# TCP セッションへの識別子付与による 複数プロセス横断可能な処理追跡法

清水裕基[†] 三好健文[†]
入江英嗣[†] 吉永努[†]

様々なネットワークサービスの普及に伴い，高信頼性を確保するため，サービスの基盤となるサーバの数が増えてきている．サーバの数を増やし並列化することで，スループットの向上や一部のサーバの死活に対応できる．しかし，サーバ構成の複雑度が増すことで，障害が発生する可能性が高くなる．障害の要因はユーザの操作ミスからアプリケーションの不完全性など多岐に渡るため，すべての障害要因を取り除く事はできない．そこで，障害が発生してもシステムが停止している時間を最小限にし，被害を軽減する手法が必要とされている．そのためには，障害の発生箇所と要因を高速に特定する仕組みが求められる．

本論文では，サービスへの処理要求に対して識別子を付与する手法を提案する．識別子は，システムコールの `accept` を修正した `iaccept` により，TCP セッションの確立毎に付与される．付与された識別子を追跡することで， 従来は難しかった複数サーバを横断しての処理追跡がおこなえる．既存のサーバログを識別子を用いて検索することで処理追跡をおこなう実験により，提案手法の有効性を示す．

# A Data Flow Trace Method over Multiple Processes by Using Additional Identifier for Each TCP Session

Hiroki SHIMIZU,[†] Takefumi MIYOSHI,[†] Hidetsugu IRIE[†] and Tsutomu YOSHINAGA[†]

Due to the rapid spread of various network services, the number of servers for providing services increases in order to improve the reliability for the services. By increasing the number of servers and parallelizing the services on them, the throughput and the availability of the services is improved. However, the complexity of the server structure increases owing to increasing the number of servers, and it heightens the possibility of disordering the system. Since the factors of disordering the system range from user operation miss to application bugs, it is impossible to remove all of them. Therefore, the scheme to minimize the out-of-service time and reduce loss by the system disordering is required. For the requirement, finding out the factor of the system disordering is strongly desired.

In this paper, a method that gives identifier for each request to service is proposed. Identifiers are given at every TCP session establishment by modified `accept` system call named `iaccept`. By tracing the given identifier, data flow of the request for the service can be traced over multiple processes. The result of experiments that traces data flow by using the given identifier and server output log shows the effectiveness of the proposed method.

## 1. はじめに

インターネットを介した多種多様なサービスが普及し，多数のユーザに提供されている．オンラインショッピングや口座の送金処理，また電子市役所など，経済及び社会的に重要な情報もインターネット越しにやり取りされている．このようなサービスを提供するシステムの場合，時間を意識せず常に処理をおこなえることや，情報流出などを防ぐ高信頼性が要求される．

サービスの基板部では大抵の場合，処理を分担し，負荷を分散させるために多くのサーバが連携して動作する．第一に処理を分担することで，サーバは特定の処理に特化することができ，単純化が可能である．サービス開発者は既存のサーバの組み合わせで基盤を用意でき，サービスの内容を作ることに集中することができる．第二にサーバを複数台用意することによる負荷分散をおこなうことで，特定のサーバに負荷が集中しないようにでき，サービスのスループット向上ができる．

しかしその一方で，障害の発生可能性は高まる．な

† 電気通信大学 大学院情報システム学研究科
Graduate School of Information Systems, The University of Electro-Communications.



ⓒ 2011 Information Processing Society of Japan

ぜならば，システム障害の要因の多くはソフトウェアバグと人為的ミスであるため[8]，組み合わせるサーバが多くなるほど，また保守しなければならないサーバが多いほど，障害の発生可能性が高まる．そのため，万が一障害が発生しても，障害の検出と復旧を素早くおこなうことで，障害が発生している時間をできるだけ短くする手法が期待される．

障害の検出には，サーバのリソース使用量の変化から検出する手法が提案されている[3)6)8)]．また，サーバが応答しない状態に陥るなど，自明に分かる場合もある．これら手法により，容易に障害の検出をおこなうことができる．

検出した障害からサービスを復旧させるためには，どのような処理要求によって障害が発生したか，また発生した箇所がどこなのかを調べる必要がある．この時最も容易に得られる情報源として，サーバのログがある．ログを用いることで，障害の要因を調査することができる．例えば Apache のログには，何時にどの IP からのどのような要求を処理したかといった情報が記録されている．同様に他のサーバでも，要求に対しておこなった処理結果がログに出力される．しかし，これらのログはサーバ単体の記録である．多くのサーバの組み合わせにより構成されるサービス全体で，入力された処理要求がどのようなサーバ連携により処理されたかを追跡する情報は存在しない．そのため，サーバの相互関係で発生する障害をログの確認からおこなうことは非常に難しく，処理要求の流れそのものを追跡する手法が必要である．

そこで本論文では，サービスに届いた処理要求に識別子を付与することで，要求が各々サーバでどのように連携して処理されたか追跡できるシステムを提案する．本提案により，複数サーバを横断しての処理追跡が可能となる．また処理の追跡により，障害箇所の特定に役立てることができる．本論文では識別子付与による処理追跡の初期評価として，容易に得られる情報源であるサーバログとの組み合わせによる処理追跡をおこなう．

2 章には，本論文が想定する問題について述べる．2.3 節では関連する研究について述べる．3 章では，提案手法の実装法や，処理の追跡手法について述べる．また評価を 4 章に示し，結果を 5 章で考察する．

## 2. 処理の追跡と関連研究

本章ではまず，想定する障害について述べ，識別子を付与することによる処理の横断追跡手法の概要を述べる．また，障害の発生箇所や要因特定が容易に出来ることを示す．最後に既存研究との比較により，既存手法の問題点について述べる．

### 2.1 障害の要因

高信頼性を求められるサービスの数は増えているが，障害の要因は減っていない．これは文献 8) によると，障害の要因はユーザの操作ミスや設定ミスのような人為的なものから，プログラムのバグなど多岐にわたるためである．障害の分類として，次の二つがある．

