動的にノード構成可変な Android クラスタにおける負荷分散と効率的通信の研究

澤田 祐樹

1 はじめに

近年モバイル端末はマルチコアプロセッサの搭載により高性能化が著しく，並列分散アプリケーションのためのプラットフォームとして新たに注目されている．我々は Android OS を搭載したモバイル端末を計算ノードとして用いたクラスタシステム (Android クラスタシステム) を開発している [1]．本システムではモバイル端末の脱退や参入に伴うノード構成の動的変更に対応するためにチェックポイントデータの取得と，チェックポイントデータからの並列処理のリスタートが可能である．クラスタシステムからノードが脱退した場合でも脱退ノード内の並列プロセスをクラスタ内の別のノードへ移譲できる．

しかし従来のシステムでは脱退ノードで実行していた並列プロセスを不可分に移譲することしかできず，ノード単位で再配置する．そのためノードが脱退したとき，脱退ノードが担っていた並列プロセスを引き継いだノードだけ並列処理の負荷が高くなる．またリスタート時において並列分散処理のフレームワークが本来行っていた通信の効率化は行われないため，通信方法が最適ではないプロセス間通信が存在する．本研究ではノード構成の動的変更時においても効率的な並列分散処理を維持するための機能を実現し，その効果を定量的に明らかにする．

2 Android クラスタシステム

本システムでは並列分散処理のフレームワークとして MPI の実装の 1 つである Open MPI を使用し，MPI アプリケーションを実行対象としている．MPI アプリケーションの実行中にクラスタからノードが脱退した場合，MPI アプリケーションの実行を通常は継続できない．そこで Android クラスタでは，モバイル端末の脱退や参入に伴うノード構成の動的変更時においても並列処理を継続するためにチェックポインティング技術を採用する．これによりアプリケーションの実行状態を保存したチェックポイントデータの取得と，チェックポイントデータからのアプリケーションのリスタートが可能である．本システムでは並列分散アプリケーションのチェックポイント/リスタートが可能である DMTCP(Distributed MultiThreaded Checkpointing)[2]) をチェックポインティングソフトウェアとして使用する．DMTCP はユーザレベルで各並列プロセスの状態を保存でき，x86 や ARM，MIPS の各命令セットに対応している．また並列処理の再配置が可能であり，クラスタシステムからノードが脱退した場合でも脱退ノード内の並列処理をクラスタ内の別のノードへ移譲することで並列処理を継続できる．

DMTCP の管理下で MPI アプリケーションを実行した場合の例を図 1 に示す．図 1 は 3 ノードで構成するクラスタシステムであり，各ノードでは MPI 並列プロセスがそれぞれ 2 プロセスずつ実行されている．Open MPI で並列処理を開始すると MPI 並列プロセスの生成/管理を行うデーモンプロセス (*orterun*/*orted*) が起動する．またホストノードではチェックポイント/リスタート処理のためにプロセスの管理/制御を行う *dmtcp_coordinator* プロセスが起動する．

3 ノード構成の動的変更に伴う追加機能

従来の DMTCP におけるリスタート機能ではノード構成の動的変更時に，並列処理の負荷の不均衡が発生し，

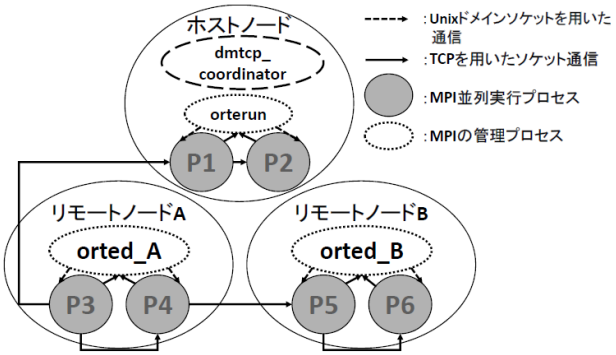

図 1: DMTCP 管理下での MPI による並列処理

さらにプロセス間の通信方法が最適化されずにリスタートする．クラスタシステム全体の性能低下となりうるこれら 2 つの問題について，図 1 を用いて詳細を述べる．

図 1 に示すクラスタ上でチェックポイントデータを取得後にリモートノード B がクラスタから脱退し，リモートノード B の並列プロセスをリモートノード A に再配置すると想定する．このときリモートノード A では 4 つの MPI 並列プロセス P3, P4, P5, P6 がリスタートし，リモートノード A のみ並列処理の負荷が増加し，クラスタ全体の性能が著しく低下する．また，同一ノード内のプロセス同士であっても TCP ソケット通信を行うため，通信効率が低い．図 1 中の P4, P5 はチェックポイント時はノード間通信であるが，ノード単位での再配置後は同一ノード内 (リモートノード A) で実行される．DMTCP はチェックポイント時に記録した通信方法と同じ方法でプロセス間通信を再構築するため，P4 と P5 間の通信は TCP ソケット通信として復元される．同一ノード内の通信は TCP よりも高速な手段が使えるため，通信方法を切り替えることで性能向上の可能性がある．

