Performance Debugging for Distributed Systems of Black Boxes

Presented by 多田 好克 (Tada Yoshikatsu)

SOSP 読み会 at 電気通信大学 on Nov. 15, 2003.

0 論文の概要

分散システムを**暗箱 (black-box) として捉え**、処理の隘路 (bottleneck) 発見の手助けをする方法論の提案、実践、評価。11 章からなり、**以下のような構成**になっている。

1. 導入
2. 問題点と目標、非目標
3. 関連研究
4. アプローチの概要
5. 提案するアルゴリズム（**２種類**）
6. トレースの取り方
7. 実験とその結果
8. –11. 今後、サマリー、謝辞、参考文献

ヘテロで手を加えられない大きなシステムのつもり。

8.〜11. 章は省略...

nesting algorithm（入れ子）と convolution algorithm（畳込み）。

1 導入

Web ベースのシステムなどの、分散**マルチタイヤ (multi-tired) システム**の虫取りや性能解析用のデータを提供する。特に、様々なソフトの寄せ集めで構成され、手の出しようのない暗箱の集合体も対象にする。つまり、ソフトやネットワークに手を入れずに有用な情報を提供する。

目指すシステムはアプリ独立で、ノード間の通信を受動的 (passive tracing) に収集する。また、そのデータをオフラインで解析し、見やすく提示する。**知らなくて良い**し、対象とする分散システムの構造とか機能、メッセージの内容などは変更する必要もない。

って、様々なサーバを呼び合うようなシステムのこと？

もちろん、知っていると、より良い情報が提供できる。

2 問題点と目標、非目標

分散システムを**構成要素のサーバやクライアント**と、それらを結ぶ**通信路**のグラフとして捉える［図１］。ノードとエッジは、それぞれに処理時間 (delay, latency) が異なり、また、**呼ばれ方によっても**処理時間は異なる。ここでは、**エッジの処理時間は考えない**。図１は RPC を元にしたシステムであるが、**メッセージ伝達型のシステム**にも対応できる。

a) **影響の大きい**因果経路 (causal path) パターンを見つける、b) 遅延の**原因となるノード**を同定する、ためのツールを提供することが目的である。

○ 対象システムの知識を必要としない	× データを提供するだけ、判断は人間
○ 対象システムに手を加えない	× 妥当なシステムを保証するものではない
○ 対象システムに外乱を与えない	× ベンチマークとしては使えない
○ ツールの使い方が簡単	× 実時間でデータを提供するものではない

データの正確さは他の解析に照らして示すが、**有用かどうか**は読者の判断に委ねる。

ノードと呼ぶ。計算機と思って良い。

エッジと呼ぶ。要するにネットワークのこと...

もちろん、同じように呼ばれても時間の分散はある。

LAN だから... 仮想遅延ノードを考えても OK だし...

e-mail システムなど...

実行回数が多く、処理時間が長い...

キャッシュする認証サーバは原因ではない...

人間の判断が必要なので、示すのは難しい...

3 関連研究

同じ方法、同じ問題意識: 文献 [11] は特別な (Mlog) 言語を使う。magpie[13] はより複雑。また、稀な事象に重きを置いている。DPM[18] はカーネルに手を入れている。商用のシステムは JVM とか.net を対象にし、**範囲が狭い**。

legacy ノードには使えない。

異なる方法、同じ問題意識: 文献 [26] は実時間が目的で、プログラマが対象システムにデータ取得ルーチンを埋め込む。ETE[10] も同じで暗箱アプローチではない。

同じ方法、異なる問題意識: Pinpoint[5,6] は異常ノード(fault node)を見つけるのが目的。文献 [3] も同様で、対象システムを**止める**こともある。文献 [2] は異常の埋込み(fault injection)をするので、暗箱ではないし、対象に対するそれなりの知識が必要。文献 [27] は侵入検出が対象、文献 [12] は WAN のネットワーク解析が対象。

つまり、対象システムに外乱を与える。

4 アプローチの概要

図２上のような状況で、図２下のような情報を提供する。ツールは奇数矢印の時刻から、偶数矢印の時間を推測する。上のような呼出し関係を知らずに、2, 6, 10 の時間を提供するのが目的。そのために、以下の手順を踏む。

トレース情報収集: これだけは実時間で実行する。 あちこちで集めたものを**時刻でソート**してまとめる。詳細は６章で...

