課題研究報告書

ヘテロジニアス環境における MapReduce の自動タスク割り当て

若井 久瑠美

主指導教員 井口 寧

北陸先端科学技術大学院大学
先端科学技術研究科
（情報科学）

令和 7 年 3 月

Abstract

In recent years, with the explosive growth of data volume and the increasing diversity of computational tasks, data centers have evolved from homogeneous cluster architectures to heterogeneous ones. This transition is driven by the widespread adoption of specialized hardware components such as GPU, ARM CPU, and FPGA. However, MapReduce were originally developed with homogeneous environments. MapReduce has proven exceptionally effective in large-scale data processing on homogeneous clusters of commodity machines, its performance can degrade significantly in heterogeneous environments.

To address this challenge, a method known as MrHeter was proposed, aiming to optimize task assignment in MapReduce running on heterogeneous clusters. Specifically, MrHeter distributes map and reduce tasks according to the actual capabilities of each node. MrHeter can dramatically reduce the execution time of MapReduce jobs, achieving improvements of 30% to 70% over traditional, homogeneous scheduling approaches.

The MrHeter method solves the calculation models MrHeter-MS and MrHeter-R to find the optimal number of task assignments. To solve these computational models, we use the average execution time of the map and reduce tasks for each heterogeneous node, but this needed to be measured manually. The more heterogeneous nodes there are, the more time-consuming it is to measure the average execution time. For this reason, the MrHeter method had the disadvantage of being expensive to use.

In this study, we will conduct a demonstration experiment to perform task allocation using Optuna, a Python hyperparameter optimization library. Optuna takes in hyperparameters and an objective function, and outputs the hyperparameter values that minimize or maximize the objective function. Therefore, it is not necessary to measure things such as average execution time in advance, and labor savings can be achieved.

The advantage of using Optuna to automatically obtain parameters to reduce MapReduce execution time is that it requires less effort to things such as measure average execution time than the MrHeter method. The main contributions of this research are as follows. The first point is that, where in the past, the parameters for task allocation had to be measured manually, by using the Python hyperparameter optimization library Optuna, manual measurement becomes unnecessary. The reason that manual measurement is no longer necessary is that when you input the hyperparameters and objective function into Optuna, you can obtain the hyperparameters that minimize the objective function, in other words, the execution time of MapReduce. The second reason is that manual measurement is no longer necessary, so the cost of measurement can be reduced, and a heterogeneous environment consisting of many different types of nodes can be constructed at low cost.

Many studies have been conducted to improve the performance of MapReduce in

heterogeneous environments. LATE is the first study in the academic field to address this issue and has been very successful. LATE identifies tasks with slow execution speeds, called stragglers, and reduces the overall execution time of the job by speculatively executing them on other nodes. TPR, Dolly, and Mantri are also studies that focus on stragglers. In heterogeneous environments, predictable and stable computing power is the main cause of stragglers. However, most research does not investigate the causes of stragglers, but only tries to eliminate their effects, so the problem is not solved at its root. On the other hand, there are also studies that optimize jobs by improving the data imbalance between nodes in a cluster. For example, there are SkewReduce, Topcluster, and Libra. The aim is to distribute data according to the processing power of the nodes, but it is difficult to determine the appropriate coefficient because there is variation depending on the application. There have also been studies on improving MapReduce performance through configuration optimization. However, there are so many parameters that affect job performance that tuning them requires multiple tests, which is inefficient.

Unlike previous research, the MrHeter method identifies the causes of the performance degradation of MapReduce in heterogeneous environments and designs a computational model to solve the problem from the roots. One drawback of the MrHeter method is that it is costly to manually search for and input performance indicators and other data that are used as input variables for the computational model. In particular, measuring the average execution time of local map tasks and remote map tasks for each type of node is costly, so we propose an automated program. We also aim to reduce MapReduce execution time by using Optuna, a black-box optimization method, to skip the manual search for optimal parameters and use automatically derived parameters.

Local map task average execution time and Remote map task average execution time measurements are costly, so we have proposed a program to automate the measurement process. This program calculates the average task execution time for a specified server based on the container ID, which is unique for each job, from the resourcemanager log. The input information required is the path to the log file, the server name to be measured, and the container ID. When measuring the Local map task, the log of the Local map task execution is extracted from the resourcemanager log, and when measuring the Remote map task, the log of the Remote map task execution is extracted from the resourcemanager log, and the path of these log files is specified as input information.

In this paragraph, we will explain how to execute MapReduce task assignment using Optuna. The flow of the code for parameter optimization using Optuna is as follows: First, the results of the previous execution using Optuna are evaluated and a parameter search is performed, and the parameters to be set for the next trial are determined. In the case of the first trial, only the parameter search is performed. Next, the parameters for each server (mapreduce.job.maps and mapreudce.job.reduces) derived by Optuna to

minimize the execution time of the job are set in the Hadoop configuration file mapred-site.xml for each server, and the server is restarted to reflect the settings. This parameter setting and server restart process is performed on all servers. After that, the job to be minimized is executed and the execution time is measured. Optuna evaluates this execution time and specifies the parameters to be used in the next trial. The program ends when the number of times the job is executed reaches the specified number.

The experiment used a heterogeneous cluster environment. We used 14 servers, including one Opteron 2 (AMD Ryzen 7 3700X 8-Core Processor 4.4GHz), one P100 (Intel(R) Xeon(R) CPU E5-2637 v4@ 3.50GHz), and icl00-icl09 (Intel(R) Celeron(R) J4105 CPU @ 1.50GHz) 10 units, and icl10-icl11 (Intel(R) Celeron(R) J4005 CPU @ 2.00GHz) 2 units.

In terms of the cost of the procedure, the procedure for the MrHeter method takes 1+(number of server types * 2) steps, and the procedure for the proposed method takes 4 steps, so we can say that the labor saving of parameter determination using Optuna has been achieved. The more server types there are in a cluster environment, the greater the benefit of using the proposed method to save labor.

From the execution results of the wordcount job, it was found that the proposed method using Optuna was able to reduce the time by 16-50% compared to the default settings of Hadoop. The previous study, the MrHeter method, reduced the time by 4-48% compared to the default settings of Hadoop, so it was found that the proposed method had execution times that were equivalent to or shorter than the MrHeter method.

In this study, the execution time of MapReduce was used as a performance benchmark. In the future, it will be necessary to consider whether it is possible to use the parameters automatically derived by Optuna in situations where it is necessary to satisfy multiple criteria other than execution time, such as power consumption and memory usage.

目 次

図 目 次

表 目 次

第1章　はじめに

GPU，ARM CPU，さらにはFPGAが広く使われるようになり，データセンターはヘテロジニアスクラスターへと発展している．しかし，MapReduceはホモジニアスクラスター向けに設計されており，ヘテロジニアス環境では性能が低い．ヘテロジニアス環境におけるMapReduceの性能低下の原因は，計算能力の異なるノード間の不合理なタスク割り当てである．この原因に基づき，ヘテロジニアス環境におけるMapReduceの実行時間を短縮するMrHeter法が提案されている[1]．MrHeter法は，MapReduce処理をマップシャッフルステージとリデュースステージに分離し，それぞれに対して最適化モデルを構築し，異種ノードの計算能力に基づいた異なるタスク割り当てを行う．ヘテロジニアス環境において，MrHeter法はオリジナルのHadoopと比較して，MapReduceの実行時間を30〜70%短縮することができる．

しかし，MrHeter法を使用するには，異種ノードそれぞれのマップタスクとリデュースタスクの平均実行時間などを知る必要がある．平均実行時間の測定は，異種ノードの数が増えれば増えるほど，手間がかかる．そこで本研究では，Pythonのハイパーパラメータ最適化ライブラリのOptuna[23]を使用してタスク割り当てを行い，実証実験を行う．Optunaは，ハイパーパラメータと目的関数を入力すると，目的関数を最小化または最大化するハイパーパラメータの値を出力する．よって事前に，平均実行時間の測定をする必要がない．

Optunaを使用してMapReduceの実行時間を短縮するためのパラメータを自動的に取得することができると，MrHeter法と比較して平均実行時間の測定の手間がかからないことが利点となる．本研究での主な貢献は以下の通りである．

- タスク割り当てのためのパラメータ測定を従来は人手で測定しなければならなかったところ，Pythonのハイパーパラメータ最適化ライブラリのOptunaを用いることで，人手での測定が不要になることである．人手での測定が不要となるのは，Optunaにハイパーパラメータと目的関数を入力すると，目的関数を最小化する，つまりMapReduceの実行時間を最小化するハイパーパラメータを得ることができるためである．
- 人手での測定が不要になることで，測定のコストを下げることができ，多数の異種ノードによって構成されるヘテロジニアス環境を低コストで構築できることである．

第2章　関連研究

2.1　はじめに

この章では本研究において使用する技術や関連研究を説明する. 本研究のテーマはヘテロジニアス環境における MapReduce の自動タスク割り当てである. Hadoop とは Apache Software Foundation（ASF）によって開発された大規模なデータセットを処理するための分散型フレームワークである. そして MapReduce は Hadoop のコンポーネントの一部でデータ処理タスクを複数のノードに分散し，並列処理によって効率的に処理するフレームワークである. MapReduce には投機的実行という機能がある. これはサーバーに機能差があった場合, 高性能サーバーで割り当てられたタスクが終わった時点で低性能サーバーで終わっていないタスクがあったら高性能サーバーでも並行して実行しより速くタスクを処理したサーバーの結果を採用すると言う機能である.

またヘテロジニアス環境における MapReduce の性能を改善するためこれまでに多くの研究がされてきた. 学術分野では LATE[3] は初めてこの問題に取り組んだ研究で非常に成功している. LATE はストラグラーと呼ばれる実行速度の遅いタスクを特定し他のノードでも投機的に実行することでジョブ全体の実行時間を短縮している. TPR[4], Dolly[5], Mantri[6] もストラグラーに注目した研究である. 一方クラスターのノード間のデータの偏りを改善することでジョブを最適化する研究も存在する. 例えば SkewReduce[7], Topcluster[8], Libra[9] などである. ノードの処理能力に応じてデータを分散させることが目的であるが, アプリケーションによってばらつきがあるため適切な係数を定めることは難しい. MrHeter 法 [1] もクラスターのノード間のデータの偏りの改善を目指した研究である. また設定の最適化から MapReduce のパフォーマンスを向上させる研究もなされた [10, 11]. しかし, ジョブの性能に影響を与えるパラメータは膨大でチューニングするには複数回のテストが必要となり非効率である [12].

2.2　Hadoop

Hadoop は, 大規模なデータセットを処理するための分散型フレームワークであり, Apache Software Foundation（ASF）によって開発されている．Hadoop は，膨大な量のデータを複数のノードに分散し，並列処理によって高速で効率的なデータ処理

を可能にする．主要なコンポーネントとしては，Hadoop MapReduce(MapReduce), Hadoop Distributed File System（HDFS），Hadoop YARN，Hadoop Common がある．

MapReduce は，データ処理タスクを複数のノードに分散し，並列処理によって効率的に処理するフレームワークである [2]．HDFS は，データを複数のノードに分散して格納し，耐障害性を提供する．Hadoop YARN は，リソース管理やジョブスケジューリングを行う．そして，Hadoop Common は他モジュールが利用するライブラリ群である [24].

2.3 MapReduce

MapReduce は 2.2 節でも述べたように Hadoop のコンポーネントの一部で, 分散並列プログラミングモデルであり大量のデータを処理するために設計されている．図 2.1 は MapReduce におけるデータの流れを表している．

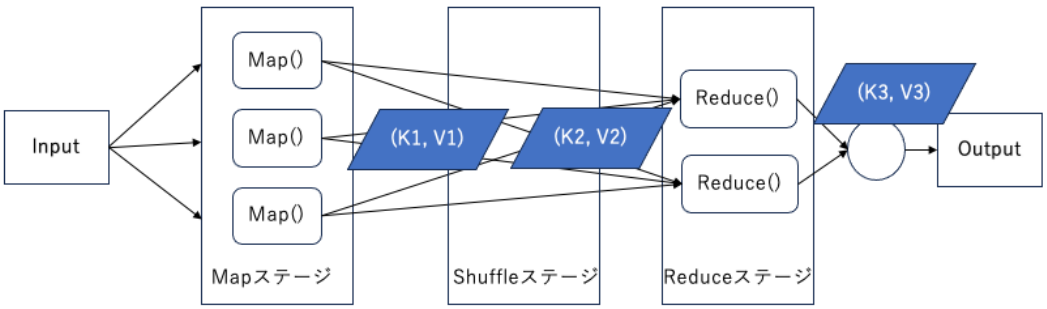

図 2.1: MapReduce におけるデータの流れ

MapReduce の典型的なジョブの実行は，以下のステップで構成される．

1. 図 2.1 の Map ステージでは, ジョブ投入後，Input ファイルは多くの部分に分割され，Map 処理に割り当てられる．
2. 図 2.1 の Shuffle ステージでは,Map タスクの入力タプル (K1, V1) を中間タプル (K2, V2) に変換し，ローカルディスクに出力する．
3. 図 2.1 の Reduce ステージでは,Reduce タスクが実施される．すべての中間タプル (K2, V2) を集め，ソートし最終的なタプル（K3, V3）つまり Output ファイルを生成する．

MapReduce は元々同種のクラスタ用に設計されている．従って，Map ステージにおいて割り当てられる Input ファイルと Reduce ステージで割り当てられる中間タプル (K2, V2) の割り当ては一様である．しかし，異なるデバイスは異なる実行時間を持つ．実行時間は計算能力によって異なるため，本来の MapReduce は異機種混在環境には合わない．ジョブの完了時間は最も遅いジョブに依存する．従って理想的なシーンは，全てのタスクが同時に終了することである [1].

2.3.1 MapReduce の投機的実行について

MapReduce の投機的実行は MapReduce の機能であり Map ステージと Reduce ステージで実行される．しかし，本研究では Map ステージにおける投機的実行が問題となるため，ここでは Map ステージの投機的実行について説明する．

図 2.2 は Map ステージにおける投機的実行の様子を表している．図の灰色の丸は高性能サーバーに割り当てられたタスクを，青色の丸は低性能サーバーに割り当てられたタスクを表現している．それぞれのサーバーに 5 つずつタスクが割り当てられているとする．サーバーに割り当てられたタスクを実行するステージを Local Map ステージと呼ぶため, 図 2.2 において該当のステージに Local Map と記載されている．高性能サーバーで 5 つのタスクを全て終えた時，低性能サーバーではまだ 2 つのタスクしか終えていない．そこで高性能サーバーでも低性能サーバーのタスクを同時に実行し始める．低性能サーバーのタスクを同時に実行する高性能サーバーのステージは Remote Map ステージと呼ばれ, 図 2.2 において該当のステージに Remote Map と記載されている. 高性能サーバーで同時にタスクが実行された結果，低性能サーバーの残り 3 つのタスクは高性能サーバーでより速く実行される．よって Map ステージの出力には高性能サーバーの実行結果が使われ，同時に行われていた低性能サーバーでのタスクの実行結果は破棄される．これが MapReduce の投機的実行である．

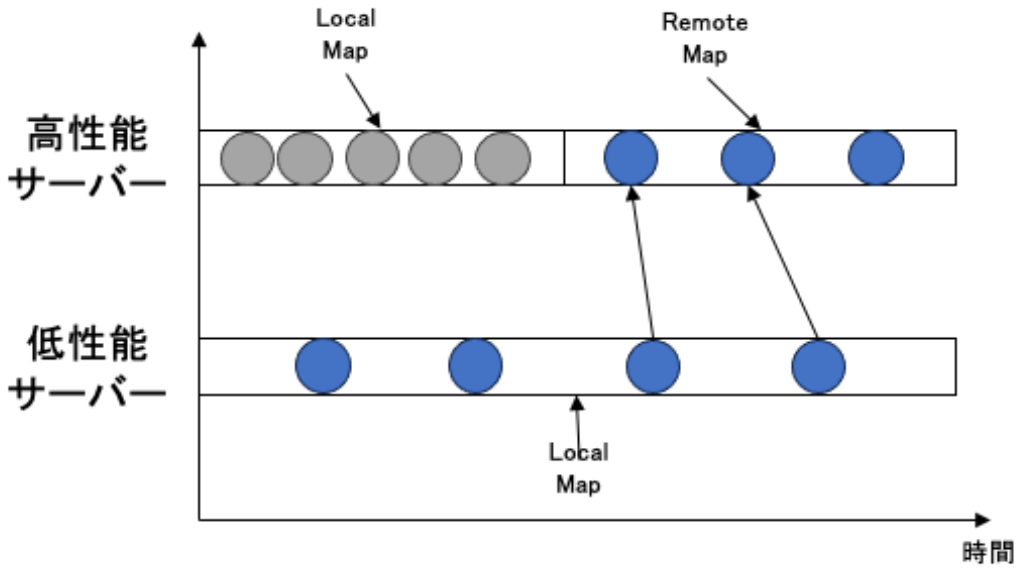

図 2.2: Map ステージにおける投機的実行

2.4 ヘテロジニアス環境における MapReduce 性能改善の研究について

ヘテロジニアス環境における MapReduce の性能を改善するためこれまでに多くの研究がされてきた. 産業界では Hadoop2.2.0+が yarn − .nodemanager.resource.cpu − vcores というパラメータを調整することで物理 CPU 上の仮想 CPU を調整することができる. しかしこの方法には大まかにしか調整できない, ジョブ実行後には調整できないという欠点がある [1].

