令和元年度 特別研究報告書

プロセスの資源待ち時間を削減する資源利用制御機構

龍谷大学 理工学部 情報メディア学科

学籍番号 ： T150467
氏名 ： 蔭山 直人

指導教員 ： 芝 公仁 助教，三好 力 教授

内容梗概

多くのプロセスは，CPU やディスクなど複数の計算機を使用して動作する．これらの資源はプロセス間で共有されるが，一部のプロセスが特定の資源を偏って使い続けると，この資源を共有する他のプロセスの資源待ち時間が増加し，スループットが低下してしまう問題がある．

本論文では，偏って資源を使い続けるプロセスに対し利用可能な資源の量の制限を設定し，他のプロセスの資源待ち時間を削減することで, システム全体のスループットを向上させる機構である Resource Manager について述べる．Resource Manager を用いた場合の資源競合時のプロセスの処理回数と，Resource Manager を用いない場合の資源競合時の処理回数を比較する実験を行う．また，Resource Manager の設定値を変更することにより，性能の限界を確かめる評価実験を行う．評価実験から，Resource Manager を用いることにより，資源競合時に資源配分を適切に行うことでシステムの性能が向上したことを確認した．

目 次

1　はじめに

多くのプロセスは，CPU やディスクなど複数の計算機を使用して動作する．これらの資源はプロセス間で共有されるが，一部のプロセスが特定の資源を偏って使い続けると，この資源を共有する他のプロセスの資源待ち時間が増加し，スループットが低下してしまう問題がある．

本論文では，偏って資源を使い続けるプロセスに対し利用可能な資源の量の制限を設定し，他のプロセスの資源待ち時間を削減することでシステム全体のスループットを向上させる機構である．Resource Manager について述べる．Resource Manager は，資源競合時に資源配分を適切に行うことでシステムの性能を向上させる．

Resource Manager は，Linux カーネルが提供する Control Groups(以降 Cgroups と呼ぶ) の機能を用いることで，プロセスをグループ単位で管理でき，利用可能な資源の量の制限値を一括で設定することができる．加えて，Cgroups では，1 つのグループで複数の資源の利用制限を設定できるため，特定の資源に依らない資源管理を行える．

以下，本論文では，3 章で Resource Manager の構成を述べ，4 章で，資源競合時の Resource Manager の動作述べる．また，5 章で評価を行い，Resource Manager の有用性を示す．

2　関連研究

Linux の CPU スケジューラは，CPU 使用時間をプロセスに均等に配分することで，公平性を保とうとする．しかし，CPU を偏って使用するようなプロセスが，CPU に負荷を掛け続けると，他のプロセスのスループットが低下してしまう問題がある [1]．

Linux の I/O スケジューラの代表として，CFQ と Deadline が挙げられる．CFQ スケジューラは，複数プロセスが公平に I/O 処理時間を分けるアルゴリズムである．Deadline スケジューラは，一定時間処理されなかった要求を優先的に処理する，リアルタイム処理向けのスケジューラである．しかし，CFQ スケジューラでは，ブロックデバイスに負荷を掛けプロセスが存在する場合，他のプロセスのスループットも低下する問題がある．Deadline スケジューラでは，プロセスの優先度を指定できないため，ユーザが優先的に処理させたいプロセスに資源を多く配分することができない [2]．

上記で述べた Linux スケジューラの問題を解消するアルゴリズムが提案されている [1，2] が，特定の資源を効率的に配分するためのアルゴリズムであるため，複数の資源を考慮した資源配分をできない．Resource Manager では，複数資源の資源配分を行うために，Linux カーネルの Cgroups を用いることで，複数の資源を考慮した資源配分の調整を行うことができる．

文献 [3] では，テストケースの再現のために Cgroups が用いられている．また，Cgroups はクラウドサービスの資源制限やコンテナ型仮想化環境に用いられている．

3 Resource Manager

3.1 Resource Managerの構成

Resource Manager の構成を図 1 に示す．Resource Manager は，Cgroups 操作機構，CPU 使用状況管理機構，I/O 使用状況管理機構の 3 つの機構で構成されている．

Cgroups 操作機構は，グループの作成や，利用可能な資源の量の制限の設定などを行う．Cgroups は Linux カーネルが提供する機能である．Cgroups にはプロセスのグループ単位での管理や，利用可能な資源の量の制限の設定を行える特徴がある．

Ggroups は，Cgroups ファイルシステムにより通常のファイル操作と同様な操作でプロセスの登録や，制限の設定をすること可能である．Resource Manager で用いる Cgroups v2 は，/sys/fs/cgroup をルートとする単一階層構造で構成される．Cgroups のルートにディレクトリを作成することで，グループを作成することができる．Cgroups のルート以外のディレクトリには，プロセスの登録や制限対象の資源の登録などの操作をするためのファイルが，グループ作成時に作成される．Cgroups で資源の利用制限を設定する際は，下記ファイルに制限値を書き込む．

