DEIM2021 A33-2

待ち時間のばらつきの軽減を目的とした 動的実行領域分割アルゴリズムの提案と評価

高山沙也加† 関澤 龍一†† 鈴木 成人††† 山本 拓司††† 小口 正人†

† お茶の水女子大学 〒 112–8610 東京都文京区大塚 2-1-1
†† 富士通株式会社 〒 105–7123 東京都港区東新橋 1-5-2 汐留シティセンター
††† 株式会社富士通研究所 〒 211–8588 神奈川県川崎市中原区上小田中 4-1-1
E-mail: †{sayaka-t,oguchi}@ogl.is.ocha.ac.jp, ††r_sekizawa@fujitsu.com,
†††{shigeto.suzuki,takuji}@fujitsu.com

あらまし 近年の HPC システムでは利用者層の拡大に伴い, 投入されるジョブ数および必要ノード数は増加しており, また, 多様化している. そのためシステム規模は増加傾向にあり, 効率的な運用がより重要視される. システムの運用において, ユーザの立場ではジョブ投入から実行されるまでの待ち時間が, 運用の立場では利用可能な計算機資源のうち実際に利用された割合である充填率が重視される. 特にあらゆる規模のジョブ投入が想定される環境では, 小規模ジョブが大規模ジョブの割り当ての妨げとなり待ち時間が著しく長い大規模ジョブが生じてしまうことがあり, この軽減が大きな課題となる. 本研究では, 大規模ジョブの待ち時間の削減を目的として, ジョブ分布やジョブ実行状況の情報から, 自動で実行領域の分割数と分割ノードサイズ, 実行ジョブのノード数範囲を決定するアルゴリズムの提案と効果検証を行う. 検証には, 「京」ジョブ統計情報から生成されたジョブミックスを利用する.

キーワード ジョブスケジューリング, HPC システム, 領域分割

Consideration of System Node Partitioning Based on Input Job Information

Sayaka TAKAYAMA†, Ryuichi SEKIZAWA††, Shigeto SUZUKI†††, Takuji YAMAMOTO†††, and Masato OGUCHI†

† Ochanomizu University 2–1–1 Otsuka, Bunkyou-ku, Tokyo 112–8610 Japan
†† Fujitsu Ltd 1–5–2 Shiodome City Center, Higashishinbashi, Minato-ku, Tokyo 105–7123, Japan
††† Fujitsu Laboratories Ltd 4–1–1 Odanaka, Nakahara-ku, Kawasaki-shi, Kanagawa 221–8588, Japan
E-mail: †{sayaka-t,oguchi}@ogl.is.ocha.ac.jp, ††r_sekizawa@fujitsu.com,
†††{shigeto.suzuki,takuji}@fujitsu.com

1. はじめに

近年の HPC システムでは利用者層の拡大に伴い, 投入されるジョブ数およびその必要ノード数は増加しており, また, 多様化している. 2019 年に計算資源の共用を終了したスーパーコンピュータ「京」のジョブ規模別の計算資源利用状況を見てみると, 必要ノード数が 1000 を超えるジョブのノード割当時間積が半数以上を占めている [1]. このように, ジョブ規模の増大化に伴い, システム規模は全体的に増加傾向にある. システムの運用において, 図 1 に示すようにユーザの立場ではジョブ投入から実行されるまでの待ち時間が, 運用の立場では利用可能な計算機資源のうち実際に利用された割合である充填率が重視される. 待ち時間はシステム側で十分なノード数を用意することで, 充填率は必要ノード数に応じてジョブ受け入れに制限を設けたり割り当てがうまくいくジョブを待ったりすることで改善可能だが, 一方を優先させることで他方が悪化する可能性がある. 特にあらゆる規模のジョブの投入が想定される環境では, 小規模ジョブが大規模ジョブの割り当ての妨げとなり待ち時間が著しく長い大規模ジョブが生じてしまうことがあり, この軽減が大きな課題となる. 投入ジョブの必要ノード数に応じてシ

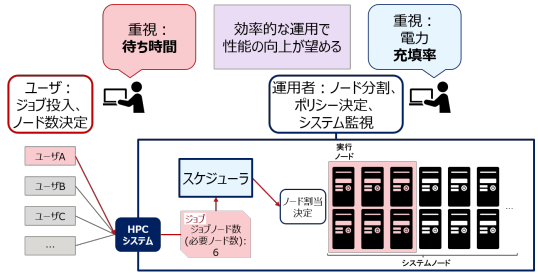

図 1: HPC システムの想定図

ステムとキューを 2 分割することで充填率と待ち時間の改善を図るという手法は実環境の運用で既に導入されている．しかし，大規模環境では 2 分割の制御では十分な効果を得られない場合がある．

本研究では，あらゆる規模のジョブが投入される環境における大規模ジョブの待ち時間の削減を目的として，過去のジョブ分布やジョブ実行状況の情報から，実行領域の分割数と分割ノードサイズ，実行ジョブのノード数範囲を決定するアルゴリズムの提案と効果検証を行う．

2. 関 連 研 究

システムの効率的な運用を目的としたジョブスケジューリング手法は既にいくつか提案されている．多様なジョブ受け入れを前提とした，不均一なマシンで構成された HPC システムの利用率とジョブの待ち時間の最適化を目的とした研究として，履歴ワークロードを用いた Portable Batch System（PBS）ベースのクラスタのジョブシミュレーションを実行する方法が提案されている [2]．この検証ではジョブスケジューリングのシミュレーション機能を持たない PBS リソースマネージャの代わりに，マウイスケジューラを用いて大規模な PBS ベースのクラスタのジョブシミュレーションを行っている．

