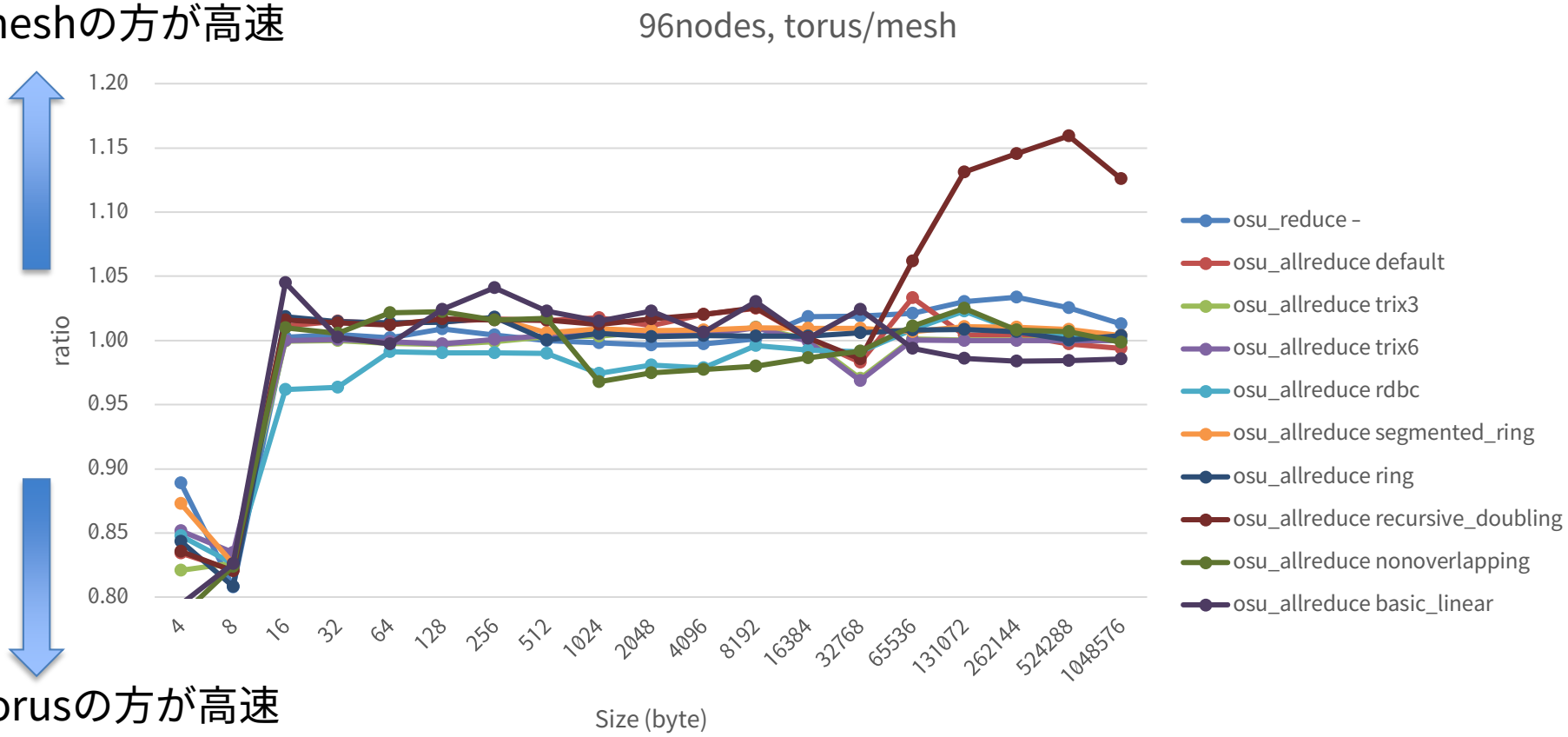
Type Iのallreduceアルゴリズム選択実験：96ノード



- グラフが重なっていてわかりにくいが、デフォルト（未指定）でほぼ最速、ただし128KBが若干遅い

Type Iのallreduceアルゴリズム選択実験：96ノード、torus対mesh

meshの方が高速

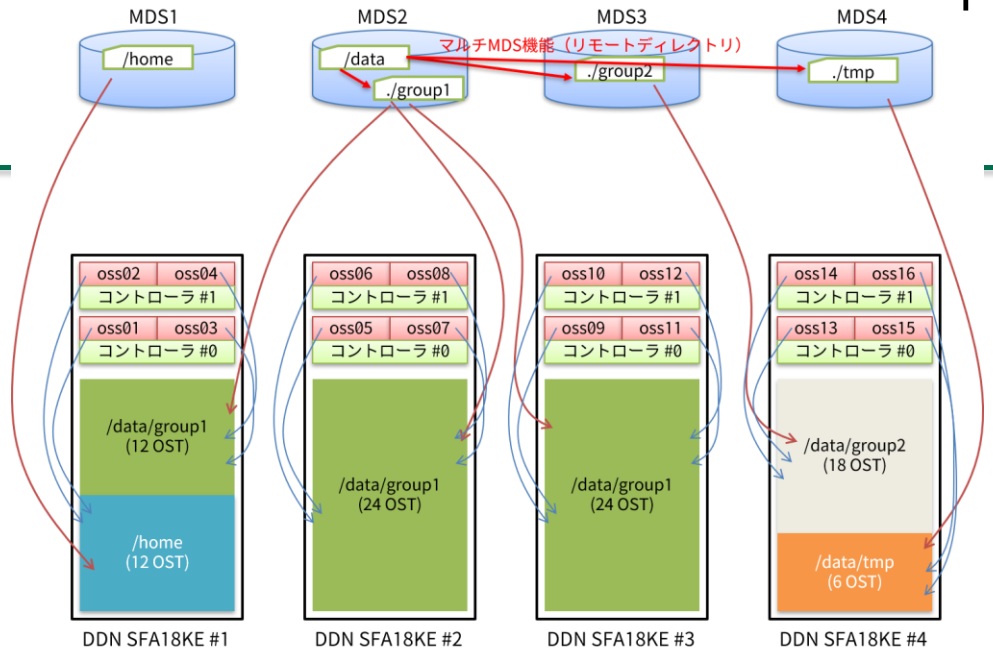


torusの方が高速

- 4Byte, 8Byteはtorusの方が明らかに高速、16byte以上ではあまり変わらない（±5%）
- recursive_doublingだけはサイズが大きな時にmeshが高速

ストレージ性能（IOR） 1/2

- IORベンチマーク性能（サービス開始直前）
  - Type Iサブシステムから測定
  - 一般的なバッチジョブでの利用を推奨しているのは /data/group1
    - File Per ProcessでR/Wともに100GB/s前後
    - Single Shared FileでR/Wともに80GB/s強



		File Per Process 性能		Single Shared File 性能	
操作対象	OST	write 平均値	read 平均値	write 平均値	read 平均値
ディレクトリ	クライアント数	[MiB/sec]	[MiB/sec]	[MiB/sec]	[MiB/sec]
/data/group1	576	93130.36	106612.15	84112.94	83286.16
/data/group2	288	33770.22	38685.99	32984.10	36369.85
/home	144	17928.59	22037.02		