- cpu.weight：読み書き可能なファイル．[1, 10000] の範囲で指定できる．グループ内の cpu.weight の総和の比率で配分される．
- io.max：読み書き可能なファイル．制限を設定するデバイスと，単位時間あたりの読み書き量，読み書きの回数を指定する．

cpu.weight では，CPU 使用時間をグループ内の cpu.weight の値の比率で配分する．io.max では，特定デバイスへの 1 秒間の読み書きする量をバイト単位で指定するか，読み書きできる回数を指定することで，制限を設定できる．

CPU 使用状況管理機構と I/O 使用状況管理機構は，Linux カーネルが持っている taskstats のデータを netlink を用いて取得する．taskstats 構造体には，タスクの資源利用状況などの情報が格納されている．CPU 使用状況管理機構と I/O 使用状況管理機構は，taskstats から CPU 使用状況や I/O 使用状況に関するデータを用いて，プロセスの CPU と I/O の使用状況を監視する．

- blkio_count：ブロックデバイスに対して操作を行った回数．
- write_bytes：ディスクに書き込みを行った総データ量．

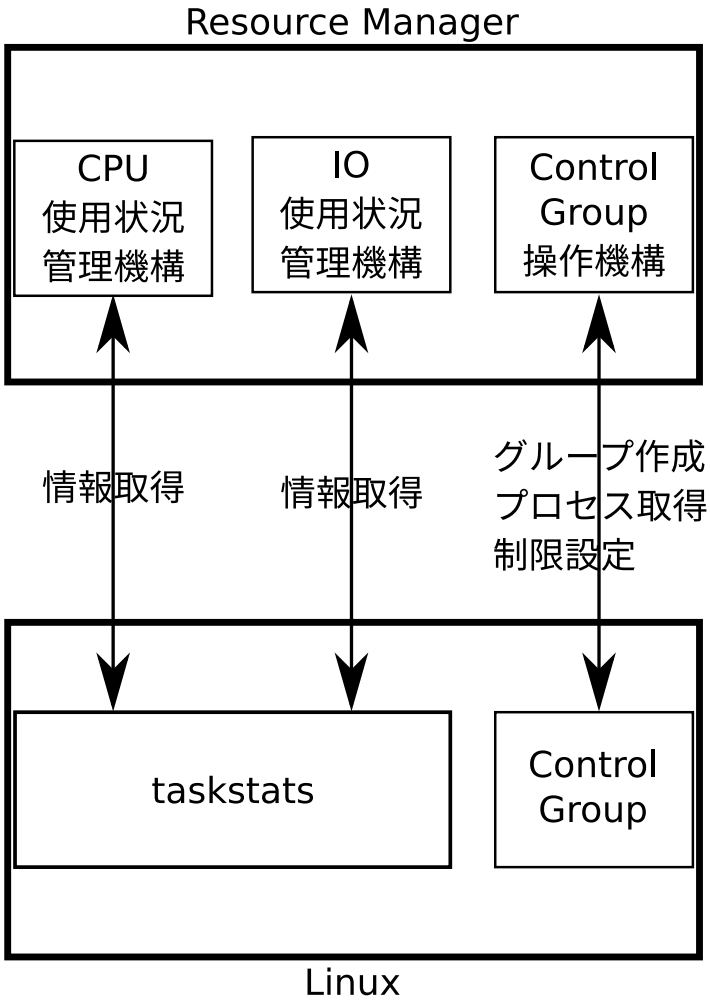

図 1　提案機構の構成

- read_bytes : ディスクから読み込みを行った総データ量.
- utime : ユーザコードで CPU を使用した時間を表す.
- stime : カーネルコードで CPU を使用した時間を表す.
- starttime : プロセスの起動時刻を表す.

また，CPU 使用状況管理機構は，taskstats から得たデータを用いてプロセスの平均 CPU 使用率を算出する.

$$total\_time = utime + stime \tag{1}$$

$$seconds = current\_time - starttime \tag{2}$$

$$cpu\_usage = 100 * (total\_time/seconds) \tag{3}$$

式 (1) で，プロセスの CPU 使用時間を算出する．式 (2) で，現在時刻からプロセスの起動時刻を引くことで，当該プロセスが稼働している時間を算出できる．最後に，式 (3) でプロセスの CPU 使用時間からプロセスの稼働時間を割ることで，平均 CPU 使用率を求めることができる．

Resource Manager は，起動時に Cgroups 操作機構を用いて，制限を設定するグループと制限を設定しないグループを図 2 の構成で作成する．Managed Group は，CPU Limited，I/O Limited，Unlimit の 3 つの子グループを持っている．CPU Limited は，CPU を偏って使用するプロセスに対し制限を設定するグループで，I/O Limited は，IO を偏って使用するプロセス対し制限を設定するグループである．Unlimited には，制限を設定しないプロセスが登録される．