スケジューリングのコストと効率のコントロールを目的として，時間軸方向をいくつかの区間に区切り，区間単位でジョブスケジューリングを行う区間スケジューリングが提案されている [3]．この検証では，充填率は直近の未来では短いスケジュール区間，遠い未来には長いスケジュール区間を使った場合に最も高く，待ち時間はスケジュール区間が長いほどノード数の少ないジョブの待ち時間が短くなることが確認されている．

また，HPC システムのジョブスケジューリングでは後述の Backfill が用いられることが多いが，その効率はジョブ実行時間の見積の正確さに依存する．利用者のジョブ実行時間見積の正確性を示す指標を導入し，その指標に応じた優先度で Backfill する手法によって通常の Backfill と比較して最大で 6 % の処理性能が向上したことが示されている [4]．

以前の研究では，異なるシステム規模，ジョブ分布条件でジョブスケジューリングを行い，充填率と待ち時間の評価を行った [5]．その結果，ジョブの必要ノード数とその分布がジョブスケジューリングの充填率と待ち時間の違いに大きく影響を与えており，効率的なシステム運用のためには実行領域の動的な変更が必要となることが明らかになった．

本研究ではジョブの必要ノード数の分布や待ち時間に注目し，自動で実行領域の分割数と分割ノードサイズ，実行ジョブのノード数範囲を決定するアルゴリズムの提案と効果検証を行う．

3. 実行領域分割アルゴリズム

HPC ユーザが最も重視する指標は自身のジョブが投入されるまでの時間である待ち時間である．特にあらゆる規模のジョブが混在する環境では小規模ジョブが大規模ジョブの割り当ての妨げになることがある．そのため大規模ジョブの待ち時間は長期化しやすく，これを短くするのが課題となる．そこで，小規模ジョブが大規模ジョブの実行の妨げとなるのを防ぎ，ジョブ規模毎に生じる待ち時間の大きなばらつきの軽減を目的とした実行領域分割アルゴリズムを提案する．アルゴリズムの概要を以下に記す．

提案アルゴリズムでは，過去の投入ジョブ分布とジョブの待ち時間，そして運用者が設定したジョブの最大最小待ち時間の差から，実行領域の分割数，各実行領域が受け入れるジョブの必要ノード数範囲，各実行領域のサイズを決定する．アルゴリズムはジョブスケジューリングから独立して動作し，一定間隔で適用する．初めに，各実行領域における最大最小待ち時間の差と運用者が設定した最大最小待ち時間の差を比較し，実行領域の分割数を決定する．いずれの実行領域でも最大最小待ち時間の差が運用者の設定を下回っていれば分割数は-1，そうでなければ+1する．次に，各実行領域が受け入れるジョブの必要ノード数範囲を決定する．この時，各実行領域のジョブ粒度が揃うよう分割する．例えば，最大必要ノード数が 10000 として分割数を 2 とすると，実行領域 1 には 1 から 100，実行領域 2 には 101 から 10000 の必要ノード数のジョブを投入する．最後に，決定した分割数とジョブの必要ノード数範囲で，各実行領域の大きさを決める．ジョブの必要ノード数範囲別にノード実行時間積の総和を求め，その比に合わせてシステムノードを分割し，各実行領域とする．この時，ジョブの必要ノード数に対して実行領域があまりに小さいとキューが詰まりやすくなるため，最大ジョブノード数の 2 倍より大きくない実行領域があった場合，該当する実行領域のノード数を最大ジョブノード数の 2 倍とし，他の実行領域はシステムノードから該当実行領域のノード数を差し引き割合で分割した大きさとする．アルゴリズムの流れを図 2 に示す．

4. 実 験 概 要

提案アルゴリズムの効果検証のために，あるジョブ条件でアルゴリズムを適用し，分割数，各実行領域が受け入れるジョブの必要ノード数範囲，各実行領域の大きさを変えた際の充填率と待ち時間の評価を行う．本来のアルゴリズムでは各実行領域における最大最小待ち時間の差と 運用者が設定した最大最小待ち時間の差を比較し分割数を決定するが，実験では効果確認のために最大最小待ち時間の差に関係なく分割数を増やし実験する．

ジョブスケジューラには Slurm Workload Manager (Slurm) [6] を用いる．ジョブデータには，後述のジョブミックスを利用する．ジョブ密度をシステムノードに対する投入ジョブ規模の

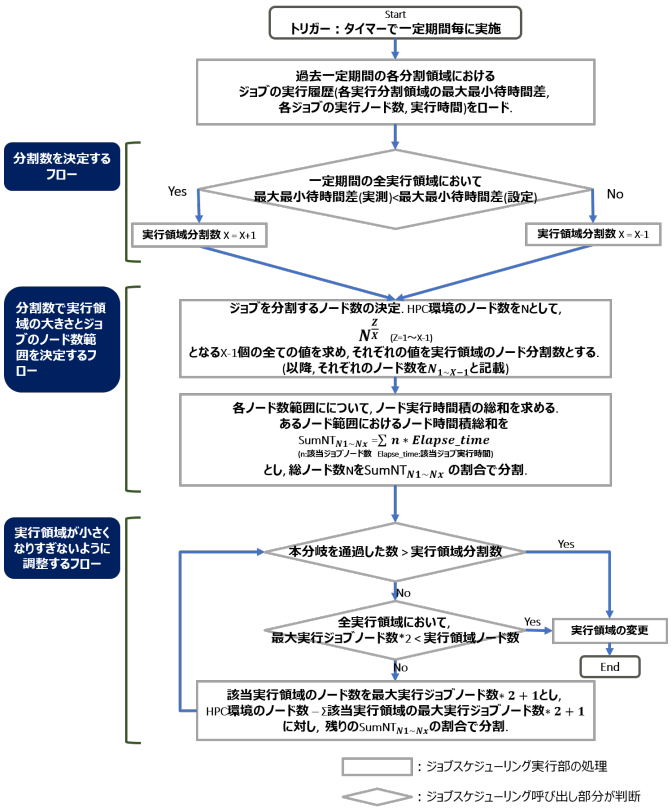

図 2: 提案アルゴリズムのフローチャート

多寡の指標とし, 次式で求めた値とする.

$$Input_Ratio = \frac{\sum_{i=0}^{n} NodeJob_i \times ElapseTime_i}{TotalSubmit \times SystemNode} \times 100$$

この時, NodeJob はそのジョブが必要とするノード数, ElapseTime はジョブの実行時間, TotalSubmit は総投入期間, SystemNode はシステム全体のノード数を表す.