2.4.1 ストラグラーに着目した研究

学術分野では LATE[3] は初めてこの問題に取り組んだ研究で非常に成功しており多くの研究者の注目を引いている. LATE はストラグラーと呼ばれる実行速度の遅いタスクを特定し他のノードでも投機的に実行することでジョブ全体の実行時間を短縮している. TPR[4], Dolly[5], Mantri[6] もストラグラーに注目した研究である. ヘテロジニアス環境では, 予想可能で安定した計算能力がストラグラーの主な原因である. しかしほとんどの研究は, ストラグラーの原因を調査せず影響を排除しようとするだけで問題を根本から解決していない.

2.4.2 ノードの処理能力に応じてデータを分散させる研究

一方クラスターのノード間のデータの偏りを改善することでジョブを最適化する研究も存在する. 例えば SkewReduce[7], Topcluster[8], Libra[9] などである. ノードの処理能力に応じてデータを分散させることが目的であるが, アプリケーションによってばらつきがあるため適切な係数を定めることは難しい. また MrHeter 法 [1] もクラスターのノード間のデータの偏りを改善することでジョブを最適化する研究の一つである. また本研究では MrHeter 法のジョブ最適化を Optuna によって自動化しタスクの省力化を目指す手法を提案する.

2.4.3 設定の最適化を目指す研究

また設定の最適化から MapReduce のパフォーマンスを向上させる研究もなされた [10, 11]. しかし, ジョブの性能に影響を与えるパラメータは膨大で 70 以上あるため, チューニングするには複数回のテストが必要となり非効率である [12].

2.5 自動化されたハイパーパラメータ最適化手法の研究について

近年，機械学習モデルの性能向上に伴い，ハイパーパラメータ最適化（HPO）は極めて重要な課題となっている．従来のグリッドサーチやランダムサーチでは計算資源の浪費が問題視され，これに代わる効率的な探索手法としてベイズ最適化をはじめとする自動化された HPO 手法が注目されている．実際，多種多様なアプローチが提案されており，それぞれが探索の効率化とコスト削減を目指している．以下では，主要な先行研究についてアプローチ，メリット，デメリットの観点から概説する．

Chen らは，Transformer を用いて大規模なチューニングデータから最適化戦略を学習する HPO アルゴリズム（OptFormer）を提案している [13]．この手法のアプローチは，過去の膨大な試行結果をテキストトークン列としてモデル化し，メタ学習により新たな HPO ポリシーを獲得する点に特徴がある．利点として，大規模データから複雑な事前分布を学習できるため，従来の Gaussian Process に基づくベイズ最適化よりも高精度で校正された予測が可能であることが示されている．一方で，最適化アルゴリズム自体の訓練に大量のデータと計算資源を要し，Transformer モデルの実装も含めて手法が複雑になるため，即座に適用できる汎用的な HPO ソリューションとしては導入のハードルが高いというデメリットも指摘できる．

Wistuba らは，低コストな部分的評価とマルチフィデリティ（多段階予算）でのリソース配分を組み合わせたベイズ最適化手法（DyHPO）を導入している [14]．この研究のアプローチでは，Gaussian Process に学習曲線の動的な挙動を取り込む新たなカーネルを導入し，さらに複数の予算レベルを考慮した獲得関数を提案することで，各候補の訓練継続可否を動的に判断する．利点として，学習曲線の初期段階から将来の性能を見積もることで，不有望な候補へのリソース浪費を削減し，探索効率を大幅に向上させている．実験では多数のデータセット・モデルで既存手法を上回る性能が示されており，限られた予算内での高効率な探索が可能となっている．その反面，複雑なカーネル設計やマルチフィデリティ対応のアルゴリズム実装が必要であるため，手法の実装・適用が難しくなるというデメリットがある．また，学習曲線の挙動を予測に利用する前提上，その仮定が成り立たないタスクでは効果が減少する可能性もある．

Mallik らは, AutoML の文脈で広く用いられる Hyperband 手法に専門家の事前知識を組み込んだ拡張手法（PriorBand）を提案している [15]．このアプローチは，人間の経験に基づく有望なハイパーパラメータの事前分布を設定し，その分布に沿って初期候補を生成した上で，多段階の予算配分戦略（Hyperband）を適用するものである．利点として，ドメイン知識を活用することで探索空間を効率的に絞り込み，わずか約 10 回程度の試行コストといった低予算下でも高精度なモデル性能に到達できることが報告されている．また，良質な事前分布を用いることで探索と活用のバランスを効果的に取り，探索の初期段階から高い性能を発揮する（anytime

性能の向上）点もメリットである．一方で，本手法は有効な事前知識の存在に依存するため，専門家の知見や簡易なプロキシタスクから適切な先験分布を得られない場合には効果が十分発揮されない可能性がある．さらに，Deep Learning 向けに設計された手法であることから，他の分野への適用には追加の検討が必要となるなど，汎用性の面での課題も残る．

Kadra らは，ディープラーニング特有の学習曲線に見られるスケーリング則（べき乗則）に着目し，多予算設定下でのモデル性能を予測する新たなマルチフィデリティ HPO の枠組みを提示している [16]．このアプローチでは，各ハイパーパラメータ構成に対して学習エポック数と性能の関係をパワー則で表現する確率的サロゲートモデルを構築し，それらのアンサンブルに基づいてベイズ最適化を行う．利点として，限られた予算段階（例：短い学習時間や一部データ）から最終的な検証性能を高精度に推定できるため，予算配分の無駄を減らしつつ有望な構成にリソースを集中できる．その結果，画像や自然言語処理を含む多数のベンチマークにおいて最先端の最適化性能を達成しており，ディープラーニングにおける HPO の実用性を大きく向上させている．デメリットとしては，学習曲線がべき乗的に伸びるという前提に依存するため，一部のモデルやデータセットではこの仮定が当てはまらない可能性がある点が挙げられる．また，複数のパワー則モデルを学習・管理する必要があるため，手法の実装と計算コストが増大する可能性も課題である．

実用面に目を向けると，Lindauer らは, 柔軟なベイズ最適化フレームワークである SMAC3 を開発し [17]，Awad らは Differential Evolution と Hyperband を組み合わせた進化的ハイパーパラメータ探索手法 DEHB を提案している [18]．SMAC3 は，順応的なサロゲートモデル（例：ランダムフォレスト）と積極的なレーシング機構を組み合わせることで，少数の評価で高性能な構成を見つけ出すことを可能にしている．このパッケージの利点は，汎用性と拡張性の高さであり，様々な最適化タスクに対してプリセットやインタフェースを提供しつつ，計算コストの削減と高い最適化精度を実現している点である．一方で，強力な汎用ツールであるがゆえにユーザーが調整すべきハイパーパラメータも多く，各ドメインに最適化する際の設定には専門知識を要する場合がある．また，内部でサロゲートモデルの学習やレース方式による評価管理を行うため，単純なランダムサーチと比較して実行基盤の実装が複雑になる可能性がデメリットとして考えられる．DEHB は，モデルフリーの進化アルゴリズムである Differential Evolution（DE）にバンディットベースの資源割り当て戦略 Hyperband を組み合わせた手法であり，ベイズ最適化に代わるシンプルかつ強力なアプローチとなっている．利点として，確率モデルを必要としない簡潔な構成でありながら，Hyperband により不良な構成を早期に淘汰し，DE によりグローバルな探索を行うことで，高い最終精度と優れた途中経過（anytime）性能を両立している．さらに，進化的手法の特性上，並列分散実行が容易でスケーラビリティに優れる点も実務での利点である．デメリットとしては，サロゲートモデルを用いない分，目的関数の形状を学習して予測に活かす

ことができず，極めて高価な評価関数に対してはモデルベースの手法よりも評価回数を要する可能性がある．加えて，進化アルゴリズムのパラメータ（例えば個体数や変異率）の設定次第では性能に影響が及ぶ場合があり，問題によってはベイズ最適化のような精緻なモデル活用が奏功するケースも考えられる．

以上のように，各先行研究はベイズ最適化を核としてそれぞれ異なる方向からHPO の自動化と効率化を達成している．具体的には，探索戦略のメタ学習，マルチフィデリティ評価，専門知識の統合，学習曲線モデルの活用，あるいは汎用的な最適化フレームワークの提供など，各手法が独自の強みを活かして探索精度の向上と計算コスト削減に寄与していることが分かる．一方で，Optuna はベイズ最適化を基盤としつつも，実装の容易さと柔軟性によりユーザーフレンドリーかつ効率的な探索を可能としている点で特徴付けられる．本研究では，これらの先行手法の知見を踏まえ，Optuna の持つ低コストで高性能な実装上のメリットに着目してMapReduce 環境への適用を図り，さらなる効率的な最適化手法の発展を目指す．

また，佐野らの報告では，Web サービス運用時のパフォーマンス向上を目的としてJava VM のパラメータ最適化に Optuna を適用した事例が紹介されている [19]．この実例は，MapReduce におけるパラメータ最適化と共通する課題（高次元な設定空間や本番環境での効率的な探索の必要性）を含んでおり，Optuna の柔軟かつ低コストな実装が実運用環境で効果的に機能することを示唆している．したがって，本研究においても同様の観点から Optuna を採用し，効率的なパラメータ探索によるシステム性能の向上を実現したいと考えている．

2.6 MrHeter 法

2.4.2 節でも触れたが, 本研究の実験において比較対象とする MrHeter 法 [1] についてさらに説明する．MrHeter 法はノードごとの計算能力に基づいたタスク割り当てを行うことでヘテロジニアス環境における MapReduce の性能を改善する手法である. MrHeter 法では MrHeter-MS と MrHeter-R という計算モデルを解いてサーバーのノードに割り振るタスク数を求める. MrHeter-MS は Local map ステージと Remote map ステージ, MrHeter-R は Reduce ステージのタスク数を求めるモデルである．

そして表 2.1 は MrHeter-MS の入力変数, 表 2.2 は MrHeter-MS の出力変数, 表 2.3 は MrHeter-R の入力変数, 表 2.4 は MrHeter-R の出力変数を表している．MrHeter-MS と MrHeter-R は表 2.1 と表 2.3 に示した性能指標を用いて解くため, 事前に計測する必要がある．

MrHeter-MS は式 2.1, 2.2 のような計算式で表される. x は Map ステージと Shuffle ステージの実行時間の合計を示しており, 式 2.1 は Map ステージ, Shuffle ステージの実行時間の合計を最小化することを示している. C_i は異種コンピューティングノードの表現で i は異種ノードの種類を表している．

変数	説明
C_{ij}	異種コンピューティングノードの表現で i は異種ノードの種類 j は数
Cl_{ij}	C_i の Local map タスク平均実行時間
Cr_{ij}	C_i の Remote map タスク平均実行時間
BW	クラスタの帯域幅 (byte/sec)
SM	実行するジョブの種類
DS, KS, TS, CS	各ブロックのデータサイズ / Reduce キー数 / 入力データのデータサイズ / ノード数
k	パラメータ

表 2.1: MrHeter-MS の入力変数

変数	説明
ml_{ij}	C_i の Local map タスク数
mr_{ij}	C_i の Remote map タスク数

表 2.2: MrHeter-MS の出力変数

変数	説明
MST	Map ステージと Shuffle ステージの合計実行時間
Ck_{ij}	C_i の Reduce タスクのキー当たりの平均実行時間
KS	Reduce キー数
p	パラメータ

表 2.3: MrHeter-R の入力変数

変数	説明
r_{ij}	C_i の Reduce タスクのキー数

表 2.4: MrHeter-R の出力変数

式 2.2 の 1 つ目の式は C_i の Local map タスク数と C_i の Local map タスク平均実行時間の乗数と C_i の Remote map タスク数と C_i の Remote map タスク平均実行時間の乗数の合計, つまり Local map ステージと Remote map ステージの実行時間の合計は Map ステージ, Shuffle ステージの実行時間の合計以下だということを示している.

次に式 2.2 の 2 つ目の式は左辺で Shuffle ステージの実行時間を示しており, これは Map ステージ, Shuffle ステージの実行時間の合計以下であることを示している.

式 2.2 の 3 つ目の式の左辺では入力データのデータサイズを各ブロックのデータサイズで割った数, つまり入力データの全ブロック数を表しており, 右辺では Local map タスク数と Remote map タスク数の合計が示されている. 入力データの全ブ

ロック数と Local map タスク数と Remote map タスク数の合計は等しいため等式で繋がれている.

次に式 2.2 の 4 つ目の式の左辺について説明する. サーバーが低性能サーバーの時, mr_{ij} つまり C_i の remote map ステージのタスク数は 0 になる. またサーバーが高性能サーバーの時, ml_{ij} つまり C_i の local map ステージのタスク数は $\frac{TS}{BS} \cdot CS$ で表される各ノードに割り当てられるタスク数と等しくなるため $(ml_{ij} - TS/BS \cdot CS) = 0$ となる. よって左辺はいずれの場合も 0 となるためこの式が成り立つ. 最後に式 2.2 の 5 つ目の式は C_i の local map ステージのタスク数は 0 以上各ノードに割り当てられるタスク数以下となることを示している.

$$\min x \quad \text{subject to} \tag{2.1}$$

$$\begin{cases} (ml_{ij} \cdot Cl_{ij} + mr_{ij} \cdot Cr_{ij}) \leq x \\ Cl_{ij} + \dfrac{k \cdot SM \cdot (ml_{ij} + mr_{ij}) \cdot DS + mr_{ij} \cdot DS}{BW} \leq x \\ TS/DS = \{ml_{11} + \cdots + ml_{ij}\} + \{mr_{11} + \cdots + mr_{ij}\} \\ mr_{ij} \cdot (ml_{ij} - TS/BS \cdot CS) = 0 \\ 0 \leq ml_{ij} \leq TS/BS \cdot CS \end{cases} \tag{2.2}$$

MrHeter-R は式 2.3, 2.4 のような計算式で表される. y は Map ステージ, Shuffle ステージ, Reduce ステージの実行時間合計を示しており, 式 2.3 は Reduce ステージの実行時間を最小化することを示している.

式 2.4 の 1 つ目の式は, Map ステージと Shuffle ステージの合計実行時間と C_i の Reduce タスクのキー当たりの平均実行時間に C_i の Reduce タスクのキー数をかけた時間, つまり Reduce ステージの実行時間を足した時間は Map ステージ, Shuffle ステージ, Reduce ステージの実行時間合計以下となることを示している.

式 2.4 の 2 つ目の式は, 全 Reduce キー数はすべてのノードの Reduce タスクのキー数の合計であることを示している.

式 2.4 の 3 つ目の式は, 各ノードの Reduce タスクのキー数と Reduce タスクのキー当たりの平均実行時間の乗数, つまり各ノードの Reduce ステージの実行時間

が等しいことを示している.

$$\min y \quad \text{subject to} \tag{2.3}$$

$$\begin{cases} MST + p \cdot r_{ij} \cdot Ck_{ij} \leq y \\ KS = r_{11} + r_{12} + \cdots + r_{21} + \cdots + r_{ij} \\ r_{11} \cdot Ck_{11} = r_{21} \cdot Ck_{21} = \cdots = r_{ij} \cdot Ck_{ij} \end{cases} \tag{2.4}$$

2.6.1 MrHeter-MS の入力変数の測定について

表 2.1 で示した MrHeter-MS の入力変数の測定から説明していく. 測定は以下のステップで行う.

1. C_i の Local map タスク平均実行時間と Remote map タスク平均実行時間の測定
2. クラスタの帯域幅 (byte/sec) の測定
3. 各ブロックのデータサイズの測定
4. Reduce キー数の測定
5. ノード数の測定
6. パラメータの測定

C_i の Local map タスク平均実行時間と Remote map タスク平均実行時間の測定について

C_i の Local map タスク平均実行時間と Remote map タスク平均実行時間の測定から説明していく. 測定は次の手順で行う.

1. 初めに実行環境となるクラスター環境においてジョブやインプットデータ等の条件を実験と同じ状態にしてジョブを 1 度実行する. クラスターのマスターサーバーの Hadoop で自動的に出力される resourcemanager のログを取得する. resourcemanager は Hadoop のジョブの進行状況を時系列で表している.
2. 次にサーバーの種類ごとに resourcemanager のログを使用して Local map タスク平均実行時間と Remote map タスク平均実行時間を測定する．Listing2.1 はタスク平均実行時間測定に使用するログの例を示しており, 今回は icl00

というホスト名のサーバーがマスターサーバーのため下線が引かれている hadoop-wakai-resourcemanager-icl00.log というログを使用する.