3.2 Resource Manager Interface

Resource Manager Interface は，プロセスの登録，プロセスの一覧を取得するためのインタフェースである．Resource Manager Interface の構成を図 3 に示す．Resouce Manager Interface では，管理グループに所属しているプロセスの取得，プロセスの登

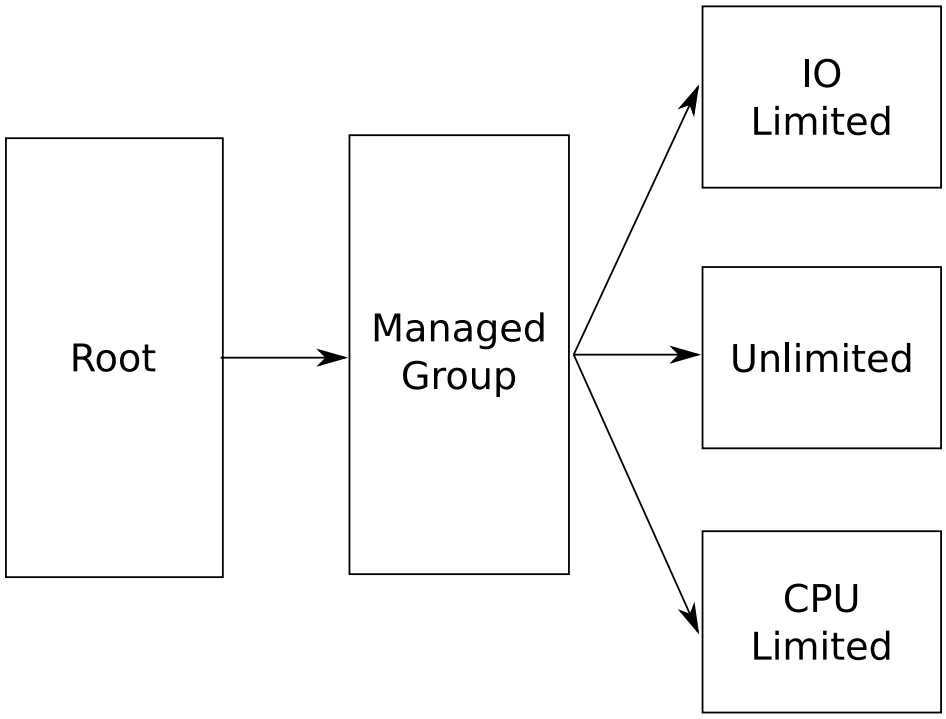

図 2　管理グループの構成

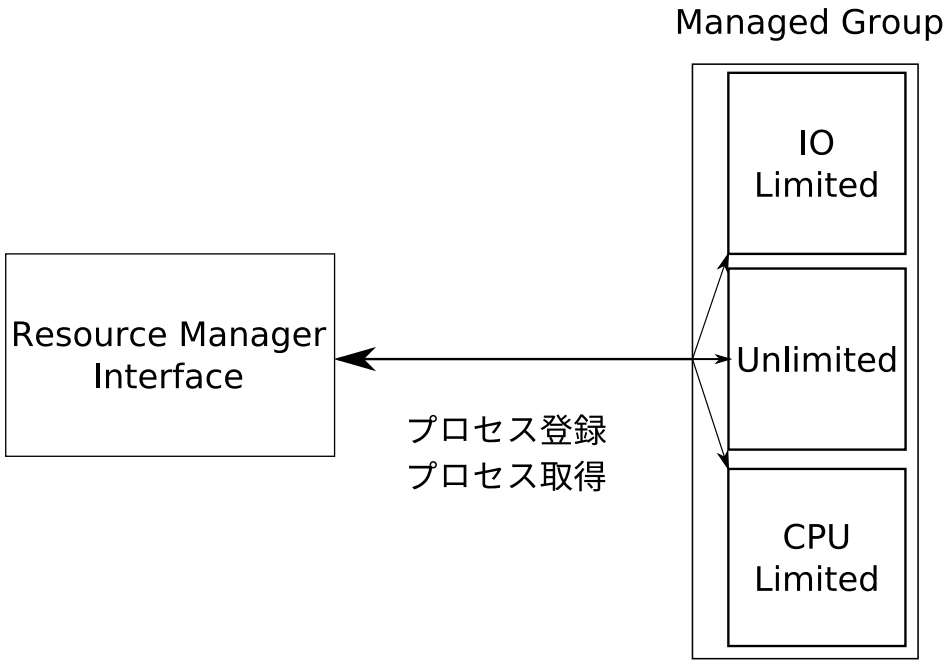

図 3　Resource Manager Interface の構成

録を Resource Manager を介さずに行う．このインターフェースでは，ユーザの判断で，管理グループにプロセスを登録することができる．これにより，ユーザが優先して処理をさせたいプロセスに，資源を多く与えることができる．

