6J-02

動的なノード数変更に対応した MPI並列処理のための負荷分散手法の提案

澤田 祐樹 † 荒井 裕介 †† 横田 隆史 †† 大津 金光 †† 大川 猛 ††
† 宇都宮大学工学部情報工学科 †† 宇都宮大学大学院工学研究科情報システム科学専攻

1 はじめに

近年モバイル端末は高性能化しており, 並列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 端末を使用し, MPI アプリケーションを並列処理するクラスタシステムを開発している. このシステムは端末の脱退, 参入に伴うノードの動的変化に対応するためにチェックポイントデータの取得と, 取得したチェックポイントデータからの並列処理の再開が可能である. しかし現状のリスタート機能においてクラスタシステムを構成するノードが動的に変化し並列処理を再開した場合, 特定のノードがノード 1 台分の並列処理を全て引き受ける. クラスタシステムに参加している全ノードが 2 プロセス割り当てられた状態で 1 台のノードが脱退した場合, 1 台のノードが脱退ノードの並列処理を負担し 4 プロセス実行となり実行プロセス数に偏りが生じる. また新規ノードが参入した場合, クラスタシステム内のあるノードで実行している 1 プロセスを新規ノードに割り当てることはできないので負荷分散することができない.

そこで各ノードにおける並列処理の負担を分散化するためのプロセスの割り当て手法を提案する.

2 Android クラスタシステム全体の概要

我々が開発している Android クラスタシステム [1] における通信環境, 並列処理環境, ノードの管理環境について述べる.

ノード間の通信は高速な通信が可能で, かつ安価で利用できる Wi-Fi を用いている。各ノードは無線通信で相互接続しクラスタシステムを構築する. また本クラスタシステムでは並列プログラムの基盤として Open MPI を用いる. 本システムは移動体である Android 端末を構成要素としているため, 通信範囲外への移動によるシステムからの脱退や, 通信範囲内へ移動してきた新規ノードのクラスタシステムへの参入が発生する場合がある. このようにノードが動的に変化する場合があるので, 並列処理に使用可能なノードを正確に把握する必要がある. そのためノードの管理はマシンファイルを用いる. マシンファイルはクラスタシステムに参加している全てのノードが個別に保持するようにし, ノードの脱退や参入が発生した場合は各々マシンファイルを更新することで並列処理に使用可能なノードを管理する. またマシンファイルは MPI アプリケーションを並列処理する際に使用するノードと各ノードに割り当てるプロセス数を決定する際にも使用する.

3 チェックポインティング/リスタート機能

Android クラスタシステムのチェックポインティング/リスタート機能について述べる. チェックポインティング/リスタート機能はプロセスに関する全ての情報が記述されたチェックポイントファイルを生成できる Distributed MultiThreaded Checkpointing (DMTCP)[2] を使用する. DMTCP 上で MPI アプリケーションを実行するとホストノードにて dmtcp_coordinator と呼ばれるプロセスが立ち上がり, dmtcp_coordinator がチェックポイント要求を中継して各プロセスへ伝達する. チェックポイント要求を受け取ったプロセスにおいて, 自プロセスのチェックポイントデータを保存する. DMTCP はノード間でのチェックポイントデータの送受信機能はないので送受信機能が必要となる. 本システムではホストノード以外のノードは定期的 (現在は 30 秒) にチェックポイントデータをホストノードへ送信する. ホストノードはリモートノードの MPI 並列処理を負担できないため, ノード数変更に伴う並列処理のリスタート時にホストノードは自身以外のノードにチェックポイントデータを送信し, ホストノード以外で並列処理を負担できるようにする.

Open MPI で並列処理を行うとホストノードでは orterun プロセスが立ち上がり実行プロセスは orterun プロセスの子プロセスとなる. リモートノードでは orted プロセスが立ち上がり実行プロセスは orted プロセスの子プロセスとなる. 現状リモートノードで実行される orted プロセスはホストノード内で実行はできないので, リモートノードが脱退した場合は orterun プロセスが実行されていないノード (ホストノード) が並列処理を負担する. そのためホストノードはどのリモートノードでもリスタートできるように全てのリモートノードのチェックポイントデータを集める.

以上の流れでチェックポイントデータを取得し, クラスタシステムを構成するノードが動的変化した場合, 集めたチェックポイントデータ (ckpt) を元に MPI アプリケーションをリスタートする.

次ページに記載されている図 1 のように 3 ノードで構成され, それぞれ 2 プロセスを実行している Android クラスタを例に 1 ノードが脱退した場合のリスタート処理について述べる. プロセス H, プロセス A, プロセス B はそれぞれホストノード, ノード A, ノード B での実行プロセスである.