3.1 プロセス単位での再配置による負荷分散

並列プロセスの再配置時，従来のノード単位での再配置に加えて複数のノードにプロセス単位で並列プロセスを分散して再配置する機能を導入する．本機能を容易に実現するために，現在全てのプロセス間通信を TCP ソケット通信に統一している．前述の例において，プロセス単位での再配置によって脱退したリモートノード B の並列プロセス P5, P6 を 1 つずつ均等にホストノードとリモートノード A に再配置できる．ホストノードとリモートノード A で実行している MPI 並列プロセスの数は均等となり，ノード間における並列処理の負荷の不均衡が解消できると考える．従来の DMTCP は並列処理をノード単位でしか再配置できないため，DMTCP に変更を加えることでこれを実現する．先の例において，リモートノード B 内の並列処理の再配置時には *orted_B*, P5, P6 の 3 つのプロセスのチェックポイントデータがなければ復元はできないがプロセス単位での復元を可能とすることで，例えば P5 はホストノード，P6 はリモートノードで復元することが可能となる．

3.2 プロセス間通信の切り替え

並列プロセスの復元時，各プロセスにおいて通信相手先プロセスの配置ノードに応じて通信方法を切り替える (最適化する) 機能を導入する．本機能ではチェックポイ

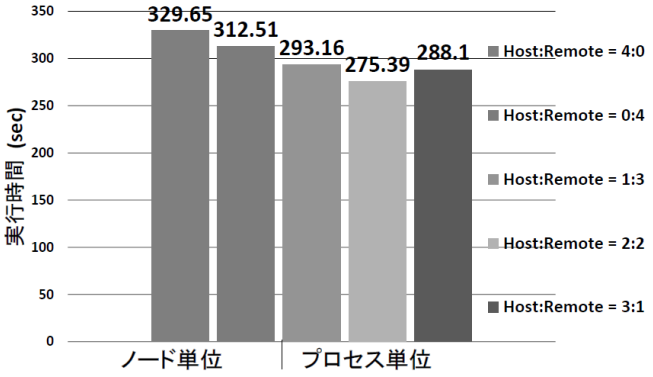

図 2: 再配置プロセス数ごとの実行時間

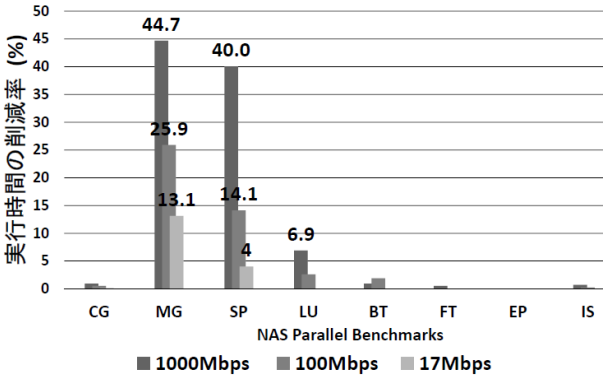

図 3: 通信方法の切り替えによる実行時間の削減率

ントデータに記録されていた通信方法がノード間通信であっても通信相手先プロセスが同一ノード内で起動していることを検知して通信方法をより高速なノード内通信に切り替えて復元する．先の例において，プロセスの再配置により TCP ソケット通信を行う P4 と P5 が同一ノードで起動した場合は，より高速な通信方法に切り替える．通信を切り替えるためにはクラスタ内で起動するプロセスの配置ノード (IP アドレス) を各プロセスが把握する必要がある．DMTCP に変更を加え，*dmtcp_coordinator* から各プロセスに対してクラスタ内で起動している全プロセスの IP アドレスを通知する．各プロセスはプロセス間通信の再構築を行う前に自身と通信相手先プロセスが起動している IP アドレスと比較する．通信相手先プロセスが同一ノードで起動している場合はより高速な UNIX ドメインソケットを用いた通信に切り替える．