4　資源競合判定と制限の設定

Resource Manager は，Control Group 管理機構を用いて作成した．グループに所属してるプロセスを管理対象とする．管理対象プロセスのブロックデバイスの使用状況と CPU 使用状況を Linux カーネルから定期的に取得，更新する．これらの情報を用いて，プロセスの負荷状況を監視し，状況に応じてプロセスを適切なグループに所属させる．詳細な動作を以下で述べる．

(1) プロセスに制限を設定 ：Resource Manager が最も資源に負荷を掛けているプロセスに対し，制限を設定する．

(2) 競合判定 ：Resource Manager が制限を設定したあと，制限が設定されていないプロセスの平均 CPU 使用率もしくは I/O 使用回数が向上した場合，資源競合が発生しているとみなす．

(3) 制限値の調整 ：制限がかかっていないプロセスが別の処理に移行し，そのプロセスの資源利用状況が変化した場合，資源競合が発生していないとみなし，制限を設定されているプロセスの制限を解除する．

Resource Manager の動作を図 4 に示す．Resource Manager は，プロセスが偏って資源を使用しているかどうかを判断するために，過去 10 秒間の平均 CPU 使用率や I/O 使用状況の閾値を設定する．閾値から偏って資源を使用していると判断されたプロセスは，I/O Limit もしくは， CPU Limit グループに登録される．

Resource Manager の管理下に置かれているプロセスの動作の具体的な例を図 5 に示す．図 4 のケースでは，プロセス 1 がブロックデバイスを偏って使い，負荷をかけるため．プロセス 2 が CPU を用いる処理に移行するまで時間がかかってしまう．Reource Manager は，プロセス 1 がブロックデバイスに負荷を掛けていると判断し，I/O Limit グループに当該プロセスを登録する．これにより，プロセス 2 の I/O 処理の待ち時間が低減され，プロセス 2 の平均 CPU 使用率が向上する．プロセス 1 に制限を設定したことにより，プロセス 2 の平均 CPU 使用率が向上したため，Resource Manager は，資源競合が発生していると判断し，プロセス 1 を I/O Limit グループ下で管理を継続する． また，プロセス 2 が CPU を用いる処理に移行した場合，プロセス 1 のブロックデバイスに対する操作回数が 0 回になる．このとき，Resource Manager は，プロセス 1 に対する制限を解除する．

