Vol.2012-ARC-202 No.27
Vol.2012-HPC-137 No.27
2012/12/14

RDMA をサポートするクラスタ向けの軽量協調キャッシング

新井 淳也[1,a)]　石川 裕[1,b)]　堀 敦史[2,c)]

概要：協調キャッシングは各ノードのファイルキャッシュを協調管理することで仮想的に全ファイルキャッシュ領域を結合したグローバルファイルキャッシュを提供する手法である。既存の協調キャッシング機構はメッセージパッシング通信によって CPU に負荷をかけていたが、近年は低負荷・低レイテンシのリモート DMA (RDMA) が HPC クラスタに普及しつつある。そこで本研究では RDMA のみを用いて協調キャッシングを行う手法を提案する。さらにアプリケーションに対し透過的に協調キャッシングを提供するライブラリを実装し、性能評価を行った。

キーワード：HPC, I/O, キャッシュ, 協調キャッシング

1. 序論

HPC クラスタ上における巨大データ処理の需要は広く存在し、高い I/O 負荷の問題を解決するために研究が行われてきた [1], [8], [12]。**分子回折像分類計算**[14] は巨大なファイルを扱う処理の一つで、端的には画像のグループを作成する一種のクラスタリングである。処理対象となる回折像全体のサイズは将来的に 300TB を超える見通しである。この分子回折像分類計算を理化学研究所計算科学研究機構のスーパーコンピュータ「京」[15] で並列分散的に行うアプリケーションの開発を現在理化学研究所と東京大学が共同で進めている。分子回折像分類計算が行う I/O の特徴は以下の 4 点にまとめられる。

読み出し偏重　入力と出力は別のファイルであり、I/O のほとんどは読み出しである。

データ再利用　以前にどこかのノードが使用した回折像を別のノードが再び要求することが多い。

低局所性　完全なランダムアクセスではないものの、要求される回折像は連続しない。例えば MapReduce[4] のような手法を適用することは容易でない。

アクセス集中　多数のノードが同時に同じ回折像を要求する。

これらの特徴のうち、読み出し偏重・データ再利用・低局所性に対して有効性を発揮する手法として**協調キャッシング**[3] がある。協調キャッシングはノード間でファイルキャッシュを協調管理することにより、仮想的に全ノードのファイルキャッシュ領域を結合したグローバルキャッシュを提供する。グローバルキャッシュはノード数に応じて拡大するため、多数のノードを使用する計算処理では巨大なファイルをキャッシュに格納することも可能になる。

しかし回折像分類計算の 4 つ目の特徴であるアクセス集中は、既存の協調キャッシング機構 [2], [3], [5], [6], [7], [9] において特定のノードへの負荷集中を発生させ性能の低下を招くことが構造的に明らかである。負荷集中が発生する状況は 2 通りある。一つ目はキャッシュブロックの所在の問い合わせ。協調キャッシングではファイルデータの断片であるキャッシュブロックに一意な ID を割り当て、その ID 毎にどのノードがどのキャッシュブロックを所有しているかの管理を行っている。キャッシュブロックの所在はしばしば複数台のサーバに分散して管理されるが、ある ID のキャッシュブロックがどこにあるかを知っているサーバは 1 台しかない分散の仕方になっている。そのため、一つの ID に対してリクエストが集中すると 1 台のサーバにアクセスが集中する結果となる。二つ目はキャッシュブロックを所有するノードからのブロック取得。キャッシュブロック ID 毎にそのブロックを所有するノードを 1 つしか記録しない手法ではその唯一のノードにアクセスが集中する。

既存手法はいずれもクライアント・サーバモデルを採用している点とメッセージパッシングによる通信を行っている点が共通しているが、メッセージパッシングベースの通

[1] 東京大学
[2] 理化学研究所
[a)] arai@is.s.u-tokyo.ac.jp
[b)] ishikawa@is.s.u-tokyo.ac.jp
[c)] ahori@riken.jp

信は CPU 負荷を生じさせることから、アクセスが集中した際のレイテンシ増大が著しい。さらにメッセージパッシングによる通信はクライアントとサーバの両方でのメッセージ応答を必要とする。このうち特にクライアントでのメッセージ応答はユーザープログラムの計算を妨害しうる。

一方、HPC クラスタの世界ではメッセージパッシングとは別の通信手段として CPU 負荷のないリモート DMA(RDMA) が広く普及してきている。RDMA とは、予め登録されたメモリ領域間においてネットワークデバイスにより CPU を介さないノード間のデータ通信を可能にする仕組みである。RDMA の特徴としては CPU 負荷が小さいことだけでなくレイテンシが小さく高速な通信が可能である点も重要である。

そこで我々は回折像分類計算の I/O 性能改善を主目的として、RDMA のみを用いた低 CPU 負荷で機能する新しい協調キャッシング手法を提案する。既存のシステムではメッセージパッシングを用いてクライアントやサーバが内部で協調処理を行ってリクエストを処理することができた。対して RDMA は被アクセス側が通信に関知しない一方向通信であるため協調処理が困難だが、そのような環境でも正確にデータを取得できるよう工夫を行っている。

