社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2007－ARC－172 (38)
2007－HPC－109 (38)
2007／3／2

ハイパフォーマンス分散時刻認証局：毎秒百万タイムスタンプ発行の実現

西川武志[†] 松岡聡[†,††]

時刻認証はデジタルデータがある時点で存在していた、改ざんされていないということを証明する手段である。我々は分散時刻認証法を開発し既存の集中型時刻認証法の性能スケーラビリティに起因する分散 DoS 攻撃への脆弱性と高価な時刻源を用いることに由来する高コスト性という大きな二つの問題を解決した。本報告では毎秒百万タイムスタンプ発行が実現可能となるような実装と動作パラメータについての検討を行った。

High Performance Distributed Time-Stamping Authority: How to Issue Millions Time-Stamp

TAKESHI NISHIKAWA[†] and SATOSHI MATSUOKA[†,††]

Time stamping is a technique to prove the existence of a digital data prior to a specific point in time. The centralized time-stamping scheme which is the main stream at present can not stand up to the concentration of numerous time-stamping requirement. So, the centralized time-stamping scheme has vulnerability to the distributed DoS(DDoS) attack. It also has high cost problem which causes using an expensive time source such as atomic clock. We soleved these problem by developing a distributed time stamping scheme. In this report, we investigated an implementation and parameter configuration those make a million time-stamp per second possible.

1. はじめに

デジタル時刻認証はデジタルデータがある時点で存在していた、改ざんされていないということを証明する手段である。その重要性は日増しに増大しており、ビジネス分野では、特に日米では法律で規定された保存電子帳票の正当性証明目的や内部統制での種々の記録の非改竄証明にはデジタル時刻認証の役割が大きなものとなって来ている。

他方で、デジタル時刻認証は科学技術分野でも知的財産優先権の確保や不正が行われていないことの証明に役立つことが期待されるにもかかわらず、その経済性、高コストであることから普及していない。実験機器の電子化、デジタル化等により出力がデジタルデータとしてのみ保存され、またバイオインフォマティクスのように学問手法自体が、従来の紙媒体の実験ノートに保存しきれない大量のデータが生成される場面では、元のデジタルデータのハッシュを生成し、元のデータそのものを第三者に渡さずに存在や非改竄の証明ができるデジタル時刻認証は原理的に有用な手段と分かっていながら普及していない。

我々は時刻源や監査に伴う経済コストと多数のタイムスタンプ要求に応えるという問題を解決出来る新たな分散時刻認証方法を提案した。[1] 分散時刻認証局を構成する時刻認証ユニット（Time-Stamping Unit:TSU）が互いに一つの時刻認証要求に複数 TSU が複数世代に渡ってタイムスタンプを発行することで全体としてお互いに自動的に評価しあい、信用出来る TSU のリストを形成して行くことで分散時刻認証局として監査プロセスをその動作に内在させてることで監査コストの低減を実現した。また極めて短時間に多数のタイムスタンプを発行することで、ファイルシステムのタイムスタンプをデジタル署名に基づいたものとするような場合の短時間に多数のタイムスタンプの発行が必要となる要望や、対タンパ性の向上が図れる。

我々が $(N, K = L + M, G)$ と名付けた提案手法で発行されたタイムスタンプの検証は、分散時刻認証局から源となるタイムスタンプから連環して複数世代に渡って発行された多数のタイムスタンプに記録された時刻から多数決や平均値や最頻値などの統計処理値に基づいている。TSU の数、N が増えれば増えるほど

† 東京工業大学
Tokyo Institute of Technology
†† 国立情報学研究所
National Institute of Informatics

統計値の分散が低減するが、本方式には世代G、各世代毎のタイムスタンプ発行要求数K、ここでKは信頼できるTSUのリストの要素数Lと$(N-L)$個のTSUからランダムに選ぶTSUの数Mという設定パラメータが存在する。

本報告では、分散時刻認証局全体として1タイムスタンプ発行要求に対し1秒間に百万タイムスタンプ発行を実現させるには、どのような実装でどのようなパラメータの組み合わせにすれば良いかを、実際にクラスタ環境で実行し検討した結果を述べる。