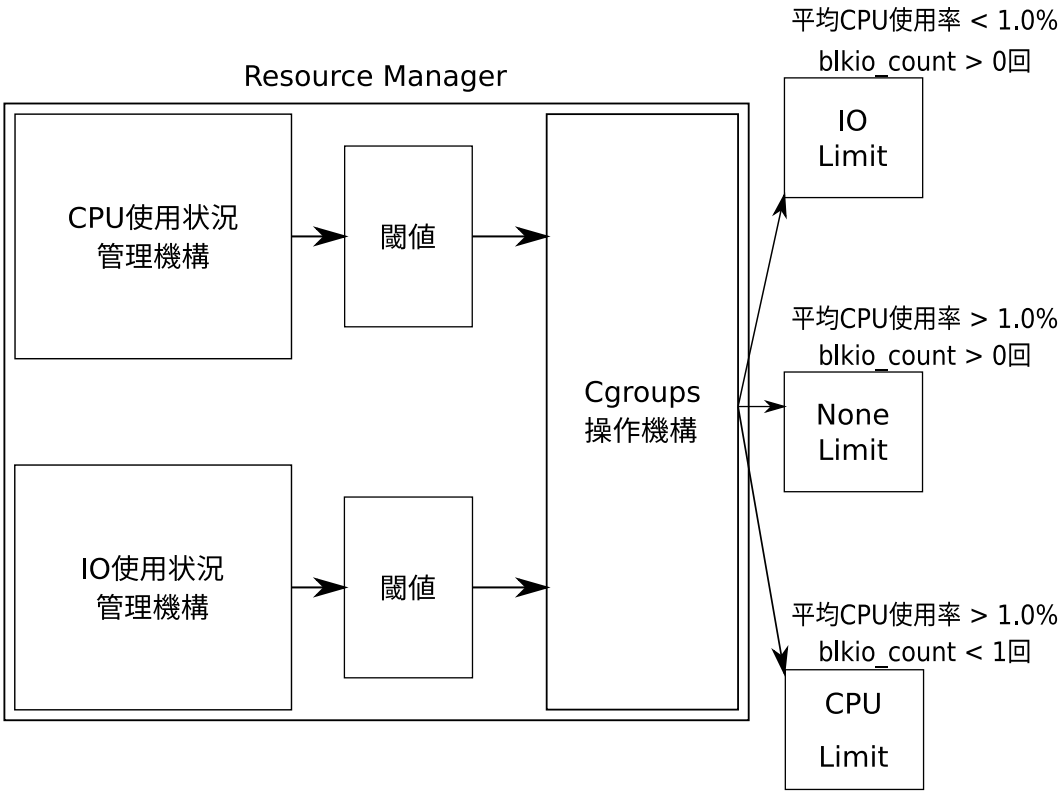

図 4　Resource Manager の動作

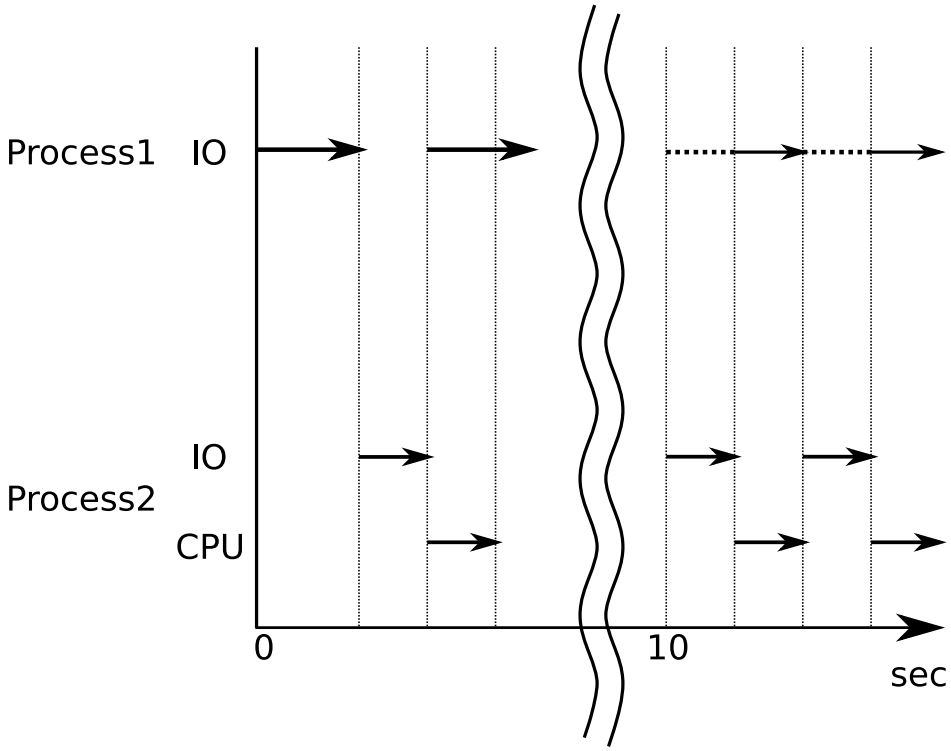

図 5　Resource Manager 動作時のプロセスの振る舞い

5　評価

本実験は表 1 に示す環境で，マルチプロセスで，資源競合が発生する状況を作り実験を行った．

5.1　制限値の変化に伴う性能の変化

本章では，設定する制限値に応じて，性能がどのように変わるか調べる実験を行った．

5.1.1　I/O

設定する I/O の制限値を 1MB から 150MB まで 1MB ずつ変化させた．実験結果を図 6 に示す．横軸は設定した制限値で，縦軸は実験プログラムのプロセスの処理回数を示す．プロセス 1 は約 2 秒ファイルに書き込みを行う．プロセス 2 は，約 1 秒ファイルに書き込みを行い，約 2 秒 CPU 処理を行う．本機構を用いた場合，プロセス 1 の処理回数が低下し，プロセス 2 の処理回数が向上したことにより，2 つのプロセスの合計処理回数が本機構を用いない場合と比較して増加した．制限値を 5MB 程度に設定した場合，2 つのプロセスの合計処理回数が多くなり，制限値をそれ以上に設定するとプロセス 2 の処理回数が向上し，Resource Manager を用いない場合の合計処理回数とおおよそ同等になり，処理回数が落ちていることがわかる．この実験結果から，制限値を低く設定しすぎると，プロセス 1 の処理回数が大幅に低下してしまい，2 つのプロセスの合計処理回数が低下し，制限値を高く設定しすぎると，制限がうまく機能せず，Resource Manager を用いない場合の合計処理回数とおおよそ同等になり，2 つのプロセスの合計処理回数が低下していることがわかる．

