Vol.2010-HPC-126 No.22
2010/8/4

並列アプリケーションのトレースログの効率的なオンライン圧縮アルゴリズムの評価

鴨志田　良和[†1]　田浦　健次朗[†1]

並列アプリケーションの膨大なトレース・ログを, イベントの重要度に基づいて効率的にオンライン圧縮するアルゴリズムを提案し, 核となるアルゴリズムの性能について評価を行う. 東京大学 HA8000 クラスタシステムを用い, 1 秒に 1300 回の通信を行うアプリケーションを用いた実験によると, データを 1/1000 に削減するためにかかる CPU 使用率 0.06%以下であり, gzip –fast コマンドと同程度であった. また, 99.9%のデータを削除しているにもかかわらず, 通信に非常に長い時間がかかっている 90 件程度のイベントをほとんどもらさず記録することができていることが分かった.

Evaluation of the Efficient On-Line Compression Algorithm for Trace-Logs of Parallel Applications

Yoshikazu Kamoshida[†1] and Kenjiro Taura[†1]

We propose a efficient on-line compression algorithm for massive trace logs, which is preferably sampling significant or interesting events in traces and we report the evaluation of the core algorithm. Experiments are done on the HA8000 cluster system in The University of Tokyo with a benchmark application, which makes each process operate about 1300 times communication per second. Experimental results show that CPU usage of compression algorithm is less than 0.06%, which is comparable to “gzip –fast” command, and that nearly all of the most time-consuming 90 events are preserved while reducing data size to 1/1,000.

†1 東京大学
The University of Tokyo

1. はじめに

今後数年から 10 年程度で普及すると予測されているペタスケール・エクサスケールの並列計算では, 並列計算にかかわるプロセス数が数十万から 100 万以上になると考えられている. 並列計算の性能チューニングを行うためには, 通信にかかる時間などをログにとってそれを解析することがよく行われるが, 100 万プロセス規模の並列計算ではそのトレース・ログのサイズが膨大になり, 記憶域や通信帯域を圧迫してしまう恐れがある. たとえば, ログの 1 レコードのサイズを 32 バイト, ログの生成速度が 1K レコード/sec だとすると, 100 万プロセスが生成するログのサイズは 32G バイト/sec となる. これだけの量のデータを, 実行中のアプリケーションへの影響が無いようにストレージに蓄積すること, また, あとからデータを解析すること, これらどちらの処理も容易には実現可能ではない. この問題を解決するためには, 出力されるログのサイズを, 効率的に, オンラインで削減することが必要である.

トレース・ログをもとに性能解析を効率的に行うためには, ログに各イベントの発生時刻と経過時間が含まれることが望ましい. しかし, これらのデータを含むログを効率的に圧縮することは困難である. 単にプロセスごとに gzip コマンド等で圧縮するだけでは 1/3 から 1/5 程度にしかサイズを削減することはできない. ログサイズの削減は, 圧縮後のデータから情報の損失なしにもとのデータを得ることができる情報圧縮と, 非可逆的なサイズの削減, すなわち, ログに含まれる主要な情報は保持しつつ, 重要ではない情報を削除することでサイズを削減することが考えられる. 本稿では後者についても区別せず圧縮と呼ぶことにする. 実際, 100 万プロセス規模のトレース・ログを低負荷に, かつ利用しやすい形で蓄積しようとする場合, ログのサイズを 1/100 から 1/1000 程度に圧縮できないと現実的ではないとわれわれは考えている.

並列アプリケーションの各プロセスが生成するログにはある程度の類似性があり, クラスタリングを行うことによって, 数個のグループに分けることができることが知られている[1)]. レコードあたり 32 バイトの例を考えると, 1/100 に圧縮するためには, レコードあたり 0.32 バイト以下の情報量で済ませる必要があるが, たとえばクラスタリングによって 6 つのクラスに分けられるとすると, $\log_2 6/8 \approx 0.323 > 0.32$ であるから, どの rank がどのクラスに属しているかという情報を単純に格納するだけで, 1/100 を達成できないことになる. このように, 1/100 やそれ以上の圧縮を考えるとき, 損失なしの情報圧縮は利用できないかもしれない.