$(N, K = L + M, G)$分散時刻認証法の実装ではタイムスタンプの発行と検証の処理時間に対する要求を非対称なもの、発行は短時間に検証は長時間でも構わないとして、短時間で多数のタイムスタンプ発行を可能とした。設定パラメータに関しては世代数Gが増大すると平均応答時間が増大する傾向にあること、$K = L + M$がNに対して小さすぎず、大きすぎずという条件が平均応答時間を小さくするということが判明した。

2. $(N, K = L + M, G)$ 分散時刻認証法

我々が提案する$(N, K = L + M, G)$分散時刻認証法では、ネットワーク上に多数の時刻認証要素（Time Stamping Unit:TSU）が相互に時刻認証し合うことで同時に多数の時刻認証要素が結託して時刻を詐称することが困難であること、同じく多数の時刻認証局がネットワーク上に分散していることで分散DoS攻撃に耐性を持つであることの仮定に基づいている。

$(N, K = L + M, G)$分散時刻認証法の実装、$(N, K = L + M, G)$分散時刻認証局は、以下の要求、仕様を満たすとする。

- 各TSUはRFC3161、"Internet X.509 Public Key Infrastructure Time-Stamp Protocol"に互換のSimple Time-Stamp Unitととして動作する。したがって各TSUが発行するタイムスタンプは一意のシリアル番号を持つ、
- 各TSUは$(N, K = L + M, G)$分散時刻認証局運用規則に則って独立に管理される、
- 各TSUは独立にインターネット上のNTPサーバと時刻を同期すること、
- 各TSUは互いに監査し合う、
- 各TSUは特定のセキュリティハードウェアを使用することなく、汎用の計算機上で動作可能であること、
- 各TSUは絶対時刻をある一定の精度でタイムスタンプとして応答可能であること、
- 簡便、効率的、頑強な統計的処理に基づく検証プロトコルを持つこと、
- Denial of Service (DoS) 攻撃に耐性を持つこと、
- 同時多数のリクエストに応えられるようにすること、
- ネットワーク切断・重大遅延等のネットワーク障害に耐性を持つこと、
- タイムスタンプの検証プロセスには十分に長い時間を要しても統計的に十分なサンプルを集める一方で発行プロセスは迅速に行うこと、
- ネットワークトラフィックを過度に消費しないこと、
- 時刻が過失により正確でないノードや悪意を持った第三者による時刻を改ざんしたノードが存在することへの耐性を持つこと
- $(N, K = L + M, G)$分散時刻認証局全体として単一障害点を持たないこと。

各TSUがRFC3161のインターネット標準時準じることで既存のRFC3161の時刻認証局ならびにその利用者を$(N, K = L + M, G)$分散時刻認証局へ参加出来ることを目指している。$(N, K = L + M, G)$分散時刻認証局はタイムスタンプを利用したい主体がタイムスタンプを要求、発行し合うとともに相互に監査を行うTSUを用意することで形成される相互結合ネットワークから構成される。各々が相互にタイムスタンプを発行、要求し合うプロセスとその発行されたタイムスタンプに納められた時刻に基づいて各TSUが他のTSUの信頼性を監査すること、相互監査をシステムに内在することで監査費用の低減をはかっている。すなわち$(N, K = L + M, G)$分散時刻認証局はタイムスタンプ利用者により自己形成されるシステムである。

2.1 タイムスタンプ発行プロセス

概略として以下の手順を繰り返す。

（1） `mk_hash` ハッシュ作成
（2） `mk_tsq` TSQ生成
（3） `tsq()` TSQをroot TSUに発行
（4） `generate TSR()` root TSUによるTSR生成、応答
（5） `dispatch g\_tsq()` root TSUによる次世代へのTSQ発行準備
（6） `g\_tsq()` root TSUによる次世代へのTSQ発行
（7） `generate TSR()` G世代TSUによるTSR生成、応答
（8） `dispatch g\_tsq()` 次世代へのTSQ発行準備
（9） `g\_tsq()` 次世代へのTSQ発行
（10） ～ 以降、(7)～(9)をG上限まで繰り返し