提案手法は実際に実装を行い、「京」と東京大学の FX10 スーパーコンピュータシステム [16] を用いて性能を評価した。

2. Anco: RDMA による協調キャッシングライブラリ

Anco は「京」及びその商用版である FX10 向けに開発された協調キャッシングライブラリである。特徴としては以下の 5 つが挙げられる。

(1) 通信には低負荷・低レイテンシである RDMA のみを用い、メッセージパッシングは一切使用しない。これによりアクセスが集中したときにもメッセージパッシングより高速にキャッシュの管理を行うことができ、スケーラビリティが向上する。

(2) ネットワーク構造はピアツーピアである。クライアント・サーバの区別が無く、全てのノードに負荷が公平にかかる。また、サーバがないため全ノードをユーザの計算のために使用できる。

(3) ノードを幾つかのグループに分割する。グループ分割により効率的なキャッシュディレクトリ構造を構築でき、さらに負荷分散も実現できる。詳しくは後のキャッシュディレクトリ構造の説明 (2.4) で述べる。

(4) RDMA を使用しないシングルスレッドアプリケーションであれば、Anco を利用するためにプログラムの変更も再コンパイルも必要ない。Anco は I/O システムコールを環境変数 `LD_PRELOAD` の設定によりフックして協調キャッシングを提供する。

(5) ファイルの読み出しのみをサポートする。読み出しが主な I/O であるアプリケーションを性能改善の対象とする。

2.1 RDMA 通信

RDMA を用いた通信では各ノードが特定のメモリ領域を通信用に登録し、その領域の間でネットワークデバイスが CPU を介さないデータの転送を行う。通信はメッセージパッシングと異なる一方向通信である。すなわち、通信手続きの呼び出しを行うのは読み出しや書き込みを行うノードのみであり、読み出しや書き込みを行われる被アクセス側のノードは何も行う必要がない。

Anco ではキャッシュブロックのデータやキャッシュディレクトリなど協調キャッシングに必要なデータのほとんどを RDMA のために登録されたメモリ領域内に配置する。以下の記述では明示的にそのことを述べないが、外部からアクセスされるデータは全て登録済みメモリ領域にあるものとする。

2.2 ローカルキャッシュ領域

協調キャッシング機構の中にはローカルのキャッシュメモリ領域と協調キャッシングで管理するグローバルキャッシュメモリ領域がメモリアドレス上で分割されているものもあるが、Anco ではローカルキャッシュメモリ領域にあるキャッシュブロックをそのまま共有するため、ローカルキャッシュメモリ領域しか存在しない。

2.2.1 構造と書き換え

ローカルキャッシュメモリ領域には、キャッシュブロックの ID、取得トークン、キャッシュブロックのデータの 3 つを一要素とする一次元配列が置かれている。これを便宜上**ローカルキャッシュ配列**と呼ぶことにする。取得トークンとはローカルキャッシュにキャッシュブロックが配置されたときに生成される数値で、過去にそのノードで生成されたどの取得トークンとも重複しない一意な値である。実際には単純なカウンタで問題ない。

キャッシュブロック ID と取得トークンは共にアトミックに書き換え可能なサイズでなければならない。なぜならこれら 2 つの値はリモートのローカルキャッシュから RDMA でキャッシュブロックを取得したときに正しいデータを取得できたかどうかを確認するために使用されるからである。Anco では RDMA のみによる通信を実現するためにロックを使用しない。ゆえに別のノードが読み出し中であるブロックのデータを破壊してしまったり、2.4 で述べるキャッシュディレクトリから得たブロックの所在が既に古い情報となっており別のブロックで置き換えられてしまっている場合などがあり得る。そこでキャッシュブロックのデータ転送開始前と後でキャッシュブロック ID と取得トークンを比較し、同じ値であれば成功（正しいブロッ

クを取得できた）と判定、違う値であれば失敗と判定する。判定に使用するためこれらの値は以下の性質を保証する。

(1) キャッシュブロック ID が有効な値であるなら、キャッシュブロック ID とキャッシュブロックのデータは必ず正しく対応する。
(2) 取得トークンが有効な値であるなら、取得トークンが一致する間はキャッシュブロックのデータの変更が行われていない。

具体的にはローカルメモリ上のキャッシュブロックの書き換えは以下の順番で行われる。

キャッシュブロック取得時:

(1) 他ノードやファイルシステムからキャッシュブロックのデータをメモリ領域へ読み込む。
(2) 取得トークンを生成し代入する。
(3) キャッシュブロック ID を代入する。

キャッシュブロック放棄時:

(1) キャッシュブロック ID を無効値に設定する。
(2) 取得トークンを無効値に設定する。（この後はキャッシュブロックのデータに使用したメモリ領域を他のキャッシュブロックのために使用できる）