われわれは, このような背景のもと, 時間データを含むような並列アプリケーションのトレース・ログのサイズを大幅に削減するための方法を提案し, その圧縮率や圧縮にかかるCPU 負荷を評価する.

本稿はこれ以降, 次のように構成される. まず 2 章で提案手法を述べたのち, 3 章で提案手法の評価を行う. そして, 4 章で本研究に関連する研究との比較検討を行い, 5 章でまとめを行う.

2. 提案手法

ログデータは多数のイベントの集合で構成され, この中には, 重要なものとそうでないものが混在する. 本研究では, ログデータの中から, 重要ではないイベントを削除し, 重要なイベントを保持することでデータ量を削減する方法を提案する.

あるログイベント e_i の重要性を, s_i とするとき,

$$p_i = \frac{s_i}{\sum_{k=0}^{k<n} s_k} \tag{1}$$

の確率で e_i を保持することにする. オンラインでログデータを圧縮するアルゴリズムは以下のとおりである.

(1) イベントの個数が一定量 (n) になるまでメモリに蓄積する
(2) 乱数を用いて, 式 1 に従って保持するべきイベントを決定する
(3) 上記のサンプリングを x 回行う

このアルゴリズムで “重要な” データのみ保持することにより, もとのデータを x/n に削減することが可能である. このアルゴリズムを適用する際, 重要性 (s_i) をどのように決定するかが, アルゴリズムの有効性を左右する. 重要性の決定に関して, われわれは, 「まれにしか発生しないイベントほど重要である」という仮定を置くことにする. イベント e_i と同一種類のイベントの発生頻度に対して単調減少する関数を s_i に設定すれば, 上の過程を満たすと考えられる.

1 種類の MPI イベントでも, たとえば実行時間の違いによって分類することで, 時間情報を含むログの圧縮を行うことができる. 以降では, イベント e_i の実行時間を d_i, 実行時間によって作成したヒストグラムの, 実行時間 t における値を $Freq(t)$ とするとき, s_i を

$$s_i = \frac{1}{f(Freq(d_i))} \tag{2}$$

のように定義することを考える. f は 0 より大きい値をとる任意の単調増加関数を使用することができるが, 3 章で述べる実験では, f の形式として,

$$f(h) = 1 \tag{3}$$
$$f(h) = h \tag{4}$$
$$f(h) = h^2 \tag{5}$$

の 3 つの場合について評価を行う. 式 3 は, すべてのイベントが同一の重要度であることを示し, 式 4 は, ヒストグラムの各階級から, 同数のサンプルを抽出することを示す. また, 式 5 は, 頻度が低いイベントに, より重みをつけてサンプリングを行うことを示す. ヒストグラム $Freq(t)$ は, t の値を, 1μsec 以下, 10μsec 以下, 100μsec 以下, 1$m$sec 以下, 10$m$sec 以下, 10$m$sec 以上の 6 種に分類して作成する.

3. 実　　験

3.1 実験環境

実験は東京大学の HA8000 クラスタシステム上で行った. HA8000 クラスタシステムは, 東京大学がサービスを提供するスーパーコンピュータシステムで, 筑波大学, 東京大学, 京都大学の 3 大学で定められた「T2K オープンスパコン仕様」[8] に基づき日立製作所が製作した 952 ノード, 約 15,000 コア, ピーク性能 140TFLOPS のクラスタ型コンピュータシステムである[7]. 各ノードは AMD quad-core Opteron(2.3GHz)4 ソケット, 合計 16 コアから構成されており, ノードあたりの記憶容量は 32GB である. この実験は, HA8000 クラスタシステム 512 ノードサービス[9] の, 東京大学情報基盤センター内での試験運用として行われた. 実験では 512 ノードを使用, 各ノードで 16 プロセスを実行し, 合計 8,192 プロセスでの実行を行った. 実験に使用した各ノードは, 4 つの Myrinet-10G(1 リンクあたり 1.25GB/sec ×双方向) でネットワークに接続し, フルバイセクションバンド幅が確保される方法で接続されている.