/data/tmp	144	12697.28	13242.41		
		write+read 合計値	169052.000	write+read 合計値	118381.525

ストレージ性能（IOR） 2/2

- サービス開始後（8月）、資源が空いているときにあまり大きくない規模で測定してみた
  - /data/group1配下、mpiexec ./ior -w -r -C -g -i 3 -vv -s 13000 -b 1m -t 1m

  - Type I, 144ノード576プロセス, POSIX
    - なぜか200MiB/sec以下しか出なくてなかなか終わらず諦めた
  - Type I, 144ノード576プロセス, MPI-IO
    - Max Write 1002.35 MiB/sec
    - Max Read 908.24 MiB/sec

  （理由は不明、問題設定が悪かったのか？）

  - Type II, 15ノード120プロセス, POSIX
    - Max Write 1048.00 MiB/sec
    - Max Read 391.36 MiB/sec
  - Type II, 15ノード120プロセス, MPI-IO
    - Max Write 918.37 MiB/sec
    - Max Read 345.59 MiB/sec

発表の目次

- 主要ベンチマークプログラムによる性能評価
- この二ヶ月の稼働状況・利用状況
- 「不老」の特徴的な機能・便利な機能・利用上の注意
  - クラウドシステム
  - 会話型バッチ（インタラクティブバッチジョブ実行）
  - ストレージ（コールドストレージ、Type IIのローカルSSD）
  - コンテナ対応
- まとめ