以降特に明示的には言及しないが、全てのローカルキャッシュ操作はこの手順で行われる。なお一つ目の条件だけではデータ破壊を防げない状況として次のようなものがある。

(1) ノード A がノード B にある、ID が X であるキャッシュブロックのデータの読み込みを開始する。
(2) ノード B がその領域のデータを別のキャッシュブロックで置き換える。
(3) ノード B が再び同一の領域に ID が X であるキャッシュブロックで置き換える。
(4) ノード A がキャッシュブロックのデータの読み込みを終える。

キャッシュブロック ID はノード A のブロック取得前後で一致するため、ノード A は正しくブロックを取得できたと誤解してしまう。また二つ目の条件だけではデータ破壊を防げない状況として次のようなものがある。

(1) ノード A がキャッシュディレクトリ（2.4 で述べる）を参照し、キャッシュブロック ID が X であるブロックがノード B にあるという情報を得る。
(2) ノード A がキャッシュディレクトリを参照してからノード B にアクセスするまでの間にノード B が該当キャッシュブロック領域を ID が Y である別のキャッシュブロックで置き換える。
(3) ノード A は ID が Y であるキャッシュブロックを ID が X であるブロックだと思い込んで転送を開始する。

2.2.2 管理リスト

ローカルキャッシュに空きがない時に新しいキャッシュブロックを取得した場合は適切にキャッシュブロックを選択し放棄する必要がある。どのブロックを放棄すべきかを決定するために以下の 2 つの管理リストを持っている。

Singlet 専用リスト Singlet[3] とは、ある一つのノードにしかコピーが所持されていないキャッシュブロックのことである。特殊な操作はあるが基本的に FIFO 構造になっている。

一般リスト ローカルキャッシュに存在するブロックのうち singlet 専用リストに所属しないブロック全てが所属している。LRU により管理される。

Snglet 専用リストの要素数と一般リストの要素数の和の最大値はローカルキャッシュ上に配置できるキャッシュブロックの最大数と一致する。また singlet 専用リストの最大要素数は、ローカルキャッシュ上に配置できるキャッシュブロック最大数に対する比率（**singlet 比率**）により決定される。

他のノードやファイルシステムから取得したキャッシュブロックは最初に一般リストに追加される。一般リストではキャッシュを所有するノード自身からのアクセスのみを検出して LRU による管理を行う。RDMA による一方向通信を用いているため別のノードからのアクセスを検知することはできず、また別のノードからアクセスがあったときは必ずアクセス元のノードへのキャッシュブロックのコピーが伴うので、例え需要が大きかったとしてもコピー元であるブロックを放棄してしまって問題ない。

一般リストに所属するブロックが LRU で放棄対象に選ばれたとき、そのブロックが singlet である場合は singlet 専用リストに移され、singlet でない場合はそのまま放棄される。この選別により singlet 専用リストへの要素の追加と削除の頻度が減るため、singlet は singlet でないブロックよりも長くローカルキャッシュに留まることができる。このような処置を行う理由は、singlet を削除してしまうと次にそのブロックに対するアクセスがあったとき必ずキャッシュミスとなることにある。しかしながら singlet を長く保持するために singlet 比率を高めると、ローカルキャッシュを所有するノード自身からのアクセスを反映したキャッシュ管理を行える領域が小さくなってしまうため、バランスを取る必要がある。

Singlet 専用リストは基本的に FIFO だが、リストに含まれるブロックにアクセスがあった場合は singlet 専用リストから削除され一般リストのもっとも最近アクセスがあったブロックとして再挿入される。Singlet 専用リストに含まれるブロックがアクセスされないまま終端に達した場合は singlet であるかどうかに関わらず放棄対象となる。

2.3 グループ分割

Anco においてノードはいくつかのグループに分割して管理される。目的は効率の良いキャッシュディレクトリ構造の構築と負荷分散にある。グループ分割の基準は MPI プロセスのランクをグループ数で割った剰余である。例え

ばグループ数が 3 なら、ランク 0 はグループ 0、ランク 1 はグループ 1、ランク 2 はグループ 2、ランク 3 はグループ 0…となる。このように分ける理由はキャッシュブロックへのアクセスの局所性をなるべく低下させることにある。具体的な説明は 2.4 の中で行う。

2.4 キャッシュディレクトリ

キャッシュの所在を管理するために**代表ディレクトリ**と**代替ディレクトリ**という 2 種類のキャッシュディレクトリを用いる。

2.4.1 代表ディレクトリ

代表ディレクトリはキャッシュブロック取得時にブロックの所在を確認するために使用される。その構造はブロックの ID を行、グループの ID を列に持つ二次元のテーブルである。各セルはそれぞれのグループにおけるそれぞれのキャッシュブロックの存在状態を表現しており、以下の 3 種類の値のいずれかを取る。

読込済所在 ノードのランクとそのノードのローカルキャッシュ配列内でのインデックスのペアを値として持つ。キャッシュブロックのデータ全体が読み込まれており、他のノードはここからブロックを転送することができる。

読込中所在 読込済所在と同じくノードのランクとそのノードのローカルキャッシュ配列内でのインデックスのペアを値として持つ。キャッシュブロックのデータは現在読込中であり準備ができていない。