- 外的要因による障害
  外的要因は，DDoS 攻撃や SQL インジェクションなどの，悪意ある攻撃がある．サーバの高負荷によってシステムのダウンを狙ったり，データベースへ意図しない入力をすることで情報が漏洩したりする．また漏洩した情報の流出により，二次被害を被る可能性もある．
- 内的要因による障害
  内的要因は，主に人為的な設定ミスや，アプリケーションの組み合わせなどによって発生する．例えば検索ワードから特殊文字をエスケープするスクリプトが，その設定の影響でデータベースに異常なクエリを発行してしまうことがある．他にも，通信の文字コードの設定の不一致などで障害が発生する．

文献 8) によると，障害のほとんどは内的要因に起因する．よって 2.2 節では内的要因に起因する障害を例に，処理の追跡による障害の発生箇所と要因の特定手法を示す．

### 2.2 処理の追跡による障害箇所と要因の解析法

ロードバランササーバ 2 台，Web サーバ 3 台，データベースサーバ 2 台で構成される検索サービスを例に，処理の追跡手法を考える．ロードバランササーバ 2 台各々に対して 3 台の Web サーバが等距離に接続され，さらに Web サーバ 3 台各々に対してデータベースサーバ 2 台が等距離に接続されている構成とする．各々複数台のサーバによって処理を分担し，負荷を分散する．この構成の場合，検索のために通りうるサーバの組み合わせは 12 通りになる．何か障害が発生した場合，12 通りの中から，障害の発生箇所と要因を見つけ出さなければならないが，これは非常に難しい．

例えば特殊文字を含む検索処理が要求されたとする．ここで，Web サーバに備わっているはずの特殊文字を置換して排除する処理が期待通りに動かず，データベースに予期しない入力が与えられたとする．例えば文字列「*」が検索処理として与えられたときに，Web サーバが置換しないままデータベースに「*」を与えると，データベースの全内容が出力される．この障害により，サービスの利用者は期待した結果を得られず，かつ，サーバに予期しない負荷がかかる．

この予期しない処理の原因を最も容易に確認する手法として，サーバのログを確認する手法が挙げられる．検索処理の場合，Web サーバには置換処理をおこなった結果，データベースサーバには「*」の入力による要求を処理した結果がログ出力される．もし各々のログが，一連の処理によって出力されているとわかれば，特殊文字処理の影響でデータベースに予期しない入力


ⓒ 2011 Information Processing Society of Japan

がおこなわれ，障害が発生したという処理流れがわかる．しかしながら，それぞれのログから上記の一連の処理を見出すことは簡単ではない．これは次のような要因による．

- ログの組み合わせ可能数はサーバ台数の増加と共に複雑化し，無限の組み合わせから特定のログ組み合わせを見出すのは不可能である．
- タイムスタンプを用いて各々ログを組み合わせることはできない．これは一般にログに出力されるタイムスタンプは粒度が荒く，秒単位であるため，同じタイムスタンプを有した複数ログが存在する可能性が高い．そのため，特定の 1 つの処理に絞り込むことができない．
- ログの出力順で各々ログを組み合わせることはできない．これはログへの書き込みタイミングはサーバによって異なるためである．

このように，既存のログ出力では処理を追跡することが難しい．

そこで，タイムスタンプや処理順序を根拠とせずにも，各ログの特定の項目が一連の処理であることを判断できる手法が必要である．これを解決するために，サービスへの処理要求に識別子を付けることで，処理をサーバを横断して追跡する手法を提案する．

検索サービスの場合，追跡する対象はサービスに入力された検索要求である．本提案により，検索要求が各々サーバでどのように処理されたかを次のように追跡することができる．

(1) 処理要求に識別子を付与する．また，識別子を付与したことをログ出力する．
(2) サーバは処理要求と識別子を一緒に受け取り，要求を処理する．
(3) 要求の処理結果をログ出力する．この時，出力内容に識別子を含める．
(4) 処理結果を他のサーバの入力とする場合，処理要求と共に識別子を渡す．
(5) 処理が終了していなければ，(2) へ戻る．

本節の想定する構成の場合，識別子を付与するのは最も上位にあるロードバランササーバである．以下すべてのサーバで識別子を共有し，ログ出力に含めることで，処理がどのサーバを横断したか追跡できるようにする．

処理の追跡には，追跡したい処理の識別子を鍵に，すべてのログファイルに検索をすればよい．処理は識別子を共有してサーバ間を転送されるため，処理結果に同一の識別子を含めることができる．実装方法と追跡方法については，3 章に示す．

### 2.3 関 連 研 究

ネットワークをトレースする手法として，パケットレイヤでは従来から様々なトレース手法が提案されている．TCP/IP にはヘッダ部にパケットがどの IP から送られてきたかが書きこまれている．ヘッダ情報は例えば TCPDump を用いることで確認することができる．しかし，この情報は偽造することが可能で[7]，また DDoS 攻撃のように大量のパケットが一度に流入した場合，情報量が爆発するため，TCPDump では対処が難しい．他にも，スイッチの FDB 情報によるトレース手法などが考えられる．また InterTrack[7] では，複数追跡技術をハッシュによって相互接続することで，レイヤや物理装置なども超えた追跡がおこなえることを示している．しかし，これら技術は本提案の考える処理の追跡には向いていない．なぜなら，追跡対象の要求がパケットとして分割されてしまうため，追跡対象数が多くなってしまうためである．障害からの素早い復帰に必要なのは，サービスへの要求が各々サーバでどのように処理されたかを追跡することであり，パケットの追跡では粒度が細かすぎる．