稼働開始から二ヶ月の運用状況

- 運用初期は重大な障害があったり再起動をともなう設定変更があったりする可能性もあり利用者に迷惑をかけるかも知れない　→　7月の間だけは無料利用可能とした
  - 一般利用・グループ利用は、費用は支払ってもらうがポイントの減算はしない、という運用
- 結果的に全系障害など致命的な問題はまったく発生せず、HW的には安定運用
  - SW的には色々と不備・不足があり対応を進めている
- 真夏の縮退運転は行われなかった
  - 大学側から要請があった場合には*-extraリソースグループを止めることで対応することを計画
  - 実際には要請はなく、止める必要はなかった
    - サブシステムにもよるが、利用率がそこまで高くはなかった
    - 利用率を考慮しても、思ったより消費電力が低かった
    - テレワーク推奨になって大学自体の電力需要が逼迫しなかった
    - などが要因として考えられる

各サブシステムの利用状況・混雑具合

- 「→」の前は7月の状況
- 「→」以降は8月の状況

- Type Iサブシステム
  - 全然流れない状況まではいかないものの、タイミングによっては利用者多数
  - →　タイミング次第、空きノードが多いこともあるし混んでいることもある
- Type IIサブシステム
  - 数ノードのジョブに1日待つほど混雑した日もあり
    - 各ジョブがどの程度GPUを使っていたのかは不明
  - →　半分以上空いていることが多い、V100をぶん回すチャンス！
- Type IIIサブシステム
  - 限定的な利用（そもそもノード数が少ない）
- クラウドシステム（バッチ）
  - 大混雑、ノード使用率100%となっていることも
  - →　相変わらずの大混雑、対策が必要（UNCAI用のノードを減らすことを検討中）
- クラウドシステム（UNCAI）
  - ほとんど使われていない　→　ほとんど使われていない

これからの運用で取り組む課題

- まだ性能や使い勝手の調査が足りていない機能
  - 機械学習の性能（Type I vs Type II）
  - Type IIサブシステムのローカルストレージの積極的な活用（特にNVMesh）
  - インストールはしてあるが性能が妥当なのかを確認できていないアプリも多い
- システム利用率の改善
  - Type IIサブシステムの利用率向上
    - これまでスパコンを使ってこなかった様々な分野にいると思われる機械学習ユーザの誘致
  - クラウドシステムのバッチジョブの混雑緩和
  - クラウドシステムのUNCAIの利用率向上
    - 1万円で1ソケットを約58日使える、手元（研究室など）のLinuxサーバの代わりに使うと便利だと思うが利用率は大変低い状況……
    - バッチジョブが混雑している間は戦略的に放置するべきかもしれない
- その他
  - コールドストレージ（2月のアップグレード後が本番）

発表の目次

- 主要ベンチマークプログラムによる性能評価
- この二ヶ月の稼働状況・利用状況
- 「不老」の特徴的な機能・便利な機能・利用上の注意
  - クラウドシステム
  - 会話型バッチ（インタラクティブバッチジョブ実行）
  - ストレージ（コールドストレージ、Type IIのローカルSSD）
  - コンテナ対応
- まとめ

クラウドシステム

- クラウドシステムは他のサブシステムと同様のバッチジョブ実行とブラウザで予約する時刻指定実行に対応
  - 2種類の時刻指定実行
    - 時刻指定バッチジョブ
      - 指定時刻になると登録しておいたスクリプトが起動
    - 時刻指定インタラクティブ実行
      - 指定時刻になると仮想マシンが起動、sshログインして利用
  - 時刻指定実行には富士通のWebシステムUNCAIを利用