シミュレーションにおいて定めたシステム側の条件を表 1 に, ジョブ条件を表 2 に記す. ノードごとの性能差やネットワークによる不都合がない環境を前提とする. また, システムを構成するノードの扱いはいずれも平等とする.

表 1: システム側の条件

システムノード数	9216
backfill	なし
離散割当	なし
Fair share	なし
トポロジ	Tree

表 2: ジョブデータの条件

ジョブの最大必要ノード数	1000
ジョブの取得期間	7 日間
ジョブ密度	95%

4.1 ジョブミックス

ジョブミックス [3] とは, 「京」ジョブ統計情報から最小必要ノード数と最大必要ノード数, そしてジョブ密度を設定し, ノード実行時間積がジョブ密度に達するまでジョブをランダムピックアップして生成されたジョブ群である. ジョブミックスの生成の際, 「京」のジョブデータは必要ノード数, 実行時間で分類されている. それぞれのグループの実際の実行時間やジョブ数, ジョブの投入頻度や分散といった統計情報を用いて, より実データに近い条件でジョブデータは生成される.

4.2 スケジューリングアルゴリズム

本研究での実験でスケジューリングアルゴリズムとして利用した First-Come-First-Served (FCFS) では, ジョブは割り当て優先順位が到着順に定められ, 処理が行われる. FCFS の利点としてはアルゴリズムがシンプルであること, ジョブ処理が公平であることが挙げられるが, ジョブの投入タイミングによっては充填率が著しく悪くなる可能性がある.

そのため, Backfill を有効にすることで, 割り当て優先順が低い場合でも優先度の高いジョブを遅延させずに, かつ前方に実行可能領域を確保できる場合は優先度の高いジョブを追い越して割り当てできるようになる. Backfill の導入の有無によるスケジューリングの違いを図 3 に示す.

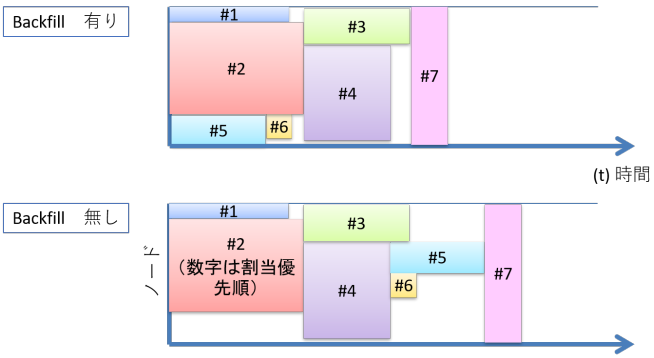

図 3: Backfill の有無によるスケジューリングの違い

4.3 Slurm Workload Manager

Slurm Workload Manager [6] はあらゆる規模の HPC システムで利用されているジョブスケジューリングシステムで, Sequoia (Lawrence Livermore National Laboratory) や Piz Daint (Swiss National Supercomputing Centre) のような TOP500 のスーパーコンピュータにも用いられている. 一般的なリソース管理プログラムと同様にジョブのスループット, システムの利用率, およびジョブの待ち時間に大きく影響する可能性のある多くのパラメータ設定が可能である. ただし, 実験条件やポリシーの変更が実稼働 HPC システムにおけるスケジューラのパフォーマンスに与える影響を確認するには数日から数週間かかる場合があるため, パラメータを変更することでシステムパフォーマンスを調整したり, 新しいポリシー選択を実装したりすることは実用的ではない.

そこで, 本研究では Slurm Simulator [7] を用いる. このシミュレータでは小規模なクラスタ構成であれば, ジョブの構成

とパラメータ条件によっては 1 時間で 17 日分のスケジューリングシミュレーションが可能である.

5. 実験結果

5.1 アルゴリズムを適用したジョブスケジューリング

実行領域分割アルゴリズムを適用しなかった場合と適用した場合とでジョブスケジューリングの結果を比較する. 表 2 の条件で生成したジョブミックス (以降, Job1 とする) を利用した. アルゴリズムを適用して決定した実行領域条件を表 3 に示す. また, 表 4 に各分割数でのジョブ粒度を示す. 分割数を増やすほど各実行領域での最大最小必要ノード数の差は小さくなるため, ジョブ粒度は大きくなる.

表 3: Job1 の実行領域条件

	実行領域サイズ	受け入れノード数上限
2 分割	193/9023	-30/-1000
3 分割	19/1081/8116	-9/-99/-1000
4 分割	11/190/1487/7528	-5/-31/-177/-1000
5 分割	7/66/636/1252/7255	-3/-15/-63/-251/-1000

表 4: Job1 の各分割数でのジョブ粒度

	ジョブ粒度
分割なし	0.001
2 分割	0.030
3 分割	0.101-0.111
4 分割	0.178-0.200
5 分割	0.252-0.333

分割アルゴリズム適用による充填率への影響を確認する. 各分割数でジョブスケジューリングを実行した際の充填率を図 4 に示す. この結果を見ると, 分割数を増やしたことによる充填率の悪化はほぼ見られない.