無効 このグループにキャッシュブロック ID に対応するキャッシュブロックは存在しない。

代表ディレクトリは全ノードの間で分散して保持される。すなわち全ノードが RDMA 通信用メモリ領域の中に二次元のテーブルとして代表ディレクトリの一部を持つ。代表ディレクトリの分割はブロック ID（行）毎にラウンドロビンで行う。例えばノードが全部で 3 つなら、ランク 0 のノードは ID 0, 3, 6, ... の行を保持し、ランク 1 のノードは ID 1, 4, 7, ... の行を保持する。

代表ディレクトリがグループ ID を列として持つということは、すなわちそのグループに属するノードのいずれからも書き換えられうるということを意味する。書き換えを行うノードが複数存在するために、競合状態の発生を回避することができない。競合状態が発生したとしても間違った所在が代表ディレクトリに書き込まれてしまうわけではないものの、読込済所在が書き込まれた後で読込中所在で上書きされてしまうことは性能上悪影響を及ぼす。故にグループのメンバは互いに同じキャッシュブロックへ同時にアクセスする確率が低い、アクセスの局所性の低い組み合わせになっているべきである。グループを連続したランクで区切ることは良い考えではない。なぜならランクが隣接するノードは隣接するデータを読むように作成されていることが多いからである。そこでグループはなるべくランクが隣接しないような集合、すなわちランクをグループ数で割った剰余が等しいノード同士の組み合わせになっている。

2.4.2 代替ディレクトリ

代替ディレクトリは代表ディレクトリに記録された所在にあるブロックが放棄されるとき、代わりに代表ディレクトリに書き込むべき所在を発見するために使用される。各グループは個別に代替ディレクトリを所有しており、その構造はブロック ID をキーとし、グループに所属するノードのランクとローカルキャッシュ配列のインデックスの対応を格納したハッシュテーブルである。複数のキャッシュブロック ID を一つのセルに対応させているため値の衝突が発生する場合がある。もし衝突した場合は上書きが行われ、以前の値は消失する。

代替ディレクトリは代表ディレクトリと同様、グループに所属するノードの間で分散して保持される。分散の仕方も同様で、代替ディレクトリの一行ごとにラウンドロビンである。

2.5 キャッシュブロックの取得と放棄

ここではこれまでに説明したシステムの構成要素を踏まえて、キャッシュブロックの取得と放棄の手順について述べる。

2.5.1 取得

回折像分類計算に見られるように、同時多発的に同じキャッシュブロックへのリクエストが発生したとしてもファイルシステムに負荷をかけないことを目標に設計を行っている。詳しくは次の取得手順の中で述べる。

(1) 代表ディレクトリの取得対象ブロック ID に対応するエントリ（行）を丸ごと RDMA で取得する。
(2) 取得したエントリ内の自身が所属するグループのセルの値によって分岐する。
  (a) セルの値が読込済所在だった場合は、エントリ全体の中からランダムに 1 つの読込済所在を選択し、そこから RDMA でキャッシュブロックを自身のローカルキャッシュ領域へ転送する。エントリの中から発見できる読込済所在は高々グループ数までなので、グループ数が多いほうがキャッシュブロック転送元の負荷を分散させることができる。一方でグループの数を増やすと代表ディレクトリのエントリ取得時に転送するデータの量が増え、またセルが増える分代表ディレクトリを更新する頻度も上がる。
  (b) 読込中所在だった場合は、そのノードの読み込みが終わるまでポーリングして待つ。
  (c) 無効値だった場合が少々複雑である。まずリクエスト元ノードは自身が所属するグループにおける取得対象ブロック ID の**代表ノード**となる。代表

ノードとは代表ディレクトリに記載されたブロック所在の持ち主である。代表ノードとなったノードは最初に対応する代表ディレクトリ中のセルを読込中所在に書き換える。次に他のグループが読込済所在を持っていればそこから、なければファイルシステムからキャッシュブロックを読み込む。最後に代表ディレクトリ中のセルを読込済所在に書き換える。

(3) 代替ディレクトリの内容を更新する。つまり自身があるローカルキャッシュ配列インデックスの位置にキャッシュブロックを手に入れたことを記録する。

以上のうち、自身が所属するグループのセルの値が読込中所在だった場合と無効値だった場合の対応はファイルシステムの負荷を抑制するために重要である。読込中所在だった場合にファイルシステムからのブロック取得を行うと、同時多発的に同じブロックへのアクセスが発生したときほとんどのアクセスがグローバルキャッシュからではなくファイルシステムからの取得になってしまう。読込中所在という状態を用意することでグループ内で既に取得中のノードがあることを知らせ、ファイルシステムへアクセスしないようにさせている。理想的には同じキャッシュブロックに対するファイルシステムへのアクセスは最大でグループ数で抑えられるようになるが、Anco ではロックを使用しないため代表ノードが複数誕生してしまう可能性があるが、の性能評価で示す通り効果は出ている。

2.5.2 放棄

放棄時はまず放棄対象のブロックが singlet であるかどうかを知る必要がある。競合状態によって singlet でないという偽陰性の判定が出てしまう可能性はあるが、その場合はファイルシステムから読み直せば済むため問題ない。