会話型バッチ（インタラクティブバッチジョブ実行）

- クラウドシステムで利用可能な「時刻指定インタラクティブ実行」とは別に、Type Iシステム、Type IIシステム、クラウドシステムでは「会話型バッチ」が利用可能
  - 対象リソースグループ：fx-interactive, cx-interactive, cxgfs-interactive, cl-interactive
- バッチジョブ実行時に対応するリソースグループを指定することで利用可能（専用オプションの追加も必要）
  - $ pjsub --interact -L rscgrp=cx-interactive
  - デフォルトでは1時間で打ち切り、オプションを付加すれば延ばすことが可能
  - $ pjsub --interact -L rscgrp=cx-interactive,elapse=2:00:00
- 利用ポイントの消費はバッチジョブ扱いとなる
  - ログインノードは（特に並列実行時の）利用ポイントの消費が大きいため、会話型バッチをうまく活用してください

コールドストレージの仕様と特徴（再掲）

フェーズ1: 2020年7月1日より稼働開始

機種名	PetaSite Library
総スロット数 （最大搭載可能カートリッジ数）	88巻
総物理容量 / 最大搭載可能容量	484 TB / 484 TB
総ドライブ数	6
ODAサーバ数	1

フェーズ2: 2021年2月1日より稼働開始予定

機種名	PetaSite拡張型 Library
総スロット数 (最大搭載可能カートリッジ数)	1,980巻
総物理容量 / 最大搭載可能容量	6 PB / 10.89 PB
総ドライブ数	20
ODAサーバ数	4

- １度書き込み（追記）のみの光ディスクストレージ
- 実験データ等の長期データ保存用
- 理論上100年データ保持可能
- 水にぬれても読み出せる
- サービス終了後ユーザに光ディスクを返却

コールドストレージの運用方針（課金方法と利用方法）

- 追記型の光ディスクのため買い切り（売り切り）での運用が必要
- 課金方式：光ディスク10枚（50TB相当）を一口として、初期費用と年間経費を徴収
  - ファイル負担経費（初回利用時のみ必要）：一口190,000円
  - ファイル管理経費（毎年必要、基本負担金とは別）：一口10,000円
  - メディア持ち込み利用の場合は管理経費のみ必要
- 利用方法
  - フロントエンド（ログインノード）からコマンド操作
  - オンサイト利用装置からコマンド操作
  - 画像処理装置から専用クライアントで操作

ローカルストレージの活用

- Type IIサブシステムと Type IIIサブシステムはユーザ用のローカルストレージを備える
  - Type II：NVMe SSD 6.4 TB/node、一部ノードにてBeeGFS/BeeOND/NVMeshを提供
  - Type III：NVMe SSD、Fibre Channel接続のローカルストレージ

- Type IIのローカルストレージについて
  - リソースグループによって使えるものに違いがあり、少々わかりにくい
    - NVMe SSDがそのまま使える
      - ローカルに接続されたSSDが使える、パスはPJM_LOCALDIR環境変数を参照
    - BeeONDが使える
      - ジョブ実行時にPJM_BEEOND=1オプションを付けておくと、PJM_BEEONDDIR環境変数にノード間で共通のディレクトリが見える
      - ジョブ終了時に消えるため、データの回収には場合はバッチジョブ内でのファイル操作が必要
    - NVMeshが使える
      - cxgfs系のリソースグループではNVMeshが利用可能、ジョブが終了してもデータは残る、申請制
  - ローカルストレージの性能については本発表終盤のベンチマーク情報を参照

Webに掲載しているジョブクラス一覧を更新しました

リソース グループ名	最大 ノード数	最大 CPUコア数	最長実行時間 (デフォルト値)	最長実行時間 (最大値)	最大 メモリ容量 (*)	ローカルストレージ			備考
						NVMeSSD 6.4TB 利用可能	BeeOND 利用可能	BeeGFS(NVMesh) 利用可能(申請制)	
cx-interactive	1	1	1時間	24時間	338 GiB	○			会話型バッチ
cx-debug	4	160	1時間	1時間	338 GiB x 4	○			短時間デバッグ用、準占有利用ノードと共有
cx-extra	8	320	1時間	12時間	338 GiB x 8	○			12時間ノード縮退運転用
cx-share	1/4(共有)	10	1時間	168時間	84 GiB	○(共有)			ノード共有(**)、資源を1/4に分割
cx-single	1	40	1時間	336時間	338 GiB x 1	○			
cx-small	8	320	1時間	168時間	338 GiB x 8	○	○		
cx-middle	16	1,280	1時間	72時間	338 GiB x 16	○	○		
cx-large	64	3,840	1時間	72時間	338 GiB x 64	○	○		
cx-special	221	8,840	unlimited	unlimited	338 GiB x 221	○			事前予約制 (†)
cx-middle2	16	1,280	1時間	72時間	338 GiB x 16	○	○		実行優先度強化型(‡)
cxgfs-interactive	1	40	1時間	168時間	338 GiB x 1			○	会話型バッチ
cxgfs-single	1	40	1時間	336時間	338 GiB x 1			○	
cxgfs-small	8	320	1時間	168時間	338 GiB x 8			○	
cxgfs-middle	16	1,280	1時間	72時間	338 GiB x 16			○	
cxgfs-special	50	2,000	1時間	72時間	338 GiB x 50			○	事前予約制 (†)
準占有利用	(契約次第)	(契約次第)	unlimited	unlimited	338 GiB x 実行ノード数	○	要相談	要相談	