3.2 使用するベンチマークプログラム

実験に用いる並列アプリケーションは, 一定時間のローカル計算と, 全プロセスでの Allreduce 処理を交互に繰り返す, 人工的に作成したベンチマークプログラムである. このプログラムは共役勾配法で疎行列を係数に持つ連立一次方程式を解く場合[10] など, 同期の多い並列アプリケーションにみられる特徴的な通信パターンをもとに作成したものである. 2^n プロセスでの実行時, 1 回の Allreduce 処理は n 回の MPI_Sendrecv を実行することで実現できる. それぞれの MPI_Sendrecv の開始時刻と終了時刻を計測してログを生成し, ログ圧縮

プロセスに渡して処理を行う．このプログラムでは, 1 秒間に約 100 回の Allreduce 処理を行い, 全部で約 34,000 回の繰り返しを行う．並列計算は $2^{13}(= 8,192)$ プロセスで実行するため, MPI イベントの発生頻度は 1 秒間に約 1,300 回, 全体で約 44 万回である．

また, ログは圧縮プロセスに渡すだけでなく, アプリケーションのメモリ上にも保持し, 実行終了後にファイルに書き出し, オフラインでの圧縮との比較に使用する．

3.3 ログ圧縮ツール

ログの圧縮は, 1 つの MPI プロセスに対して 1 つのプロセスを割り当てて行う．圧縮プロセスは, それぞれの MPI プロセスとパイプで通信を行い, ログの処理を行う．圧縮を行うまでに蓄積するイベントの個数 (n) は 100,000 とし, サンプリング数 (x) は 100 とする．今回行う実験では, MPI イベントの種類はすべて MPI_Sendrecv である．このイベントの異なる実行時間に対して, 2 章で述べたようにヒストグラムを作成し, イベントのサンプリングを行う．

実験では, 圧縮プロセスの使用 CPU 時間を計測することにより, オンライン圧縮にかかる負荷を測定する．また, 圧縮の効率, 選択されたイベントの分布について評価を行う．

3.4 実 験 結 果

3.4.1 CPU 使用率

提案するアルゴリズムでログを圧縮する際の CPU 使用率 (%) を, **図 1** に示す．CPU 使用率の値は, 1 コア使い切っている状態を 100%としたときの相対値で, 全 MPI プロセスの平均値である．`f=1` は 2 章の式 3 を使用した場合, `f=h` は式 4 を使用した場合, `f=h*h` は式 5 を使用した場合である．また, 比較のために, `gzip` コマンドと `gzip --fast` コマンドでログをオフラインで圧縮した場合の結果を載せる．gzip の結果は, gzip コマンドの実行時間をアプリケーションの実行時間 (339 秒) で割った値である．図より, f の形式によって値はほとんど変わらず, `gzip --fast` コマンドと同程度の CPU 使用率で圧縮を行うことができることが分かる．またその CPU 使用率は, 平均して 0.06% 程度であり, アプリケーションの動作を大きく妨害しないと期待できる．

3.4.2 データ削減率

図 2 にログ圧縮のデータ削減率を示す．縦軸には, 圧縮後のログサイズの, 圧縮前のサイズに対する割合を示す．`gzip` コマンドでは, 圧縮後のログのサイズはもとのログファイルのサイズの 3 分の 1 程度であるのに対し, 提案手法ではあらかじめ設定したサンプル数 x と蓄積するイベント数 n から決定される $1/1,000$ にデータを削減できていることがわかる．なお, $f = h * h$ の場合にサイズがさらに小さくなっているのは, 複数回のサンプリングで同じイベントが選択されたため, 選択されたイベントの数が x よりも少なくなっているからである．