5.1.2　CPU

設定する制限値は，1 と 5 から 50 まで 5 ずつ変化させた．実験結果を図 7 に示す．横軸は設定した制限値で，縦軸は処理回数を示す．プロセス 1 は，約 5 秒間 CPU 処理を行う．プロセス 2 は，約 1 秒間ファイル書き込みを行い，約 2 秒間 CPU 処理を行う．本機構を用いた場合，5.1.1 章の実験結果と同様に，資源を偏って使用し，制限を設定されたプロセスの処理回数が低下し，制限が設定されていないプロセスの処理回数が向

表 1　実験環境

OS	Ubuntu18.4.0 LTS
Kernel	Linux 4.19.91
CPU	Intel(R) Core(TM) i5-4570 3.20GHz
HDD	Hitachi HDS72302 2TB

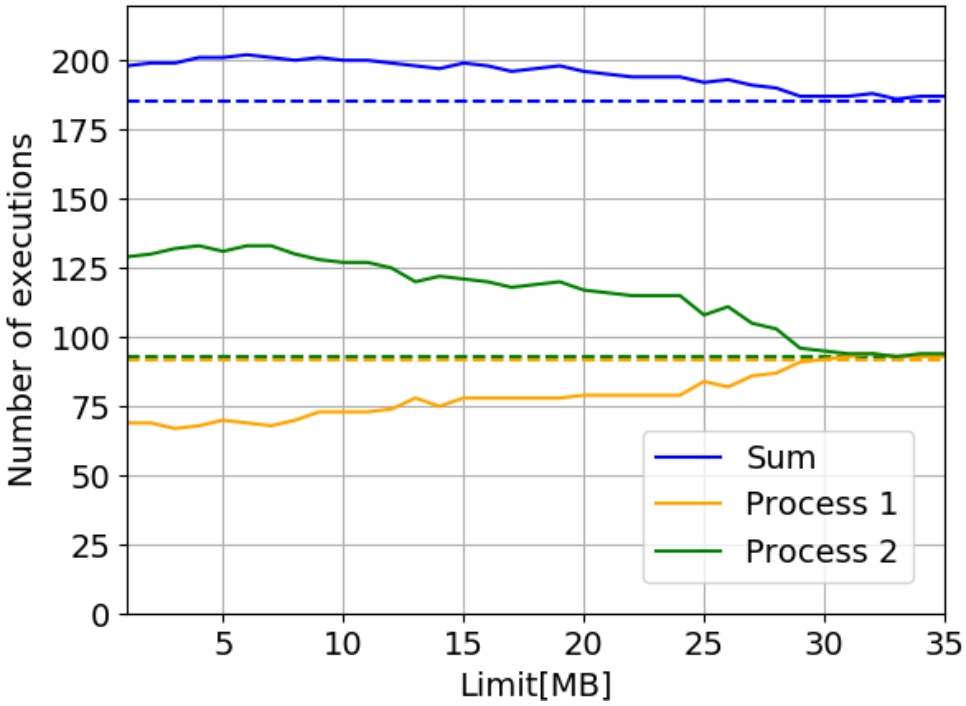

図 6　制限値を変化させたときの処理回数

上したことにより，2 つのプロセスの合計処理回数が本機構を用いない場合と比較して増加した．実験結果より，制限値を最低値に設定すると 2 つのプロセスの合計処理回数が，最も多くなることがわかる．制限値を高く設定するに連れて，本機構を用いない場合と同じような振る舞いになる．

5.2　プロセス監視の間隔の変化に伴う性能の変化

本節では，プロセスを監視する間隔を変化させることで，性能の変化を調べる実験を行った．実験結果を図 8 に示す．実験プログラムは 5.1.2 章と同様で，横軸はプロセス監視の間隔，縦軸は処理回数を示す．プロセス監視の間隔がプロセスの処理時間よ