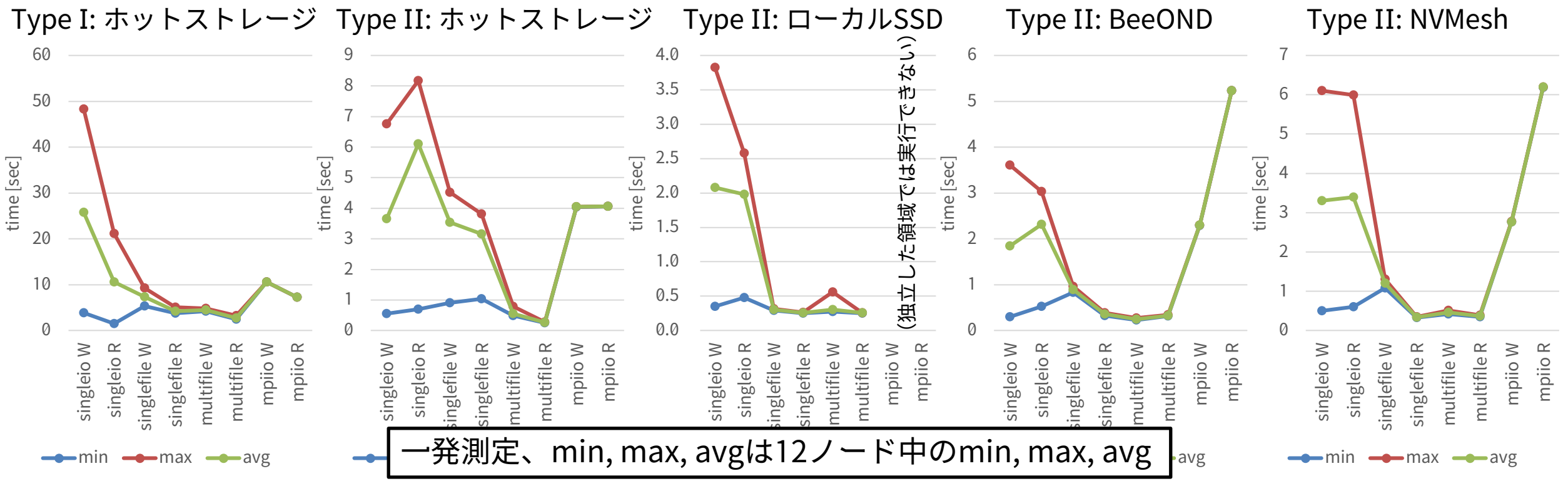
(少しはわかりやすくなったか……？)

ストレージ性能（自作のテストプログラム）

- 実際の使い方を想定したシンプルなテストプログラムを作成し、運用中に測定してみた
- プログラム内容（いずれもファイルのopen/close時間を含む）
  1. singleio：マスタープロセスのみがファイル操作、配付と集約はMPI通信
     - ノード数が増えた場合に全プロセス分を持てないことを想定し、逐次的に1対1通信
  2. singlefile：全プロセスがfread/fwriteで1ファイルを読み書き
  3. multifile：全プロセスが個別の1ファイルをfread/fwrite
  4. mpiio：MPI-IOで1ファイルを読み書き（MPI_File_set_view, MPI_File_write/read_all)
- 問題設定（プロセス数と容量）
  - 12プロセス、1プロセス/1ノード（Tofu-Dを考慮（12:mesh）、比較のためType IIも12ノード）
  - {1M,10M,100M,1G}elements/processで測定、今回は100M版を提示（10M以上は同様の傾向）
  - データ型は全てdouble型（8byte）
    - 100M elements/processはプロセス当たり800MB、1秒で終了すれば9600MB/sec相当
- githubに公開してあるため興味がある人はご自由にお試しください
  - https://github.com/exthnet/iotest