Proposal of Load Balancing Technique for MPI Parallel Computation Allowing Dynamic Change of the Number of Nodes

†Yuki Sawada, ††Yusuke Arai, ††Takashi Yokota, ††Kanemitsu Ootsu and,††Takeshi Ohkawa
Department of Information Science, Faculty of Engineering, Utsunomiya University (†)
Department of Information Systems Science, Graduate School of Engineering, Utsunomiya University (††)

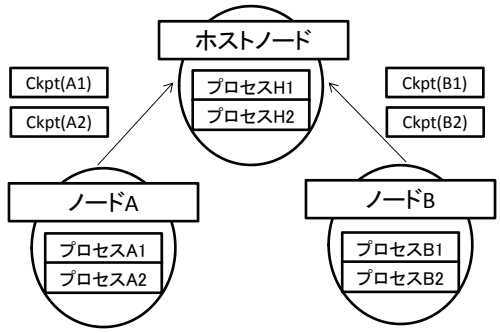

図 1: 3 ノードで構成されるクラスタシステム

この例ではノード B が脱退するものとし, ノード A が脱退ノード B の実行プロセスを継承して実行する. ホストノードは集めた ckpt のうち脱退ノード B の ckpt をノード A に送信する. ckpt は 30 秒ごとに更新されるが ckpt を更新する前に並列処理が中断した場合は更新前の ckpt を使用する. MPI アプリケーションを実行中にノード B が脱退し並列処理が中断された後, 取得した ckpt をもとにリスタートした場合, ノード A がノード B の並列処理を負担しノード A の実行プロセス数は 4 プロセス, ホストノードの実行プロセス数は 2 プロセスとなる.

前述のように DMTCP によるリスタートはノード単位でプロセスが割り当てられる. そのため各ノードにおいて MPI 並列処理の負荷に偏りが生じる.

4 動的なノード数変更後における並列処理の負荷分散

3 節で述べたように DMTCP を用いたリスタートは各ノードにおいて MPI 並列処理の負荷に偏りが生じるのでリスタート時に負荷分散できるようにする. 現在のクラスタシステムでのノード単位でのプロセスの割り当てではなく, プロセス単位で各ノードに実行プロセスを割り振り, MPI 並列処理の負荷に偏りを減らすことでクラスタシステム全体の処理性能向上を目指す. 3 節の図 1 を例とすると脱退ノード B の 2 つの実行プロセスをホストノードとノード A に 1 プロセスずつ割り振ることで負荷分散を実現する. またプロセス単位での割り当ての際, 同一ノード内で通信をしていたプロセス同士が負荷分散によりノード間通信となる場合がある. Open MPI で実行されるプロセスは, 本システムにおいてノード内通信は shared memory 通信, ノード間通信は TCP 通信を用いるが, プロセス割り当てを容易にするためにプロセス間通信は TCP 通信のみに設定する. リスタート時における実行プロセスは主に以下の 3 STEP で決定する.

- STEP 1. 全ての ckpt からプロセス情報を読み込む
- STEP 2. プロセス情報においてプロセス間の親子関係の情報が正しいか確認
- STEP 3. 親プロセス, 子プロセス (実行プロセス) の順に生成

この処理をチェックポイントファイル生成時に DMTCP によって自動生成されるリスタート処理のヘルパ用シェルスクリプトを変更することによって実現する. ヘルパスクリプトではホストノード, リモートノード両ノードにて dmtcp_restart コマンドを実行するように記述されている. また各ノードでは dmtcp_restart コマンドを実行する際, リスタートさせたいプロセスの ckpt をコマンドラインオプションの引数として指定している. そのため負荷分散したいプロセスの ckpt を割り当てたいノード上で実行される dmtcp_restart コマンドのコマンドラインオプションの引数に加えることで負荷分散を実現する.

割り当てたいノードを仮にノード A とする. STEP 1 において, 負荷分散したいプロセスの ckpt をノード A 上で実行される dmtcp_restart コマンドのコマンドラインオプションの引数に加えると ckpt からプロセスの情報を読み込むことはできる. しかし STEP 2 におけるプロセスの親プロセス・子プロセス確認の際に負荷分散したいプロセスはノード A ではないノードで生成されたため親子関係を持たない. そのためリスタート時に生成されない.

プロセスを他ノードに負荷分散する場合は, 割り当てたいノードにおいて orted プロセスの子プロセスとして新たに親プロセス・子プロセスの関係を築くことにより実行プロセスとして生成されると考える.

5 おわりに

本稿ではリスタート時における DMTCP の実行プロセス決定方法について述べた. またクラスタシステムを構成するノードの変化時における MPI 並列処理の負荷分散手法の提案を行った.

今後の課題としてリスタート時にプロセス間の親子関係を上書きし, 並列処理の負荷分散を可能とすることが挙げられる. また本稿において通信方式はプロセス割り当てを容易にするために TCP 通信のみに設定したが shared memory 通信は TCP 通信より高速なので, 実行プロセス割り当て時にプロセス間の通信方法もプロセスの実行ノードに応じて変更する必要がある.

謝辞

本研究は，一部日本学術振興会科学研究費補助金（基盤研究 (C)24500055，同 (C)24500054，同 (C)25330055，若手研究 (B)2573006）の援助による

参考文献

[1] 荒井 裕介ほか: “端末の動的な参加・脱退を支援する無線接続型 Android クラスタシステムの実装”, 信学技報, Vol.114, No.155 , pp.143-148, 2014.

[2] Jason Ansel, Kapil Arya and Gene Cooperman: “DMTCP: Transparent Checkpointing for Cluster Computations and the Desktop”, 23rd IEEE International Parallel and Distributed Processing Symposium, IPDPS, 2009.