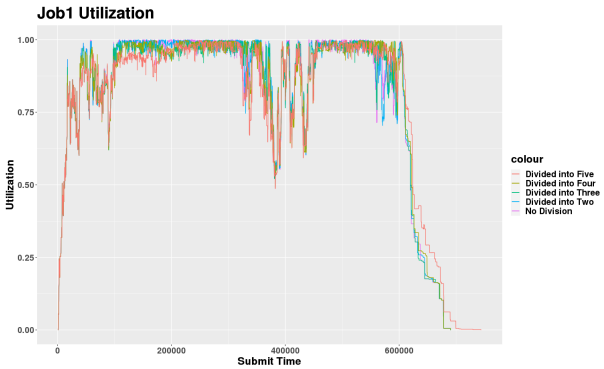

図 4: 各分割数での充填率

次に, Job1 の各分割数での最大最小待ち時間の差を図 5 に示す. 分割した場合では, 最大最小待ち時間の差が最も大きい分割領域の結果を出力している. アルゴリズムを適用した場合では 2, 3, 4 分割では分割なしと比べて最大最小待ち時間の差は大幅に小さくなっている. この結果から, 特定のジョブのみ待ち時間が大幅に増加してしまう問題はアルゴリズムの適用によって軽減できていることが確認できる. ただし, 5 分割では他の実験結果と比べて最大最小待ち時間の差が大幅に増えてしまっている. この結果を見ると, 分割数を増やせば必ずしも良くなるわけではないことがわかる.

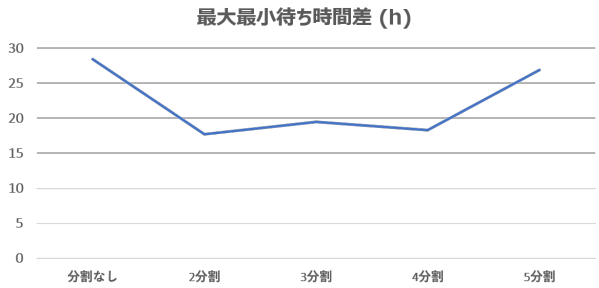

図 5: 最大最小待ち時間の差

実行領域を分割せずにジョブスケジューリングを行った場合と, 提案アルゴリズムに則って実行領域を 2 分割した場合での最大最小待ち時間の差を図 6 に示す. 受け入れるジョブの必要

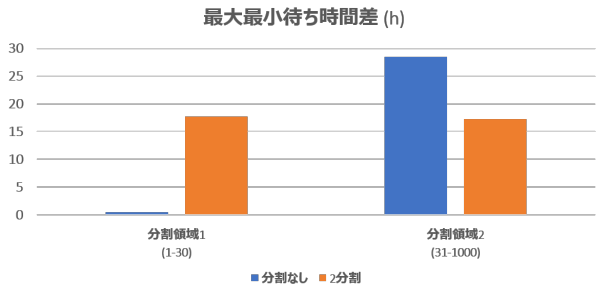

図 6: 分割なしと 2 分割の最大最小待ち時間の差

ノード数範囲が小さい順に番号を振っており, 必要ノード数に応じて投入領域を分けられた各ジョブ群での最大最小待ち時間の差を評価の対象とする. また, 分割なしの実験結果でも, 比較のため分割した場合と同様のジョブの必要ノード数範囲でジョブを分けて最大最小待ち時間の差を見た. 図 6 を見ると, 分割なしの実験結果では, 実行領域 1 の小規模ジョブ群の待ち時間の差は著しく小さいのに対し, 実行領域 2 の大規模ジョブ群では大幅に待たされているジョブが生じているのが確認できる. アルゴリズムに則って実行領域を 2 分割した場合では分割なしと比べると実行領域 1, 2 の最大最小待ち時間の差の違いは小さく, アルゴリズムの目的であるジョブ規模毎に生じる待ち時間の大きなばらつきの軽減は達成できていると言える. 分割数 3, 4, 5 の最大最小待ち時間の差を図 7 - 9 に示す. いずれの分割数でも分割なしと比べてジョブ規模毎の最大最小待ち時間の差の違いは小さいが, 分割数 5 では最大最小待ち時間の差のばらつきが大きい. これは, 分割数を増やしたことで各分割領域のジョブ粒度のばらつきが大きくなってしまったためであると考えられる.

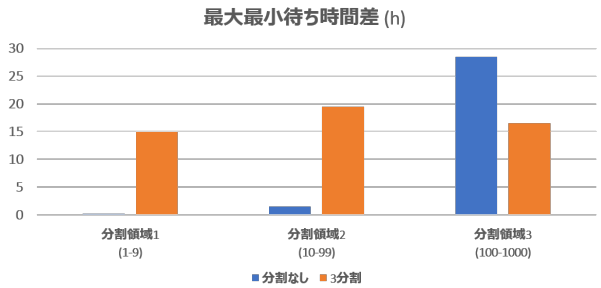

図 7: 分割なしと 3 分割の最大最小待ち時間の差

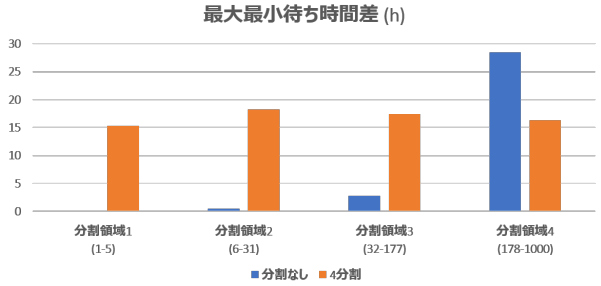

図 8: 分割なしと 4 分割の最大最小待ち時間の差

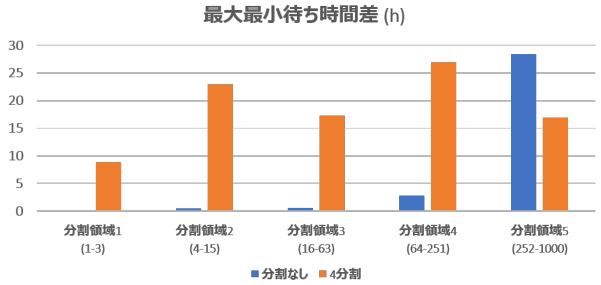

図 9: 分割なしと 5 分割の最大最小待ち時間の差

5.2 ジョブミックス比較

ジョブ密度と最大最小必要ノード数, 投入期間を Job1 と同じ条件で生成したジョブミックス (以降, Job 2, Job 3 とする) で同様に実験を行った. ただし, 生成したジョブミックスはそれぞれジョブの投入タイミングや総投入数が若干異なる. 表 5 に生成した各ジョブミックスの詳細を記す.