仮想化技術により，I/O をモニタリングする手法も考えられている．BitVisor[5] はその一つで，様々な I/O を監視し，データ書き込みの暗号化をおこなう．BitVisor 同士の通信も暗号化される．暗号化したデータに対して識別子を付与することで，データの追跡をおこなうことができると期待できる．しかし，Bitvisor はハードウェア依存であることや，仮想モニタ上にサーバ環境を再構築しなければならず．環境移設のコストが高い．また，すべてのデータ I/O に対して監視をおこなうため，システム全体のパフォーマンスが低下する．パフォーマンスの低下はサービスの応答時間低下につながり，特にリアルタイム性が高く要求されるシステムにおいて致命的である．

また，サーバ群をフレームワーク Izna[4] により再構成するという提案もある．Izna は軽量なプロセスマイグレーションを用いたフレームワークで，クライアントからの要求を軽量プロセスに結びつける．この軽量プロセスをサーバ間で移送し，負荷が低いマシンが処理をおこなう．軽量プロセスのヘッダ情報に識別子を付けることで，処理要求のサーバ間移動を追跡することができると考えられる．しかし，既存のサービスを Izna フレームワーク上に再設計する必要がある．既存のサーバは利用できないため，移植のためのコストが高い．

ログの解析には，Autonomic Computing[2] により，自動化ツールが提供されている．Autonomic Computing では，ログなどの情報を収集し，記述したルールや集合知に基づき障害の検出，復旧を行う．しかしログ同士の関係は，ルール記述ないし集合知による経験によって導きだされるため，常に完全なトレースができるとは限らない．

## 3. 提案と実装

### 3.1 提 案 手 法

本節では，サービスに届いた要求に識別子を付与す



ⓒ 2011 Information Processing Society of Japan

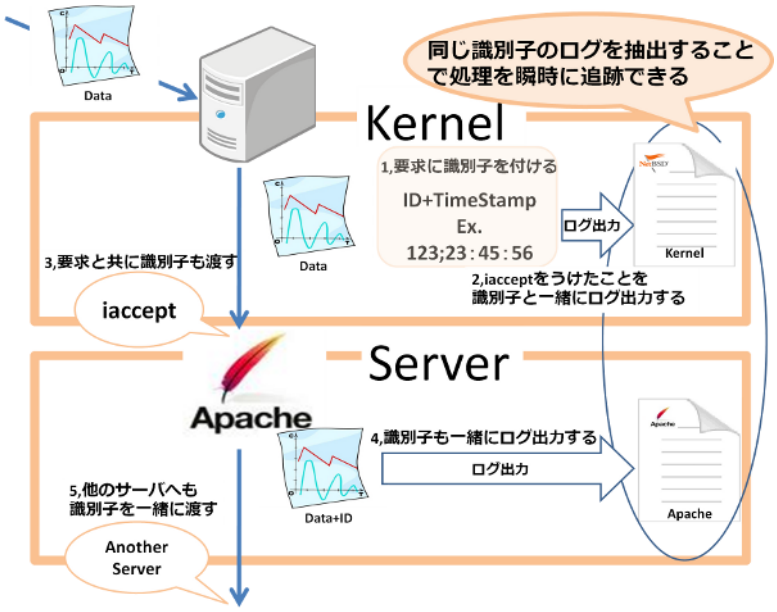


**図 1** 識別子を用いた処理の流れの追跡例
Fig. 1 An example how to trace data flow with identifier

ることで，要求が各々サーバでどのように処理されたか追跡できるシステムを提案する．2.3 節より，本提案が実現すべき点は次のようにまとめられる．

(1) 処理を追跡する要求に対して，適当な粒度による追跡が可能であること．
(2) 既存のサーバ環境ないしサーバ自体をを劇的に変化させることなく，少しの修正により追跡可能にすることができること．
(3) 追跡のための処理以外では，パフォーマンス低下が起こらないようにすること．

これらを考慮し本提案では，識別子の付与をサービスへの処理要求ごとにおこなう手法を考える．提案するシステムの全体図を**図 1** に示す．

外部と通信をおこないたいサーバは，accept を呼び出して待機する．TCP セッションの確立要求が届いたとき，OS カーネルは accept の処理をおこないソケットへの接続を受け付け，ソケットを参照できるファイルディスクリプタや接続先の IP 情報などをサーバに渡す．サーバは受け取ったファイルディスクリプタを読み書きすることで，外部との通信が可能になり，処理要求を受け取ることができる．そこで本提案では，従来の accept 処理に加え，識別子を渡すシステムコール iaccept を OS カーネルに新規作成する．識別子は iaccept が呼ばれた回数を数えるカウンタとし，iaccept が呼ばれる毎に処理要求に識別子をつける (1)．また iaccept を処理したことを示すログ出力をおこなう (2)．このログにより，サーバが何らかの要因で応答不能に陥った場合でも，iaccept により処理要求をサーバに渡したことを確認することができる．これは処理要求がどの段階まで送られたかを示すのに重要な情報となる．以後このログを他と区別するために，iaccept ログと呼ぶ．iaccept ログは，iaccept の処理を終了したタイミングのタイムスタンプと iaccept を処理したことを知らせるメッセージ，処理に割り当てられた識別子を出力する．タイムスタンプを付与することで，iaccept に対応したサーバを複数台稼動させたり，カウンタのループにより識別子が衝突した場合でも，区別できる可能性が高くなる．

サーバは iaccept によって処理要求を受け取るが，この時従来の accept によって得ていた情報に加え，識別子を得ることができる (3)．要求を処理した後，処理結果をログに記すとき，識別子をともに出力するようにする (4)．また，他のサーバに処理を引き継ぐ場合，要求と共に識別子を一緒に渡すようにする (5)．このことで識別子を複数サーバで共有でき，処理を追跡することが可能となる．

本提案システムを用いて，HTTP/1.0 を用いて 3 つの Web ページ閲覧をおこなったときの識別子付与動作を示す．HTTP/1.0 におけるサーバ・クライアント間の通信は**図 2** のようにおこなわれる．3 つのページを読み込むために，3 回の TCP セッションの確立がおこなわれる．TCP セッションの確立時，サーバでは iaccept を呼び出す．この結果受け取ったファイルディスクリプタを読み書きする事で，HTTP プロトコルに則ったメッセージの処理をおこなう．iaccept は 3 回処理されるので，識別子は 3 回付与され，それぞれ