4 評価

プロセス単位での再配置とプロセス間通信の切り替えによって，効率的な並列処理を維持可能かどうか評価する．評価では実現した機能を使用した場合と使用しない場合において，リスタート後の MPI アプリケーションの実行時間を比較する．現在，Android OS 上で DMTCP を動作させる環境を実現できていないため，Linux PC (CPU:Core i7 4770, 動作周波数:3.4GHz, コア数:4, メモリ:32GB) を複数台利用したクラスタを用いて評価を行う．DMTCP のバージョンは 2.3.1 である．

4.1 プロセス単位での再配置

3 ノードで構成するクラスタシステムから 1 ノードが脱退し，2 ノードでリスタートするケースを想定する．各ノードで 4 つの並列プロセスが実行されているものとする．このケースにおいて，脱退したノードで実行されていた 4 プロセスをクラスタに残った 2 ノードに任意のプロセス数再配置し，MPI アプリケーションがリスタートしてから終了するまでの実行時間を計測する．評価プログラムには N クイーンプログラム (N=19) を使用し，ノード間の通信方法は Wi-Fi である．図 2 は再配置したプロセス数ごとの実行時間を示し，凡例はクラスタ内に残っているホストノード (Host) とリモートノード (Remote) それぞれに対して新たに再配置したプロセス数である．図 2 中の左 2 本のバーがノード単位での再配置，残り 3 本のバーがプロセス単位で任意のプロセス数再配置した場合の実行時間である．図 2 より，ノード単位での再配置と比較してプロセス単位での再配置により実行時間を削減できる．同じプロセス単位での再配置であっても実行時間が異なる原因はノード間通信における通信負荷の違いであると考える．図 2 において，クラスタに残った 2 ノードにそれぞれ 3 プロセスと 1 プロセス再配置する場合にホストノードに 3 プロセス再配置した場合の実行時間がリモートノードに 3 プロセス再配置した場合の実行時間よりも短いのはノード間通信の回数が少ないことが原因である．ゆえに通信負荷を考慮してノード間通信を減らすようなプロセスの再配置を行うことが必要である．

4.2 プロセス間通信の切り替え

3 ノードで構成するクラスタシステムから 1 ノードが脱退し，2 ノードでリスタートするケースを想定する．3 ノードのうち 1 ノードは 2 プロセス実行し，残り 2 ノードは 1 プロセス実行する．脱退するノードは 2 プロセス実行しているノードであり，クラスタ内の 2 ノードに 1 プロセスずつ再配置する．評価プログラムには NAS Parallel Benchmarks(NPB) の 8 つのプログラムを使用する．SP 以外の問題サイズはクラス C，SP はクラス A としている．ノード間通信には 1000Base-T を用いる．プロセス間通信方法を切り替えなかった場合と比較して，切り替えを行うことでどの程度実行時間を削減できるかを図 3 に示す．図 3 より，MG と SP ではプロセス間通信の切り替えにより実行時間を削減できた．その他 6 つのプログラムにおいては効果が見られなかった．原因はノード内のプロセス間通信の切り替えに比べて，切り替えが行えないノード間通信の通信量が多く，通信負荷が大きいことが考える．また，プロセス間の通信形態も原因の一つであり，集団通信を行う MPI アプリケーションでは通信方法の切り替えによる効果が見られない．

5 おわりに

本研究ではノード構成の動的変更時におけるシステムの性能低下を緩和するための 2 つの機能を実現し，評価を行った．プロセス単位での再配置を適用することで，ノード単位での再配置と比較して実行時間を最大 11.8%できた．またプロセス間通信方法を切り替えることで，通信帯域が乏しい状況においてもプログラムによっては実行時間を 13.1%削減できることを確認した．

システムの性能低下をさらに緩和し，効率的な並列処理を維持するために，今後の課題はクラスタ内のノードの性能に応じて再配置するプロセスを決定する機能を実現することである．

参考文献

[1] Y. Sawada, Y. Arai, K. Ootsu, T. Yokota, and T. Ohkawa, “An Android Cluster System Capable of Dynamic Node Recofiguration”, Proc. 7th International Conference on Ubiquitous and Future Networks (ICUFN 2015), pp.689-694, 2015.

[2] J. Ansel, K. Arya, and G. Cooperman, “DMTCP: Transparent Checkpointing for Cluster Computations and the Desktop”, 23rd IEEE International Parallel and Distributed Processing Symposium (IPDPS2009), pp.1-12, 2009.