時刻同期は OK と思っている。また、時刻順のソートも対象外...

因果経路やパターン推測: ここからはオフラインの処理。トレースは大量で**不完全**であることが前提。処理時間、使用メモリ量は少なく、堅牢である。アルゴリズムの詳細は５章で...

タイムアウトや再送なども含まれる。

視覚化: あまり重要とは考えていない（手が回っていない？）。

5 提案するアルゴリズム

アルゴリズムは２つある。１つはRPC を対象にした入れ子(nesting algorithm)アルゴリズム、もう１つはメッセージ伝達を対象にした畳込み(convolution algorithm)アルゴリズム。

5.1 入れ子アルゴリズム

トレース情報は**基本的には**（時刻、送り手、受け手、call/return）の４つ。 で、以下の４つの手順を踏む。

他の情報も取れれば組み入れることはできる。

呼出しペアと入れ子の同定(FindCallPairs): B→Cの call は $T_{opencalls}$ というハッシュに覚えて、C→Bの return が来れば対応づける。複数の call が先行した時には、最も**遅い物から順に**return に対応付ける。 call/return がマッチしないものは**捨てる**。そのペアを $T_{callpairs}$ に記録。そのペアに、$T_{opencalls}$ 内で、 この時刻以前にBを call した**∗→Bの集合**を記録する。

括弧内は図４の関数名。

妥当な入れ子の決定 (ScoreNestings): **ペア** (B, C, k, l)は、 多数のペア (A, B, i, j) に含まれる。その中から因果の深いものを同定する。 方法は全ての親に対して、(k-i) **を横軸**にしたヒストグラム（貢献度...）を作る。ただし、ヒストグラム全体の面積は１に正規化する。図５の例では、４つの場合が出来るが、$(t_3 - t_1)$ と $(t_4 - t_2)$ が同じで、他の２つより高い値を示す。

つまり、入れ子にする。多くの場合、これで OK と言っている...

親の決定 (FindNestedPairs): ヒストグラムに基づいて親を決める。 基本的には値の高いものを親とするが、以下の**３種の状況**では値を $1/x$ にするというペナルティを課す。

問題はない...

- 図５のように同じ親に複数回出てきた子は、 その**回数の２乗**を x とする。
- 既に対応付いている親子。気持ちは、その回数のｎ乗を x にしたいが、ｎは０で x は１、ペナルティなし。
- 親に対応付いている子の数。これも、上と同じでペナルティなし。

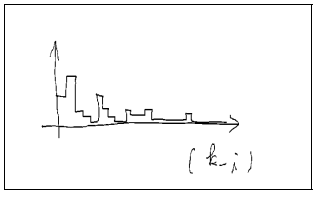
つまり、全ての親ネ。

(A, B, i, j) はA→Bが時刻 i に、B→Aが時刻 j に観測されたペアを表す。

(k-i) を短い時間間隔で離散化したものを横軸にする。しかも、対数軸でやるようだ... 1m 秒〜２時間を $\log_{1.05}$ で 340 に分けたらしい。

と言いながら、実践では１つ目だけで、後の２つは無視。

２乗という値は経験値。



なお、値が同じ場合は時間の短い親に対応付ける。

呼出し経路の決定 (FindCallPath, CreatePathNode): 個々のペアに対して、親が居なければルートに登録して、CreatePathNode を呼び出す。CreatePathNode は、再帰的に実行され、ノードの実行時間と呼出しまでの時間を収集して回る。