ⓒ 2011 Information Processing Society of Japan

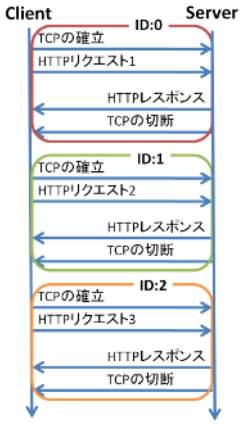


**図 2** HTTP/1.0 におけるページ要求処理フロー図
Fig. 2 Flow diagram of HTTP/1.0's page request process

を独立した処理として追跡可能になる．また iaccept の呼び出し毎に，iaccept ログが出力される．サーバは要求を処理後，ログ出力に識別子を共に出力する．このことで，iaccept ログとサーバログを横断した処理追跡が可能になる．この動作は他のプロトコルでも同様で，本提案手法は accept を呼び出す TCP 通信全てに用いることができる．

### 3.2 実装方法

システムの実装は OS カーネル部とサーバ部に分かれる．各々の書き換えにより，処理追跡がサーバを横断しておこなえるシステムが構成される．本論文ではカーネル部の実装を，NetBSD5.1-GenericOS に対しておこなった．しかし，実装の内容は汎用的なもので，他の Posix 準拠な accept を有している OS の場合，同様の実装をおこなうことができる．

OS カーネル部では NetBSD5.1-GenericOS カーネルに対して iaccept の実装をおこなう．はじめに，iaccept が識別子を渡す方法として，次の二つの手法を検討する．

(1) 構造体を拡張し，識別子を構造体の要素とする手法
(2) 識別子を独立させ，第 4 の引数とする手法

(1) の構造体を拡張して識別子を付与する場合，サーバは特に書き換えることなく，従来通り accept を用いることで識別子を得られることが期待される．しかし，例えば TCP/IP を用いたプログラミングでは，sockaddr 構造体を sockaddr_in 構造体とみなして要素を取り出すといった操作がよくおこなわれる．そのため，構造体の書き換えによる整合性を保つために，多くの OS カーネル部を書き換えなければならない．そこで，(2) の手法を採用し，accept とは独立した iaccept を作成し，第 4 の引数に識別子を渡すこととする．accept とは別に iaccept を作成したことで，サーバ側も iaccept を用いるための書き換えが必要となる．しかし，サーバの開発者は accept を用いるか iaccept を用いるか選択することができる．そのため，パフォーマンスが低下する部分を最小限に抑えることができるというメリットも生まれる．以上の検討から，iaccept の引数を**図 3** のように定める．

次に，iaccept の実装を示す．システムコール accept は uipc_syscalls.c に定義されている．これに，accept と同様の処理に加え識別子の付与，タイムスタンプの取得，ログ出力機能を付けたシステムコール iaccept を新規作成する．識別子は符合なし 32bit 整数で定義される iaccept の呼び出し回数を数えるカウンタである．またタイムスタンプはシステムコールの microtime によって得られるエポックタイムとする．iaccept ログ出力には，システムコールの log を用いる．log は指定された文字列を/dev/log に出力する．後に syslog デーモンにより，/var/log/messages に書き込まれる．これらの実装により，iaccept は次のように処理をおこなう．

(1) iaccept は accept と同様の処理をおこない，TCP セッションの確立要求に対して各種情報のコピーとソケットを読み書きするファイルディスクリプタを作成する．
(2) (1) の処理の結果をエラーナンバを用いて確認し，問題がなければ (3) へ進む．問題がある場合，通常の accept 同様，エラーナンバを呼び出し元へ返す．
(3) 識別子を付与し，ユーザランドのメモリへ値をコピーする．このことで，iaccept の引数に識別子を設定する．
(4) タイムスタンプを取得し，iaccept を処理したことをログ出力する．ログには，タイムスタンプと iaccept を処理したことを示すメッセージ及び識別子を出力する．
(5) 識別子のカウンタをすすめる．

最後にサーバ部の実装を示す．サーバは，従来 accept を呼び出していた部分を iaccept に書き換え，accept で受け取っていた情報に加え識別子を受け取れるようにする．また，受け取った要求を処理し，ログを出力するとき，処理結果と共に受け取った識別子を出力するようにする．サーバの iaccept 対応のための書き換えは図 3 に示したように，accept と比べ，識別子を受け取るための引数が増えただけである．また受け取った引数を処理結果と共にログに出力するだけである．このことから，容易な書き換えにより処理が追跡できることが期待される．実際に書き換えるのに要した手間については，4.1 節に示す．

### 3.3 追跡手法

サーバは各々要求を処理した結果をログ出力するが，3.2 節の実装により，iaccept を呼び出すことで処理の識別子を得ている．そのためログには，処理結果な


ⓒ 2011 Information Processing Society of Japan

```
#include <sys/socket.h>
int  accept(int socket, struct sockaddr *addr, socklen_t *addrlen)
int iaccept(int socket, struct sockaddr *addr, socklen_t *addrlen, unsigned int *id)
```

**図 3** iaccept の関数定義
Fig. 3 A definition of systemcall iaccept

どの情報と共に識別子を出力している．また OS カーネルは iaccept を受けたことをログ出力している．これら複数のログファイルを横断して処理を追跡するには，追跡したい処理の識別子を鍵に検索すれば良い．このことで，特殊なツールや手法を用いることなく処理を高速かつ容易に追跡することができる．追跡の例を 4.1 節に示す．

## 4. 評　　価

3 章に述べたシステムを実装し，iaccept の実用性と処理性能を評価する．評価をおこなう点は次のとおりである．

(1) iaccept の実用性の評価をおこなう．実用性の評価基準は，iaccept を用いることで，accept と同様の情報と共に識別子を得ることができることを示すことである．さらには，受け取った識別子をサーバ間で共有し，処理をサーバを横断して追跡可能であることを示すことである．また既存の障害検出手法に信頼性を低下させることなく組み合わせが可能であることを示すことである．
(2) iaccept の処理性能の評価をおこなう．処理性能の評価基準ははオーバヘッドが実用的な範囲に収まることを示すことである．