※1秒=9600MB/sec

テスト結果：12ノード、1プロセス/1ノード、プロセスあたり100M要素



一発測定、min, max, avgは12ノード中のmin, max, avg

- multifileが良い性能：max値、左から順にW/R=4.82/3.27, 0.79/0.28, 0.56/0.26, 0.27/0.34, 0.52/0.39
  - 実はType IがType IIよりかなり遅い、ただしホットストレージは計算ノードとストレージ両方の負荷の影響を受けるため測定タイミングの影響もあるかもしれない？
- mpiioの性能が今ひとつ、特にBeeOND・NVMeshが遅い
  - 東工大TSUBAME3でも試してもらったが、同様にBeeONDのmpiioは遅い。使い方があっていないようだ。
    - （野村さん@東工大 ご協力ありがとうございます）
- SSDによる劇的な性能向上は観測できていないが、singlefileでも高性能、他のジョブの影響も受けない利点有。

コンテナ対応

- Type IIサブシステムではSingularityが利用可能
  - Dockerコンテナの利用が可能
  - もちろんコンテナからGPUも利用可能
  - 従来のスパコンユーザ以外にも使ってもらいたい
- どのような手順が必要なのか、どのくらい簡単に使えるのかがうまく伝えられていないと思われるので、本資料で具体的な手順を紹介する
  - もちろん、Webにも掲載して伝えていく
- コンテナ利用上の基礎知識
  - `module load singularity`でsingularityコマンドが利用可能になる
  - 「不老」の計算ノードは外部ネットワークに接続できない構成となっているが、ジョブ実行時にjobenv=singularityオプションを追加することで接続が可能になる
    - 外部のリポジトリからコンテナイメージを拾ってくることが可能になる
  - --nvオプションを指定すれば計算ノード上のGPUを利用可能
  - 会話型バッチ（cx-interactive、cxgfs-interactiveリソースグループ）を使えば対話型でのコンテナ実行も可能

訂正
Type IIサブシステムはjobenv指定無しでも外部ネットワークアクセス可能でした（jobenv指定がないとSingularityを利用する際に困るのは変わりません）

コンテナ利用例

- Dockerコンテナを拾ってきて、TensorFlowのMNISTサンプルをGPUを使って実行する
  - サンプルプログラム（mnist_sample.py）

```
import tensorflow as tf

mnist = tf.keras.datasets.mnist

(x_train, y_train),(x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(512, activation=tf.nn.relu),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test)
```

コンテナ利用例：会話型バッチ、MNISTサンプル 1/4

- コンテナの準備
  - 赤文字は実行したコマンド、それ以外はコマンドによって得られた出力

```
$ pjsub --interact -L rscgrp=cx-interactive,jobenv=singularity
[INFO] PJM 0000 pjsub Job 24138 submitted.
[INFO] PJM 0081 .connected.
[INFO] PJM 0082 pjsub Interactive job 24138 started.
$ module load singularity
$ singularity shell docker://tensorflow/tensorflow:latest-gpu
INFO:    Converting OCI blobs to SIF format
INFO:    Starting build...
Getting image source signatures
Copying blob 7ddbc47eeb70 done
省略
Copying blob 43988fbe2254 done
Copying blob 019211ded66e done
Copying config 7815e5bf64 done
Writing manifest to image destination
Storing signatures
2020/08/29 15:41:12  info unpack layer: sha256:7ddbc47eeb70dc7f08e410a6667948b87ff3883024eb41478b44ef9a81bf400c
省略
2020/08/29 15:41:41  info unpack layer: sha256:019211ded66e3878a78e08489b7edfc9be46737f1646bb89cb5cac7688d77456
INFO:    Creating SIF file...
Singularity>
```

←会話型バッチを起動、jobenv=singularityオプションを付けておく

←dockerリポジトリからコンテナを取得
jobenv=singularityオプションが付いていないとここ（Creating SIF file...の直後）で
ERROR : Failed to create mount namespace: Operation not permitted
エラーが出てしまう