タイムスタンプの発行要求からログの記録までの詳細は次の通りである。(1) デジタルデータからハッシュを作成する。(2) 利用者はRFC3161に従って時刻認

証を行いたいファイルのハッシュからタイムスタンプクエリを作成する。(3) タイムスタンプ要求 (TSQ) を RFC3161 に従ったものと、本手法で拡張したプロトコルを用いて TSU に発行する。(4) TSU はタイムスタンプ応答 (TSR) を生成し、a) 非同期に TSR を要求元に返す。(5) 次世代の TSU、K 個を $(N, K = L+M, G)$ アルゴリズムに従って選択し TSQ の発行準備をするとともに、現在何世代目かを記録するフラグを 1 減ずる。(6) 次世代 TSU へ TSQ を $(N, K = L+M, G)$ アルゴリズムによる要求 (g_tsq) として発行する。(7) 次世代 TSU による TSR 生成、応答が行われ、a) 非同期に TSR を要求元に返す。受け取った前世代 TSU は g_tsq による TSR を手元に保存し、c) どの次世代 TSU からの TSR であったかを記録する。従ってタイムスタンプ時刻は TSQ が TSU に届いて処理される過程で認証されるため、TSQ 要求から TSR 処理、ログの記録までの全体の処理が終わった時刻よりも前の時刻が記録される。(8) タイムアウトになっていないか、世代数が上限に達していないかを確認し、いずれでも無い場合、再び次世代の TSU、K 個を $(N, K = L+M, G)$ アルゴリズムに従って選択し TSQ の発行準備をするとともに、現在何世代目かを記録するフラグを 1 減ずる。(9) 次世代 TSU へ TSQ を $(N, K = L+M, G)$ アルゴリズムによる要求 (g_tsq) として発行する。(10) 以降、(7)〜(9) を G 上限まで繰り返していく。

G が 3 以上に対して各々の G 世代の TSU は $(G-1)$ 世代から受け取ったタイムスタンプ要求を自己と $(G-1)$ 世代の直接上流の TSU を除いた $(N-2)$ の TSU の内から $K = (L+M)$ 個を選んでこれまでと同様に世代数を減じて転送要求する。残り世代数が 0 になれば末端 TSU として次世代に転送要求を出さない。その結果、各 G 世代で $1, K, K^2, ..., K^{G-1}$ 個のタイムスタンプ要求がなされ、全体では $(K^G-1)/(K-1)$ 個のタイムスタンプ要求がなされる。その結果、最大で $(K^G-1)/(K-1)$ 個のタイムスタンプが生成されるが、現実の実装では途中のネットワーク遮断等なんらかの理由により応答が無い TSU があることが考えられる。

ここで root TSU によって発行されたタイムスタンプを root TS と呼び、発行を要求した時刻を t_0、root TS に署名された時刻を t_1 とする。次世代に転送されたタイムスタンプを trns TS と名付ける。この時刻を t_{trns} とする。この $t_{trns} - t_0$ の統計値を t_{Stat} とすると $t_0 + t_{Stat}$ として時刻が認証される。t_{trns} の各種統計処理値としては平均値、分散、最頻値等が考えられる。

今回の報告では平均値を用いて t_0 から 1 秒間に百万タイムスタンプを発行する際の実装や設定パラメータの検討を行った。

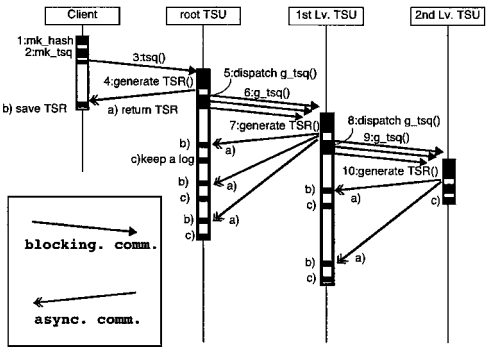

図 1 シングルスレッド版 TSQ タイムチャート
Fig. 1 TSQ time chart of Single-thread version.

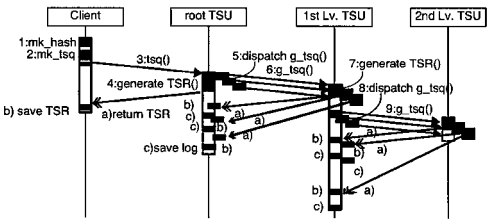

図 2 マルチスレッド版 TSQ タイムチャート
Fig. 2 TSQ time chart of Multi-thread version.