このアルゴリズムは、（トレースしたメッセージの個数）×（ノードごとの並列度の平均）程度の**時間、空間**で実行できる。

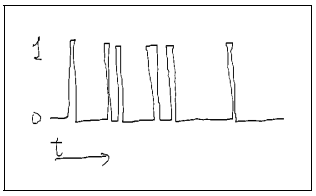
と言っても、時刻によるソートはどうするの？

5.2 畳込みアルゴリズム

このアルゴリズムは、メッセージの集合から因果関係を見つけ出す。トレースをエッジごとのメッセージに分け、それを**時刻シグナルのように処理**する。出力はA→B→A→Cのような制御の因果とその処理時間。トレース情報は（時刻、送り手、受け手）。アルゴリズムは**以下の手順**を踏む。

パルスのトリガ関係を調べる方法と同じ…

図6だけど、再帰でややこしいので、説明はプログラムに即していない。

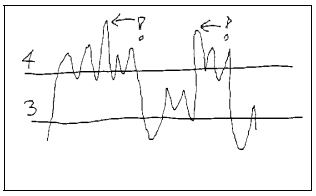


1段目の因果関係 (FindPathsFromRoot): 指定された^{root}根が送り手となるトレースの集合を求める（A→＊）。

2段目の因果関係 (ProcessNode): 個々の受け手（＊）について、さらに次の受け手の集合を求める（B→＊'）。 このA→B→＊' に対して…

畳込み (FindCausedMessages, FindCorrelation): A→BとB→＊' の２つの時刻の差ごとの共起頻度をとる。つまり、時刻を短い間隔で離散化し、そこにメッセージが存在すれば１、なければ０という**関数ｓ（ｔ）**を作る。で、 A→BとB→＊' の時刻をｓ（ｔ）で**畳込む**と、２つのメッセージの時間を横軸にした共起頻度グラフ［図７］ができる。

$(f \bigotimes g)_t = \sum_{x=-\infty}^{+\infty} f_x g_{t-x}$



４は経験的な値…

３も経験的な値…

後は適当…： 図７のようなデータから因果の候補を決める。

- 平均より標準偏差の**４倍以上**値の大きい山を求める。
- ただし、平均より標準偏差の**３倍以下**の谷が無いと候補とはしない。

全体の処理は…： という処理を深さ優先で再帰的に行い、因果経路を求める。

問題点として、以下のようなことが述べられている。

遅延の分散が小さいと良い結果が得られる。分散が大きい場合は図６の 22 行のｖを大きくする。

遅延の長いものとか異常に短いものを除くとより良い結果が得られるが、**紙面の都合で省略**。また、他にも多くの機能を加えて結果を改善しているが、ここでは述べない。でも、７章の結果はこれらの効果を**反映させてある**。

何！

許せん !!

必要な空間はＯ（ｍ＋Ｓ）。ｍは ProcessNode の扱うメッセージ数。ＳはＴ／μで、Ｔは最も長い因果の時間、μは離散化した単位時間。必要な時間は○（ｅｍ＋ｅＳ log Ｓ）で、ｅは出力のエッジ数。第２項の○（Ｓ log Ｓ）は畳込みの FFT の計算時間。経験的には第２項の時間が支配的。

5.3 ２つのアルゴリズムの比較

入れ子はRPC が前提。畳込みはどのような状況にも使える。入れ子では**A→B→C（→B）→A**とか、 遅延書き戻しのような**リターンした後に処理を依頼する**ものには対応できない。他方、畳込みはA→B→C→B→Aのパターンに対して、A→B→Aも出力してしまう。畳込みは稀に発生するパターンや分散の大きいパターンは検出できない。

つまり、入れ子を飛ばしてリターンする。

A→B→Aの後に、B→C→Bが実行される。

5.4 可視化

文献 [8] の “dot” プログラムというのがあるらしい。図８が入れ子の出力例。子供も含むノードの実行時間、子供を呼び出すまでの平均処理時間が書かれている。また、図９が畳込みの出力例。メッセージの総数と、遅延時間が頻度の多い順に並べられている。