表 5: 各ジョブミックスの詳細

	ジョブ投入数	平均必要ノード数	最大ノード実行時間積
Job1	2515	164.58	23040
Job2	2521	172.81	19008
Job3	2461	172.57	21888

各ジョブミックスの最大最小待ち時間の差を図 10 に示す. Job 2, Job 3 では分割なしと比べて分割アルゴリズム適用によって最大最小待ち時間の差は小さくなっている. ただし, 最大最小待ち時間の差が最も小さくなる分割数は異なっている. また, Job 1 は Job 2, Job 3 とは異なり分割しない方が最も最大最小待ち時間の差が短い. これら結果から, 投入ジョブの必

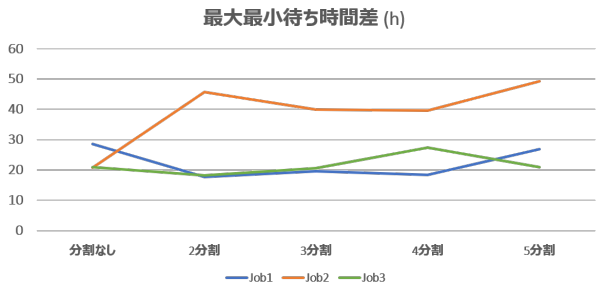

図 10: 各ジョブミックスの最大最小待ち時間の差

要ノード数分布や実行領域, ジョブ密度が同じでもジョブの投入タイミングによって最適な分割数は異なり, 想定されるジョブ規模に大きな変化がなくても定期的にアルゴリズムを適用し分割領域と分割条件を変える必要があると言える.

5.3 分割がうまく機能しないジョブミックスの分析

本項では Job 1 で提案アルゴリズムがうまく機能しなかった原因の分析を行う. 図 11-13 は 2 分割適用時の, それぞれ Job 1, Job 2, Job 3 の各ジョブの投入時間と待ち時間をを表している. Job 1 と Job 3 は待ち時間が長いジョブでも 20 時間程度で収まっているのに対し, Job 2 ではジョブ投入開始から 150000 秒頃に待ち時間が 40 時間近いジョブが発生しており, このジョブが最大最小待ち時間の差の悪化の原因であると考えられる.

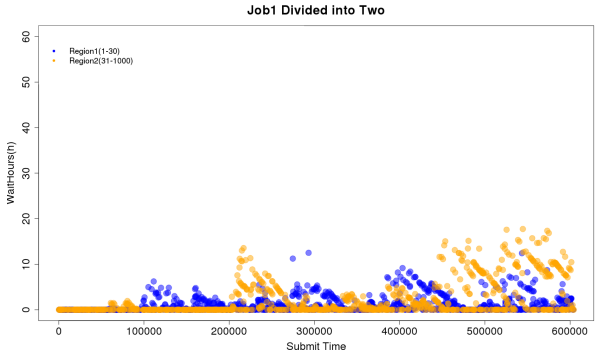

図 11: Job1 のジョブの投入時間と待ち時間

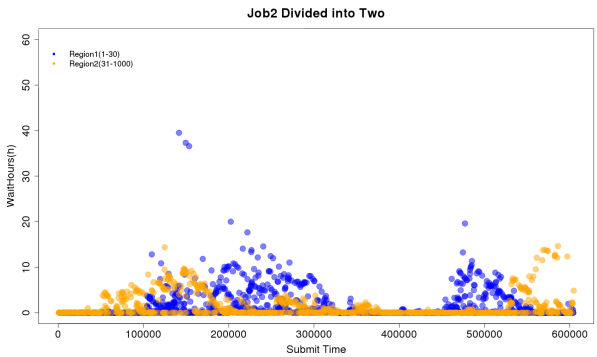

図 12: Job2 のジョブの投入時間と待ち時間

次にジョブのジョブの投入タイミングと投入密度を確認する.

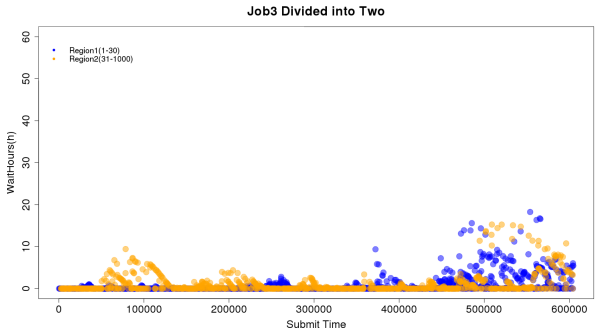

図 13: Job3 のジョブの投入時間と待ち時間

Job 2 において最大最小待ち時間の差が大きかった実行領域 2 に限定して,

$$\frac{\sum_{i=0}^{n} NodeJob_i \times ElapseTime_i}{NodeSize \times Interval}$$

の式から一定時間隔での実行領域における投入密度を求める. 式において, NodeJob はそのジョブが必要とするノード数, ElapseTime はジョブの実行時間, NodeSize は実行領域のノード数, Interval はジョブを集計する時間隔を表す. Interval は 20000 秒とする. 図 14 に 20000 秒間隔で集計した各ジョブの投入密度を示す. Job 2 は 100000 秒と 120000 秒で投入密度が

