第57回 プログラミング･シンポジウム 2016.1.8-10

深化する記憶装置階層のための大規模データ処理基盤の提案

松宮 遼[1,a)] 遠藤 敏夫[2,b)] 大山 恵弘[1,c)]

概要：Hadoop や Spark といった大規模データ処理基盤の登場により，開発者は並列分散処理についての深い知識がなくても大規模データを処理するプログラムを記述できるようになった．本発表では，不揮発性メモリを加えた新しい階層を持つ計算機のための大規模データ処理基盤の Hiskell を提案する．Hiskell では，システム管理者によって事前に与えられた記憶装置の情報をもとに，アクセス頻度の高いデータを高速な記憶装置に，アクセス頻度の低いデータを低速な記憶装置に動的に配置する．

キーワード：並列分散システム，大規模データ処理，記憶装置階層，不揮発性メモリ

1. 背景

統計処理やグラフ解析で用いられるデータは年々増大している．そのようなデータは，Hadoop [1] や Spark [6] といった大規模データ処理基盤を用いて並列で処理されることが多い．Hadoop や Spark では，Map や Reduce といった並列処理における頻出パターンを抽象化させたもの (スケルトン) の組み合わせを開発者に記述させる．ノード間での同期やロックの処理はスケルトンの内部に隠蔽されているため，開発者は容易に大規模データ処理のプログラムを記述できる．

計算機における記憶装置の階層は長い間，揮発性メモリと HDD，そして HDFS [4] などの分散ファイルシステムにより構成されてきた．そのため既存の大規模データ処理基盤ではそのような階層を仮定している．しかし最近は，揮発性メモリと HDD だけでなく，NAND ゲート型の SSD をはじめとする不揮発性メモリも搭載している計算機が存在している．そのような計算機では，揮発性メモリよりも大容量で低速な記憶装置として不揮発性メモリを扱うことで，並列処理の速度を向上させる場合があることが知られている [3, 8]．

不揮発性メモリは 2015 年現在，NAND ゲートを用いたものや eMMC を用いたものが普及している．しかし今後は ReRAM や MRAM を用いたものも普及する可能性がある．これらの新しい不揮発性メモリは，既存の不揮発性メモリよりも高価であるが，高速である．言い換えれば，既存の不揮発性メモリよりも高速で揮発性メモリよりも大容量な記憶装置として，既存の不揮発性メモリを含めた記憶装置階層に追加される可能性がある．つまり不揮発性メモリを仮定した計算機では，複数の不揮発性メモリが混在している計算機も仮定する必要がある．

[1] 電気通信大学
[2] 東京工業大学
[a)] r.matsumiya@ol.inf.uec.ac.jp
[b)] endo@is.titech.ac.jp
[c)] oyama@inf.uec.ac.jp

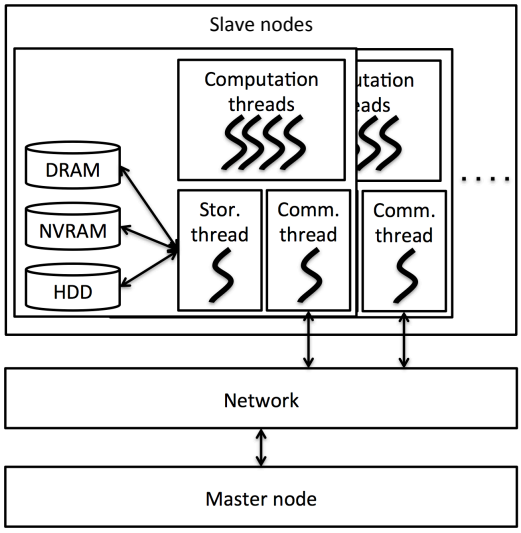

図 1 Hiskell の構成

```
1  void hist(const Array<int> *src,
2      Array<Pair<int, int>> *dst)
3  {
4    Array<Pair<int, int>> *intermediate
5      = new Array<Pair<int, int>>();
6    map(src,
7        intermediate,
8        [](int x)
9        {
10         return Pair<int, int>(x, 1);
11       });
12   reduce_by_key(intermediate,
13       dst,
14       [](int x, int y)
15       {
16         return x + y;
17       });
18   delete intermediate;
19 }
```