←コンテナ起動完了

コンテナ利用例：会話型バッチ、MNISTサンプル 2/4

- MNISTのサンプルを実行

```
Singularity> python ./mnist_sample.py
2020-08-29 15:44:22.593081: I tensorflow/stream_executor/platform/default/dso_loader.cc:48] Successfully opened dynamic library libcudart.so.10.1
2020-08-29 15:44:25.566391: I tensorflow/stream_executor/platform/default/dso_loader.cc:48] Successfully opened dynamic library libcuda.so.1
2020-08-29 15:44:25.566468: E tensorflow/stream_executor/cuda/cuda_driver.cc:314] failed call to cuInit: UNKNOWN ERROR (-1)
2020-08-29 15:44:25.566512: I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:169] retrieving CUDA diagnostic information for host: cx110
2020-08-29 15:44:25.566522: I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:176] hostname: cx110
2020-08-29 15:44:25.566596: I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:200] libcuda reported version is: Not found: was unable to find
libcuda.so DSO loaded into this program
2020-08-29 15:44:25.566637: I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:204] kernel reported version is: 440.64.0
2020-08-29 15:44:25.566951: I tensorflow/core/platform/cpu_feature_guard.cc:142] This TensorFlow binary is optimized with oneAPI Deep Neural Network
Library (oneDNN)to use the following CPU instructions in performance-critical operations:  AVX2 AVX512F FMA
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
2020-08-29 15:44:25.588987: I tensorflow/core/platform/profile_utils/cpu_utils.cc:104] CPU Frequency: 2100000000 Hz
2020-08-29 15:44:25.591255: I tensorflow/compiler/xla/service/service.cc:168] XLA service 0x5de8470 initialized for platform Host (this does not
guarantee that XLA will be used). Devices:
2020-08-29 15:44:25.591295: I tensorflow/compiler/xla/service/service.cc:176]   StreamExecutor device (0): Host, Default Version
Epoch 1/5
1875/1875 [==============================] - 3s 2ms/step - loss: 0.2219 - accuracy: 0.9333
Epoch 2/5
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0963 - accuracy: 0.9708
Epoch 3/5
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0707 - accuracy: 0.9776
Epoch 4/5
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0531 - accuracy: 0.9831
Epoch 5/5
1875/1875 [==============================] - 3s 1ms/step - loss: 0.0438 - accuracy: 0.9852
313/313 [==============================] - 0s 923us/step - loss: 0.0688 - accuracy: 0.9800
```

- CUDAに関する警告メッセージは出ているが、実行自体はできている

コンテナ利用例：会話型バッチ、MNISTサンプル 3/4

- singularity shellやsingularity execに --nvオプションを付けておけば ノード内の4GPUを全て利用可能
  - 搭載されているGPUを確認する nvidia-smiコマンドの実行結果→

```
$ singularity shell --nv docker://tensorflow/tensorflow:latest-gpu
Singularity> nvidia-smi
Sat Aug 29 15:45:53 2020
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 440.64.00    Driver Version: 440.64.00    CUDA Version: 10.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla V100-SXM2...  On   | 00000000:3D:00.0 Off |                    0 |
| N/A   29C    P0    42W / 300W |      0MiB / 32510MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   1  Tesla V100-SXM2...  On   | 00000000:3E:00.0 Off |                    0 |
| N/A   28C    P0    41W / 300W |      0MiB / 32510MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   2  Tesla V100-SXM2...  On   | 00000000:B1:00.0 Off |                    0 |
| N/A   29C    P0    41W / 300W |      0MiB / 32510MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   3  Tesla V100-SXM2...  On   | 00000000:B2:00.0 Off |                    0 |
| N/A   30C    P0    41W / 300W |      0MiB / 32510MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

コンテナ利用例：会話型バッチ、MNISTサンプル 4/4

- GPUを使える状態（--nv）でMNISTサンプルを実行した場合