3. 実装評価

今回、$(N, K = L+M, G)$ 分散時刻認証手法、tsagrid プログラムを Java Servlet として実装した。JDK1.5 系、Tomcat5.5.x 系ならびに時刻認証用の API 群として BouncyCastle[2] を用いた。実装に当たって、タイムスタンプの発行は一意のシリアル番号付与を必要とするが、それ以外の場所は並列処理可能である。また複数の TSU からの TSR は非同期に返ってくるので TSR 処理をクライアント側でシングルスレッドで処理していてはとても大きなオーバヘッドとなり、短時間に大量のタイムスタンプ発行は実現出来ない。一方で TSQ 発行、複数 TSU の選択、タイムスタンプ生成、TSR 応答処理をマルチスレッド化した場合、複数 TSU の個数 K を大きく取ると TSU の処理時間や処理に必要な計算資源、特にメモリ要求量が増大し、結果として応答時間が悪化することが予想される。このことを検証するため、タイムスタンプのシリアル番号付与および TSR 応答処理以外の部分についてシングルスレッド版とマルチスレッド版を実装し、評価を行った。

図 1 にシングルスレッド版、図 2 にマルチスレッド版でのタイムスタンプ発行プロセスのタイムチャートを示す。

図 1、図 2 を比較して明らかなようにマルチスレッ

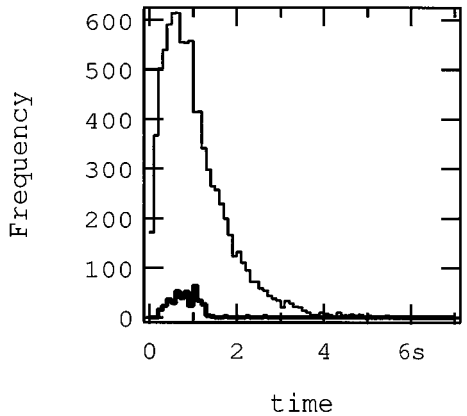

図 3 シングルスレッド版による $G = 16, L = 1, M = 1$ でのタイムスタンプ時刻と全処理時間の分布

Fig. 3 Distribution of time-stamping time and total processing time at $G = 16, L = 1, M = 1$ by single-tread version.

ド版が短時間に処理が完了することが期待される。

3.1 動作検証

$(N, K = L + M, G)$ 分散時刻認証手法を実装した tsagrid プログラムを OrionMultisystems 社製 Orion DS-96(CPU:Transmeta Efficeon 1.2GHz、各ノード 2GB RAM、合計 96 ノード) の均一クラスタにインストールし、シングルスレッド版、マルチスレッド版それぞれにおいて設定パラメータ $(N, K = L+M, G)$ を変化させ、動作検証を行った。本報告での動作検証では各 TSU からの次世代への全処理に対するタイムアウト時間は目標とする 1 秒以内の多数の応答に対して十分大きなタイムアウト時間と見なせる 5 秒に設定した。各設定パラメータでは 400 回の分散タイムスタンプ発行を行い、最初の 100 回を初期状態とし、後の 300 回を平衡状態とし、平衡状態での統計値により動作検証を行った。

3.2 シングルスレッド版とマルチスレッド版の比較

シングルスレッド版での応答待ち時間が大きい例として、図 3 に $G = 16, L = 1, M = 1$ におけるタイムスタンプ時刻（太線）と全処理時間（細線）を示す。タイムスタンプ時刻の平均値は 0.578 秒であったが、一連の処理をシングルスレッドで処理するため待ち時間が多くログに記録された全処理が完了するまでの時刻はタイムスタンプ時刻よりもロングテールとなっている。

図 4 に $G = 3, L = 1, M = 1$ という $(N, K = L+M, G)$ 分散時刻認証法での最小設定パラメータでのシングルスレッド版とマルチスレッド版の全処理時間の分布を示す。横軸が TSQ 発行 TSU ノード番号、縦軸が TSR 生成 TSU ノード番号であり、0 秒に近ければ黒く 1 秒に近くなれば白くなるようにプロットし