これらの項目を評価するためにおこなう実験は，次のとおりである．

(1) サーバを iaccept を用いるように対応をする．また得られた識別子をサーバ間で共有することで，サーバを横断した処理追跡をおこなう (4.1 節)．
(2) iaccept を利用することによるオーバヘッドがどの程度になるかの評価をおこなう (4.2 節)．
(3) iaccept ログのタイムスタンプとサーバログのタイムスタンプを組み合わせによる，既存の障害検出手法への組み合わせをおこなう．この時，既存の処理時間取得手法と比較し，信頼性を低下させることなく組み合わせが可能であることを示す (4.3 節)．

これら実験により，iaccept を用いることによる，メリット及びデメリットを整理することができ，本提案手法の有用性を評価することができる．

各評価に用いる環境を**表 1** に示す．

**表 1** 実験の共通環境
Table 1 Common environments of experiment

| | |
|---|---|
| CPU | Intel Core2Duo E6750 2.66GHz |
| Memory | 2Gb |
| Kernel | NetBSD 5.1 GENERIC+iaccept 対応 |
| NetWork | 学内 LAN 100Mbps |

**表 2** 識別子による追跡実験のための環境追記
Table 2 Additional environments of request trace experiment

NetBSD サーバの環境

| | |
|---|---|
| Pound | 2.4.5 |

Apache サーバの環境 (4 台共通)

| | |
|---|---|
| CPU | Intel Xeon 3.00GHz |
| Memory | 1Gb |
| Kernel | Ubuntu10.04 kernel 2.6.32-32-generic |
| Apache | 2.2.14(Ubuntu) |
| ApacheBench | 2.3 |

### 4.1 識別子の付与によるサーバを横断する追跡実験

iaccept による識別子の付与により，サーバを横断した処理をログを用いて追跡可能か実験をおこなう．実験に用いたサービスの構成は，Pound が稼働する NetBSD マシン 1 台に Apache が稼働する Ubuntu マシン 4 台が等距離で繋がっている構成とする．NetBSD マシンは iaccept の処理がおこなえるるよう OS カーネルを修正したものを用意し，その上で iaccept を用いるように修正した Pound を実行する．Pound を実行するマシン環境は表 1 の通りだが，本実験における追加の環境を**表 2** に示す．

Pound はロードバランサで，Apache は Web サーバである．外部からの要求はまず Pound へ到達し，いずれかの Apache サーバへ転送される．転送先は Pound の負荷分散アルゴリズムによって決まるため，事前に知ることはできない．Apache は Pound によって転送された要求を処理し，要求元に返す．

Pound は accept を実行したら子プロセスを生成して，親プロセスは次の accept を待つ．子プロセスはリクエストを転送先へ渡す処理をする．この動作はサーバにおいて一般的な動作で，Pound の修正結果は他のサーバにも同様に有効であるといえる．親プロセスの処理は pound.c で，子プロセスの処理は http.c に書かれている．両者合わせた行数は，読みやすくするための改行やコメントなどを含め 2000 行程度であ


ⓒ 2011 Information Processing Society of Japan

る．これらに対して iaccept に対応させる修正をおこなう．親プロセス部では，iaccept への対応と，受け取った識別子を子プロセスへ渡す処理を記述する．子プロセス部では，識別子を親プロセスから受け取る処理と，受け取ったものを HTTP ヘッダへ載せる処理を記述する．またログ出力に処理結果と共に識別子を出力するようにする．これら修正に要した行数は 10 行であった．

Apache は LogFormat を自由に記述できる機能を標準で有している．本実験では，Pound から Apache への識別子転送を HTTP ヘッダによっておこなう．Apache の LogFormat は特定の HTTP ヘッダを出力する機能を有しているので，コンフィグファイルである apache.conf に%{HTTP ヘッダ名 }i と記述することで，識別子を処理結果とともにログ出力する．そのため，Apache のコード書き換えは 0 行である．

本節の実験において，これら修正した OS カーネル及びサーバによって得られるログは次の 3 つである．

(1) iaccept ログによる要求受け渡し通知ログ
(2) Pound による要求転送ログ
(3) Apache による要求処理ログ

Pound への処理要求の発行は，ApacheBench を用いて総数 10 万回，並列 100 アクセスとした．ApacheBench は Web サーバのスループットなどを計測するのに用いられる標準的なベンチマークである．ApacheBench による要求発行が終了したら，iaccept ログから任意の識別子を 10 個抽出して，それら各々を鍵に Pound 及び各々Apache サーバのログの検索をおこなった．結果 10 回の試行に対してすべての回で処理を抽出でき，サーバを横断した処理流れを明らかにできた．追跡結果の例を図 4，図 5，図 6 に示す．なお，セキュリティの観点から IP アドレスを一部 x に置換している．各々ログの最後に付与された数字が識別子で，各々図は識別子 155 を鍵に追跡できたことを示している．この実験から得られた考察は，5.1 節に述べる．

### 4.2 iaccept を用いることによるオーバーヘッドの測定実験

iaccept は accept に比べ，ログの出力機能及び識別子，タイムスタンプの取得機能が増えている．そのため，accept に比べて処理時間が増える．これは特にサーバで利用する場合，スループットの低下につながり好ましくない．そこで，どの程度のオーバーヘッドが発生するのか測定する．処理時間の測定は，各々システムコールが呼び出された直後からサーバが呼び出し結果を受け取った直後までとする．各々のシステムコールを用いるメッセージサーバを作成し，システムコールの処理に要する時間を測定する．メッセージサーバは各々システムコールの結果を受け取った後に，乱数の平方根の計算を 1 億回おこなう．この計算には 2 秒ほどかかる．計算が終わったら，クライアントに「Hello」とメッセージを送信して，コネクションを切断するものである．メッセージサーバを実行した環境を表 1 に示す．