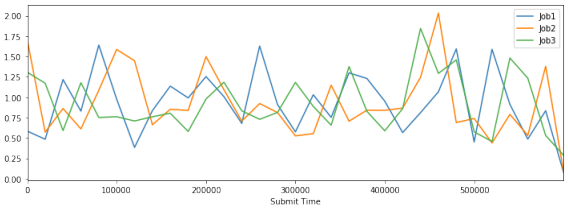

図 14: 各ジョブの投入密度

高くなっており, 実行領域に対してジョブ投入が集中している. また, Job 2 において最も投入密度が高いのは 460000 秒だが, 次の時間隔では 0.75 程度まで下がっている. そのため, ある程度の期間投入密度が高くなければ著しく待ち時間が長いジョブは発生しづらいと考えられる. 提案アルゴリズムででは分割される実行領域のサイズはジョブの投入間隔を考慮していないため, Job 2 のように投入タイミングによっては最大最小待ち時間の差の改善が見られないことがある. そのため, 投入タイミングに偏りが発生した場合の対処を考える必要がある.

5.4 異なるジョブ分布を用いた実験

必要ノード数が 1 から 100 のジョブが 90%, 101 から 1000 のジョブが 5%のジョブ群を混合させた混合分布で実験を行った. 混合分布を用いて, 各分割数でジョブスケジューリングを実行した際の充填率を図 15 に示す. 「京」分布の充填率と比べると, 分割数を増やすほど充填率の時系列の変化に段差が生じるようになり, 実行終了までに要する時間も長くなっている. 段差は特定の実行領域にジョブが投入されないままだと生じやすい. そのため, この結果からジョブ投入が一部の実行領域に偏っていることがわかる. ノードサイズは必要ノード数範囲別

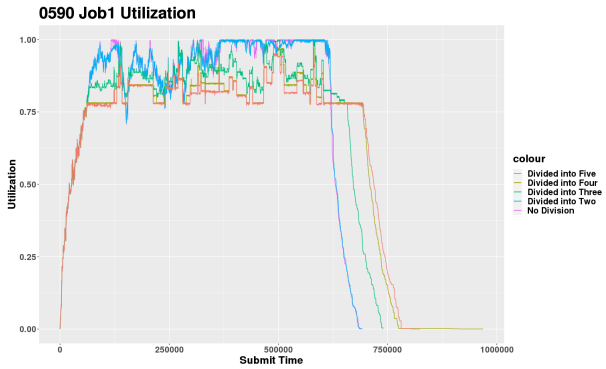

図 15: 混合分布における各分割数での充填率

にノード実行時間積の総和を求め, 比で分割しているが, 実行領域のサイズの下限を最大必要ノード数の 2 倍にしており, 最大必要ノード数の 2 倍より小さい実行領域があった場合はサイズを最大必要ノード数の 2 倍にして, 他の領域も再度サイズを決め直している. そのため, 実行領域によっては最大必要ノード数に対してジョブのノード実行時間積が小さく, 最大必要ノード数の 2 倍だとジョブに対して余ってしまい代わりに他の実行領域を大きく圧迫してしまう可能性がある. 混合分布では小規模ジョブの割合が「京」分布と比べて大きく, 大規模ジョブ群を受け入れる実行領域の下限によって他の実行領域が大きく圧迫されていると考えられる.

次に, 混合分布の投入タイミングの異なる三つのジョブミックスの, 分割数別の最大最小待ち時間の差を図 16 に示す. 各ジョブミックスは 5.2 と同様に Job 1, Job 2, Job 3 とする. 5.1 の結果と比べると 4 分割以降の悪化が特に著しい. また, 各

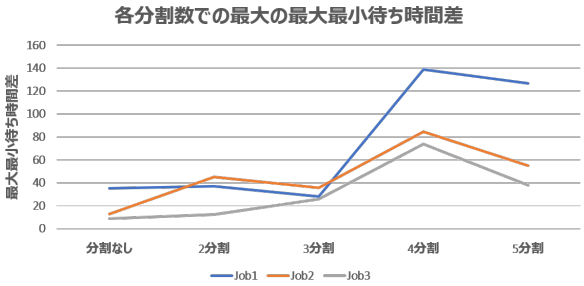

図 16: 混合分布における最大最小待ち時間の差

領域の最大最小待ち時間の差と割り当てノードサイズを比較したところ, 最小ノードサイズの適用によるノード数の変動割合の大きい分割領域ほど, 分割なしと比べて最大最小待ち時間の差の悪化が著しいことがわかった.

5.5 最小ノードサイズの変更

5.4 の実験では, 5.1 と比べて大規模ジョブの投入割合が小さく, 最小ノードサイズが適用されることが多かった大規模ジョブ群を受け入れる領域に無駄が多かった可能性が考えられる. これまでの実験では, ノード実行時間積の比で決定したノードサイズと最小ノードサイズとして受け入れるジョブの最大必要ノード数の 2 倍実行領域を確保してから割り当てし直した (以降, 条件 1 とする) ノードサイズを適用していた. そこで, 最小

ノードサイズを実行領域が受け入れるジョブの最大必要ノード数と同じにして (以降, 条件 2 とする) 再度実験を行う. 実験には 5.4 と同様のジョブを用いた.

条件 2 を適用し, 混合分布でジョブスケジューリングを実行した際の充填率を図 17 に示す. 図 15 の結果と比べて改善が確認出来た.