Listing2.2 は hadoop-wakai-resourcemanager-icl00.log の一部である. ログには Local map タスク実行時と Remote map タスク実行時のログが含まれているのだが, Local map タスク実行時と Remote map タスク実行時のログの識別にはサーバーごとのコンテナの使用状況を利用した. サーバーごとにコンテナが 8 つ使用できるのだがそのコンテナが全て使われていたら Local map タスク実行時, それ以外の場合は Remote map タスク実行時と判断した. Listing2.2 の 1 行目から opteron2 というホスト名のサーバーで 8 つのコンテナが使用されていることが分かる. 一方 12 行目からは opteron2 のサーバーで 4 つのコンテナが使用されていることが分かる. よってこのようなログの場合は 11 行目までを Local map タスク実行時, 12 行目からを Remote map タスク実行時と判断している.

また container_1721048858041_0001_01_000XXX のような ID がジョブごとに割り当てられているので, そのジョブのステータスが ACQUIRED から RUNNING に変わった時刻をジョブの開始時刻とし, ジョブのステータスが RUNNING から COMPLETED に変わった時刻をジョブの終了時刻とする. このようにしてサーバーごとにジョブ数とそれぞれのジョブの実行時間を測定し, 全ジョブの実行時間の合計をジョブ数で割ることでタスク平均実行時間を算出することができる.

Listing 2.1: タスク平均実行時間測定に使用するログ

```
1  wakai@icl00:~/opt/hadoop-3.3.6/logs$ ls -ltr
2  total 796572
3  -rw-rw-r-- 1 wakai wakai 0 Dec 29 2023 SecurityAuth-
       wakai.audit
4  -rw-rw-r-- 1 wakai wakai 818 Jul 2 2024 hadoop-wakai-
       datanode-icl00.out.5
5  -rw-rw-r-- 1 wakai wakai 2954 Jul 2 2024 hadoop-wakai-
       nodemanager-icl00.out.5
6  -rw-rw-r-- 1 wakai wakai 2954 Jul 2 2024 hadoop-wakai-
       nodemanager-icl00.out.4
7  -rw-rw-r-- 1 wakai wakai 818 Jul 2 2024 hadoop-wakai-
       datanode-icl00.out.4
8  -rw-rw-r-- 1 wakai wakai 2954 Jul 2 2024 hadoop-wakai-
       nodemanager-icl00.out.3
9  -rw-rw-r-- 1 wakai wakai 818 Jul 2 2024 hadoop-wakai-
       datanode-icl00.out.3
10 -rw-rw-r-- 1 wakai wakai 2954 Jul 2 2024 hadoop-wakai-
       nodemanager-icl00.out.2
11 -rw-rw-r-- 1 wakai wakai 1040 Jul 2 2024 hadoop-wakai-
       datanode-icl00.out.2
```

```
12 -rw-rw-r-- 1 wakai wakai 2954 Jul 3 2024 hadoop-wakai-
     nodemanager-icl00.out.1
13 -rw-rw-r-- 1 wakai wakai 818 Jul 3 2024 hadoop-wakai-
     datanode-icl00.out.1
14 -rw-rw-r-- 1 wakai wakai 2961 Jul 4 2024 hadoop-wakai-
     nodemanager-icl00.out
15 drwxr-xr-x 2 wakai wakai 4096 Jul 9 2024 userlogs
16 -rw-rw-r-- 1 wakai wakai 159895997 Jul 15 2024 hadoop-
     wakai-nodemanager-icl00.log
17 -rw-rw-r-- 1 wakai wakai 216124846 Jul 15 2024 hadoop-
     wakai-datanode-icl00.log
18 -rw-rw-r-- 1 wakai wakai 1040 Jul 15 2024 hadoop-wakai-
     datanode-icl00.out
19 -rw-rw-r-- 1 wakai wakai 3428 Sep 28 22:28 hadoop-wakai
     -historyserver-icl00.out.5
20 -rw-rw-r-- 1 wakai wakai 4177 Oct 8 21:22 hadoop-wakai-
     historyserver-icl00.out.4
21 -rw-rw-r-- 1 wakai wakai 3419 Oct 12 07:51 hadoop-wakai
     -historyserver-icl00.out.3
22 -rw-rw-r-- 1 wakai wakai 3428 Oct 20 22:40 hadoop-wakai
     -historyserver-icl00.out.2
23 -rw-rw-r-- 1 wakai wakai 3419 Oct 22 21:55 hadoop-wakai
     -historyserver-icl00.out.1
24 -rw-rw-r-- 1 wakai wakai 3428 Oct 23 22:59 hadoop-wakai
     -historyserver-icl00.out
25 -rw-rw-r-- 1 wakai wakai 5435072 Oct 24 07:47 hadoop-
     wakai-historyserver-icl00.log
26 -rw-rw-r-- 1 wakai wakai 268435543 Nov 4 17:10 hadoop-
     wakai-resourcemanager-icl00.log.1
27 -rw-rw-r-- 1 wakai wakai 2977 Nov 9 00:30 hadoop-wakai-
     resourcemanager-icl00.out.5
28 -rw-rw-r-- 1 wakai wakai 818 Nov 9 00:30 hadoop-wakai-
     namenode-icl00.out.5
29 -rw-rw-r-- 1 wakai wakai 2977 Nov 9 00:47 hadoop-wakai-
     resourcemanager-icl00.out.4
30 -rw-rw-r-- 1 wakai wakai 818 Nov 9 00:47 hadoop-wakai-
     namenode-icl00.out.4
31 -rw-rw-r-- 1 wakai wakai 2970 Nov 9 01:05 hadoop-wakai-
     resourcemanager-icl00.out.3
32 -rw-rw-r-- 1 wakai wakai 818 Nov 9 01:05 hadoop-wakai-
     namenode-icl00.out.3
33 -rw-rw-r-- 1 wakai wakai 2970 Nov 9 01:22 hadoop-wakai-
     resourcemanager-icl00.out.2
34 -rw-rw-r-- 1 wakai wakai 818 Nov 9 01:22 hadoop-wakai-
     namenode-icl00.out.2
35 -rw-rw-r-- 1 wakai wakai 2970 Nov 9 13:25 hadoop-wakai-
```

```
    resourcemanager-icl00.out.1
36 -rw-rw-r-- 1 wakai wakai 7096 Nov 9 16:23 hadoop-wakai-
    namenode-icl00.out.1
37 -rw-rw-r-- 1 wakai wakai 2970 Jan 10 15:58 hadoop-wakai
    -resourcemanager-icl00.out
38 -rw-rw-r-- 1 wakai wakai 7096 Jan 10 16:00 hadoop-wakai
    -namenode-icl00.out
39 -rw-rw-r-- 1 wakai wakai 125987334 Jan 11 08:41
    hadoop-wakai-resourcemanager-icl00.log
40 -rw-rw-r-- 1 wakai wakai 39634249 Jan 11 08:41 hadoop-
    wakai-namenode-icl00.log
```

Listing 2.2: resourcemanager のログの一部

```
1 2024-07-27 22:45:28,812 INFO org.apache.hadoop.yarn.
    server.resourcemanager.scheduler.common.fica.
    FiCaSchedulerNode: Assigned container
    container_1721048858041_0001_01_000142 of capacity <
    memory:1024, vCores:1> on
    host opteron2:40529, which has 8 containers, <
    memory:8192, vCores:8> used and <memory:0, vCores:0>
    available after allocation
2 2024-07-27 22:45:28,813 INFO org.apache.hadoop.yarn.
    server.resourcemanager.scheduler.capacity.ParentQueue:
     assignedContainer queue=root usedCapacity=1.0
    absoluteUsedCapacity=1.0 used=<memory:106496, vCores
    :103> cluster=<memory:106496, vCores:104>
3 2024-07-27 22:45:29,516 INFO org.apache.hadoop.yarn.
    server.resourcemanager.rmcontainer.RMContainerImpl:
    container_1721048858041_0001_01_000142 Container
    Transitioned from ALLOCATED to ACQUIRED
4 2024-07-27 22:45:29,807 INFO org.apache.hadoop.yarn.
    server.resourcemanager.rmcontainer.RMContainerImpl:
    container_1721048858041_0001_01_000142 Container
    Transitioned from ACQUIRED to RUNNING
5 2024-07-27 22:45:30,370 INFO org.apache.hadoop.yarn.
    server.resourcemanager.rmcontainer.RMContainerImpl:
    container_1721048858041_0001_01_000121 Container
    Transitioned from RUNNING to COMPLETED
6 2024-07-27 22:45:32,956 INFO org.apache.hadoop.yarn.
    server.resourcemanager.rmcontainer.RMContainerImpl:
    container_1721048858041_0001_01_000140 Container
    Transitioned from RUNNING to COMPLETED
7 2024-07-27 22:45:39,838 INFO org.apache.hadoop.yarn.
    server.resourcemanager.rmcontainer.RMContainerImpl:
    container_1721048858041_0001_01_000135 Container
    Transitioned from RUNNING to COMPLETED
```

```
8 2024-07-27 22:45:41,004 INFO org.apache.hadoop.yarn.
    server.resourcemanager.rmcontainer.RMContainerImpl:
    container_1721048858041_0001_01_000137 Container
    Transitioned from RUNNING to COMPLETED
9 2024-07-27 22:45:41,818 INFO org.apache.hadoop.yarn.
    server.resourcemanager.rmcontainer.RMContainerImpl:
    container_1721048858041_0001_01_000139 Container
    Transitioned from RUNNING to COMPLETED
10 2024-07-27 22:45:41,819 INFO org.apache.hadoop.yarn.
    server.resourcemanager.scheduler.capacity.allocator.
    AbstractContainerAllocator: assignedContainer
    application attempt=
    appattempt_1721048858041_0001_000001 container=null
    queue=default clusterResource=<memory:106496, vCores
    :104> type=RACK_LOCAL requestedPartition=
11 2024-07-27 22:45:41,819 INFO org.apache.hadoop.yarn.
    server.resourcemanager.rmcontainer.RMContainerImpl:
    container_1721048858041_0001_01_000143 Container
    Transitioned from NEW to ALLOCATED
12 2024-07-27 22:45:41,819 INFO org.apache.hadoop.yarn.
    server.resourcemanager.scheduler.common.fica.
    FiCaSchedulerNode: Assigned container
    container_1721048858041_0001_01_000143 of capacity <
    memory:1024, vCores:1> on
    host opteron2:40529, which has 4 containers, <
    memory:4096, vCores:4> used and <memory:4096, vCores
    :4> available after allocation
13 2024-07-27 22:45:41,819 INFO org.apache.hadoop.yarn.
    server.resourcemanager.scheduler.capacity.ParentQueue:
     assignedContainer queue=root usedCapacity=0.96153843
     absoluteUsedCapacity=0.96153843 used=<memory:102400,
     vCores:99> cluster=<memory:106496, vCores:104>
14 2024-07-27 22:45:42,620 INFO org.apache.hadoop.yarn.
    server.resourcemanager.rmcontainer.RMContainerImpl:
    container_1721048858041_0001_01_000138 Container
    Transitioned from RUNNING to COMPLETED
15 2024-07-27 22:45:42,623 INFO org.apache.hadoop.yarn.
    server.resourcemanager.rmcontainer.RMContainerImpl:
    container_1721048858041_0001_01_000143 Container
    Transitioned from ALLOCATED to ACQUIRED
16 2024-07-27 22:45:42,634 INFO org.apache.hadoop.yarn.
    server.resourcemanager.rmcontainer.RMContainerImpl:
    container_1721048858041_0001_01_000141 Container
    Transitioned from RUNNING to COMPLETED
17 2024-07-27 22:45:43,635 INFO org.apache.hadoop.yarn.
    server.resourcemanager.rmcontainer.RMContainerImpl:
```

```
    container_1721048858041_0001_01_000143 Container
    Transitioned from ACQUIRED to RUNNING
18 2024-07-27 22:45:44,147 INFO org.apache.hadoop.yarn.
    server.resourcemanager.rmcontainer.RMContainerImpl:
    container_1721048858041_0001_01_000130 Container
    Transitioned from RUNNING to COMPLETED
19 2024-07-27 22:45:44,150 INFO org.apache.hadoop.yarn.
    server.resourcemanager.scheduler.capacity.allocator.
    AbstractContainerAllocator: assignedContainer
    application attempt=
    appattempt_1721048858041_0001_000001 container=null
    queue=default clusterResource=<memory:106496, vCores
    :104> type=NODE_LOCAL requestedPartition=
20 2024-07-27 22:45:44,151 INFO org.apache.hadoop.yarn.
    server.resourcemanager.rmcontainer.RMContainerImpl:
    container_1721048858041_0001_01_000144 Container
    Transitioned from NEW to ALLOCATED
```

クラスタの帯域幅 (byte/sec) の測定について

次にクラスタの帯域幅 (byte/sec) の測定についてである．測定には iperf という最大帯域幅を測定するためのツールを使用した [20]．バージョンは 3.9 を使用した．Listing2.3 は帯域幅を測定するコマンドを実行した様子である．この結果から 937Mbits/sec を本実験の帯域幅として使用している．

Listing 2.3: 帯域幅測定時のコマンド

```
1 wakai@opteron2:~/opt/iperf-3.9$ iperf3 -s
2 - - - - - - - - - - - - - - - - - - - - - - - - -
3   [ ID] Interval Transfer Bitrate
4   [ 5] 0.00-10.05 sec 1.10 GBytes 937 Mbits/sec
       receiver
5 -----------------------------------------------------------

6 Server listening on 5201
7 -----------------------------------------------------------
```

各ブロックのデータサイズの測定について

次に各ブロックのデータサイズについてである．これは表4.2にあるようにHadoopの設定で DFS block size を 128MB にしているためこの値を使用する．

Reduce キー数の測定について

Reduce キー数を表す KS はジョブ実行時の Hadoop の出力ファイルの Reduece キーをカウントした. Listing2.4 は wordcount のジョブ実行後の出力ファイルを表示した所である. この出力ファイルの内下線を引いた part-r-00000 という出力ファイルの一部を抜粋したのが Listing2.5 である. wordcount の場合出力ファイルを見るとどの単語が入力データに何回出現したかが分かる. 例えば 6 行目を見ると”!important” という文字列が 12 回出現したことが分かる. この文字列の種類が Reduce キー数となるため, wordcount 実行後の出力ファイルの文字列の種類をカウントした. ジョブが grep の場合は出力ファイルには grep の条件の文字列と一致する文字列が含まれる行が出力されており, Reduce キー数はこの行数となるため, wordcount と同様にジョブ実行後の出力ファイルの行数をカウントする.

Listing 2.4: ジョブ実行後の出力ファイル

```
1  wakai@icl00:~/opt/hadoop-3.3.6/etc/hadoop$ hadoop fs -ls
       /user/wakai/mapred/benchmarks/wordcount/output
2  WARNING: HADOOP_PREFIX has been replaced by HADOOP_HOME.
     Using value of HADOOP_PREFIX.
3  Found 101 items
4  -rw-r--r-- 1 wakai supergroup 0 2025-01-10 16:40 /user/
     wakai/mapred/benchmarks/wordcount/output/_SUCCESS
5  -rw-r--r-- 1 wakai supergroup 6583041 2025-01-10 16:40
     /user/wakai/mapred/benchmarks/wordcount/output/part-r-00000
6  -rw-r--r-- 1 wakai supergroup 6655376 2025-01-10 16:40
       /user/wakai/mapred/benchmarks/wordcount/output/part-r
     -00001
7  -rw-r--r-- 1 wakai supergroup 6595255 2025-01-10 16:40
       /user/wakai/mapred/benchmarks/wordcount/output/part-r
     -00002
8  ...
9  /user/wakai/mapred/benchmarks/wordcount/output/part-r
     -00097
10 -rw-r--r-- 1 wakai supergroup 6545203 2025-01-10 16:40
       /user/wakai/mapred/benchmarks/wordcount/output/part-r
     -00098
11 -rw-r--r-- 1 wakai supergroup 6544781 2025-01-10 16:40
       /user/wakai/mapred/benchmarks/wordcount/output/part-r
     -00099
```

Listing 2.5: wordcount 実行後の出力ファイルの一部

```
1  !!!</span></h2> 6
2  !=-1) 132
3  !confirm( 6
```

```
4  !width=20|MF 12
5  !xmlDoc.firstChild) 6
6  "!important" 12
7  ""Aegean 6
8  ""Aye, 6
9  ""Baruch 12
10 ""Catalina 6
11 ""Forward, 12
12 ""Notability"" 6
13 ""Opposing 12
14 ""Public" 6
15 ""Shouting" 6
16 ""USS 12
```

ノード数の測定について

ノード数は同じ種類のサーバーが何台あるかという数を割り当てた．実験環境については詳しくは 4.2 節で説明しているが, 例えば AMD Ryzen7 3700X 8-Core Processor 4.4GHz はクラスター環境で 1 台使用しているので CS は 1 としている．

パラメータの測定について

最後にパラメータであるが, Zhang らの MrHeter-MS の入力変数の測定結果 [1] から逆算したところ k=-0.44 と求められたのでこの値をパラメータとして使用している．

2.6.2 MrHeter-R の入力変数の測定について

表 2.3 で示した MrHeter-R の入力変数の測定を説明していく．測定は以下のステップで行う．

1. Map ステージと Shuffle ステージの合計実行時間の測定
2. C_i の Reduce タスクのキー当たりの平均実行時間と Reduce キー数の測定
3. パラメータの測定

Map ステージと Shuffle ステージの合計実行時間の測定について

Map ステージと Shuffle ステージの合計実行時間である MST はジョブ実行後に resourcemanager のログを確認することで測定できる．Listing2.6 はログにおいて Reduce ステージが始まる箇所である．container_1721048858041_0001_01_000XXX の

ようなIDがジョブごとに割り当てられている. Mapステージのジョブは Mapステージの最初に一斉にスタートする. よって8行目で Mapステージのジョブが全て完了した後に新たに割り当てられたジョブの IDである container_1721048858041_0001_01_000463 は Reduceステージのジョブということになる. よって container_1721048858041_0001_01_000463 のジョブが始まるまでの時間を Mapステージと Shuffleステージの合計実行時間である MST とすることができる.