**表 3** iaccept によるオーバーヘッドの測定結果
Table 3 A result of iaccept overhead

| | |
|---|---|
| Average of accept processing time[s] | 0.0164 |
| Average of iaccept processing time[s] | 0.0310 |

メッセージサーバへのアクセスは ApacheBench を用い，100 回の逐次アクセスとする．結果は**表 3** の通りである． 表 3 より，iaccept は accept と比較して約二倍の処理時間を要することがわかった．オーバーヘッドが増えることについて，考察を 5.2 節で述べる．

### 4.3 iaccept ログのタイムスタンプの併用による処理時間計測実験

処理時間の計測は，障害の予測や検出をおこなうのに有効な手法の一つである[3)8)]．そこで本節では，iaccept ログのタイムスタンプを起点とし，これにサーバログのタイムスタンプを組み合わせることで，サーバが要求の処理に有した時間を計測する実験をおこなう．この実験から，既存手法との組み合わせにより障害の予知や検出がおこなえるか，またその精度を示す．

一般にサーバは処理時間をログ出力する機能を有している．しかし iaccept を用いることで，TCP セッションの確立タイミングを誤差なく捉えることができる．このことで処理時間計測のための始点が他のプロセスに左右されないため，処理時間が一定の値に定まることが期待される．文献 8) では異常予知のために機械学習を用いている．機械学習において，学習データの分散値は小さい方が良いデータと言える．これは分散値が小さいことは，ある点に向かってデータがより集合しているといえるためである．そこで，iaccept ログのタイムスタンプとサーバログのタイムスタンプを併用した処理時間計測法 (a) とサーバが標準的に有している処理時間計測法 (b) との精度の比較をおこなう．比較の基準を次に示す．

(1) iaccept ログを併用した処理時間 (Ave a) と，サーバ単体による処理時間 (Ave b) の差の比較
(2) iaccept ログを併用した処理時間 (Var a) と，サーバ単体による処理時間 (Var b) の分散値の比較

実験のために iaccept を用いたメッセージサーバを作成した．このサーバは次のように動作する．

(1) サーバは iaccept を呼び出す．この時 iaccept ログにはタイムスタンプが出力される．これは処理時間 (a) の始点となる．
(2) サーバは iaccept の呼び出し結果を受け取った後に，時間取得の関数を呼び出す．これは処理時間 (b) の始点となる．
(3) 乱数の平方根の計算を 1 億回おこなう．後にクライアントに「Hello」とテキストを送信し，コ



ⓒ 2011 Information Processing Society of Japan

```
/netbsd: 1310585465.439684   iaccept 155
```

**図 4** iaccept ログの例
Fig. 4 An example of iaccept log

```
172.21.67.143, xxx.xxx.xxx.xxx 172.21.67.147 - -
[14/Jul/2011:04:31:05 +0900] "GET / HTTP/1.0" 200 148 ""
"Opera/9.80 (Windows NT 6.1; U; ja) Presto/2.9.168 Version/11.50" 155
```

**図 5** Pound ログの例
Fig. 5 An example of Pound log

```
172.21.67.143, xxx.xxx.xxx.xxx 172.21.67.147 - -
[14/Jul/2011:04:31:05 +0900] "GET / HTTP/1.0" 200 487 "-"
"Opera/9.80 (Windows NT 6.1; U; ja) Presto/2.9.168 Version/11.50" 155
```

**図 6** Apache ログの様子
Fig. 6 An example of Apache log

ネクションを切断する．

(4) サーバは時間取得の関数を呼び出す．これは (a), (b) 両方における処理時間の終点となる．

この時 (2) と (4) における時間取得はサーバによっておこなわれる．プロセスが動作するタイミングはスケージューラによって決められるため，特定のタイミングに定まらない．サーバ単体による処理時間計測法 (b) では，(2) と (4) における時間取得タイミングが共に，スケジューラによって左右されるため，処理時間の分散値が本提案手法よりも大きくなると予想される．

また，サーバ環境は時間と共に変化し，様々なパフォーマンスが悪化することが指摘されている[6]．パフォーマンスの悪化に伴い，サーバに割り振られる CPU 時間は減っていく．サーバ環境の悪化は異常な状態なので，サーバの処理時間の変異が現れるはずである．この時処理時間の分散値が大きくなることで，サーバの稼動状態が安定してないことを知らせることができる．そこで，サーバ環境へ負荷をかけることでパフォーマンスが悪化している状態を再現し，本論文提案手法 (a) とサーバが有している処理時間計測手法 (b) で，どちらのほうが分散値が大きくなるかの比較をおこなう．

サーバに負荷をかけるのに，既存ツールの stress[1] を用いる．負荷として，CPU 時間を長く利用するものを 8 プロセス，HDD 読み書きを繰り返すものを 4 プロセス実行した．CPU 時間を長く利用するプロセスは乱数の平方根を計算する処理を無限回繰り返す．HDD 読み書きを多数発行するプロセスは，alerm システムコールを実行し，数秒後に割り込み処理として．usleep を実行する．その後，1MB 分の乱数を書き込む動作を無限回繰り返す．これら 12 個のプロセスを実行した後に，top によりロードアベレージが 12 付近，メモリ使用率が 100%近くになることを確認し，実験をおこなった．

またサーバをリアルタイムプライオリティ化，つまり CPU 処理優先度を最も高い状態に設定することによる比較もおこなう．このことで，サーバプロセスは最も優先度が高くスケジューリングされるため，上に示した負荷など他のプロセスの影響を削減することができ，(a) と (b) 両者の処理時間の差が小さくなることが期待される．プロセスをリアルタイムプライオリティに設定するには，システムコール sched_get_priority_max を呼びだし，得られた結果を sched_setscheduler に設定することでできる．スケジューリングポリシーは SCHED_RR を指定した．この時得られたリアルタイムプライオリティ値は 191 であった．リアルタイムプライオリティとしてプロセスを実行するためにはルート権限が必要なため，すべての実験はルート権限を与えおこなう．