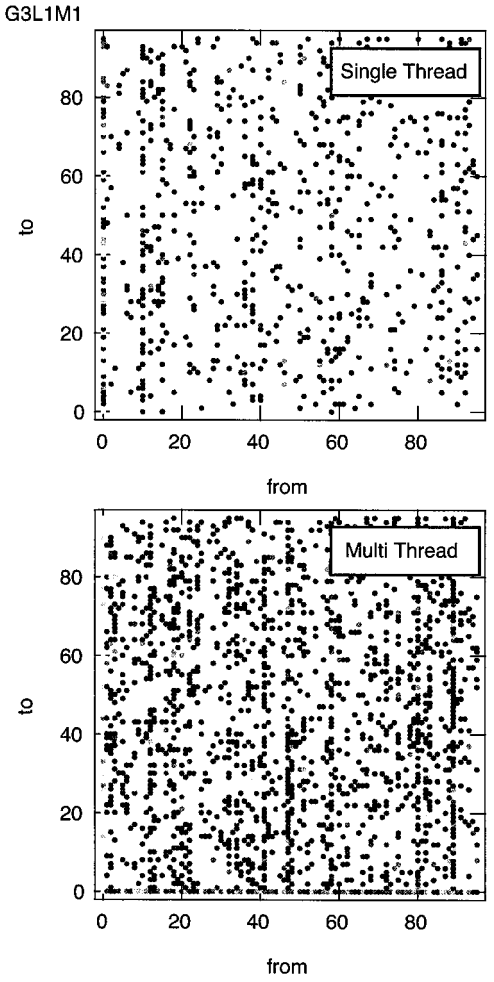

図 4 シングルスレッド版とマルチスレッド版の比較：全処理時間の分布 $G = 3, L = 1, M = 1$

Fig. 4 Comparison with single-thread and multi-thread version by distribution of time-stamping time and total processing time at $G = 3, L = 1, M = 1$

ている。1 秒以内に全処理を終えているノードの割合はシングルスレッド版で 8.88 %、マルチスレッド版で 25.82 %となっている。このようにマルチスレッド版はシングルスレッド版より 3 倍もの効率でタイムスタンプ発行要求をこなしていることがわかる。

しかしながらマルチスレッド実行、すなわち設定パラメータ K を増やすとマルチスレッド実行による計算資源消費による副作用としての処理時間増大の影響がある。図 5 に $G = 3, L = 1, M = 1$ の処理でのマルチスレッド版での TSR 生成時間の一例を示す。横軸はタイムスタンプ発行のシリアル番号増分である。180 から 200 の間隔で TSR 生成時間が 2 桁増大して

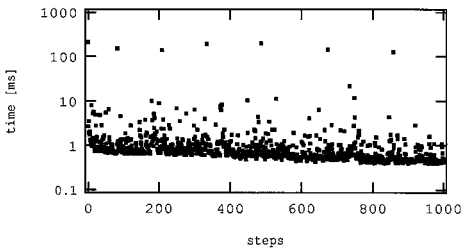

図 5 マルチスレッド版における TSR 生成時間
Fig. 5 Time of generating TSR

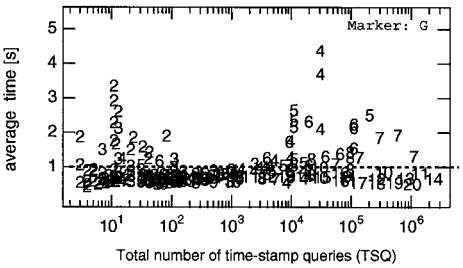

図 6 タイムスタンプ時刻の平均値の各パラーメタへの依存性
Fig. 6 Average time by time-stamping dependency of parameters.