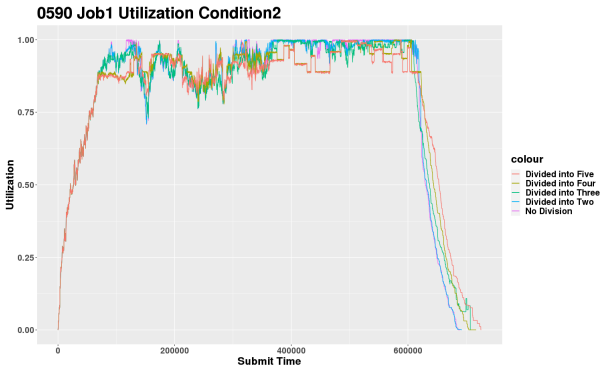

図 17: 混合分布における各分割数での充填率 - 条件 2

次に, 混合分布の三つのジョブを用いて条件 2 をジョブスケジューリングに適用した際の, 分割数別の最大最小待ち時間の差を図 18 に示す. こちらは, 条件 1 適用時の結果と比べるとや

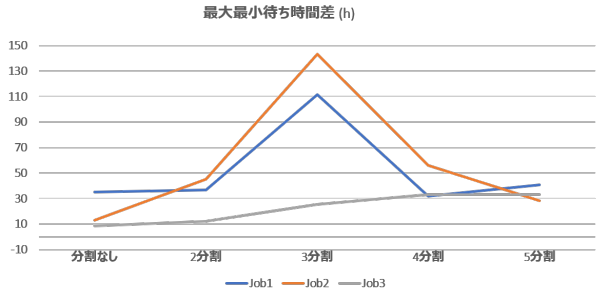

図 18: 混合分布における最大最小待ち時間の差 - 条件 2

や安定しているが, 3 分割時の最大最小待ち時間の差の悪化が著しい. ただし, Job 3 のみ 3 分割時の著しい悪化はなく, 3 分割以降緩やかに最小待ち時間の差が増加している.

表 6 と表 7 は, それぞれ Job 1 の 3 分割と 4 分割での, ノード実行時間積の比で決定したノードサイズと条件 2 適用後のノードサイズの比較である. 3 分割では条件 1 の方が, 分割数 4 では条件 2 の方が最大最小待ち時間の差の結果が良かった. これは, 元のノード実行時間積の多寡が影響していると考えられる. 分割数 3 の分割領域 3 は 941.9 と, ジョブの最大必要ノード数と比べると比較的大きめである. そのため, 実行領域に比較的余裕のある条件 1 の方がうまく機能したと考えられる. それに対して分割数 4 の分割領域 4 は 472.3 とやや小さめである. そのため, 条件 1 ではやや無駄が多く, 他の領域が圧迫されてしまったと考えられる.

表 8 と表 9 は, それぞれ Job 2 の 3 分割と 4 分割での, ノード実行時間積の比で決定したノードサイズと条件 2 適用後のノードサイズの比較である. ただし, 表 8 はいずれの領域もノード

表 6: ノードサイズ比較 - 3 分割 - Job1 - 条件 2

	分割領域 1	分割領域 2	分割領域 3
時間積分割	65.8	8208.3	941.9
条件 2 適用後	65.3	8159.7	1000

表 7: ノードサイズ比較 - 4 分割 - Job1 - 条件 2

	分割領域 1	分割領域 2	分割領域 3	分割領域 4
時間積分割	26.3	1539.9	7177.5	472.3
条件 2 適用後	24.7	1447.0	6744.4	1000

実行時間積分割時点で条件 2 を満たしていたため同じ条件である. Job 1 と同様, 3 分割では条件 1 の方が, 分割数 4 では条件 2 の方が最大最小待ち時間の差の結果が良かった. ただし, 分割なしの時点で最大最小待ち時間の差が Job 1 ほど大きくないためか分割による改善はほぼ見られなかった.

表 8: ノードサイズ比較 - 3 分割 - Job2 - 条件 2

	分割領域 1	分割領域 2	分割領域 3
時間積分割	61.2	8124.4	1030.5
条件 1 適用後	61.2	8124.4	1030.5

表 9: ノードサイズ比較 - 4 分割 - Job2 - 条件 2

	分割領域 1	分割領域 2	分割領域 3	分割領域 4
時間積分割	23.5	1432.0	7328.2	432.4
条件 1 適用後	21.9	1339.4	6854.7	1000

表 10 と表 11 は, それぞれ Job 2 の 3 分割と 4 分割での, ノード実行時間積の比で決定したノードサイズと条件 2 適用後のノードサイズの比較である. ただし, 表 10 はいずれの領域もノード実行時間積分割時点で条件 2 を満たしていたため同じ条件である. また, Job 3 は最大必要ノード数が 856 となっていたため条件 2 適用時の分割数 4 の分割領域 4 の最小ノードサイズも 856 となっている. Job 3 では条件 1 よりも条件 2 適用時の方が全体的に最大最小待ち時間の差が小さくなっている. しかし, 分割領域 4 の最大最小待ち時間の差は条件 1 より大きくなっている. ノード実行時間積で決定したノードサイズは 419.8 と小さめだが, 最大必要ノード数に対するノードサイズの割合は 0.49 であり, Job 1 の分割数 3 の分割領域 3 や Job 2 の分割数 4 の分割領域 4 よりも大きい. そのため, 条件 2 では分割領域 4 に投入されるジョブ群に対してノード数が不十分であったと考えられる.

表 10: ノードサイズ比較 - 3 分割 - Job3 - 条件 2

	分割領域 1	分割領域 2	分割領域 3
時間積分割	70.7	6944.2	2201.1
条件 1 適用後	70.7	6944.2	2201.1

表 11: ノードサイズ比較 - 4 分割 - Job3 - 条件 2

	分割領域 1	分割領域 2	分割領域 3	分割領域 4
時間積分割	23.5	1572.4	7200.3	419.8
条件 1 適用後	22.4	1494.4	6843.2	856

5.6 アルゴリズムの改善

5.5 の実験結果を踏まえて, ノード実行時間積によって決定されたノードサイズに応じて最小ノードサイズを可変とする. 先の実験結果から 0.49 を最小ノードサイズ決定の閾値としている. 最大実行ジョブノード数に対する割り当て実行領域の割合が 0.49 未満であれば最小ノードサイズは各実行領域の受け入れる最大実行ジョブノード数と同数, そうでなければ最小ノードサイズは最大実行ジョブノード数の 2 倍設ける.