Listing 2.6: hadoop-wakai-resourcemanager-icl00.log において Reduce ステージが始まる箇所

```
1 2024-07-27 22:53:17,252 INFO org.apache.hadoop.yarn.
    server.resourcemanager.rmcontainer.RMContainerImpl:
    container_1721048858041_0001_01_000358 Container
    Transitioned from RUNNING to COMPLETED
2 2024-07-27 22:53:17,303 INFO org.apache.hadoop.yarn.
    server.resourcemanager.rmcontainer.RMContainerImpl:
    container_1721048858041_0001_01_000416 Container
    Transitioned from RUNNING to COMPLETED
3 2024-07-27 22:53:17,751 INFO org.apache.hadoop.yarn.
    server.resourcemanager.rmcontainer.RMContainerImpl:
    container_1721048858041_0001_01_000414 Container
    Transitioned from RUNNING to COMPLETED
4 2024-07-27 22:53:23,060 INFO org.apache.hadoop.yarn.
    server.resourcemanager.rmcontainer.RMContainerImpl:
    container_1721048858041_0001_01_000411 Container
    Transitioned from RUNNING to COMPLETED
5 2024-07-27 22:53:23,124 INFO org.apache.hadoop.yarn.
    server.resourcemanager.rmcontainer.RMContainerImpl:
    container_1721048858041_0001_01_000412 Container
    Transitioned from RUNNING to COMPLETED
6 2024-07-27 22:53:23,425 INFO org.apache.hadoop.yarn.
    server.resourcemanager.rmcontainer.RMContainerImpl:
    container_1721048858041_0001_01_000357 Container
    Transitioned from RUNNING to COMPLETED
7 2024-07-27 22:53:24,782 INFO org.apache.hadoop.yarn.
    server.resourcemanager.scheduler.capacity.allocator.
    AbstractContainerAllocator: assignedContainer
    application attempt=
    appattempt_1721048858041_0001_000001 container=null
    queue=default clusterResource=<memory:106496, vCores
    :104> type=OFF_SWITCH requestedPartition=
8 2024-07-27 22:53:24,784 INFO org.apache.hadoop.yarn.
    server.resourcemanager.rmcontainer.RMContainerImpl:
    container_1721048858041_0001_01_000463 Container
    Transitioned from NEW to ALLOCATED
```

C_i の Reduce タスクのキー当たりの平均実行時間と Reduce キー数の測定について

次に C_i の Reduce タスクのキー当たりの平均実行時間である Ck_{ij} と Reduce キー数である KS の測定についてである. Ck_{ij} は Reduce ステージの実行時間を Reduce キー数で割って算出する. Reduce ステージの実行時間は Reduce ステージでジョブが開始してから終了するまでの時間であり, ジョブ実行後に resourcemanager のログを確認することで測定できる. KS は 2.6.1 節の Reduce キーの測定についてという項目で説明した方法で求めることができる.

パラメータの測定について

最後にパラメータの p は Zhang らの実験 [1] で使用されている p=1 という値を使用した.

2.6.3 MrHeter 法により最適化されたタスク割り当てについて

図 2.3，図 2.4 は MapReduce の処理を Local map ステージ，Remote map ステージ，Shuffle ステージ，Reduce ステージで表した図で横軸が時間を表している．図中の Local Map は Local Map ステージの実行時間を表している. Remote Map, Shuffle, Reduce も同様にそれぞれのステージの実行時間を表している. また Map Shuffle と表示のある青い縦線は両方のサーバーの Shuffle ステージが終わる時刻を, Reduce と表示のある青い縦線は両方のサーバーの Reduce ステージの終わる時刻を表している. 図 2.3 は MrHeter 法によって最適化される前のタスク割り当てを表している. 異なる計算能力を考慮しない不適切なタスク割り当てで、高性能サーバーと低性能サーバーの Shuffle ステージと Reduce ステージで終了時刻がずれている.

図 2.4 は MrHeter 法によって最適化された後のタスク割り当てを表している. 図 2.3 と比較して高性能サーバーに着目すると Remote map タスク数の割り当てが減り, Reduce タスク数の割り当てが増えている. このように MrHeter 法で算出した最適な Local map タスク数，Remote map タスク数、Reduce タスク数を使用することで，高性能サーバーと低性能サーバーの Shuffle ステージと Reduce ステージで終了時刻が揃っている.

2.7 Optuna の内部動作とベイズ最適化の詳細

Optuna は，機械学習モデルのハイパーパラメータ最適化を自動化する Python ライブラリであり，実行時に探索空間を動的に定義できる [21]. 各試行（*Trial*）でユーザ定義の目的関数が評価され，その結果に基づいて最適なパラメータ組合せを見出すスタディ（*Study*）を構築する.

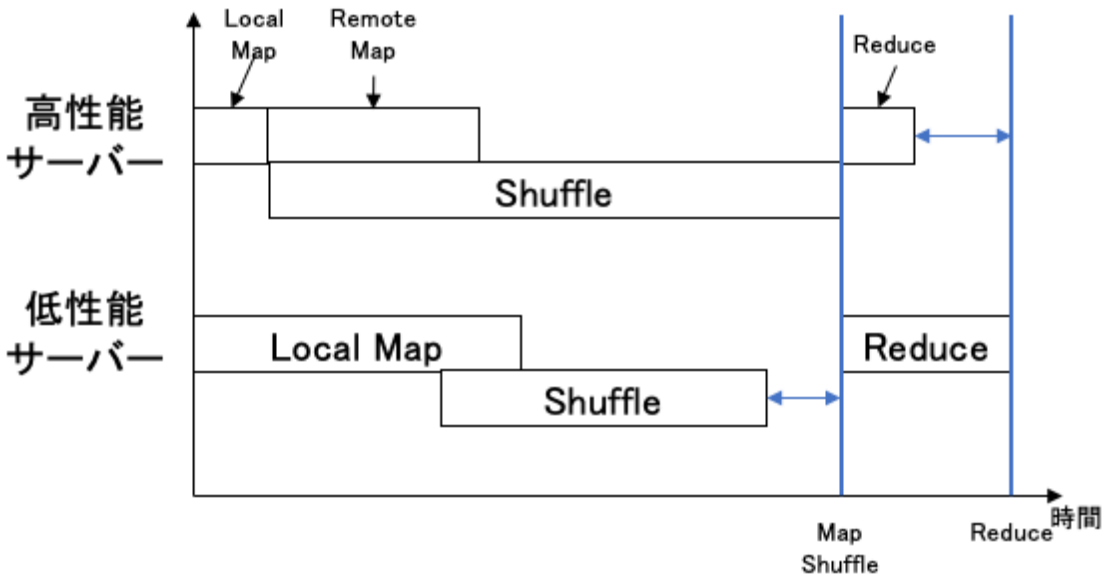

図 2.3: 最適化前のタスク割り当て

Optuna はデフォルトでベイズ最適化の一種である TPE（Tree-structured Parzen Estimator）を採用している．まず初期段階でランダム試行によりデータを収集し，良好な評価値をもたらすパラメータの分布 $l(x)$ とそれ以外の分布 $g(x)$ を推定する．その後，新たな試行では $l(x)/g(x)$ の比率を最大化するパラメータが選ばれ，探索と活用のバランスを取りながら効率的に最適解へ収束する [22].

目的関数は，例えば以下のように定義される．本例は $f(x) = x^2$ の最小化問題を扱うものである．

```
import optuna

def objective(trial):
    x = trial.suggest_float("x", -10, 10)
    return x ** 2

study = optuna.create_study(direction="minimize")
study.optimize(objective, n_trials=50)

print("Best value:", study.best_value)
print("Best params:", study.best_params)
```

上記コードでは，Optuna は初期ランダム試行で得たデータを基に TPE サンプラーを用いて，最も有望なパラメータ x を探索する．パラメータ x の探索範囲は-10 か

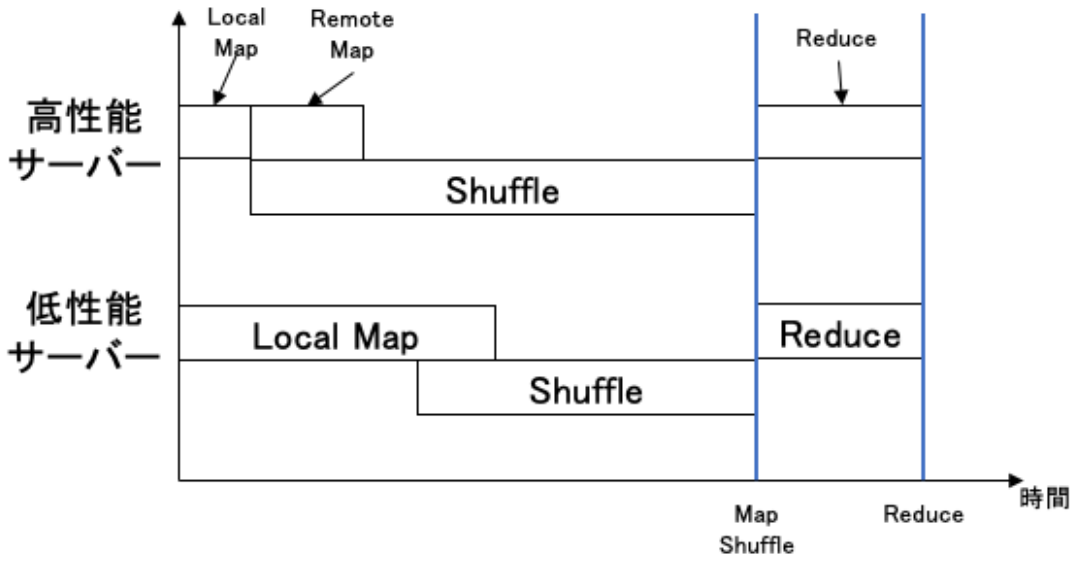

図 2.4: 最適化後のタスク割り当て

ら 10 の間と指定している. 最終的に，最良の目的関数値とそのときのパラメータ組合せが出力される. こうして Optuna は，ベイズ最適化の枠組みを活用しながら柔軟な探索空間定義と効率的なパラメータ探索を実現している [21].

Optuna を使用すると，手動でハイパーパラメータをチューニングする必要がなくなり，より効率的に最適なパラメータを求めることが可能になる.

本手法では, 目的関数は MapReduce におけるジョブの実行時間となる.

2.8 まとめ

ヘテロジニアス環境における MapReduce の性能を改善するためこれまでに多くの研究がされてきた. 学術分野では LATE[3] は初めてこの問題に取り組んだ研究で非常に成功している. LATE はストラグラーと呼ばれる実行速度の遅いタスクを特定し他のノードでも投機的に実行することでジョブ全体の実行時間を短縮している. TPR[4], Dolly[5], Mantri[6] もストラグラーに注目した研究である. ヘテロジニアス環境では, 予想可能で安定した計算能力がストラグラーの主な原因である. しかしほとんどの研究は, ストラグラーの原因を調査せず影響を排除しようとするだけで問題を根本から解決していない. 一方クラスターのノード間のデータの偏りを改善することでジョブを最適化する研究も存在する. 例えば SkewReduce[7], Topcluster[8], Libra[9] などである. ノードの処理能力に応じてデータを分散させることが目的であるが, アプリケーションによってばらつきがあるため適切な係数を定めることは難しい. また設定の最適化から MapReduce のパフォーマンスを向上

させる研究もなされた [10, 11]. しかし, ジョブの性能に影響を与えるパラメータは膨大でチューニングするには複数回のテストが必要となり非効率である [12].

自動化されたハイパーパラメータ最適化手法の研究においては，トランスフォーマーを用いた OptFormer[13]，マルチフィデリティ評価の DyHPO[14]，事前知識を活用する PriorBand[15]，学習曲線のスケーリング則に基づく手法 [16] など，各種自動化 HPO 手法の先行研究を概観した．これらは柔軟なベイズ最適化や低コストな評価戦略を通じて，効率的なパラメータ探索を実現している．一方，Optuna は実装の容易さと柔軟性を兼ね備え，Java VM 最適化 [19] の事例にも示されるように MapReduce 等のシステム最適化にも有用である．本研究では，これらの知見を踏まえ Optuna の利点を活かした最適化手法の発展を目指す.

MrHeter 法 [1] はそれまでの研究と違い, ヘテロジニアス環境における MapReduce のパフォーマンス低下の原因を特定し, 根本から解決するための計算モデルを設計した. MrHeter 法の欠点としては, 計算モデルの入力変数となる性能指標等を人手で探索し入力するためコストがかかる点が挙げられる. 特に 2.6.1 節のステップ 1 で挙げた C_i の Local map タスク平均実行時間と Remote map タスク平均実行時間の測定はコストがかかるため, 3 章で自動化のプログラムを提案する. またブラックボックス最適化手法を用いて人手での最適なパラメータの探索を省略し，自動的に導出されたパラメータを利用することで MapReduce の実行時間短縮も目指す. 次の章では本研究における提案手法について説明する.

第3章　提案手法

3.1　はじめに

第 2 章で説明した MrHeter 法は MrHeter-MS と MrHeter-R という計算モデルを解いてサーバーのノードに割り振るタスク数を求める手法であった. そして各計算モデルの入力変数となる Local map タスク平均実行時間や Remote map タスク平均実行時間等の測定が前もって必要であった. そこで Local map タスク平均実行時間や Remote map タスク平均実行時間の測定を自動化するプログラムを提案する.

またブラックボックス最適化手法を用いて人手での最適なパラメータの探索を省略しタスクの省力化を目指す手法も提案する. 具体的にはハイパーパラメータ最適化のための Python ライブラリである Optuna を使用する. Optuna を各ノードの最適なパラメータの探索処理に使用することで人手での探索の手順を省略することができコストの省力化につながると考えている.

3.2 節では MrHeter 法におけるログ測定の自動化について, 3.3.2 節では提案手法で探索を行う Hadoop のパラメータについて, 3.3.3 節では提案手法の実装について説明する.

3.2　MrHeter 法におけるログ測定の自動化について

Listing3.1 は Local map タスク平均実行時間と Remote map タスク平均実行時間の測定を自動化する Python プログラムである. このプログラムでは resourcemanager のログからジョブごとに一意となるコンテナ ID を元に指定されたサーバーのタスク平均実行時間を算出する. 入力情報として 36 行目にログファイルのパス, 44 行目に測定の対象となるサーバー名, 46 行目にコンテナ ID を指定する必要がある. また Local map タスク測定時には Local map タスク実行時のログを Remote map タスク測定時には Remote map タスク実行時のログを resourcemanager のログから抽出して Listing3.1 のコードで指定するようにした. Local map タスク実行時と Remote map タスク実行時のログの識別にはサーバーごとのコンテナの使用状況を利用した. サーバーごとにコンテナが 8 つ使用できるのだがそのコンテナが全て使われていたら Local map タスク実行時, それ以外の場合は Remote map タスク実行時と判断した.