(1) 代表ディレクトリと代替ディレクトリのキャッシュブロック ID に対応するエントリを RDMA で読み込む。どちらにも自身の所有するブロックの所在しか記録されていなければ singlet であり、そうでなければ singlet でない。Singlet だった場合、放棄対象のブロックが一般リストからの放棄であるなら放棄をキャンセルし、singlet 専用リストへ移動され、別の一般リスト内のブロックを放棄対象としてリトライする。放棄対処のブロックが singlet 専用リストからの放棄であるなら、放棄対象が singlet であるかないかに関わらず放棄される。

(2) (放棄処理を継続する場合) 自身の所有するブロックの所在が代表ディレクトリに記録されていた場合、代わりの所在を発見して置き換える役目を負う。代替ディレクトリを参照して同じグループのノードが放棄対象であるブロックを所有しているかどうかを確認し、もしそのようなノードがあれば代表ディレクトリの情報を置き換える。なければ無効値で置き換える。

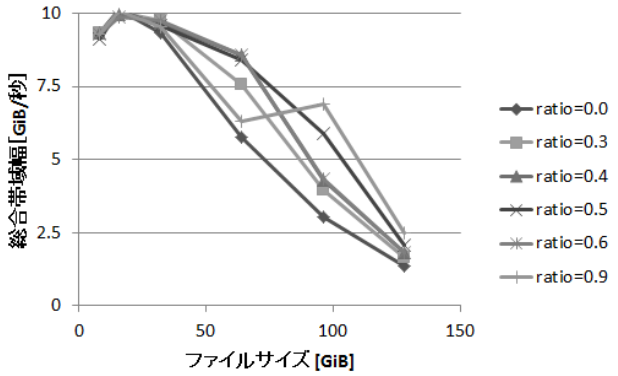

図 1 Singlet 比率とファイルサイズ毎のランダムアクセス性能（12 ノード）

(3) 代替ディレクトリの内容を実際のローカルキャッシュの内容に合わせて更新する。

3. 評価

評価はスーパーコンピュータ「京」を用いたものと東京大学の FX10 を用いたものの 2 種類があるため、実験ごとに記載する。「京」と FX10 は共にジョブのファイルを高速なローカルファイルシステムに移動してから計算を始めるステージングをサポートしており、全ての実験でステージングを利用している。実験ではキャッシュブロックの大きさは 1MiB、ローカルキャッシュ領域のサイズは 1 ノードあたり 10GiB で固定とし、1 ノードあたり 1 プロセスの対応を与える。また凡例に登場する “POSIX” とは Anco でフックしない状態での `read` システムコールを、“MPI-IO” とは MPI-IO[10] の collective I/O をそれぞれ意味する。

3.1 最適な singlet 比率の探索

2.2.2 で述べたように、singlet 比率は大きすぎても小さすぎても性能が低下する。そこで実際にランダムアクセスを行わせることによって適切な singlet 比率を探索した。FX10 の 12 ノードでランダムアクセスする対象のファイルサイズを変化させ、singlet 比率毎の総合帯域幅 (aggregate bandwidth) の変化を観測した結果が図 1 である。最初に ratio = 0.0, 0.3, 0.6, 0.9 を試し、その後に性能の良かった 0.6 周辺に点を追加した。

結果を見ると ratio = 0.0 が全体を通して最も性能が低い。Singlet 専用リストを設けたことが性能向上に貢献しているといえる。一方 ratio = 0.9 は大きなファイルサイズに対して効率が良いことが分かる。Singlet の割合が高まるとノード間で重複して所有するキャッシュブロックの数が減少することが理由だと考えられる。ratio = 0.3 から ratio = 0.6 にかけては小規模な変動に留まっているが、押しなべて性能がよいのは ratio = 0.5 の時である。よってこの先の実験では singlet 比率として 0.5 を使用する。