表 1 全処理時間の G への依存性、$L=1, M=1$ の時の処理時間 1 秒以内の割合
Table 1 Percentage with in one second in total processing time dependency of G, at $L=1, M=1$

G	Percentage
3	25.82
4	58.64
5	78.52
6	72.31
7	40.67
8	23.65
9	17.36
10	13.50
11	12.80
12	13.19
13	18.23
14	13.11
15	16.19
16	12.50

いることが読み取れる。この原因はメモリ領域の確保と開放に伴う未使用メモリ領域の Java VM によるガーベージコレクションの影響であると推測される。

3.3 $K=L+M, G$ の各パラーメタへの依存性

先に述べたようにむやみに K を増やすことは処理時間増大を招くことが予想される。では代わりに世代数 G を増やせば良いかというと単純ではない。今回はクラスタ内のネットワークを使用しているがインターネット上に TSU を配置した場合には TSU 間のネットワークの遅延の影響が大きくなる。単純に各 TSU 間のネットワーク遅延時間が等しいと仮定すれば世代数 G が 2 倍になればネットワーク遅延時間も 2 倍となると予想される。

表 1 に $L=1, M=1$ すなわち $K=2$ とし、G を 3 から 16 と変化させたときの全処理時間が 1 秒以内の割合を示す。

G を 3 から増やして行くと 5 で最大となり後は 10 %前後に減少していく傾向が見られた。これらから G を増やすと世代間のネットワーク遅延時間の影響が応答時間に強く影響することが分かる。

様々な (G, L, M) の組み合わせについてタイムスタンプ時刻の平均応答時間の平均値の各パラーメタへの依存性を測定した結果を図 6 に示す。なお、$G=2$、すなわち 1 世代だけ複数の TSU に TSQ を発行する場合、すなわち Bonnecaze 等の提唱する *kamongn* 法による分散時刻認証局[3),4)] では平均時間が 1 秒を超える事が多く見られる。これは全くランダムに TSU を選択するため応答時間が遅い TSU を選ぶ可能性が $(N, K=(L+M), G)$ 分散時刻認証法より大きいためである。

図 6 からは、総 TSQ が少ない領域では全 TSU から応答時間が遅い TSU を選択する可能性が高いため平均応答時間が大きくなる傾向があり、100 から 5000 前後で平均応答が 1 秒以内に収まる領域があり、総 TSQ が 1 万が増えると小さな G では必然的に $K=L+M$ を大きくせざるを得ず、スレッド数が増え、応答時間が遅くなる事等が読み取れる。

結果として図 6 に示すように、今回の検証環境ではマルチスレッド版を利用し毎秒百万タイムスタンプ発行を達成した設定パラメータの組は $(G, L, M, t_{avg}) = (20, 1, 1, 0.467)$, $(11, 2, 2, 0.798)$, $(14, 2, 1, 0.625)$, $(15, 2, 1, 0.618)$, $(15, 2, 2, 0.769)$ であった。

4. ま と め

本報告では、自己に時刻認証局機能を含みネットワーク上の多数の同じプログラムと時刻認証をし合うことで、既存の時刻認証基盤の経済コスト、性能スケーラビリティ、耐障害性の諸問題を解決する $(N, K=(L+M), G)$ 分散時刻認証法により、毎秒百万タイムスタンプ発行が実現可能となるような実装と動作パラメータについての検討を行った。タイムスタンプ発行のシリアル番号付与以外をマルチスレッド化するとともに応答処理の非同期化を実装することで、適切な設定パラメータを選択すれば毎秒百万タイムスタンプ発行が期待出来ることが明らかとなった。 今後の課題としては、実際のインターネット上で運用した場合の動作検証を予定している。

謝辞 謝辞本報告は国立情報学研究所からの委託業

務、NAREGI β 1 運用のための GridVM 用の追加開発、並びに VO 課金・分散時刻認証局開発およびデジタル知財存在証明システムでの試験運用の業務委託を受けて行ったものである。

参 考 文 献

1) 信学技報, CPSY2006-17, pp.25-30 (2006-8)
2) http://www.bouncycastle.org/
3) A. Bonnecaze, P. Liardet, A. Gabillon, K. Blibech, A Distributed Time Stamping Scheme, Proc. of the IEEE conference on Signal and Image Technology and Internet Based Systems. November 2005, Yaoundé Cameroon.
4) A. Bonnecaze, P. Liardet, A. Gabillon, and K. Blibech,Secure Time Stamping Schemes: A Distributive Point of View, Annals of Telecommunications, Vol. 61, No. 5–6, pp. 662–681 (2006).