Listing2.2 は hadoop-wakai-resourcemanager-icl00.log の一部である．1 行目から opteron2 のサーバーで 8 つのコンテナが使用されていることが分かる．一方 12 行目からは opteron2 のサーバーで 4 つのコンテナが使用されていることが分かる．よってこのようなログの場合は 11 行目までを Local map タスク実行時, 12 行目からを Remote map タスク実行時と判断している．よって Cl_{ij} の測定においてはまずジョブを実行する．その後 Local map タスク実行時のログを抽出して Listing3.1 のプログラムで Local map タスク平均実行時間を計算する．ログの抽出と Local map タスク平均実行時間の計算はサーバーの種類ごとに行う．また Cr_{ij} の測定も Cl_{ij} と同様に行う．

Listing 3.1: タスク平均実行時間をログから計算する Python プログラム

```
1  import re
2  from datetime import datetime, timedelta
3
4  def calculate_transition_time(log_file, container_ids):
5      total_time = timedelta() # 合計時間を格納するための変数
6      valid_containers = 0 # 実行時間を計測できたコンテナの数
7
8      for container_id in container_ids:
9          acquired_time = None
10         completed_time = None
11
12         # コンテナに基づく正規表現 ID
13         pattern_acquired = re.compile(rf'{container_id}␣
               Container␣Transitioned␣from␣ACQUIRED␣to␣RUNNING'
               )
14         pattern_completed = re.compile(rf'{container_id}␣
               Container␣Transitioned␣from␣RUNNING␣to␣COMPLETED
               ')
15
16         with open(log_file, 'r') as file:
17             for line in file:
18                 if pattern_acquired.search(line):
19                     acquired_time = datetime.strptime(line.
                           split("␣INFO")[0], '%Y-%m-%d␣%H:%M:%S
                           ,%f')
20                 elif pattern_completed.search(line):
21                     completed_time = datetime.strptime(line.
                           split("␣INFO")[0], '%Y-%m-%d␣%H:%M:%S
                           ,%f')
22                     break # 完了したのでループを抜ける
23
24         if acquired_time and completed_time:
25             transition_time = completed_time -
                   acquired_time
```

```
26              total_time += transition_time
27              valid_containers += 1
28
29      if valid_containers > 0:
30          average_time = total_time / valid_containers
31          return average_time
32      else:
33          return None
34
35  # ログファイルのパス
36  log_file_path = '/log_wordcount_150GB/log_opteron2_local.
        txt'
37
38  # 結果を保存するリスト
39  ids = []
40
41  # ログファイルを読み込む
42  with open(log_file_path, 'r') as file:
43      for line in file:
44          if 'opteron2' in line:
45              # 正規表現を使用してを抽出 ID
46              match = re.search(r'
                    container_1721048858041_0001_01_\d{6}', line)
47              if match:
48                  ids.append(match.group())
49
50  # 抽出されたを表示 ID
51  # print(",\n".join(f"'{container_id}'" for container_id in
        ids))
52
53  # 関数を呼び出して平均実行時間を計算
54  average_time_taken = calculate_transition_time(
        log_file_path, ids)
55  if average_time_taken:
56      print(f'Average␣execution␣time␣for␣the␣containers␣is␣{
            average_time_taken}.')
57  else:
58      print('Could␣not␣calculate␣average␣execution␣time.')
```

3.3 Optunaによる MapReduce タスク割り当て

3.3.1 最適化の対象となるジョブの性質

Optunaの最適化過程では, Optunaの最適化過程に用いる元データを基に最適化しているため, その元データとかけ離れたデータで最適化が進むと, 元データと異なったケースでは最適化が不適合となる.

本研究ではOptunaを用いてMapReduceのタスク割り当ての最適化を行っているため, 提案手法で求めたパラメータを使用する際には, Optunaで最適化した際に実行するジョブとそのジョブが入力とするデータは異なるが, 処理の内容は概ね同様のジョブを動かすことを前提としている.

3.3.2 探索を行うHadoopのパラメータについて

MrHeter法では, 表3.1で示したパラメータを探索している. 探索対象となるml_{ij}, mr_{ij}, r_{ij}は計算モデルMrHeter-MSとMrHeter-Rの出力変数と一致する.MrHeter法の再現実験においてこれらの最適化されたパラメータをHadoopのパラメータに設定しようとしたのだが, ml_{ij}とmr_{ij}については本研究で使用しているHadoopのバージョン3.3.6では合致するパラメータを見つけることができなかった[25]. よって代わりに表3.2に示すHadoopのパラメータにMrHeter法によって算出した最適化されたパラメータを設定することで評価することとする. mapreduce.job.mapsにはMrHeter法で算出したml_{ij}とmr_{ij}の和を設定する. mapreduce.job.reducesにはMrHeter法で算出したr_{ij}を設定する. mapreduce.job.mapsとmapreduce.job.reduces共にmapreduce.framework.nameが「local」の場合設定は無視されるという条件が付いているが, 表4.2にあるように実験環境ではmapreduce.framework.nameにyarnを設定しているためこれら2つの設定は有効である.

また提案手法においてOptunaで探索するパラメータは表3.2に示すパラメータとする. 実装については詳しくは3.3.3節で説明するが, 表3.2の2つのパラメータにOptunaによって最適化された値を設定して試行する手法となる.

変数名	説明
C_{ij}	異種コンピューティングノードの表現でiは異種ノードの種類jは数
ml_{ij}	C_iのLocal mapタスク数
mr_{ij}	C_iのRemote mapタスク数
r_{ij}	C_iのReduceタスクのキー数

表3.1: MrHeter法で最適化されるパラメータ

パラメータ名	説明
mapreduce.job.maps	ジョブあたりのデフォルトのマップタスク数. ただし mapreduce.framework.name が「local」の場合設定は無視される.
mapreduce.job.reduces	ジョブあたりのデフォルトのリデュースタスク数. 通常クラスタのリデュース能力の 99%に設定されノードが故障してもリデュースをシングルウェーブで実行できるようにする.mapreduce.framework.name が「local」の場合設定は無視される.

表 3.2: 評価実験において設定する Hadoop のパラメータ

3.3.3 実装について

Optuna による MapReduce タスク割り当ての実行方法を説明する. 図 3.1 は Optuna によるパラメータ最適化のコードのフローチャートである. また Listing 3.2 は本研究で使用した Optuna によるパラメータ最適化のコード例であり, 実行時に変更する必要がある箇所はイタリック体で下線を引いている.

図 3.1 の流れに沿って実装を説明する. Optuna によるパラメータ最適化のコードの流れとしてはまず Optuna による前回の実行結果の評価とパラメータ探索が行われ, 次回の試行で設定するパラメータが決定する. 初回の試行の場合はパラメータ探索のみ行われる. Listing 3.2 では 80 行目からの objective 関数によって Optuna による評価とパラメータ探索が行われている.

次に Optuna によってジョブの実行時間を最短にするため導出された各サーバーのパラメータ (mapreduce.job.maps と mapreudce.job.reduces) を各サーバーの mapred-site.xml という Hadoop の設定ファイルに設定し, サーバーを再起動して設定を反映させる. このパラメータ設定とサーバー再起動の処理は全サーバーに行う. Listing 3.2 では 1 行目から始まる parameter_set 関数において, Optuna 指定のパラメータをマスターサーバーに設定しサーバーの再起動が行われている. また 18 行目から始まる parameter_set_second 関数において Optuna 指定のパラメータをノードサーバーに設定しサーバーの再起動が行われている.

その後実行時間を最短化する対象のジョブを実行し実行時間を測定する. Listing 3.2 においては 69 行目から始まる execute_job 関数でジョブの実行と実行時間の測定が行われている. この実行時間を Optuna が評価し次の試行で使用するパラメータを指定する. ジョブの実行回数が設定回数に達したらプログラムを終了する.

探索したパラメータは Hadoop の設定ファイルの一つである mapred-site.xml の mapreduce.job.maps と mapreduce.job.reduces である. mapreduce.job.maps と mapreduce.job.reduces はそれぞれ 0〜100 の間で探索するように設定した. Listing 3.2 の 81〜84 行目で 0〜100 の間で探索するように設定している. 範囲を 0〜100 にしたのは Xiao Zhang らが MrHeter 法によって最適化したパラメータが Local map タス

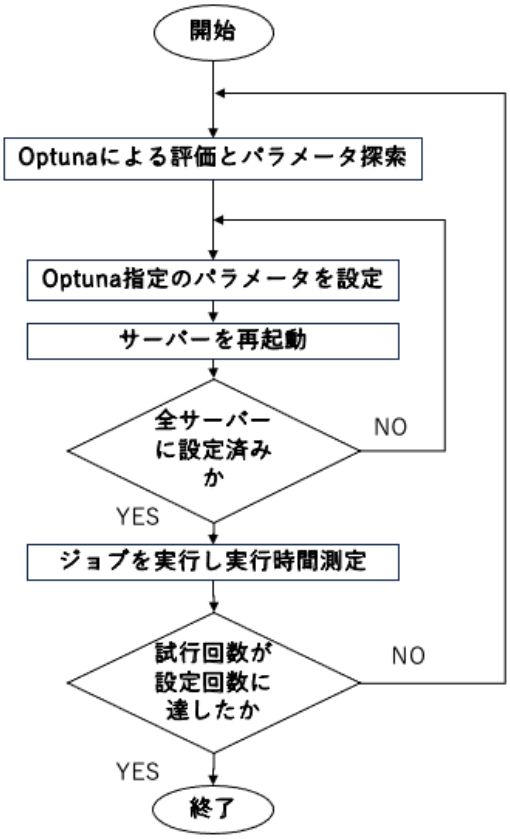

図 3.1: Optuna によるパラメータ最適化のコードのフローチャート

ク数は 0〜3, Remote map タスク数は 0〜19, Reduce タスク数は 1〜52 といずれも 0〜100 の範囲にあるためである. Optuna を使用してジョブの実行時間を最小化するパラメータを探索した. Listing 3.2 の 89 行目において direction に'minimize' を設定することで実行時間を最小化するパラメータを探索している.

本研究では, 分散 Hadoop システムの設定と管理を自動化するために, 以下の 2 つの主要なライブラリを使用した.

1. **'xml.etree.ElementTree'**：Python 標準ライブラリで, XML ファイルの解析と編集を行うために使用した. Hadoop の設定ファイル（例：'mapred-site.xml'）は XML 形式で記述されているため, このライブラリを用いて各サーバーの mapreduce.job.maps と mapreduce.job.reduces のパラメータを動的に読み込み, 変更, 保存した. これにより, Optuna により最適化されたパラメータへの設定変更が可能となる.

2. **'paramiko'**：リモートサーバと SSH で通信を行うための Python ライブラリで, 本研究では, 分散環境における複数のノードに対して設定変更を適用し, サービスを再起動するために使用している. Paramiko を用いて各サーバーに

SSH 接続を確立し, 設定ファイルの編集や必要なサービスの再起動などの操作を自動化している.

これらのライブラリにより、Hadoop 環境における設定パラメータのチューニングが効率的に行うことができ, 分散システム全体の自動化された構成管理が可能となる.

MrHeter 法ではモデルを使って最適な Local map タスク数，Remote map タスク数，Reduce タスク数を算出するが，Optuna を使用することでそれらの処理を自動化することができる．実行時間が変わらなければ，提案手法を用いることでタスクの省力化ができると言える.

Listing 3.2: Optuna によるパラメータ最適化

```
def parameter_set(map_task_master_server,
    reduce_task_master_server):
    tree = ET.parse(</path/to/mapred-site.xml>>)
    root = tree.getroot()

    for prop in root.findall('.//property'):
        name = prop.find('name')
        value = prop.find('value')
        if name and name.text == 'mapreduce.job.maps':
            value.text = str(map_task_master_server)
        if name and name.text == 'mapreduce.job.reduces':
            value.text = str(reduce_task_master_server)

    tree.write(</path/to/mapred-site.xml>>)

    # サーバ再起動
    process = subprocess.call(<< server restart command
        >>, shell=True)

def parameter_set_second(map_task_num, reduce_task_num,
    ip_address):

    # リモートサーバーで実行されるスクリプト
    remote_script = """
import xml.etree.ElementTree as ET
import subprocess
import time

# ファイルのパス
xml_file_path = /path/to/mapred-site.xml

with open(xml_file_path, 'r') as file:
    tree = ET.parse(file)
```

```
31      root = tree.getroot()
32
33  # プロパティを探す
34  for property in root.findall('.//property'):
35      name = property.find('name')
36      value = property.find('value')
37      if name is not None and name.text == 'mapreduce.job.
            maps':
38          # 新しい値を設定
39          value.text = str({})
40      if name is not None and name.text == 'mapreduce.job.
            reduces':
41          # 新しい値を設定
42          value.text = str({})
43
44  # ファイルに書き込み
45  with open(xml_file_path, 'wb') as file:
46      tree.write(file)
47
48  # サーバー再起動
49  process = subprocess.call(server restart command, shell=
        True)
50
51  """.format(map_task_num, reduce_task_num)
52
53      # セッションの確立
54      ssh_client = paramiko.SSHClient()
55      ssh_client.set_missing_host_key_policy(paramiko.
            AutoAddPolicy())
56      ssh_client.connect(ip_address, username=username,
            password=password)
57
58      # リモートサーバーで実行
59      stdin, stdout, stderr = ssh_client.exec_command('
            python3␣-c␣"{}"'.format(remote_script))
60
61      # 出力を表示
62      for line in stdout.readlines():
63          print(line.strip())
64
65      # 接続を閉じる
66      stdin.close()
67
68  def execute_job(map_task_master_server,
        reduce_task_master_server, ...):
69      parameter_set(map_task_master_server,
```

```
        reduce_task_master_server)
70      ...
71      parameter_set_second(map_task_node_server_last,
            reduce_task_node_server_last,
            ip adress node server last)
72
73      start = time.perf_counter()
74      subprocess.call(execute job command, shell=True)
75      end = time.perf_counter()
76
77      return (end - start)
78
79  def objective(trial):
80      map_task_master_server = trial.suggest_int('
            map_task_master_server', 0, 100)
81      reduce_task_master_server = trial.suggest_int('
            reduce_task_master_server', 0, 100)
82      map_task_node_server1 = trial.suggest_int('
            map_task_node_server_first', 0, 100)
83      reduce_task_node_server1 = trial.suggest_int('
            reduce_task_node_server_first', 0, 100)
84      ...
85      return execute_job(map_task_master_server,
            reduce_task_master_server, ...)
86
87  def execute_optuna():
88      study = create_study(direction='minimize')
89      study.optimize(objective, n_trials=trial counts)
90
91  if __name__ == '__main__':
92      execute_optuna()
```

3.4 まとめ

3 章では提案手法について説明した.

MrHeter 法の欠点としては, 計算モデルの入力変数となる性能指標等を人手で探索し入力するためコストがかかる点が挙げられる. 特に 2.6.1 節のステップ 1 で挙げた C_i の Local map タスク平均実行時間と Remote map タスク平均実行時間の測定はコストがかかるため, 測定を自動化するプログラムを提案した.

MrHeter 法において探索を行う Hadoop のパラメータは, MrHeter 法では Local map タスク数, Remote map タスク数, Reduce タスクのキー数の 3 つであった. しかし Hadoop のパラメータで Local map タスク数と Remote map タスク数に合致するものを見つけることができなかった. そのため MrHeter 法の再現実験ではジョブあ

たりのデフォルトのマップタスク数の設定に Local map タスク数と Remote map タスク数の和を設定することとする.

提案手法の実装には, ハイパーパラメータ最適化のための Python ライブラリの他に, XML ファイルの解析と編集が行える Python 標準ライブラリの xml.etree.ElementTree とリモートサーバと SSH で通信を行うための Python ライブラリである paramiko を使用した.

4 章では 3 章で説明した提案手法を実際に実行し評価を行っていく.

第4章　実験・評価

4.1　はじめに

4 章では実験・評価を行う．4.2 節では実験環境について説明する．4.4 節では MrHeter 法と提案手法の手順を比較し, 提案手法のコストを評価している. 4.6 節では実験で測定した実行時間を示している．実験ではデフォルトの Hadoop の設定, MrHeter 法, Optuna を用いた提案手法の 3 つの実行時間を比較した. 実行したジョブは wordcount と grep の 2 種類で, 入力データは Wikipedia のデータを使用しており, データサイズは 50GB, 150GB, 300GB を使用した.

4.2　実験環境

実験では，ヘテロジニアスなクラスター環境を使用している．図 4.1 に実験環境を示す.

サーバーは 14 台使用しており，内訳は高性能なサーバーとして AMD Ryzen7 3700X 8-Core Processor 4.4GHz が 1 台 (ホスト opteron2) と，Intel(R) Xeon(R) CPU E5-2637 v4 @ 3.50GHz が 1 台 (ホスト p100)，低性能なサーバーとして Intel(R) Celeron(R) J4105 CPU @1.50GHz が 10 台 (ホスト icl00〜icl09) ，Intel(R) Celeron(R) J4005 CPU @ 2.00GHz が 2 台 (ホスト icl10〜icl11) である．Intel(R) Xeon(R) CPU E5-2637 v4 @ 3.50GHz は GPU は使用していない．マスターサーバー (ここではホスト icl00 に割り当て) が 1 台で他がノードサーバーの構成となっている．また全てのサーバーで Ubuntu を使用しているがバージョンは Intel(R) Xeon(R) CPU E5-2637 v4 @ 3.50GHz が 20.04.6, Intel(R) Celeron(R) J4105 CPU @1.50GHz が 22.04.5, 他のサーバーは 22.04.4 である. Java Development Kit(JDK) も全サーバーにインストールしたがバージョンは 11.0.21 である.

また表 4.1 に実験環境のサーバーを示す.