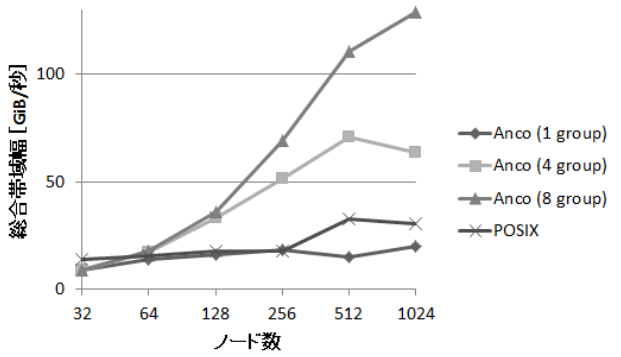

図 2 連続アクセスのスケーラビリティ

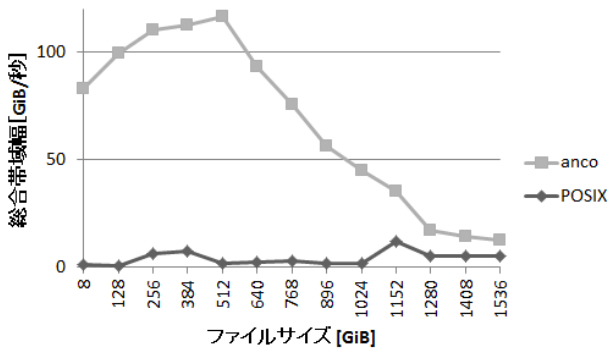

図 3 ランダムアクセスの総合帯域幅（128 ノード）

3.2 連続アクセス

「京」上において全ノードが同じファイルを同時に先頭から順に読み始めた際の総合帯域幅を計測した。ファイルサイズは 4GiB で、1 回の read 毎に 256KiB のデータを読み込んでいる。目的は回折像分類計算のアクセスパターンの特徴であるアクセス集中を再現することにある。結果を図 2 に示す。図からは POSIX が横ばいであるのに対し Anco がノード数に応じて帯域を拡大していることが読み取れる。POSIX では I/O ノードとのアクセスのための帯域がボトルネックになっているのに対し、Anco では 1 つのノードが読み込みを完了すれば他のノードはそこからデータを転送できるため、ノード間の帯域を有効活用できていると考えられる。また、Anco の中でもグループ数が多いほうがよりよいスケーラビリティを示している。2.5.1 で述べたように、グループ数が多いほうが取得元のノードを分散させることができるという点が影響しているのだろう。

3.3 ランダムアクセス

FX10 の 128 ノードを用いて、8GiB から 1536GiB のファイルについて各プロセスが独立に全体をランダムにアクセスした際の総合帯域幅を計測した。回折像分類計算のアクセスパターンの特徴である低ローカリティを反映し性能を評価することが目的である。なおアクセスは単位は 1MiB としている。つまり 1MiB の連続データを読んでから、次のランダムなオフセットへ移動してまた 1MiB の連続データを読んでいる。1 ノードあたり 10GiB のローカルキャッシュ領域を提供しているため、128 ノードの協調キャッシングで利用できるメモリ領域の大きさは総合で 1280GiB になる。それを踏まえてグラフを見ると、高い総合帯域幅を総合キャッシュ領域の大きさまでは維持できていないことが分かる。理由としては、singlet でないキャッシュブロック、つまり複数のノードが重複して所有するキャッシュブロックの存在が挙げられる。Singlet 比率として 0.5 を使用しているものの、singlet 専用リストに属するキャッシュブロックが他のノードへコピーされれば、singlet 専用リストにありながら実際には singlet でないキャッシュブロックが存在し得る。それが総合キャッシュ領域の半分に満たないファイルサイズ 512GiB が総合帯域幅のピークであることの理由だと推測される。しかしながら 3.1 で計測したように singlet 比率によってファイルサイズに対する総合帯域幅は変化するため、アプリケーションのワーキングセットの大きさに応じて singlet 比率を変更することにより性能を向上できる可能性がある。

3.4 ストライドアクセス

FX10 上で IOR ベンチマーク [13] を用いて、異なるブロックサイズでストライドアクセスした際の総合帯域幅を計測した。HPC アプリケーションで頻繁に使用されるアクセスパターンに対する性能を評価することで、回折像分類計算やそれと似たアクセスパターンのアプリケーション以外にも広く利用できるかどうかを検証することが目的である。なお IOR のブロックサイズと Anco のキャッシュブロックサイズは別のものであることに注意が必要である。IOR はファイル全体をブロックの並びと見做し、プロセス毎にブロック番号をずらしてアクセスする。プロセス数が N であるなら、プロセス 0 が 1 番目のブロックを、プロセス 1 が 2 番目のブロックを、というように進み、プロセス N は N+1 番目のブロックを読む。そしてまたプロセス 0 は N+2 番目のブロックを読む。実験では IOR のパラメータのうちブロックサイズに実験で明示した値を与えている他は IOR のデフォルト値を使用している。実験の結果を図 4 に示す。いずれのケースでも Anco の 1 グループか 4 グループのどちらかは全体的に良い性能を発揮している。ブロックサイズ 16KiB のときに 4 グループのほうが性能が良いのは、一つのブロックを同時に要求するノードの数が多いからだと考えられる。キャッシュブロックが 1MiB、IOR のブロックが 16KiB なので、1 つのキャッシュブロックに 1MiB/16KiB=64 ブロックが含まれることになるため、最大 64 ノードが同時に同じキャッシュブロックを要求する。2.5.1 で述べたようにグループ数が多いほうがキャッシュブロック転送元ノードの負荷が分散されるため、グループ数は多いほうが有利になる。一方、ブロック