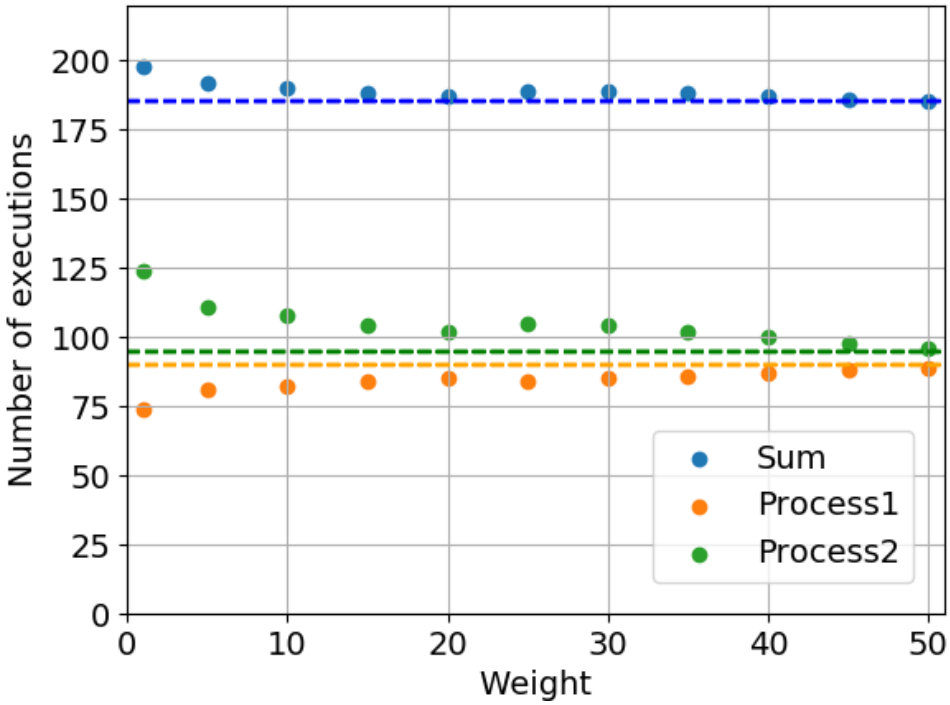

図 7　制限値を変化させたときの処理回数

り短いと，十分にプロセスの資源利用状況を取得できないため，適切に資源配分ができず，CPU を偏って使用するプロセス 1 の処理回数が多くなり，2 つのプロセスの合計処理回数が低下した．プロセス監視の間隔を 3 秒以降に設定した場合，十分にデータを蓄積できるため，適切に資源配分ができ，2 つのプロセスの合計処理回数が向上した．また，プロセス監視の間隔が 3 秒や 4 秒など，適切に資源配分を行えるプロセス監視の間隔の中でも短い時間の場合，2 つのプロセスの合計処理回数が最も向上した．これは，データ蓄積が完了するまでの待ち時間が短くなるため，Resource Manager が資源配分を開始する時間が早くなり，プロセス起動時間内で，Resource Manager の恩恵を受けられるからだと考えられる．

5.3　プロセス数に伴う性能の変化

本章では，プロセスの数を増加させることで，性能の変化を調べる実験を行った．プロセス A は約 1 秒 CPU 処理を行い，プロセス B は約 0.5 秒 CPU 処理を行ったあと，約

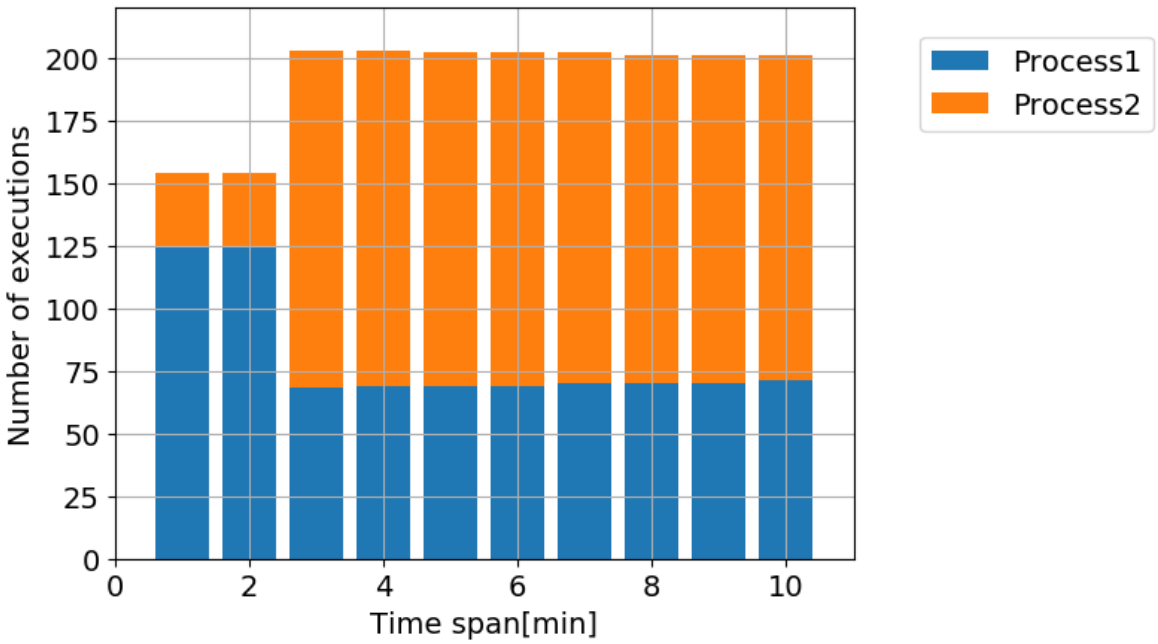

図 8　プロセス監視間隔の変化に伴う性能の変化

3 秒間ファイル書き込みを行う．プロセス A とプロセス B の合計数が 7 個になるまで実験を行った．実験結果を図 9 に示す．横軸は，プロセス数を示し，縦軸は Resource Manager を用いない場合の処理回数の比率を表す．Resource Manager を用いた場合，プロセス数が増えても処理回数が，Resource Manager を用いない場合よりも多くなったが，プロセス数が少ない場合，処理回数が Resource Manager を用いない場合の処理回数とほぼ同等になっている．