表 4.1 の i(異種ノードの種類) と j(数) は, 表 2.1 の変数 C_{ij} の説明にあるように, 異種コンピューティングノードの表現で i は異種ノードの種類, j は同じ種類のノード中の数を示し, ij の組み合わせでサーバーが一意に特定できる.

実験で使用した Hadoop のバージョンは 3.3.6 である [25]. また, 表 4.2 は実験環境で使用した Hadoop のパラメータである. 表 4.3 は探索するパラメータの Hadoop

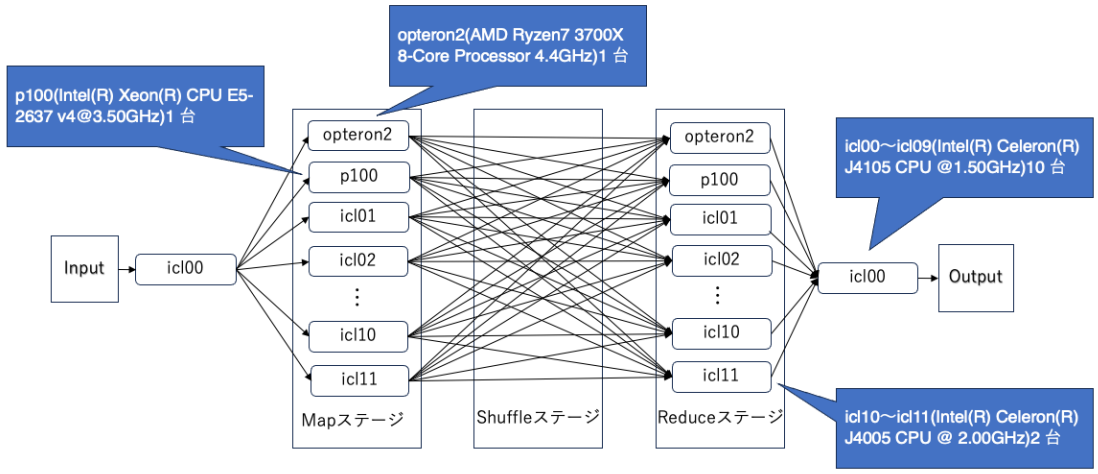

図 4.1: 実験環境

ホスト名	サーバー	i(異種ノードの種類)	j(数)
opteron2	AMD Ryzen7 3700X 8-Core Processor 4.4GHz	1	1
p100	Intel(R) Xeon(R) CPU E5-2637 v4 @ 3.50GHz	2	1
icl00	Intel(R) Celeron(R) J4105 CPU @1.50GHz	-	-
icl01	Intel(R) Celeron(R) J4105 CPU @1.50GHz	3	1
icl02	Intel(R) Celeron(R) J4105 CPU @1.50GHz	3	2
icl03	Intel(R) Celeron(R) J4105 CPU @1.50GHz	3	3
icl04	Intel(R) Celeron(R) J4105 CPU @1.50GHz	3	4
icl05	Intel(R) Celeron(R) J4105 CPU @1.50GHz	3	5
icl06	Intel(R) Celeron(R) J4105 CPU @1.50GHz	3	6
icl07	Intel(R) Celeron(R) J4105 CPU @1.50GHz	3	7
icl08	Intel(R) Celeron(R) J4105 CPU @1.50GHz	3	8
icl09	Intel(R) Celeron(R) J4105 CPU @1.50GHz	3	9
icl10	Intel(R) Celeron(R) J4005 CPU @ 2.00GHz	4	1
icl11	Intel(R) Celeron(R) J4005 CPU @ 2.00GHz	4	2

表 4.1: 実験環境のサーバー

のデフォルトの設定である. 実験で比較対象として Hadoop のデフォルトの設定で実行時間を測定しているが, その際には表 4.3 の値が設定される.

4.3 提案手法の実験内容について

提案手法の実験時の設定内容について説明する. 表 4.4 では提案手法のプログラムに設定した値についてまとめている. 設定した箇所は Listing 3.2 においてイタ

パラメータ	設定値
mapreduce.job.reduce.slowstart.completedmaps	1
mapred.job.shuffle.merge.percent	0.66
mapreduce.framework.name	yarn
DFS block size	128MB
DFS replication factor	1
speculative execution enabled	Yes
Maps to be completed before reduce creation	100%

表 4.2: Hadoop のパラメータ

パラメータ	設定値
mapreduce.job.maps	2
mapreduce.job.reduces	1

表 4.3: Hadoop のデフォルトのパラメータ値

リック体で下線が引かれていた箇所である. /path/to/mapred-site.xml はパラメータを設定する mapred-site.xml のパス, execute job command は wordcount や grep を実行するコマンド, trial counts は試行回数を示している.

またプログラム実行後は Optuna のダッシュボード機能を使用して試行結果を確認することができる [21]. 図 4.2 は Optuna のダッシュボード画面である. ダッシュボードを開くには以下のようなコマンドを実行する.

```
% pip install optuna-dashboard
% optuna-dashboard sqlite:///wordcount_50GB.db
```

コマンドの 1 行目で Optuna のダッシュボード機能をインストールしている. 2 行目ではプログラム実行時に出力した wordcount_50GB.db という名前の db をダッシュボードで可視化している. コマンド実行後 http://localhost:8080 にアクセスするとダッシュボード画面を見ることができる. 図 4.2 と図 4.3 は入力データが 50GB の wordcount のジョブを 20 回試行した際の db を可視化したダッシュボードである.

図 4.2 の上部の History というグラフは Optuna がパラメータを変更し 20 回試行した結果を示している. 横軸が試行回数, 縦軸が実行時間 (sec) になっており実行時間の推移を知ることができる. それ以外のグラフは今回は用いない. History というグラフにおいて試行回数が 0 と 8 の時, 他の試行と比較して実行時間が極端に短くなっている. マスターノードの resourcemanager のログを確認するとジョブが失敗していたため試行を除外する. Listing4.1 はジョブが失敗した時のログの例である. 失敗したジョブを除外すると試行回数が 3 の時が最も実行時間が短いと分かる.

Listing 4.1: ジョブが失敗しているログ

設定した箇所	wordcount	grep
/path/to/mapred-site.xml	/home/wakai/opt/hadoop-3.3.6/etc/hadoop/mapred-site.xml	/home/wakai/opt/hadoop-3.3.6/etc/hadoop/mapred-site.xml
execute job command	hadoop jar /home/wakai/opt/hadoop-3.3.6/share/hadoop/mapreduce/hadoop-mapreduceexamples-3.3.6.jar wordcount /user/wakai/mapred/benchmarks/wordcount/input/wikipedia_50GB /user/wakai/mapred/benchmarks/wordcount/output	hadoop jar /home/wakai/opt/hadoop-3.3.6/share/hadoop/mapreduce/hadoop-mapreduce-examples-3.3.6.jar grep /user/wakai/mapred/benchmarks/wordcount/input/wikipedia_50GB /user/wakai/mapred/benchmarks/grep/output 'technology—science—information'
trial counts	20	20

表 4.4: 提案手法のプログラムに設定した値について

```
1 2024-10-13 13:20:22,233 ERROR org.apache.hadoop.yarn.
    server.resourcemanager.ResourceTrackerService:
    Received finished container :
    container_1728790621912_0001_01_000001 for unknown
    application application_1728790621912_0001 Skipping.
```

図 4.3 は各試行の設定パラメータを見ることができるダッシュボード画面である. 図 4.2 の History のグラフにおいて試行回数が 3 の時, 実行時間が短いと確認したが, 図 4.3 は試行回数が 3 の時の試行の設定パラメータを確認した画面である.

画面中央の Parameter という部分において map_task_opteron2 27, reduce_task_opteron2 88, …という設定パラメータを確認することができる. map_task_opteron2 は opteron2 というホスト名の設定パラメータで, 表 2.2 でいうと変数 ml_{ij} と mr_{ij} を足したパラメータにあたる. opteron2 は表 4.1 から i が 1, j が 1 ということが分かる. よって ml_{11} と mr_{11} を足したパラメータは 27 という最適なパラメータが得られている.

同様に reduce_task_opteron2 は opteron2 というホスト名の設定パラメータで, 表 2.4 でいうと変数 r_{ij} にあたる. よって r_{11} は 88 という最適なパラメータが得られている.

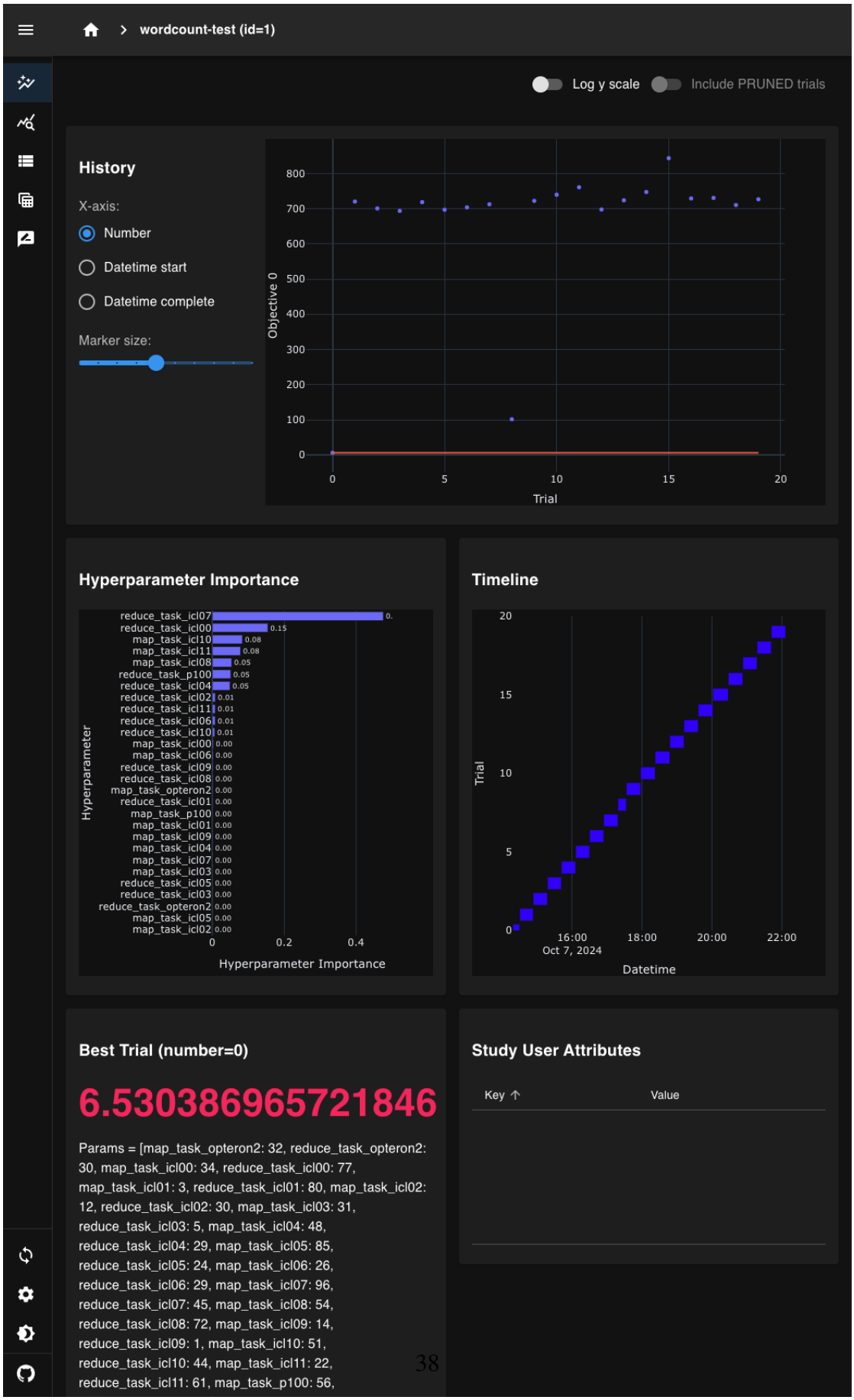

図 4.2: Optuna のダッシュボード画面

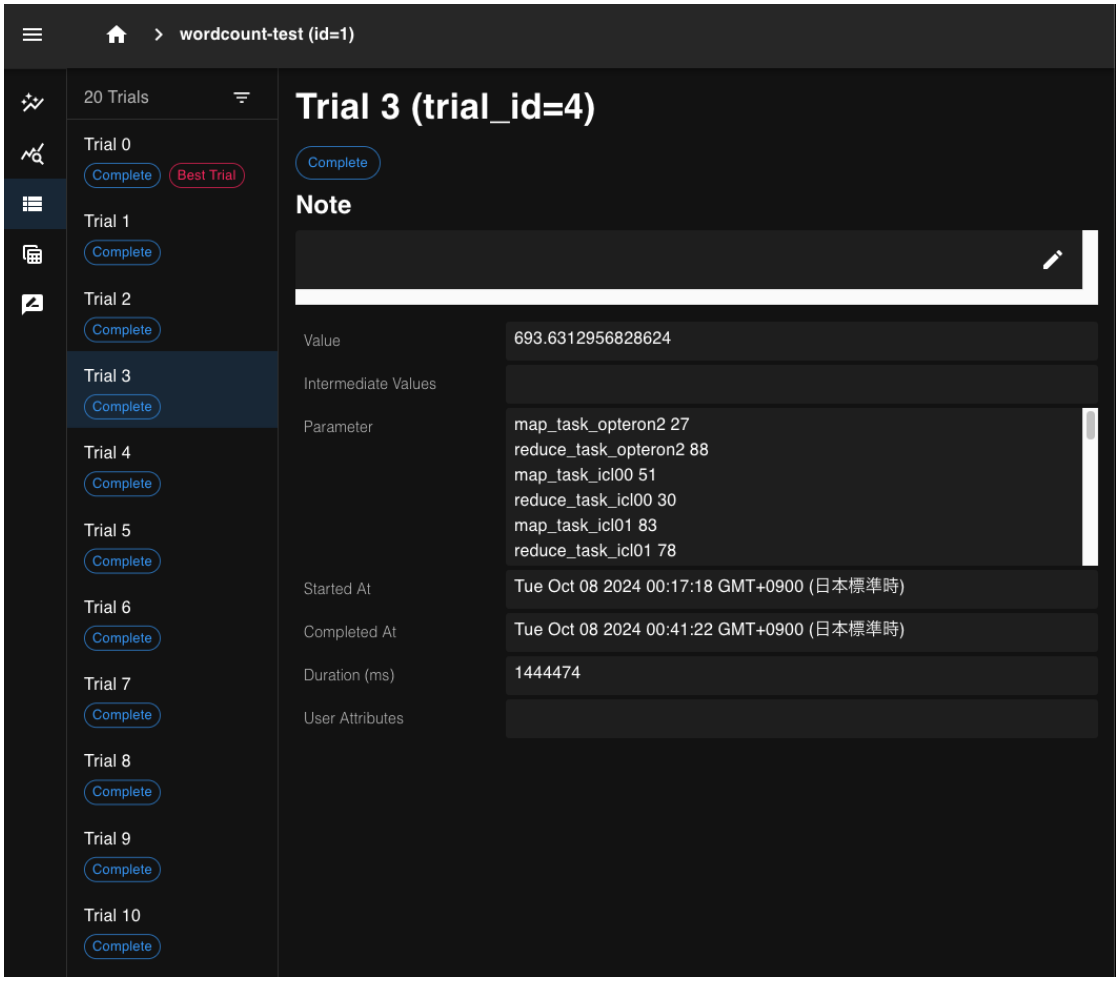

図 4.3: 各試行の設定パラメータのダッシュボード画面

4.4 Optunaによるパラメータ決定の省力化評価

MrHeter法と自動化MrHeter法と提案手法のOptunaによるタスク割り当ての手順を比較する．自動化MrHeter法とはMrHeter法のLocal mapタスク平均実行時間とRemote mapタスク平均実行時間の測定を自動化したPythonプログラムを使用した際のMrHeter法である．図4.4はMrHeter法の手順, 図4.5は自動化MrHeter法の手順, 図4.6は提案手法の手順を表している．また表4.5はMrHeter法, 自動化MrHeter法, Optunaを用いた提案手法の手順を比較した表である．

MrHeter法もしくは自動化MrHeter法は計算モデルを解いてサーバーのノードに最適なタスク数を求める手法である．よって手順として最適化したいジョブを実行し表2.1と表2.3で示した計算モデルを解くのに必要となる性能指標を計測, その後計算モデルを解くことが必要となる．

MrHeter法と自動化MrHeter法を比較するとLocal mapタスク平均実行時間とRemote mapタスク平均実行時間の測定の手順を自動化できたことから自動化MrHeter法の方がタスクを省力化できたと言える．ここで自動的に得られたパラメータは, Cl_{ij}(Local mapタスク平均実行時間) と Cr_{ij}(Remote mapタスク平均実行時間) にあたる．そして, 自動化前のMrHeter法での Cl_{ij} と Cr_{ij} の測定の手順は2.6.1節のステップ1にあたる．

