2B-6

MapReduceを用いたログ間の依存関係ツリーの抽出アルゴリズムの提案

張　一凡　　竹内　格

日本電信電話株式会社 情報流通プラットフォーム研究所

1. 概要

ある状態から操作によって別の状態に変化するシステムを考える.同一状態に対して複数の操作種類が存在する場合,状態遷移図はツリー状に繋がり,初期状態の数だけのツリーを構成する.本研究では大量のログデータに含まれる上記を例とするツリーを高速に抽出するアルゴリズムを提案する.

大量なログデータの分析には分散処理が有効である.しかし,ツリー構造の抽出にはログ間の関連分析や,関連ログから続く連鎖的な検索処理が必要となるため,処理の分散化が困難であった.提案アルゴリズムではMapReduce でのソート処理を活用し,これらの課題を解決する.

2. ツリー抽出のメリット

例としてレコメンデーションではユーザの操作の遷移ツリー（過去の履歴）を活用することで精度向上が期待できる.つまり 1→2→3 と順番に操作したユーザに対して次の操作を予測するとき,操作 3 との近似度から操作 4 を予測する既存技術より, 1→2→3→4 となる過去の事実に基づいたツリーを参照し操作 4 を予測する方が,レコメンデーションとして精度が高いと考えられる.

また,故障等の原因究明などにおいても,故障機器のログから状態の遷移ツリーを抽出することで,故障の原因を特定できるなど,幅広い分野でメリットを発揮する.

3. 大規模ログでのツリー抽出課題

一般的に大規模ログには複数の変遷ツリーが含まれるため,予め検索開始点が与えられることを前提とした既存技術(Distributed Tree Search) [1]の適用は難しい.ログを起点に並列にツリー抽出を行う時,同一ツリーを複数のプロセスで処理する事が頻発し,重複処理による効率低下が懸念される.このため,処理プロセスをツリー単位に分割するため,ログをツリー毎に分離する手法の考案が課題となる.

4. 提案手法

提案手法では MapReduce を駆使しログから根,枝,葉の抽出を並列で処理(図 1Map1～Reduce1)する.次に「根」毎にツリー構築処理(図 1Map)を行い,処理の並列度を高める.

提案手法では 2 段の MapReduce を用いる(図 1).MapReduce の概要は [2]を参照.

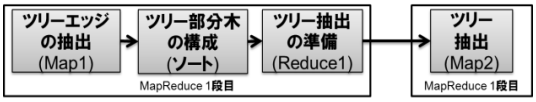

図 1 提案アルゴリズムの構成

以下では図 2 のログを例として提案アルゴリズムにおける各処理の概要を示す.

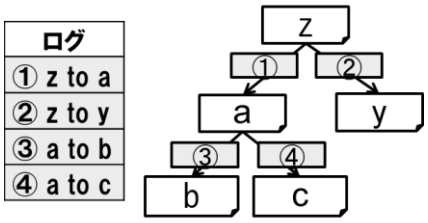

図 2 ログと抽出される状態遷移ツリー

〈ツリーエッジの抽出(Map1)〉

Map では一行ずつログを読出し,抽出プロセスが実行される.該当プロセスでは,「a to c」（商品 a を見てから商品 c を見た）を例とするログの入力に対して a:to_c と c:from_a の双方を中間データとして出力する（これらを「ツリーエッジ」と呼ぶ）.この処理はログの行毎に分散処理される.

〈部分木の構成(ソート)〉

ソートでは Map1 が出力したツリーエッジをそれぞれが持つ Key で集約する.図 2 のログのソート結果として図 3 左側の形で Key(z)と Value(to_a,to_y)による集計値が

MapReduce algorithm for finding dependency trees from BigData

Zhang Yifan, Takeuchi Kaku (Nippon Telegraph and Telephone Corporation)

Email:tyou.iifan@lab.ntt.co.jp

生成,出力される.MapReduce ではソートは分散し自動実行される.