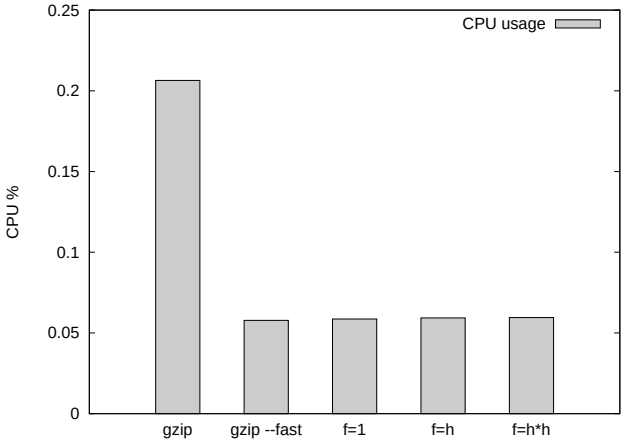

図 1 ログ圧縮プロセスの CPU 使用率
Fig. 1 CPU usage of log-compression process

3.4.3 データの分布

図 3 に, 圧縮前と圧縮後のデータ分布をヒストグラムで示す．縦軸は対数スケールで示したイベントの件数である．`Original` は圧縮前のデータ分布である．`f=1` では, 各イベントを等確率でサンプリングするため, 頻度が小さい $10msec$ を超える場合を 1 件も選択できていない．`f=h` では, ヒストグラムの各階級からほぼ同じ件数だけサンプリングできていることがわかる．また, `f=h*h` では, 3.4.2 で示したように, データのサイズは $1/1,000$ 以下になっているにもかかわらず, まれに (90 回程度) しか発生していない, MPI イベントの実行時間が $10msec$ を超える場合を, ほとんどもらさず保持することができていることがわかる．

4. 関 連 研 究

大規模な並列計算を想定した, トレースログを削減する研究は, ここ数年活発になりつつある．MPI アプリケーションを実行して, あとから再生可能な MPI イベントトレースを生成する研究[6] では, ノード内での圧縮とノード間での圧縮の, 2 つの方法を使ってログのサイズを大幅に削減している．ノード内での圧縮では, プログラムの中のあるループの中で発

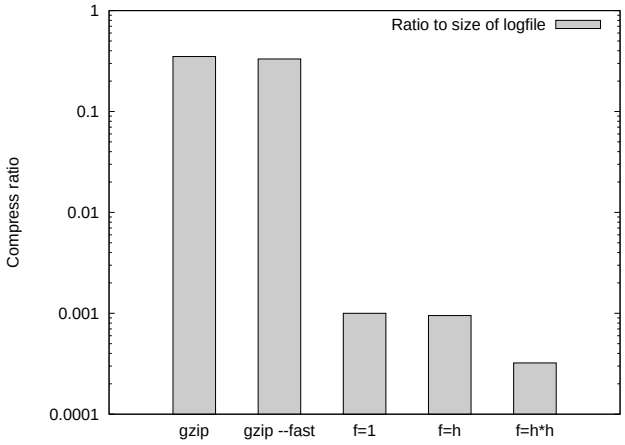

図 2 ログ圧縮のデータ削減率
Fig. 2 Data reduction ratio of log compression

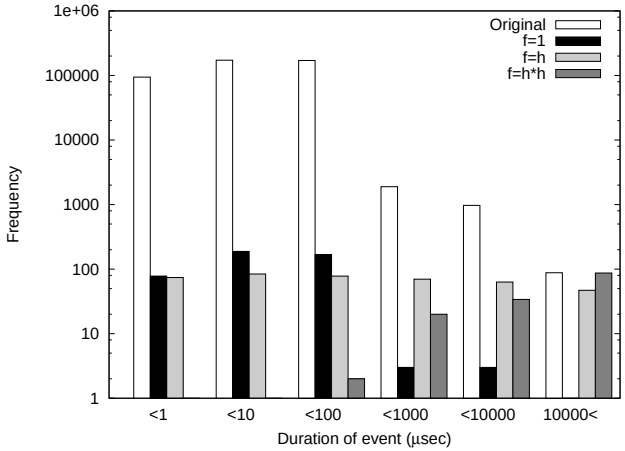

図 3 圧縮前後のデータの分布
Fig. 3 Distribution of event data

生する MPI イベントの列を表す RSD(Regular Section Descriptor)[3] を, 多重ループに拡張[4] したものを用い, 連続するイベント列をランレングス圧縮している. また, ノード間では, 通信相手の rank を自分からの相対値として保持し, 類似のログのマージすることで圧縮を行う. この手法自体は時間データを保持することを想定していない.