一方Optunaを用いた提案手法では必要な手順はOptunaによるパラメータ最適化のコードを作成し, プログラムを実行, 失敗した試行を除外し, 最適なタスク数を求めることである．プログラムの詳細については3.3.3節で説明している．また, 失敗した試行を除外し, 最適なタスク数を求める手順の詳細については4.3節で説明している．

MrHeter法の手順は1+(サーバーの種類*2) ステップ, Optunaを用いた提案手法の手順は4ステップである．ヘテロジニアス環境の場合サーバーの種類は2種類以上となるため, MrHeter法の手順のステップ数は5以上となる．ステップ数で比較するとOptunaを用いた提案手法の手順の方が少ないステップ数であることからOptunaによるパラメータ決定の省力化が実現できていると言える．

また, MrHeter法はサーバーの種類ごとに性能指標の測定が必要となる項目がある．つまりサーバーの種類が増えるにつれて測定の手間が増えると言える．一方Optunaを用いた提案手法ではサーバーの種類が増えたとしても手間が増えることが無い．よってクラスター環境においてサーバーの種類が多くなる程Optunaを用いた提案手法を使用して省力化するメリットが大きくなる．

4.5 Optunaで得られたパラメータの有効性

表4.6はWikipedia50GBのデータセットでwordcountを実行した際のMrHeter法とOptunaの最適化したパラメータを表した表である．

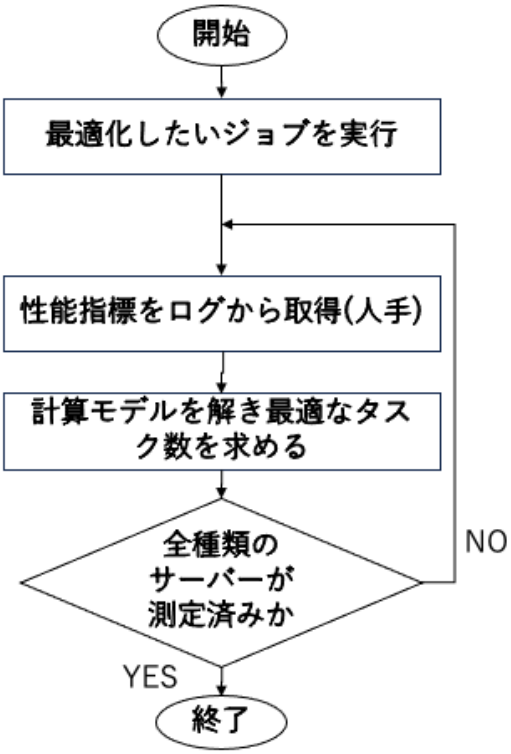

図 4.4: MrHeter 法の手順

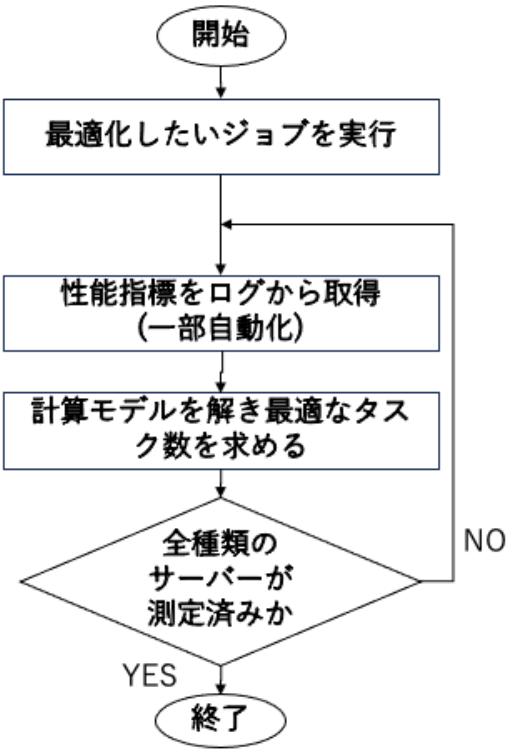

図 4.5: 自動化 MrHeter 法の手順

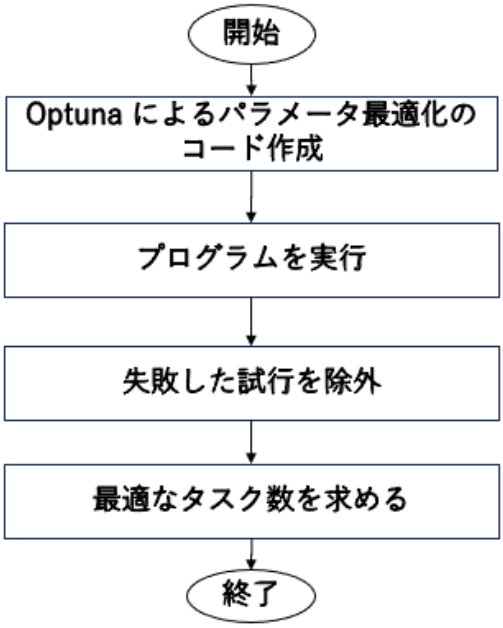

図 4.6: Optuna を用いた提案手法の手順

表 4.5: MrHeter 法と提案手法による手順比較

手法	手順
MrHeter 法	1. 最適化したいジョブを実行する (人手) 2. 次の手順をサーバーごとに繰り返す a. 計算モデルを解くのに必要な性能指標をログから取得 (人手) b. 計算モデルを解き、最適なタスク数を求める (人手)
自動化 MrHeter 法	1. 最適化したいジョブを実行する (人手) 2. 次の手順をサーバーごとに繰り返す a. 計算モデルを解くのに必要な性能指標をログから取得 (一部自動) b. 計算モデルを解き、最適なタスク数を求める (人手)
Optuna	1. Optuna を用いたコードを作成する (テンプレートを用いて人手) 2. プログラムを実行する (自動) 3. 失敗した試行を除外 (人手) 4. 最適なタスク数を求める (自動)

i(異種ノードの種類) と j(数) という列は表 2.1 の変数 C_{ij} の i と j に対応しており, 異種コンピューティングノードの表現で i は異種ノードの種類, j は同じ種類のノード中の数を示している.

MrHeter という列の map と reduce はそれぞれ表 2.2 の変数 ml_{ij}(C_i の Local map タスク数) と変数 mr_{ij}(C_i の Remote map タスク数) を足した数と表 2.4 の変数 r_{ij}(C_i の Reduce タスクのキー数) に対応している. Optuna という列の map と reduce も同様である.

Optuna の最適化したパラメータは提案手法実行後に図 4.3 のような Optuna のダッシュボード機能の中でも各試行の設定パラメータを確認することができる画面から取得した. MrHeter 法では一律に map ステージのパラメータが 1, reduce ステージのパラメータが 100 となったこともあり, サーバーの種類ごとに相関は見られない.

サーバー名	i(異種ノードの種類)	j(数)	MrHeter		Optuna	
			map	reduce	map	reduce
opteron2	1	1	1	100	27	88
p100	2	1	1	100	78	71
icl01	3	1	1	100	83	78
icl02	3	2	1	100	70	78
icl03	3	3	1	100	6	69
icl04	3	4	1	100	29	100
icl05	3	5	1	100	46	14
icl06	3	6	1	100	49	81
icl07	3	7	1	100	24	0
icl08	3	8	1	100	5	64
icl09	3	9	1	100	59	42
icl10	4	1	1	100	9	3
icl11	4	2	1	100	76	55

表 4.6: Wikipedia50GB の wordcount で最適化したパラメータ

4.6 実行時間

wikipedia のデータセット [26] を使用して wordcount, grep のジョブを実行した結果はそれぞれ表 4.7, 4.8 のようになる． また表 4.7, 4.8 の結果を棒グラフにしたのが図 4.7, 4.8 である. データサイズは 50GB, 150GB, 300GB をそれぞれ使用する.

wordcount の実行結果から, Optuna を利用した提案手法は Hadoop のデフォルトの設定と比較して 16〜50%短縮することができた. 先行研究の MrHeter 法は Hadoop

のデフォルトの設定と比較して 4〜48%短縮している. よって, Optuna を利用した提案手法は MrHeter 法と比べて実行時間が同等か短くなることが分かる.

grep の実行結果からは, データサイズが 50GB の時は Optuna の実行時間は Hadoop に比べて約 75%, MrHeter 法に比べて約 79%短縮していることが分かる. 一方データサイズが 150GB の時は Optuna の実行時間は Hadoop に比べて約 5%増加, MrHeter 法に比べて約 8%短縮していることが分かる. データサイズが 300GB の時は Optuna の実行時間は Hadoop に比べて約 5%減少, MrHeter 法に比べて約 10%短縮していることが分かる. よって, Optuna を利用して提案手法は grep においてはデータサイズ 50GB では MrHeter 法と比べて約 79%短縮と大幅に短縮できるがデータサイズ 150GB〜300GB と増えると, 短縮の割合は約 8〜10%と小さくなることがわかる.

この結果から実行するジョブによっては, データ量が増えると Hadoop のパラメータチューニングで得られる実行時間短縮に限界があると推測される. これ以上実行時間を短縮するとしたらリソースを増やすなどの方法が考えられる.

データサイズ	Hadoop	MrHeter	Optuna
50GB	1440	1385	693
150GB	3511	1838	1812
300GB	4429	3578	3722

表 4.7: wordcount の実行時間 (sec)

データサイズ	Hadoop	MrHeter	Optuna
50GB	438	528	111
150GB	1084	1228	1139
300GB	2175	2303	2083

表 4.8: grep の実行時間 (sec)

4.6.1 提案手法の測定結果における信頼区間について

本研究では Hadoop の実行時間を短縮するパラメータ検索を行うにあたり, 得られた実行時間を統計的に評価する必要がある. 特に実行時間のばらつきを考慮した上で，ある程度の信頼度をもって結果を示すために信頼区間の導入を行う. 以下にその理由を述べる.

1. **実行時間計測の不確実性**：実行環境にはネットワーク負荷やクラスタ構成, ジョブの種類などさまざまな要因が介在する. そのため, 同じパラメータ設定でも実行時間にばらつきが生じるのが一般的である. こうしたばらつきを無視して,「最短実行時間」だけを評価すると実際の運用では再現しにくい結果となる可能性がある.

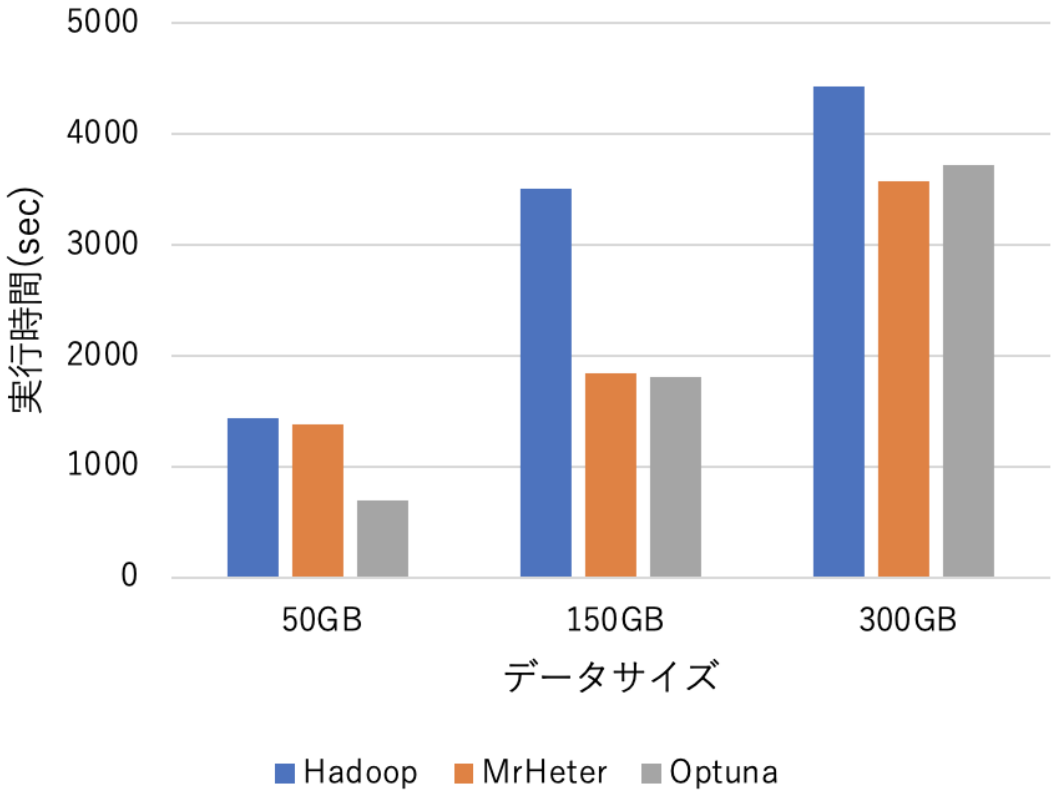

図 4.7: wordcount の実行時間

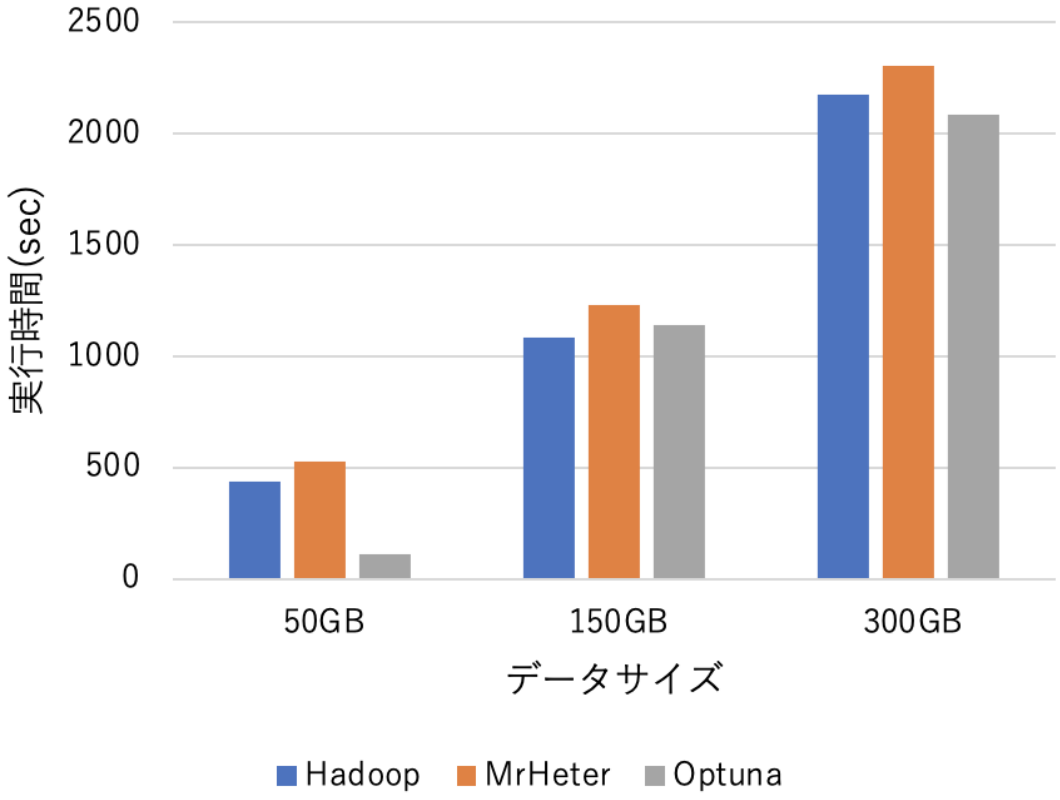

図 4.8: grep の実行時間

2. **信頼区間の意義**：信頼区間とは, サンプルから推定した母集団の平均がどの範囲にあると考えられるかを一定の確率で示すものである. またこの確率は信頼水準である. 95%の信頼区間だとしたら, 95%の確率でこの区間に母集団の平均実行時間が入ると言える.

3. **試行回数を評価する観点**：

    - 1 点目は信頼区間が十分に狭くなるかである. 例えば 20 回試行した結果の平均実行時間の 95%信頼区間が±数十秒程度に収まるのであれば, 追加の試行をしても得られる平均値は信頼区間の範囲である可能性が高い.
    - 2 点目は実測結果のばらつきを踏まえて, 信頼区間幅が実運用上許容可能かどうかである. 例えば 30 秒程度のばらつきは問題にならないのであれば, その範囲に収まっていれば試行回数は十分と言える.