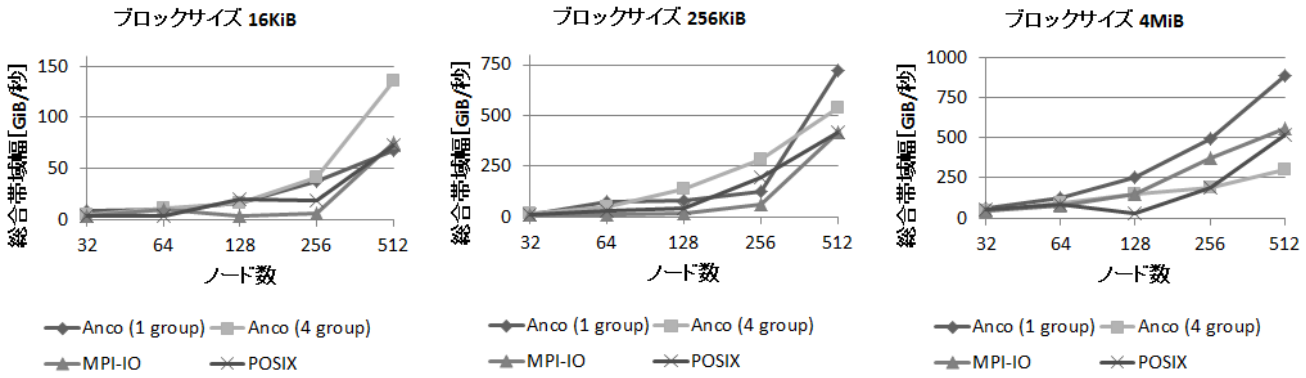

図 4 ストライドアクセスのスケーラビリティ

サイズ 256KiB、ブロックサイズ 4MiB と大きくなるにつれ 1 グループの場合のほうが良い性能を示すようになる。この現象については調査が不足しており、現在のところ原因は明らかになっていない。

4. 関連研究

代表的な 2 つの協調キャッシング手法を取り上げる。

Dahlin らが提案した *N-Chance*[3] は最も古い協調キャッシング手法である。サーバが全てのキャッシュブロックの所在を管理しており、クライアントはサーバへリクエストを投げて目的のキャッシュブロックを取得する。

N-Chance の特徴はキャッシュブロックのフォワーディングにある。各キャッシュブロックはそれぞれ残りフォワード回数を記録するカウンタを持っている。各ノードはローカルメモリ上のキャッシュブロックを LRU で管理しており、あるブロックが放棄対象に選ばれたとき、そのブロックの残りフォワード回数が 1 以上なら放棄せずランダムに選んだノードへそのブロックをフォワード（転送）する。フォワードされるたびに残りフォワード回数のカウンタはデクリメントされるが、そのキャッシュブロックに対しいずれかのノードからアクセスがあったときは残りフォワード回数のカウンタを元の値にリセットする。この仕組みは I/O の活発なノード、すなわちキャッシュブロックの入れ替わりが激しいノードから I/O の不活発なノード、すなわち同じキャッシュブロックを長くメモリに留めていられるノードへキャッシュブロックを移動させる効果があるため、グローバルキャッシュでのキャッシュヒット率が上昇する。

フォワード処理は転送先メモリ領域を確保するためにロックが必要になるため、RDMA のみを用いて実現するのが困難である。本研究の提案手法がメッセージパッシングを併用することで N-Chance のようなフォワーディングを実現した場合にどの程度パフォーマンスが変化するかの調査は今後の課題である。

Sarkar らが提案した Hint-based な手法 [11] はサーバへキャッシュブロックの所在を問い合わせる回数を減少させた。各ノードはキャッシュブロックの所在に関する「ヒント」情報を与えられている。ヒント情報は必ずしも最新のキャッシュブロックの所在を含んでいるわけではないという意味で不正確だが、サーバへ所在を問い合わせる前にヒント情報に基づいてキャッシュブロックの探索を行うことによってサーバへのアクセス回数を減少させることができる。

これら 2 つの既存手法は構造的に同一キャッシュブロックへのアクセス集中に弱い。まず N-Chance はサーバにアクセスが集中する。さらに Hint-based はヒント情報のサイズを小さく保つためキャッシュブロックの所在を 1 ヵ所しか記録しない。そのためその一つのノードへキャッシュブロックの取得リクエストが集中する。

5. 結論

データインテンシブな HPC アプリケーションのうち、特に回折像分類計算のようにノード間でデータの再利用が発生し局所性が低いアクセスパターンを持つものに対しては協調キャッシングが有効である。しかし回折像分類計算は多数のノードが同時に同じキャッシュブロックを要求するという特徴があり、既存の協調キャッシングシステムはメッセージパッシングによる CPU 負荷のためにこの種のアクセスに対する性能低下が激しくなることが推測される。CPU 負荷を軽減する手法としては通信手段として RDMA を用いることが考えられる。RDMA は近年 HPC クラスタに普及しつつあり、低負荷、低レイテンシな一方向通信を提供する。

そこで本研究ではより低負荷な機構を構築するため RDMA のみによって協調キャッシングを行う手法を提案する。提案手法はクライアント・サーバの区別のないピアツーピアなネットワーク構造を持ち、さらに効率的なキャッシュディレクトリ管理と負荷分散のためにノードをグループへ分割して管理している。

提案手法の性能は連続アクセス、ランダムアクセス、ス

トライドアクセスの 3 つについて評価し、協調キャッシングを行わない POSIX システムコール呼び出しより高速化することを確認した。観測された性能についてはまだ分析が十分でない部分もあるため、今後より精密な評価を行っていく必要がある。

参考文献

[1] Akcelik, V., Bielak, J., Biros, G., Epanomeritakis, I., Fernandez, A., Ghattas, O., Kim, E. J., Lopez, J., O'Hallaron, D., Tu, T. and Urbanic, J.: High Resolution Forward And Inverse Earthquake Modeling on Terascale Computers, *Supercomputing, 2003 ACM/IEEE Conference*, p. 52 (online), DOI: 10.1109/SC.2003.10056 (2003).