改善後のアルゴリズムの効果検証のために追加実験を行った. 検証には 5.4 で用いた Job 1, Job 2, Job 3 と, 新たに生成した混合分布のジョブミックス Job 4, Job 5, Job 6 を用いる.

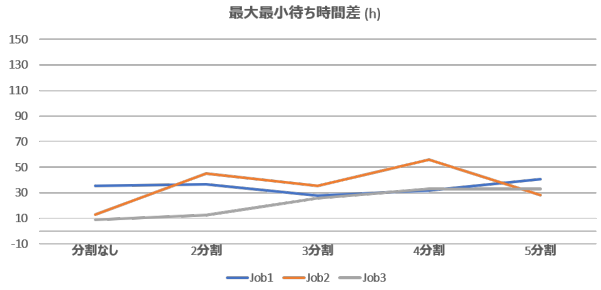

図 19: Job1-3 の最大最小待ち時間の差 - 改善後

Job 1, Job 2, Job 3 に改善後のアルゴリズムを適用した際の最大最小待ち時間の差を 19 に示す. 5.4 の結果と比べると, 全体的に最大最小待ち時間の差が小さくなっており, 改善されていると言える.

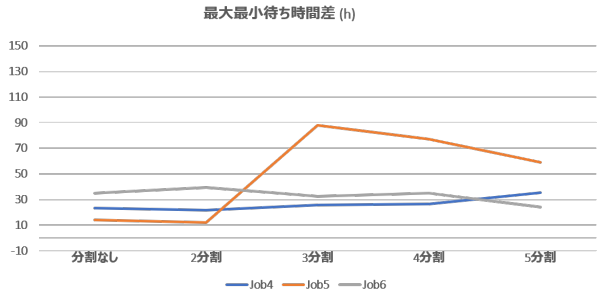

図 20: Job4-6 の最大最小待ち時間の差 - 改善後

図 20 に Job 4, Job 5, Job 6 における最大最小待ち時間の差を示す. Job 5 のみ分割数 3 以上の最大最小待ち時間の差が大きく悪化しており, 最小ノードサイズの調整で全分割数で最大最小待ち時間の差が解決するわけではないと考えられる. しかし, 5.4 の実験結果と比較して, いずれのジョブでも分割アルゴリズムの適用によって分割なしよりも最大最小待ち時間の差が小さくなっている分割数があり, 改善されていることが確認できた.

6. まとめと今後の予定

あらゆる規模のジョブが投入される環境における大規模ジョブの待ち時間の削減を目的として, 過去のジョブ分布やジョブ実行状況の情報から, 自動で実行領域の分割数と分割ノードサイズ, 実行ジョブのノード数範囲を決定するアルゴリズムの提案と効果検証を行った. 提案アルゴリズムでは, 過去の投入ジョブ分布とジョブの待ち時間, そして運用者が設定したジョブの最大最小待ち時間の差から, 実行領域の分割数, 各実行領域が受け入れるジョブの必要ノード数範囲, 各実行領域のサイズを決定する. 実験結果から, アルゴリズムの適用によって充填率の悪化なく最小待ち時間の差が改善されることが示された. ただし, ジョブの投入タイミングに偏りが生じていたり, 分割なしの時点で著しく待ち時間の大きいジョブが生じていない場合は分割アルゴリズムを適用してもスケジューリングに改善は見られなかった. アルゴリズムを修正し最小ノードサイズを可変とすることで, 混合分布の最大最小待ち時間の差が著しく悪化する問題はやや改善された.

今後の課題として, 投入されるジョブ分布や密度が同じ条件でもジョブの投入タイミングに偏りが生じる場合や, 混合分布のように最大最小待ち時間の差があまり大きくない場合の更なるスケジューリング改善が挙げられる.

謝　　辞

本研究の一部はお茶の水女子大学と富士通研究所との共同研究契約に基づくものである.

文　　献

[1] 「京」の稼働状況. *https://www.r-ccs.riken.jp/jp/k/machine-status/*, Accessed: November 2020.

[2] Georg Zitzlsberger, Branislav Jansík, and Jan Martinovič. Job simulation for large-scale pbs based clusters with the maui scheduler. *BIG DATA ANALYTICS, DATA MINING AND COMPUTATIONAL INTELLIGENCE 2018 THEORY AND PRACTICE IN MODERN COMPUTING 2018*, p. 137.

[3] 山本啓二, 宇野篤也, 関澤龍一, 若林大輔, 庄司文由ほか. 区間スケジューリングを用いたジョブスケジューリングの性能評価. 研究報告ハイパフォーマンスコンピューティング (HPC), Vol. 2014, No. 3, pp. 1–5, 2014.

[4] 滝澤真一朗, 高野了成ほか. 正確な実行時間指定を促すジョブスケジューリング. 研究報告ハイパフォーマンスコンピューティング (HPC), Vol. 2018, No. 2, pp. 1–9, 2018.

[5] 高山沙也加, 関澤龍一, 鈴木成人, 山本拓司, 小口正人. 投入ジョブ情報を踏まえたシステムノードのパーティション分割の考察. 第 12 回データ工学と情報マネジメントに関するフォーラム 論文集, Vol. 2020, pp. H4–4, 2020.

[6] Slurm Workload Manager. *https://slurm.schedmd.com/*, Accessed: November 2020.

[7] Nikolay A Simakov, Martins D Innus, Matthew D Jones, Robert L DeLeon, Joseph P White, Steven M Gallo, Abani K Patra, and Thomas R Furlani. A slurm simulator: Implementation and parametric analysis. In *International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems*, pp. 197–217. Springer, 2017.