4.6.2 提案手法の測定結果における信頼区間の評価

まずパラメータ探索における試行結果から, 実行時間の平均値および標本標準偏差を求める. その上で各実験に対して 95%信頼区間を算出する. ただし提案手法は optuna の試行回数は 20 回と設定しているが, 試行する時にエラー等が起きてジョブが成功しない試行がある場合その結果は平均や標本標準偏差を計算する時には含めない. また信頼区間は以下の式によって算出する [27]. この式は μ が母集団の平均値, $\bar{x}$ が標本の平均値, t が自由度 n-1 の t 分布表から両側 α%の枠外に出ないための上側臨界値, s が標本標準偏差, n がデータ数を表す.

$$\mu = \bar{x} - t \times \frac{s}{\sqrt{n-1}}$$

表 4.9 は提案手法におけるデータサイズ 50GB の wordcount の実行時間を表している. 試行回数が 1, 9 回目はエラー等の影響でジョブが成功していないため計算対象から除く. 表 4.9 の計測結果を上記の式で解くと, $\bar{x}$=726, t が自由度 17 の t 分布表から両側 2.5%の枠外に出ないための上側臨界値なので t=2.11, s=34, n=18 なので, μ = [709, 743] となる.

表 4.10 は提案手法の測定結果から算出した信頼水準 95%の信頼区間について示した表である. 例えばジョブが wordcount でデータサイズが 50GB の場合, 信頼水準 95%の信頼区間は 709〜743 秒であった. ここから真の平均実行時間は 709〜743 秒の範囲にあるということが 95%の信頼水準で推定できる. 信頼区間は± 17 秒に収まった.

また図 4.9 は Optuna のダッシュボード画面で, 提案手法におけるデータサイズ 50GB の wordcount の実行時間を表したグラフになっている. 横軸が試行回数で縦軸が実行時間 (sec) を表している. 試行回数 20 回の内の最短実行時間は試行回数が

3(表 4.9 における試行回数 4 回目) で記録している. このグラフを見ると, エラー等の影響でジョブが成功せず極端に短い実行時間を記録している試行回数 0, 8(表 4.9 における試行回数 1, 9 回目) を除くと, 実行時間は収束する傾向にあることが分かる. このことから提案手法においては, 20 回程度の試行を行うことで十分に局所解に近づいている可能性が高いと判断できる.

試行回数	実行時間 (sec)
1	7
2	720
3	701
4	694
5	718
6	697
7	704
8	712
9	102
10	722
11	740
12	761
13	697
14	724
15	747
16	843
17	729
18	730
19	710
20	727

表 4.9: 提案手法におけるデータサイズ 50GB の wordcount の実行時間

ジョブ	データサイズ (GB)	試行回数	平均 (sec)	標本標準偏差	信頼区間 (sec)
wordcount	50	18	726	34	709〜743
wordcount	150	18	2013	342	1843〜2183
wordcount	300	19	4144	832	3743〜4545
grep	50	19	132	16	124〜140
grep	150	19	1192	35	1175〜1209
grep	300	19	2185	38	2167〜2203

表 4.10: 提案手法の信頼水準 95%の信頼区間について

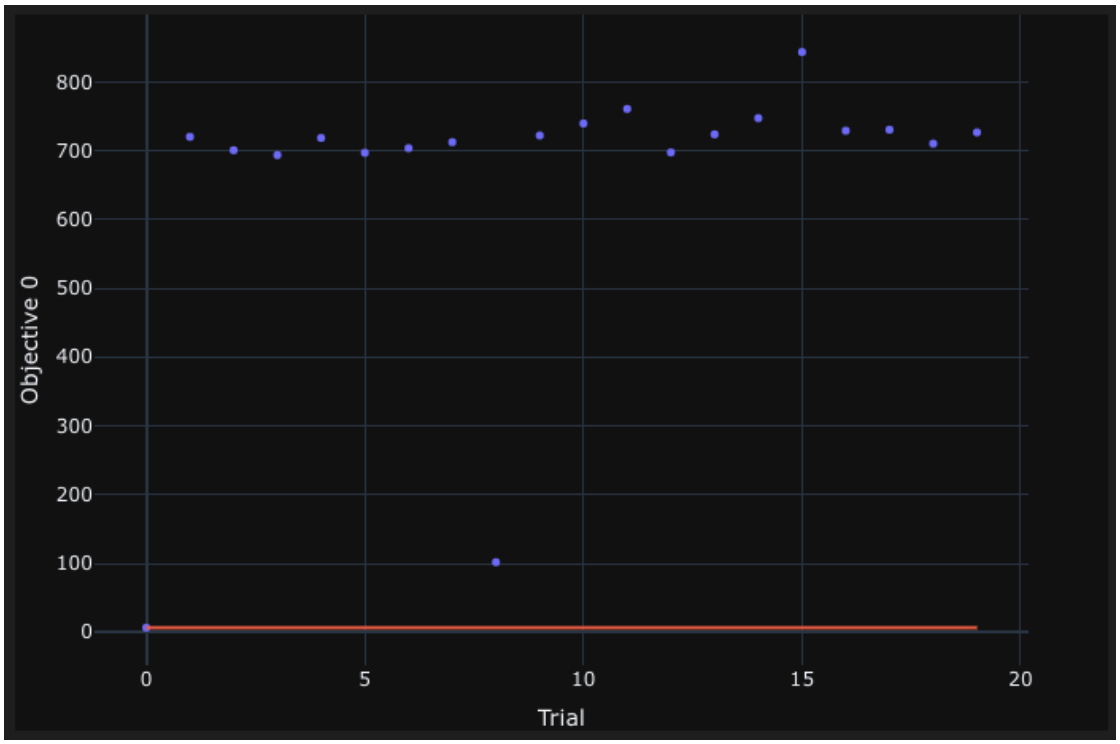

図 4.9: 提案手法におけるデータサイズ 50GB の wordcount の実行時間

4.7 まとめ

4 章では実験・評価を行なった．今回の実験ではヘテロジニアスなクラスター環境を使用した．サーバーは 14 台使用しており，内訳は opteron2(AMD Ryzen7 3700X 8-Core Processor 4.4GHz)1 台，p100(Intel(R) Xeon(R) CPU E5-2637 v4 @ 3.50GHz)1 台，icl00〜icl09(Intel(R) Celeron(R) J4105 CPU @1.50GHz)10 台, icl10〜icl11(Intel(R)Celeron(R) J4005 CPU @ 2.00GHz)2 台である．

4.4 節では手順のコストについて評価した. MrHeter 法と自動化 MrHeter 法を比較すると Local map タスク平均実行時間と Remote map タスク平均実行時間の測定の手順を自動化できたことから自動化 MrHeter 法の方がタスクを省力化できたと言える. MrHeter 法の手順は 1+(サーバーの種類*2) ステップ, 提案手法の手順は 4 ステップかかることから Optuna によるパラメータ決定の省力化が実現できた. また, MrHeter 法はサーバーの種類ごとに性能指標の測定が必要となる項目があるのでサーバーの種類が増えるにつれて測定の手間が増えると言えるが Optuna を使用した提案手法ではサーバーの種類が増えたとしても手間が増えることが無いのでクラスター環境においてサーバーの種類が多くなる程提案手法を使用して省力化するメリットが大きくなる.

4.6 節では実行時間を評価した. wordcount の実行結果から, Optuna を利用した提案手法は Hadoop のデフォルトの設定と比較して 16〜50%短縮することができた. 先行研究の MrHeter 法は Hadoop のデフォルトの設定と比較して 4〜48%短縮して

いるため提案手法は MrHeter 法と比べて実行時間が同等か短くなることが分かった. grep の実行結果からは, Optuna を利用した提案手法の実行時間はデータサイズ 50GB では MrHeter 法と比べて約 79%短縮と大幅に短縮できるがデータサイズ 150GB〜300GB と増えると, 短縮の割合は約 8〜10%と小さくなることが分かった. この結果から実行するジョブによっては, データ量が増えると Hadoop のパラメータチューニングで得られる実行時間短縮に限界があると推測される.

4.6.2 節では提案手法の信頼水準 95%の信頼区間を示した. 例えば実行するジョブが wordcount でデータサイズが 50GB の場合, 信頼水準 95%の信頼区間は 709〜743 秒であり, 信頼区間は± 17 秒に収まった. また提案手法におけるデータサイズ 50GB の wordcount の実行時間を表したグラフを見ると実行時間は収束する傾向にあることが分かる. このことから提案手法においては, 20 回程度の試行を行うことで十分に局所解に近づいている可能性が高いと判断できた.

第5章　おわりに

本研究では, Optuna を用いたヘテロジニアス環境における MapReduce タスク割り当てを提案し, 既存のパラメータを人手で探索し入力する必要がある MrHeter 法と比較することで, 省力化の評価を行なった. 提案手法では Optuna を用いてパラメータを自動で導出した. MrHeter 法の手順は 1+(サーバーの種類*2) ステップ, Optuna を用いた提案手法の手順は 4 ステップである. よって Optuna を用いた提案手法はヘテロジニアス環境における MapReduce タスク割り当てにおいて MrHeter 法と比較してタスクの省力化をすることができた.

一方, grep の実行時間はデータサイズが 50GB の時は Optuna の実行時間は Hadoop に比べて約 75%, MrHeter 法に比べて約 79%短縮しているが, データサイズが 150GB の時は Optuna の実行時間は Hadoop に比べて約 5%増加, MrHeter 法に比べて約 8%短縮, データサイズが 300GB の時は Optuna の実行時間は Hadoop に比べて約 5%減少, MrHeter 法に比べて約 10%短縮となった. wordcount の結果と比較して提案手法の Hadoop に対する実行時間短縮の割合が少ないことから, データ量が増えると Hadoop のパラメータチューニングで得られる実行時間短縮に限界があると推測される. データ量と Hadoop のパラメータチューニングで得られる実行時間短縮の関係については今後の研究で確かめたいと考えている.

今回の研究では, MapReduce の実行時間を性能の基準としていた. 実行時間以外の消費電力, 使用メモリなど複数の基準を満たす必要がある場面において Optuna によって自動的に導出されたパラメータを利用することができるか確かめることが今後の課題として考えられる.

参考文献

[1] Xiao Zhang, Yanjun Wu, and Chen Zhao, ”MrHeter: improving MapReduce performance in heterogeneous environments,” Cluster Computing, mo.19, pp.1691-1701, Sep.2016. DOI: 10.1007/s10586-016-0625-2

[2] Apache Software Foundation, ”MapReduce Tutorial,” Apache Software Foundation, https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html, Dec 2024.

[3] Matei Zaharia, Andy Konwinski, Anthony D. Joseph, Randy Katz, and Ion Stoica, ”Improving MapReduce Performance in Heterogeneous Environments,” 8th USENIX Symposium on Operating Systems Design and Implementation, pp.29-42, Dec.2008.

[4] Xia Zhaol, Kai Kang, YuZhong Sun, Yin Song, Minhao XU, and Tao Pan, ”Insight and Reduction of Map Reduce Stragglers in Heterogeneous Environment,” IEEE International Conference on Cluster Computing (CLUSTER), pp.1–8, 2013.

[5] Ganesh Ananthanarayanan, Ali Ghodsi, Scott Shenker, and Ion Stoica, ”Effective Straggler Mitigation: Attack of the Clones,” 10th USENIX Symposium on Networked Systems Design and Implementation, pp.185–198, 2013.

[6] Ganesh Ananthanarayanan, Srikanth Kandula, Albert Greenberg, Ion Stoica, Yi Lu, Bikas Saha, and Edward Harris, ”Reining in the Outliers in Map-Reduce Clusters using Mantri,” OSDI, vol.10, pp.24, 2010.

[7] YongChul Kwon, Magdalena Balazinska, Bill Howe, and Jerome Rolia, ”Skew-Resistant Parallel Processing of Feature-Extracting Scientific User-Defined Functions,” Proceedings of the 1st ACM Symposium on Cloud Computing, SoCC ’10, pp.75–86, 2010.

[8] Gang Chen, Yongwei Wu, Jie Wu, and Weimin Zheng, ”TopCluster: A hybrid cluster model to support dynamic deployment in Grid,” Journal of Computer and System Sciences, VOL.79, ISSUE.2, pp.201-215, Mar.2013.

[9] Qi Chen, Jinyu Yao, and Zhen Xiao, ”LIBRA: Lightweight Data Skew Mitigation in MapReduce,” IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, VOL.26, NO.9, pp.2520-2533, Sep.2015.

[10] Herodotos Herodotou, Fei Dong, and Shivnath Babu, ”MapReduce Programming and Costbased Optimization? Crossing this Chasm with Starfish,” Proceedings of the VLDB Endowment, Vol.4, No.12, 2011.

[11] Dazhao Cheng, Jia Rao, Yanfei Guo, and Xiaobo Zhou, ”Improving MapReduce Performance in Heterogeneous Environments with Adaptive Task Tuning,” Proceedings of the 15th International Middleware Conference, pp.97–108, 2014.

[12] Min Li, Liangzhao Zeng, Shicong Meng, and Jian Tan, ”MRONLINE: MapReduce online performance tuning,” Proceedings of the 23rd International Symposium onHigh-performance Parallel and Distributed Computing, pp.165–176, Jun.2014. DOI:10.1145/2600212.2600229

[13] Yutian Chen, Xingyou Song, Chansoo Lee, Zi Wang, et al. 2022. ”OptFormer: Towards Learning Universal Hyperparameter Optimizers with Transformers.” In NeurIPS.

[14] Martin Wistuba, Arlind Kadra, and Josif Grabocka. 2022. ”DyHPO: Supervising the Multi-Fidelity Race of Hyperparameter Configurations.” In NeurIPS.

[15] Neeratyoy Mallik, Edward Bergman, Carl Hvarfner, Danny Stoll, et al. 2023. ”PriorBand: Practical Hyperparameter Optimization in the Age of Deep Learning.” In NeurIPS.

[16] Arlind Kadra, Maciej Janowski, Martin Wistuba, and Josif Grabocka. 2023. ”Scaling Laws for Hyperparameter Optimization (Deep Power Laws).” In NeurIPS.

[17] Marius Lindauer, Katharina Eggensperger, Matthias Feurer, André Biedenkamp, et al. 2022. ”SMAC3: A Versatile Bayesian Optimization Package for Hyperparameter Optimization.” Journal of Machine Learning Research.

[18] Noor Awad, Neeratyoy Mallik, and Frank Hutter. 2021. ”DEHB: Evolutionary Hyperband for Scalable, Robust and Efficient Hyperparameter Optimization.” In IJCAI.

[19] 佐野正太郎, 秋葉拓哉, 今村秀明, 太田健, 水野尚人, 柳瀬利彦, ”Optuna によるブラックボックス最適化,” pp.108-110, 株式会社 オーム社, 東京, 2023.

[20] Jon Dugan, Seth Elliott, Bruce A. Mah, Jeff Poskanzer, and Kaustubh Prabhu, ”Iperf.fr,” ikoula HEBERGEUR CLOUD, https://iperf.fr/, Jan 2025.

[21] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. 2019. ”Optuna: A Next-generation Hyperparameter Optimization Framework.” In KDD.

[22] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. 2011. ”Algorithms for Hyper-Parameter Optimization.” In NIPS.

[23] Preferred Networks, Inc, ”OPTUNA,” Preferred Networks, Inc, https://optuna.org/, Nov 2024.

[24] Apache Software Foundation, ”Apache Hadoop,” Apache Software Foundation, https://hadoop.apache.org/, Nov 2024.

[25] Apache Software Foundation, ”Apache Hadoop 3.3.6,” Apache Software Foundation, https://hadoop.apache.org/docs/r3.3.6/, Dec 2024.

[26] Faraz Ahmad, Seyong Lee, Mithuna Thottethodi and T. N. Vijaykumar, ”PUMA Benchmarks and dataset downloads,” Faraz Ahmad, Seyong Lee, Mithuna Thottethodi and T. N. Vijaykumar, https://engineering.purdue.edu/ puma/datasets.htm, Nov 2024.

[27] 大村平 ”統計のはなし,” pp.120-124, 株式会社 日科技連出版社, 東京, 2005.

研究業績

1. 若井 久瑠美, 井口 寧. ”ヘテロジニアス環境における mapreduce の自動タスク割り当て”. 信学技法 CPSY2024-43, 第 124 巻, pp.7-12, Online, Dec. 16 2024.

謝辞

本論文をまとめるにあたり, 多くの方々にご助力をいただきました. ここに深く感謝の意を表します.

まずはじめに, 本研究の指導教員である井口寧教授には, 研究テーマの選定から学会発表に至るまで常に的確な助言と温かいご指導を賜りました. とりわけ, 研究上の問題点を明確にするための助言や, 新たな視点の提示など, 学問的探求を進める上で多大なるご指導を頂いたことに心より感謝申し上げます.

また, 中間審査をお引き受けいただいた金子峰雄教授, 田中清史教授には建設的なご意見・ご指摘をいただき, 研究内容を発展させる大きな糧となりました. 厚く御礼申し上げます.

そして最後に, 大学院在籍中, 日々の生活面・精神面で支えてくれた家族や同僚にも, 感謝の言葉を捧げます. 皆様の励ましや理解があったからこそ, 学業と研究に専念し, 無事に本論文を完成させることができました.

以上の方々をはじめ, 本研究に関わったすべての方にあらためて深く感謝申し上げます.