大規模環境における動的実行領域分割ジョブスケジューリング手法の提案と評価

理学専攻・情報科学コース　1940651　高山沙也加　(指導教員：小口正人)

1　はじめに

近年の HPC システムでは利用者の増加と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, 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が投入される環境では, 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ステムとキューにパーティションを設け 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, 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を分析し, 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.

2　提案アルゴリズム

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. 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, その比に則ってシステムノードを分割し, 各実行領域とする. この時, 最大ジョブノード数の 2 倍より小さ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引いて再度ノード実行時間積の比で分割した大きさとする.

3　実験

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, 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. 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. 実験用に生成したジョブミックスの詳細を表 1 に示す. 表 1 の条件で生成したジョブミックスを以降 Job 1 , Job 2 , Job 3 とする. 実験には Slurm Simulator[2] を用いる.

表 1: 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　実験結果

4.1　アルゴリズムを適用したジョブスケジューリング

Job 1 を用いて, 各分割条件でジョブスケジューリングを実行した際の充填率を図 1 に示す. 分割アルゴリズム適用前後で変

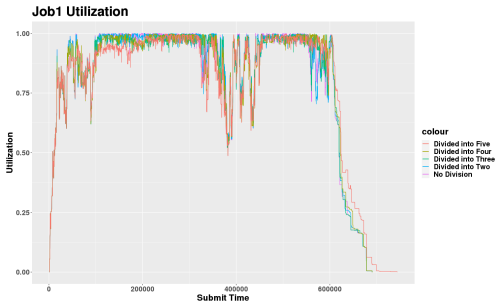

図 1: 各分割数でジョブスケジューリングした際の充填率

化はほとんどないことがわかる. これらの結果から, 分割アルゴリズムによる充填率への悪影響はほぼなく, 分割数を増やしても同様であると考えられる.

次に, Job 1 , Job 2 , Job 3 の各分割数での最大最小待ち時間の差を図 2 に示す. 分割した場合では, 最大最小待ち時間の差が

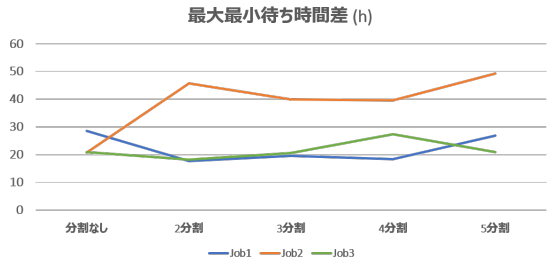

図 2: 各分割数での最大最小待ち時間の差

最も大きい分割領域の結果を出力している. 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, 分割数を増やせば必ずしも良くなるわけではない. Job 1 , 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が, 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 2 は Job 1, 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小さい. これらの結果から, 投入ジョブの必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.

4.2 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 のジョブで密度が 90%, 101 から 1000 のジョブで密度が 5% のジョブ群を混合させた混合分布で実験を行った. 生成した各ジョブミックスは 4.1 と同様に Job 1, Job 2, Job 3 とする. 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 3 に示す. 「京」分

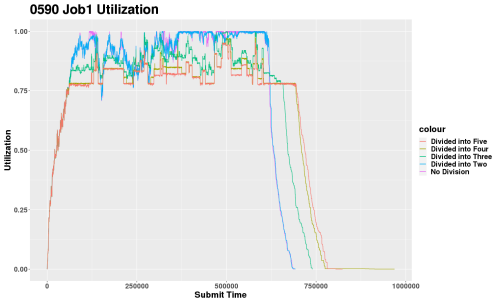

図 3: 混合分布における各分割数での充填率

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.

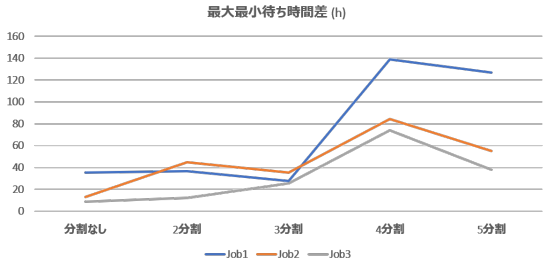

図 4: 混合分布混合分布における各分割数での最大最小待ち時間の差

Job 1, Job 2, Job 3 の, 分割数別の最大最小待ち時間の差を図 4 に示す. 4.1 の結果と比べると 4 分割と 5 分割での悪化が特に著しい. ノード実行時間積の比で決定したノードサイズを確認すると, 最大最小待ち時間の差が著しく増加している分割数の実行領域で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の割合で決定したノードサイズが小さい. そのため, 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.

4.3 アルゴリズムの改善

4.2 の実験結果のように, 最小ノードサイズを受け入れる最大ジョブノード数の二倍にすると他の領域を圧迫することがある. そこで, 最小ノードサイズを可変にして再度実験を行った. ノードサイズの最大必要ノード数に対する割合に閾値を設け, 0.49 以上であれば最大必要ノード数の 2 倍, そうでなければ最大必要ノード数と同数とする. 実験に用いたジョブミックスは 4.2 で用いたものと同条件で生成し, Job 4, Job 5, Job 6 とする.

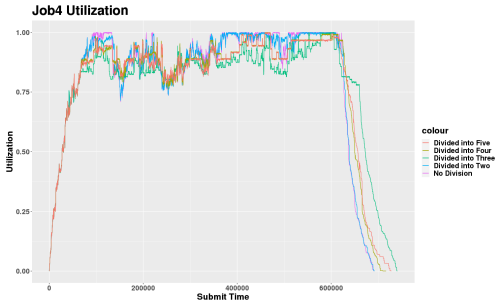

図 5: 混合分布における各分割数での充填率 - 改善後

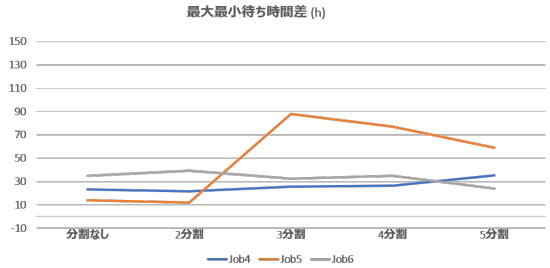

図 6: 混合分布混合分布における各分割数での最大最小待ち時間の差 - 改善後

図 5 は改善後のアルゴリズムを用いた, 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である. 図 3 と比べると分割数を増やしたことによる充填率の悪化が軽減されている.

Job 4, Job 5, Job 6 の, 分割数別の最大最小待ち時間の差を図 5 に示す. こちらも, 図 4 と同様分割数によっては著しく悪化が見られるジョブミックスがあるが, いずれかの分割数で分割なしよりも最大最小待ち時間の差の改善が見られた.

5 おわりに

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を分析し, 自動で実行領域の分割数, 各実行領域が受け入れるジョブの必要ノード数範囲, 各実行領域のノードサイズを決定するアルゴリズムの提案と効果検証を行った.

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る, または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. 最小ノードサイズを可変にしたところ, この問題はやや改善が見られた.

今後の課題として, 最小待ち時間の差の改善があまり見られない場合の更なるスケジューリング改善が挙げられる.

参考文献

[1] 「京」の稼働状況. *https://www.r-ccs.riken.jp/jp/k/machine-status/*, Accessed: January 2021.

[2] 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.