図 2 Hiskell を用いたプログラムの例

2. Hiskell

2.1 構成

本発表では深化する記憶装置階層のための大規模データ処理基盤 *Hiskell* を提案する．Hiskell は C++11 で記述されているライブラリで，開発者は Hiskell が提供するヘッダファイルをインクルードすることで Hiskell を用いたプログラムが開発可能となる．Hiskell の構成を図 1 に示す．Hiskell を利用するシステムは，マスタースレーブ型により構成されている．Hiskell では，Hadoop や Spark と同様に，スケルトンの組み合わせを開発者に記述させる．作成されたプログラムは，まずマスターノード上で実行され，各スレーブノードにタスクの割り当てを行う．

1 つのスレーブノードは，1 つのコミュニケーションスレッド，1 つ以上の計算スレッド，1 つのストレージスレッドからなる．コミュニケーションスレッドは，マスターノードや他のスレーブノードとの通信を行うスレッドである．計算スレッドは，マスターノードからの命令に応じて計算処理を行うスレッドである．ストレージスレッドは，計算スレッドが行う処理のために記憶装置間のデータ転送を行うスレッドである．

スレーブノードは起動すると，システム管理者によって事前に与えられた設定ファイルを読み込む．この設定ファイルには，各記憶装置の容量や速度といった情報が記述されている．

2.2 Hiskell を用いたプログラム

Hiskell を用いた場合のプログラムの一例として，1 次元整数配列におけるヒストグラムを算出するプログラムのソースコードを図 2 に示す．`hist()` は整数型の配列 `src` を受け取り，そのヒストグラムをキーバリューペアの配列 `dst` として出力する．例えば，`src` に `[3,5,2,1,2,3,4,4,2]` を与えると，`dst` には `[(1,1),(2,3),(3,2),(4,2),(5,1)]` が代入される．ここで，配列を表すクラス `Array` は C++の標準ライブラリ (STL) によって与えられるものではなく，Hiskell で定義されたものである．つまり，`src`, `dst`, `intermediate` の 3 つは Hiskell によって管理されているデータである．

6 行目の `map()` は第一引数に与えられた配列について，その配列の全ての要素に第三引数に与えた無名関数を適用するものである．無名関数の

返り値は第二引数に与えられた配列に格納される．このソースコードでは，src の任意の要素 x について，キーバリューペア (x, 1) を作成し，intermediate に格納している．ここでキーは x，バリューは 1 である．

12 行目の reduce_by_key() は，第一引数として与えたキーバリューペアについて，同じキーを持つ全てのペアのバリューを第三引数に与えた無名関数で結合するものである．結合された結果は第二引数に与えられた配列に格納される．この例では intermediate の任意のキーについて，そのキーに対応する全てのバリューの総和を計算する．計算結果は，そのキーと求められた総和によるキーバリューペアの形で dst に格納される．

2.3 データの配置

Hiskell 上で管理されているデータはスレーブノード間で分割される．分割されたデータは，各スレーブノードのストレージスレッドによって，そのスレーブノード内の記憶装置に配置される．この時，データが配置される記憶装置上の領域は，そのデータを表す変数が new された時に確保され，その変数が delete された時に解放される．

全てのデータが揮発性メモリの容量内に収まるのならば，全てのデータを揮発性メモリに配置すればよい．しかし Hiskell では，揮発性メモリの容量を超えるデータを扱うことも仮定する．そのような場合，アクセス頻度が高くなると考えられるデータは高速な記憶装置にストレージスレッドが動的に配置する．このストレージスレッドの処理は，計算スレッドの処理とは非同期である．