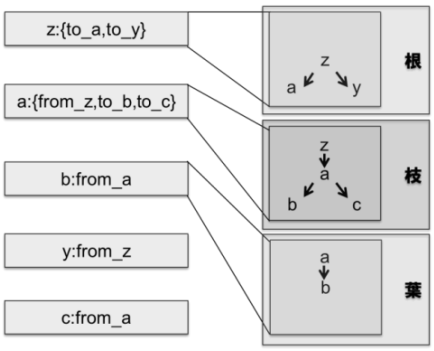

図 3 提案手法:ソート結果

〈ツリー抽出の準備(Reduce1)〉

ソートで出力される図 3 の Value に着目すると,from_, to_が含まれるかどうかで,図 3 右側の様に Key の示す状態が派生ツリーにおいて根,枝,葉のいずれに相当するかが容易にわかる.

ツリーは根から枝,葉を繰り返し取得し,追加すること(Map2 相当)で構成できる.その構成を可能にするため Reduce1 では根,枝,葉を以下のように処理する.

・根に当たるデータ(図 3 z:{to_a, to_y})を受信したプロセスは,根をトレース処理の分散処理単位の起点としファイルに出力する.

・枝に当たるデータ(図 3 a:{from_z, to_b, to_c})を受信したプロセスでは,根から枝をたどってツリーを構成する(Map2)処理で枝をたどれるように,データベースに登録する.ただし,データベース登録時では,根から枝,葉へのデータ取得を想定し,派生先(to_b, to_c)の情報は残すが,派生元(根)方向の情報(from_z)は省略する.

・葉に当たるデータ(図 3 b:from_a)を受信したプロセスでは,葉は根か枝のいずれかから必ず to_として参照されており,不要なため削除して出力,登録を行わない.

Reduce1 では入力ログに記載された状態毎に分散処理される.

〈ツリー抽出(Map2)〉

Map2 では Reduce1 の出力に従い,根毎に処理が実行される.各プロセスでは入力された根からの派生先(図 3 の場合 ”to_a”,”to_y”)を取得し,派生先を Key として枝が登録されているデータベーステーブルに対して再帰検索を行う.再帰検索により根から関連する全ての枝,葉を取得でき,大量のログから図 2 右の派生図を抽出できる.

Map2 では枝,葉の情報を再帰検索より取得しているが,検索先情報が葉(図 3 to_y, to_b, to_c)の場合は Reduce1 により情報を登録されておらず,この検索も省略できる.その省略を実現した改良版アルゴリズムで性能評価を行なった.

5. 性能評価

実装評価の結果(図 4),ログ量の増加にともなってスループットが下記の考え方に基づいて導出した理論上限に漸近する結果が得られた.

本来,提案手法は処理の並列数でスケールアウトするが,枝の検索に用いる HBase が性能ネック [3]となるため,理論上の上限値は約 10000 ログ/秒でおさえられるためである.入力ログ数が少ない時のスループットの低下は,Hadoop の分散オーバヘッドによるものであり,チューニングによって性能の向上が期待できる.

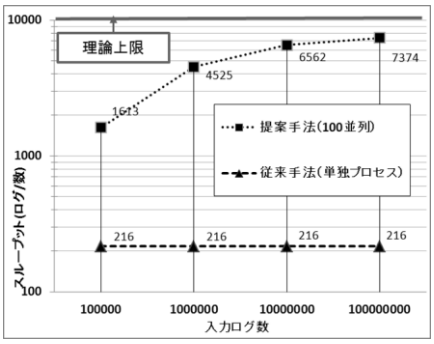

図 4 性能評価(環境:PC8 台,CPU:Xeon 4 コア*2, MEM:48GB, HDD:5TB)

6. まとめ

本研究では MapReduce を利用したツリー抽出アルゴリズムを提案した.評価の結果,外部の処理ネックは存在するものの,現状でも単独プロセス処理よりスループットが 30 倍以上と大幅に向上し,クラウドサイズのデータ処理に利用できると期待している.

[1] M. J. Quin, Parallel Programming in C with MPI and OpenMP, McGraw-Hill Education, 2008.

[2] T. Wbite, Hadoop, O'reilly, 2010.

[3] B. F.Cooper, “Yahoo! cloud serving benchmark,” 2010.03.