```
Singularity> python ./mnist_sample.py
2020-08-29 15:46:04.101633: I tensorflow/stream_executor/platform/default/dso_loader.cc:48] Successfully opened dynamic library libcudart.so.10.1
2020-08-29 15:46:06.099728: I tensorflow/stream_executor/platform/default/dso_loader.cc:48] Successfully opened dynamic library libcuda.so.1
2020-08-29 15:46:06.224375: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1716] Found device 0 with properties:
pciBusID: 0000:3d:00.0 name: Tesla V100-SXM2-32GB computeCapability: 7.0
coreClock: 1.53GHz coreCount: 80 deviceMemorySize: 31.75GiB deviceMemoryBandwidth: 836.37GiB/s
2020-08-29 15:46:06.225950: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1716] Found device 1 with properties:
中略
pciBusID: 0000:3d:00.0 name: Tesla V100-SXM2-32GB computeCapability: 7.0
coreClock: 1.53GHz coreCount: 80 deviceMemorySize: 31.75GiB deviceMemoryBandwidth: 836.37GiB/s
中略
2020-08-29 15:46:07.401088: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1858] Adding visible gpu devices: 0, 1, 2, 3
中略
Epoch 1/5
2020-08-29 15:46:11.574009: I tensorflow/stream_executor/platform/default/dso_loader.cc:48] Successfully opened dynamic library libcublas.so.10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.2177 - accuracy: 0.9356
Epoch 2/5
1875/1875 [==============================] - 2s 1ms/step - loss: 0.0965 - accuracy: 0.9712
Epoch 3/5
1875/1875 [==============================] - 2s 920us/step - loss: 0.0686 - accuracy: 0.9784
Epoch 4/5
1875/1875 [==============================] - 2s 877us/step - loss: 0.0533 - accuracy: 0.9826
Epoch 5/5
1875/1875 [==============================] - 2s 836us/step - loss: 0.0426 - accuracy: 0.9863
313/313 [==============================] - 0s 967us/step - loss: 0.0728 - accuracy: 0.9775
```

- GPUに関する様々な情報が出力されている
- GPUを使ったため実行時間が短くなっている

コンテナ利用例：バッチジョブ、MNISTサンプル

- 同様にMNISTサンプルをバッチジョブで実行する場合
  - ジョブスクリプト内で singularity exec --nv で実行　→
  - もちろん、外部にPythonスクリプトを用意しておいて実行しても良い
    ↓

```
#!/bin/bash
#PJM -L rscgrp=cx-extra
#PJM -L node=1
#PJM -L elapse=10:00
#PJM -L jobenv=singularity
#PJM -j
#PJM -S

module load singularity
singularity exec --nv docker://tensorflow/tensorflow:latest-gpu ¥
python mnist_sample.py
```

```
#!/bin/bash
#PJM -L rscgrp=cx-extra
#PJM -L node=1
#PJM -L elapse=10:00
#PJM -L jobenv=singularity
#PJM -j
#PJM -S

module load singularity
singularity exec --nv docker://tensorflow/tensorflow:latest-gpu python -c ¥
"import tensorflow as tf

mnist = tf.keras.datasets.mnist

(x_train, y_train),(x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(512, activation=tf.nn.relu),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test)"
```

発表の目次

- 主要ベンチマークプログラムによる性能評価
- この二ヶ月の稼働状況・利用状況
- 「不老」の特徴的な機能・便利な機能・利用上の注意
  - クラウドシステム
  - 会話型バッチ（インタラクティブバッチジョブ実行）
  - ストレージ（コールドストレージ、Type IIのローカルSSD）
  - コンテナ対応
- まとめ

まとめ

- 「不老」における主要ベンチマークの結果、
  「不老」稼働開始から最初の二ヶ月の稼働状況・利用状況、
  「不老」のいくつかの機能　について紹介しました

- 「不老」は特徴の異なる複数の計算サブシステムから構成されており、様々な利用者・利用方法が想定されるため、まだ十分にサポートしきれていない機能なども残されているのが現状です
- 不明な点などはWebのQ&Aシステム（または我々まで直接）お問い合わせください
- 人員・金銭・契約など様々な制限があるため全てのリクエストにすぐに対応できるわけではありませんが、頑張ってサポートしていきます
  - 「スーパーコンピュータ「不老」の利用に関するFAQ/Tips」
  - 「スーパーコンピュータ「不老」基本マニュアルおよび関連資料」
  - に継続して情報を掲載しているため、参考にしてやってください