以上の実験条件をまとめると，次のようになる．

(1) メッセージサーバのみを実行した場合 (Mes)
(2) メッセージサーバと負荷をかけるプロセスを同時に実行した場合 (Mes+st)
(3) メッセージサーバにリアルタイムプライオリティを与えて実行した場合 (RTMes)
(4) メッセージサーバにリアルタイムプライオリティを与えて実行したものと，負荷をかけるプロセスを同時に実行した場合 (RTMes+st)

表 1 の環境でメッセージサーバを実行した，ApacheBench を用いて，メッセージサーバに対して 10 回の逐次のアクセスをおこない，各々の処理の平均時間 (Ave) と分散値 (Var) をとった．結果を**表 4** に示す． 表 4 から，各々負荷をかけないときに比べて，負荷をかけたときの方がサーバの処理時間が伸びていることがわかる．また，メッセージサーバをリアルタイムプロセスに設定することで，負荷をかけた場合とそうでない場合において処理時間にほとんど差がないことがわかる．これら結果から，考察を 5.3 節に


ⓒ 2011 Information Processing Society of Japan

**表 4** 処理時間の比較結果
Table 4 A result of comparison of processing time

| | Mes | Mes+st | RTMes | RTMes+st |
|---|---|---|---|---|
| Ave a[s] | 2.11 | 8.69 | 2.11 | 2.12 |
| Ave b[s] | 2.11 | 8.65 | 2.11 | 2.12 |
| Var a | 0.01612 | 15.32493 | 0.01477 | 0.01157 |
| Var b | 0.01613 | 15.25536 | 0.01477 | 0.01158 |

述べる．

## 5. 考察と議論

### 5.1 識別子の付与によるサーバを横断する追跡実験結果の考察

この実験により，識別子を付与することで，サーバを横断した処理追跡をおこなえることが示せた．このことから，従来サーバ単体の状態を知らせていただけのログが，要求の処理状態を追跡できるものへと改善された．また，識別子を鍵にログファイルの検索をおこなうことで，処理結果の抽出がおこなえることを示せた．このことから，汎用的なツールで処理を高速に追跡できることが示せた．既存のサーバを iaccept に対応させるのは，非常に容易であることが示せた．

iaccept ログの出力を用いて，不慮のサーバ障害によってサーバログが出力されない場合でも，サーバに処理を渡したことが確認できる．このことで，どのサーバまで処理要求が到達したのかを判断でき，障害箇所の精査に役に立つ．iaccept ログの出力にはシステムコールの log を用いた．この時，log の引数に LOG_KERN を指定することで，OS カーネルからの書き込みであることを記すことができる．log は指定された文字列を/dev/log に書き込む．後に syslog デーモンにより/var/log/messages へ書きこまれる．syslog は標準で，/dev/log の内容と/var/log/messages の内容をシステムコールの fsync によって強制的に書きだし，整合性を保つ．そのため，不意の電源遮断などによるログ消失の可能性が高い場合でも，多くの情報をファイルに書きだしてあるため，ログの消失を最小限に留めることができる．fsync はログの信頼性を高めることに貢献しているが，一方で，log を実行するたびに fsync を実行するため，システム全体のパフォーマンスが落ちることを確認した．今後，ログの信頼性とパフォーマンスの劣化に着目し，syslog に代替できるログシステムの検討が必要である．

### 5.2 iaccept を用いることによるオーバーヘッドの計測実験結果の考察

実験結果より，iaccept の処理時間は accept の二倍であることがわかった．これは iaccept が発行する多量のログ書きこみ命令により，システム全体のパフォーマンスが低下した影響によるものと考えられる．このことから，iaccept を用いると，TCP セッションの確立に accept 以上の時間がかかり，サービスのパフォーマンスが悪くなる．しかしこの点はサーバの並列化との組み合わせで解決できる．並列化はサーバの負荷分散に有効であるが，処理の流れの複雑化が問題だった．しかし，iaccept を用いることで，処理の流れを明らかにでき，障害箇所の特定と，要因の検出に役立てることができる．また iaccept はサーバの並列化に影響を与えない．これらのことより，iaccept とサーバの並列化により，お互いの良い点を活かす組み合わせができる．また，識別子を付与しなければならない箇所のみ iaccept を用い，その他の部位では accept を用いることで，サービスへのオーバーヘッドを最小限に留めることができることが示せた．今後分散環境における iaccept の実験をおこない，このオーバヘッドが実サービスにどの程度の影響を与えるかを評価したい．

### 5.3 iaccept ログのタイムスタンプの併用による処理時間計測結果の考察

処理時間の計測結果には，本提案手法とサーバが有している処理時間計測法に大きな差は現れなかった．特に本論文が期待した程の差は現れなかった．これは実験に用いたメッセージサーバが単純な仕組みになっているためと考えられ，今後実サーバの修正により追実験をおこなうことで明らかにしていきたい．

実験ではメッセージサーバにリアルタイムプライオリティを与えた．このことで stress に CPU 処理時間を奪われず，安定した処理時間になることを示せた．リアルタイムプライオリティを設定するためにはルート権限でサーバを実行する必要がある．しかし，一般的にサーバに対してルート権限は付与してはならない．これは脆弱性などによりサーバが乗っ取られた場合，攻撃者にサーバのルート権限をそのまま委譲してしまう可能性が高いためである．よって，サーバをリアルタイムプライオリティで実行することは推奨されない．