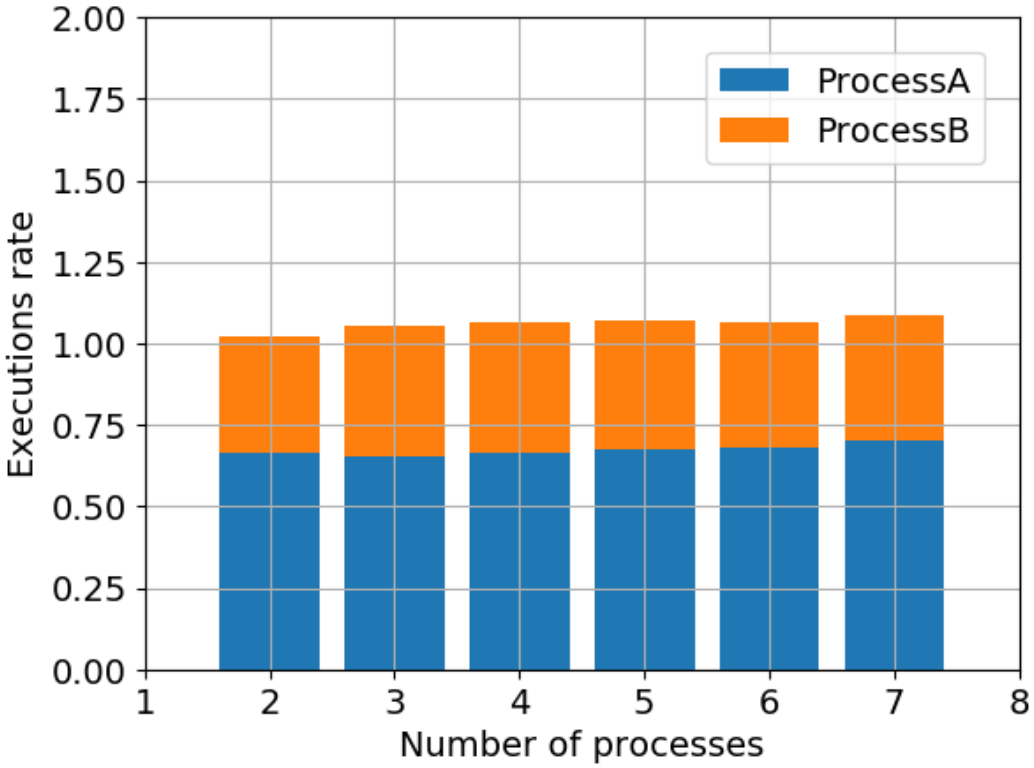

図 9　プロセス数に伴う性能の変化

6　おわりに

本論文では，複数プロセスが同一資源を使用する際の資源配分を自動で行う Resource Manager について述べた．Resource Manager では，管理対象のプロセス間で競合が発生しているか判断し，競合が発生している場合，競合が発生している資源を最も使用しているプロセスに対して，利用可能な資源の量の制限を設定する．これによって他のプロセスの待ち時間を減らし，システム全体の性能を向上させることが可能である．

本論文の評価により，資源に依らず，資源配分を適切に行うことで，システムの性能を向上させることができた．また，複数プロセスに対し，制限を設定することで制限を設定されていないプロセスを優先的に処理することが可能である．

謝辞

本研究にあたり，終始熱心なご指導とご教鞭を頂いた，芝公仁助教に心から感謝いたします．また，本研究を進めるにあたり有意義なご助言を頂いた芝研究室の院生の方に深く感謝いたします．最後に，日頃参考となる貴重なご意見やご協力を頂いた芝研究室の皆様に感謝いたします．

参考文献

[1] 谷本輝夫，佐々木広，三輪　忍，中村宏
：メニーコアプロセッサにおける競合とスケーラビリティを考慮したスレッドスケジューリング，*IPSJ SIG Technical Report*, Vol. 2011-HPC-132 No.31, pp. 1–7 (2019).

[2] 高村成道，鵜川始陽，石崎英哉
：低優先度処理を指定可能なリアルタイム処理向け I/O スケジューラ，*IPSJ SIG Technical Report*, Vol. 2014-OS-128 No.8, pp. 1–10 (2014).

[3] 岩井良成，杉木章義，棟朝雅晴
：Cgroups を利用した Hadoop における落ちこぼれタスクのリソース制限による再現，*IPSJ SIG Technical Report*, Vol. 2017-OS-140 No.13, pp. 1–6 (2017).