動的なデータ配置のためには，各データのアクセス頻度を適宜予測する必要がある．スケルトンを用いたプログラミング環境では，どのスケルトンをどの変数について適用するかが分かれば，この予測は可能である．例えば，配列の全ての要素に対して同一関数を適用し，その結果を返す map 処理の場合，入出力となる配列の各要素には高々 1 回しかアクセスしないため，一度処理された配列の要素は，その map 処理が実行されている間は二度とアクセスされない．このようにして Hiskell では，スケルトンとそのスケルトンに与えられた変数の情報をもとに，各データのアクセス頻度を予測する．

3. 関連研究

Watkins ら [5] は並列処理プログラミングフレームワークの Legion [2] に対して，本研究が対象とする，揮発性メモリ，不揮発性メモリ，HDD，分散ファイルシステムからなる新しい記憶装置階層に対応する拡張を行った．Legion で管理されるデータ領域にアクセスする際は，どのようなアクセスを行うか (読み込みのみなのか，書き込みのみなのか，読み書き両方行うのか) や，そのデータ領域に対する排他制御はどのようになっているかという情報を開発者がソースコード中に記述する必要がある．Hiskell では，そのような情報はスケルトンによって隠蔽されるため，開発者が改めてソースコード中に記述する必要はない．

佐藤ら [8] は SSD と GPU を用いたソート処理のプログラミング手法を提案した．滝澤ら [7] は深化する記憶装置階層に向けて，MapReduce 処理系を対象としたデータアクセスの局所性を考慮したタスク配置について検討している．彼らが対象としている記憶装置階層は京コンピュータに限定されるものであり，本研究が対象としている不揮発性メモリは含まれていない．

4. まとめと今後の予定

本発表は，不揮発性メモリの普及に伴う新しい記憶装置階層に向けた大規模データ処理基盤 Hiskell を提案するものである．今後は本発表での内容と，本発表で行った議論の結果をもとに Hiskell を実装し，公開する予定である．

謝辞 本発表を行うにあたり，東京大学の佐藤重幸博士より有益な助言を頂いた．また本研究の一部は JST CREST の支援を受けている．

参考文献

[1] Apache: Welcome to Apache Hadoop!, https://hadoop.apache.org/.
[2] Bauer, M., Treichler, S., Slaugher, E. and Aiken,

A.: Legion: Expressing Locality and Independence with Local Regions, *Proceedings of the 25th International Conference for High Performance Computing, Networking, Storage and Analysis (SC '12)* (2012).

[3] Midorikawa, H., Tan, H. and Endo, T.: An Evaluation of the Potential of Flash SSD as Large and Slow Memory for Stencil Computations, *Proceedings of the 2014 International Conference on High Performance Computing and Simulation (HPCS '14)* (2014).

[4] Shvachko, K., Kuang, H., Radia, S. and Chansler, R.: The Hadoop Distributed File System, *Proceedings of the IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST '10)* (2010).

[5] Watkins, N., Jia, Z., Shipman, G., Maltzahn, C., Aiken, A. and McCormick, P.: Automatic and Transparent I/O Optimization With Storage Integrated Application Runtime Support, *Proceedings of the 10th Parallel Data Storage Workshop (PDSW '15)* (2015).

[6] Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., Franklin, M. J., Shenker, S. and Stoica, I.: Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing, *Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI '12)* (2012).

[7] 滝澤真一郎，松田元彦，丸山直也：局所性を考慮した大規模並列タスクのワークフロー実行に向けて，情報処理学会研究報告ハイパフォーマンスコンピューティング，Vol. 2015-HPC-151 (2015).

[8] 佐藤 仁，溝手 竜，松岡 聡：GPU アクセラレータと不揮発性メモリを考慮した外部ソート，情報処理学会研究報告ハイパフォーマンスコンピューティング，Vol. 2015-HPC-150 (2015).