[2] Cortes, T., Girona, S., Labarta, J. and Computadors, D. D. D.: PACA: A Cooperative File System Cache for Parallel Machines, *In Proceedings of the 2nd International Euro-Par Conference*, pp. 477–486 (1996).

[3] Dahlin, M. D., Wang, R. Y., Anderson, T. E. and Patterson, D. A.: Cooperative caching: using remote client memory to improve file system performance, *Proceedings of the 1st USENIX conference on Operating Systems Design and Implementation*, OSDI '94, Berkeley, CA, USA, USENIX Association (1994).

[4] Dean, J., Ghemawat, S. and Inc, G.: MapReduce: simplified data processing on large clusters, In OSDI ' 04: Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation, USENIX Association (2004).

[5] Feeley, M. J., Morgan, W. E., Pighin, E. P., Karlin, A. R., Levy, H. M. and Thekkath, C. A.: Implementing Global Memory Management in a Workstation Cluster, *SIGOPS Oper. Syst. Rev.*, Vol. 29, No. 5, pp. 201–212 (online), DOI: 10.1145/224057.224072 (1995).

[6] Huber Jr., J. V., Chien, A. A., Elford, C. L., Blumenthal, D. S. and Reed, D. A.: PPFS: a high performance portable parallel file system, *Proceedings of the 9th international conference on Supercomputing*, ICS '95, New York, NY, USA, ACM, pp. 385–394 (online), DOI: 10.1145/224538.224638 (1995).

[7] Isaila, F., Malpohl, G., Olaru, V., Szeder, G. and Tichy, W.: Integrating collective I/O and cooperative caching into the "clusterfile" parallel file system, *Proceedings of the 18th annual international conference on Supercomputing*, ICS '04, New York, NY, USA, ACM, pp. 58–67 (online), DOI: 10.1145/1006209.1006219 (2004).

[8] Kwon, Y., Nunley, D., Gardner, J. P., Balazinska, M., Howe, B. and Loebman, S.: Scalable clustering algorithm for N-body simulations in a shared-nothing cluster, *Proceedings of the 22nd international conference on Scientific and statistical database management*, SSDBM'10, Berlin, Heidelberg, Springer-Verlag, pp. 132–150 (online), available from ⟨http://dl.acm.org/citation.cfm?id=1876037.1876051⟩ (2010).

[9] Liao, W.-K., Coloma, K., Choudhary, A. and Ward, L.: Cooperative Client-Side File Caching for MPI Applications, *Int. J. High Perform. Comput. Appl.*, Vol. 21, No. 2, pp. 144–154 (online), DOI: 10.1177/1094342007077857 (2007).

[10] Message Passing Interface Forum: MPI-2: Extensions to the Message-Passing Interface. `http://www.mpi-forum.org/docs/docs.html`.

[11] Sarkar, P. and Hartman, J.: Efficient cooperative caching using hints, *Proceedings of the second USENIX symposium on Operating systems design and implementation*, OSDI '96, New York, NY, USA, ACM, pp. 35–46 (online), DOI: 10.1145/238721.238741 (1996).

[12] Schadt, E. E., Linderman, M. D., Sorenson, J., Lee, L. and Nolan, G. P.: Computational solutions to large-scale data management and analysis, *Nat Rev Genet*, Vol. 11, No. 9, pp. 647–657 (online), available from ⟨http://dx.doi.org/10.1038/nrg2857⟩ (2010).

[13] The Regents of the University of California: IOR HPC Benchmark — Free System Administration software downloads at SourceForge.net. `http://sourceforge.net/projects/ior-sio/`.

[14] Tokuhisa, A., Taka, J., Kono, H. and Go, N.: Classifying and assembling two-dimensional X-ray laser diffraction patterns of a single particle to reconstruct the three-dimensional diffraction intensity function: resolution limit due to the quantum noise., *Acta crystallographica. Section A, Foundations of crystallography*, Vol. 68, No. Pt 3, pp. 366–381 (online), DOI: 10.1107/S010876731200493X (2012).

[15] 独立行政法人理化学研究所計算科学研究機構：スパコン京コンピュータ｜独立行政法人理化学研究所 計算科学研究機構 (AICS). `http://www.aics.riken.jp/`.

[16] 東京大学情報基盤センタースーパーコンピューティング部門：FX10 スーパーコンピュータシステム（Oakleaf-FX）[東京大学情報基盤センタースーパーコンピューティング部門]. `http://www.cc.u-tokyo.ac.jp/system/fx10/`.