実環境においてサービスを構成するサーバで，優先度を高く設定してはならない．このことから，サーバ環境が悪化することで，サーバの処理時間も共に伸びることになる．負荷をかけるプロセスを同時に走らせている状態でメッセージサーバの処理時間を計測した場合，本提案である iaccept ログとサーバログの組み合わせによる処理時間計測法の方が，サーバ単体の処理時間計測法よりも分散値が大きかった．これは本提案手法のほうが処理時間の計測区間が広いため，より多くの CPU 時間を負荷をかけているプロセスに奪われた結果と考えられる．分散値が大きいことは，処理時間のばらつきが大きいことを示し，これはシステムの異常状態とみなさなければならない．そのため，本論文提案手法による処理時間測定法の方が，文献 8) における処理時間計測法に適しているといえる．

### 5.4 議　　論

#### 5.4.1 Apache と本提案手法の，処理時間とみなすタイミングの違いの検討

4.3 節では処理時間の比較をおこなっているが，



ⓒ 2011 Information Processing Society of Japan

Apache が出力する処理時間は 4.3 節の計測タイミングとは異なる．本提案でみなす処理時間は TCP セッションの確立時を起点とするが，Apache の出力する処理時間は，最初の HTTP 処理要求の受け取りを起点としている．Web サーバの分野で Apache はデファクトスタンダードとなっており，他の Web サーバも同様のタイミングを処理時間として出力している．しかし Apache のタイムスタンプ付与方法は，次のようなプログラムにより問題となる．

(1) クライアントはサーバに対して接続要求を発行し，サーバは accept を実行し，クライアントからの要求を待つ．
(2) クライアントは処理要求を送信せず，待機する．
(3) サーバはタイムアウトエラーにより，クライアントを切断する．

Apache は最大接続数を制限するため，このようなアクセスを大量に発行することで応答不能状態となる．しかしクライアントが処理要求を送信しないため，ログには攻撃を受けている状態が出力されず，タイムアウトが発生したタイミングで初めて検出できる．iaccept ログを用いると，大量の接続要求をサーバに渡したことを確認できる．このことから，iaccept 後のタイミングから処理時間を計測するのは妥当である．

また，各々のログを用いて，Apache の処理前，処理後を確認することができる．両者のログを組み合わせることで，Apache が現在どれだけの量の接続を受けているかを判断することができ，また処理時間の計測により障害状態を検出することができる．

#### 5.4.2 識別子の付与手法の検討

本論文では，識別子の付与のためにシステムコールの iaccept を作成した．しかし，サーバによる識別子付与方法も考えられる．例えば 2.2 節の環境の場合，ロードバランササーバが識別子を付与することで同様のことが可能である．しかし，サーバによる識別子付与の場合，特定の構成でのみしか用いることができず，汎用性がない．例えば Web サーバに識別子を渡すための専用ロードバランサを作ることはできるが，これをメールサーバに応用することはできない．また，識別子を生成するサーバに対する妨害や攻撃を受けることによって，攻撃者に都合のよい識別子を生成される可能性があり得る．OS カーネルによる実装により，汎用的に識別子を付与できる．またサーバ書き換えのような攻撃を受けない．さらには iaccept ログに OS カーネルによる書き込みであることが記されるため，サーバによる実装より信頼度の高いログを出力することができる．以上のことから，識別子の付与をシステムコールの iaccept でおこなうことは妥当である．

## 6. ま と め

本論文では，サービスに届いた要求に識別子をつけることで，処理要求が各々サーバでどのように処理されたか追跡できるシステムの提案をした．提案システムを構築し評価をおこない，従来は難しかったサーバごとのログを横断し，処理の結果を追跡することができることを示せた．また既存の障害検出手法と組み合わせることで，障害検出がおこなえることを示した．

今後の課題として，まず iaccept に対応したサーバを増やすことを目標とする．対応したサーバを増やすことで．HTTP 以外のプロトコルや，多段サーバ横断に関しても提案手法の有用性を確認したい．また，既存の障害に対する評価をおこなう．本論文ではログに識別子を付与することで追跡をおこなったが，ログを用いずとも追跡がおこなえるシステムの検討をおこなう．また，syslog に代わるログシステムの検討により，高速化及び高信頼化を目指す．さらには，実サービス環境における有用性，特に大規模分散環境での有用性に関しての評価をおこなう．

## 参 考 文 献

1) stress
http://weather.ou.edu/ãpw/projects/stress/.
2) IBM. Ibm research — autonomic computing
http://www.research.ibm.com/autonomic/.
3) 山西健司. データマイニングによる異常検知. 共立出版, 2009.
4) 上野康平, 笹田耕一. 軽量なプロセスマイグレーションを可能とするフレームワーク. 情報処理学会研究報告. [システムソフトウェアとオペレーティング・システム], Vol. 2010, No. 15, pp. 1–8, 2010-07-27.
5) 品川高廣, 榮樂英樹, 谷本幸一, 面和成, 他. 準パススルー型仮想マシンモニタ bitvisor の設計と実装 (os-4:仮想化,2008 年並列/分散/協調処理に関する『佐賀』サマー・ワークショップ (swopp 佐賀 2008)). 情報処理学会研究報告. [システムソフトウェアとオペレーティング・システム], Vol. 2008, No. 77, pp. 69–76, 2008-07-30.
6) 前田彩, 山田浩史, 岩田聡, 吉田哲也, 河野健二. リクエストのリソース使用量の変化に着目したソフトウェア老化検出手法. 日本ソフトウェア科学会ディペンダブルシステム研究会第 7 回ディペンダブルシステムワークショップ 2009, pp. 59–68, 2009.
7) 大江将史, 門林雄基. 階層型 ip トレースバック機構の実装と検証 (ネットワーク管理)(〈特集〉インターネットアーキテクチャ技術論文). 電子情報通信学会論文誌. B, 通信, Vol. 86, No. 8, pp. 1486–1493, 2003-08-01.
8) 中村友洋. Web アプリケーションの障害を予測する アクセス時間解析方式の提案 (インターネットシステム). 情報処理学会論文誌. コンピューティングシステム, Vol. 47, No. 12, pp. 349–357, 2006-09-15.



ⓒ 2011 Information Processing Society of Japan