6 トレースの取り方

個々のメッセージに対するデータ量は少ないが、長時間に渡り頻度も高い。ネットワークの帯域とかスケーラビリティとかが問題となる。 データ収集は**ネットワークの受動的なモニタリング**

packet sniffing とも言う。

が基本だが、アプリケーション、ミドルウェア、カーネルなどに手を入れる方法でも、本論文の解析手法は有効である。

つまらん仕事ぢゃ。

ネットワークの受動的なモニタリングでは、複数のパケットから**メッセージを再構成する**必要があるかも知れない。また、近年の**スイッチではモニタリングできない**。

port mirroring という機能を持ったスイッチがある。また、個々のホストでデータを得るという方法もある。

情報源が異なるものは時刻に従って統合する。2つのアルゴリズムは時刻のズレに対してもある程度対応できる。また、分散時刻問題は NTP で何とかなると考える。

7 実験とその結果

けど、より正確な因果経路導出と我々の方法とを比較できる。

実験は実際のトレースデータではやっていない... また、トレースは暗箱アプローチで取ったもの**ではない**。以下の様々な方法でトレースを用意した。

トレース生成ソフト: maketrace コマンドを作った。様々なパターンを生成、実験できる。

J2EE: J2EE で 130 万メッセージ、3時間のトレースを2つ作った。

電子メールの Received ヘッダ: 良いトレースではないが、RPC では無いものの例として... 2ヵ月で 11,683 通、81,044 の Received ヘッダを集めた。

紙面の都合と、大した結果は得られなかったからだそうだ。

その他: 分散ファイルシステムの物なども集めたが、**以下では述べない**。

以下、それぞれの結果。

だが、見るものでは無い...

7.2 マルチタイヤシステム: 図 10 のようなシステムを想定し、maketace した。WS2 と AUTH の間に 200 m秒の遅延を入れたものとの2つで実験。**図 11 が出力**。左から頻度の高い順に並んでいる。図 12 が入れ子の、図 13 が畳込みの出力の一部...

J2EE の使い方のデモソフト？

7.3 J2EE: J2EE の PetStore system。 /mylist.jsp ノードの呼出しに 50 m秒の遅延を入れたものとの2つで実験。出力された中で重要そうな因果経路の1つが図 14 と図 15。

7.4 電子メールのヘッダ: 畳込みを実験。30 秒のスライスだとすべての遅延が0となった。スライスを5秒にすると、0以外の遅延も観測された。図 16 が結果。

詳細は省略。

図 17 はマルチタイヤシステムの出現頻度の高いNパターンに対する false negative の割合。ほとんど、1/N 以下の割合に納まっている。図 18 は病的な例で、入れ子が**どう間違えるかを示した**もの。図 19 はメッセージの並列度が高い時の時間見積もりを失敗する様子を示したもの。図 20 は呼出し時間の分散を増加させた時の様子。図 21 はトレース生成が追い付かない時にメッセージを捨てることを想定した時の様子。1%以下ならば大丈夫。図 22 は時刻にズレがある時の様子。マルチタイヤシステムの WS2 の時刻を数 10 m秒進め、それに対して入れ子の比較に幅を持たせた (comp) と貢献度のグラフを平滑化した (smooth) 場合の様子。

表1は様々なメッセージ数とトレース時間による空間、時間使用量の実測値。入れ子は 1.7 G の Pen4 で Linux2.4.20 の値。畳込みは 667 M の AlphaServer で、Tru64 Unix V5.1 の値。特に畳込みの処理時間ははスライスの幅に依存するが、それが○（S log S）の線に乗ることを図 23 に示した。

8〜11 今後、サマリー、謝辞、参考文献

と言いながら、単に今後の話だけだが...

8〜11 章は**以下の通り**。

- より実践的なトレースで実験したいが、データの入手が困難。
- ロックのあるというか同期するシステムには適応できない。
- 入れ子の処理を、時間の窓を作ってその中で処理をする。処理にフェーズのあるシステムでは、短期間にだけ見られるパターンが抽出できる。また、使用空間量が減らせる。