また, すべてのノードからログを収集するのではなく, 一部のノードだけをサンプリングしてデータを取得し, 全体のサイズを削減する方法が提案されている[5],[2]. これをさらに拡張し, 各ノードのログを並列にクラスタリングして, その結果を次回以降のノードのサンプリングに反映させる研究も行われている[1].

われわれの提案する圧縮手法はイベントの頻度をもとに重要そうなイベントを判定するものであり, 上記の両方の手法とも組み合わせて利用することで, より高いデータ圧縮率を達成できると考えられる.

5. おわりに

本稿では並列アプリケーションの膨大なトレースログを, イベントの重要度に基づいて効率的にオンライン圧縮するアルゴリズムを提案し, その CPU 使用率や圧縮率などについて評価を行った. 本研究は, 現時点ではアルゴリズムの基本的な部分の評価を行った段階である. 今後は主に以下の点を考慮して研究を進め, 実用的なソフトウェアとして完成度を高める予定である.

- 入れ子ループや, より複雑な通信パターンへの対処
- ノード間のデータ類似性に着目した, データ圧縮
- PMPI 等を用いた, 低オーバーヘッドで, より一般的なログ生成処理の実装
- 取得したログデータから実際の問題解決に役立つ情報の (自動) 抽出

謝辞 本研究の一部は文部科学省次世代 IT 基盤構築のための研究開発「e-サイエンス実現のためのシステム統合・連携ソフトウェアの研究開発」の支援を受けている.

参 考 文 献

1) Gamblin, T., de Supinski, B.R., Schulz, M., Fowler, R.J. and Reed, D.A.: Clustering Performance Data Efficiently at Massive Scales, *In IEEE International Conference on Supercomputing (ICS)* (2010).
2) Gamblin, T., Fowler, R. and Reed, D.A.: Scalable Methods for Monitoring and Detecting Behavioral Equivalence Classes in Scientific Codes, *In IEEE International Parallel and Distributed Processing Symposium (IPDPS)* (2008).

Vol.2010-HPC-126 No.22
2010/8/4

3) Havlak, P. and Kennedy, K.: An Implementation of Interprocedural Bounded Regular Section Analysis, *IEEE Transactions on Parallel and Distributed Systems*, Vol.2, No.3, pp.350–360 (1991).
4) Marathe, J., Mueller, F., Mohan, T., Supinski, B. R.D., Mckee, S.A. and Yoo, A.: METRIC: Tracking Down Inefficiencies in the Memory Hierarchy via Binary Rewriting (2003).
5) Mendes, C.L. and Reed, D.A.: Monitoring Large Systems Via Statistical Sampling, *International Journal of High Performance Computing Applications*, Vol.18, No.2, pp.267–277 (2004).
6) Noeth, M.J., Xiaosong, D., Dr, M. and Xie, T.: Scalable compression and replay of communication traces in massively parallel environments, *In IEEE International Parallel and Distributed Processing Symposium (IPDPS)* (2007).
7) T2K オープンスパコン (東大)：http://www.cc.u-tokyo.ac.jp.
8) The T2K Open Supercomputer Alliance: http://www.open-supercomputer.org.
9) HA8000 クラスタシステム512 ノードサービス：
http://www.cc.u-tokyo.ac.jp/use_info/512node.
10) 中島研吾：マルチコアクラスタにおける有限要素法アプリケーションのための階層型領域間境界分割に基づく並列前処理手法，情報処理学会 研究報告 (HPC-119-18)，No.119, pp.103–108 (2009).