セッションマイグレーションを用いた負荷分散SSLクラスタの提案と評価

初谷良輔[†] 齋藤孝道[††]

Web システムにおいて安全な通信路を確保するため，セキュリティプロトコルの 1 つである SSL（Secure Socket Layer）が広く利用されている．しかしながら，SSL を用いた通信のコストは，TCP のそれと比べて高いものとなり，サーバへの SSL 通信の集中は，サーバ負荷の急激な上昇を招くことがある．これに対する解決策の 1 つとして，SSL Web システムの負荷分散クラスタがある．しかしながら，既存の負荷分散クラスタリング方式では，負荷の偏向や過多な SSL セッションの生成により，クラスタの処理性能を引き出すことが困難な場合がある．そこで，本論文では，既存のクラスタリング方式における問題点を明らかにし，それらに対する解決策として，クラスタ内に展開されたサーバ間で SSL セッションを動的にマイグレーションすることが可能なクラスタリング方式の提案と実装，および，その性能評価について示す．

Load-balancing SSL Cluster by Using Session Migration

Ryosuke Hatsugai[†] and Takamichi Saito[††]

Secure Socket Layer (SSL) is widely utilized to establish secure Web communications. However, an SSL communication needs more computational power than a TCP one. Therefore, convergence of numerous SSL access to server makes the server's load rise to critical level. As a solution for it, load-balancing cluster for SSL Web system is popular to apply. However, there is possibility that the existing load-balancing schemes degrade total system's performance, because of out of balance of loads. In this paper, we reconsider the existing load-balancing clusters for SSL Web system, we propose a new load-balancing cluster that can dynamically migrate SSL session from one to other servers. We also show its implementation and evaluate it from the viewpoint of its performance.

1. はじめに

インターネット通信の安全性を確保するため，セキュリティプロトコルの 1 つである SSL（Secure Socket Layer）[1] やその後継にあたる TLS（Transport Layer Security）[2] が広く利用されている（以降，これらをあわせて SSL と呼ぶ）．SSL を用いることで，通信主体の認証による真正性，通信の暗号化による機密性，および，MAC（Message Authentication Code）による通信データの完全性を確保することができる．しかしながら，SSL は通信の安全性を確保する一方で，処理コストの高い暗号技術を多用するため，SSL を利用する Web（以降，SSL-Web と表記する）サーバにアクセスが集中することで，負荷が急激に上昇し，結果としてサーバの処理性能を低下させることがある[3)〜5)]．

これに対する解決策として，負荷分散クラスタがある．SSL-Web システムにおける負荷分散クラスタの構成例には，負荷分散装置（Load Balancer）[6] や SSL アクセラレータのような SSL リバースプロキシ[6)〜8)] を利用したものがある．たとえば，前者の方式では，クライアントからの HTTPS 通信を負荷分散装置がその後方に配置された SSL-Web サーバ群に対して割り振ることで，サーバ間での HTTPS 通信の分散化を図る．一方，後者の方式では，SSL リバースプロキシがクライアントからの HTTPS 通信を処理し，そのうえで，後方に配置された Web サーバ群に対して HTTP 通信の分散化を行うことで，システム全体の処理性能を向上させる．

しかしながら，これらのような既存方式では，クラスタを構成する各サーバに対して SSL 通信が均等に分散せず，そのときには，一部のサーバに対して大量

† NRI セキュアテクノロジーズ株式会社
NRI SecureTechnologies, Ltd.
†† 明治大学
Meiji University

の SSL の処理が集中する可能性がある．また，一連の SSL 通信において，クライアントとクラスタ間で複数の SSL セッションを新規に確立することがあり，このような過多な SSL セッションの生成は，クライアントだけでなく，クラスタが保有する計算機資源も過剰に消費することになる．これらの問題は，負荷分散クラスタが提供すべき要件の 1 つでもあるシステム全体の処理性能の向上を妨げる要因といえる．

そこで本論文では，SSL-Web システムに対する既存の負荷分散クラスタリング方式について考察し，その問題点を明らかにする．また，SSL-Web システムの処理性能の向上を目的とし，SSL セッションをサーバ間で動的にマイグレーション☆することが可能な負荷分散クラスタリング方式の提案と実装について示す．提案方式では，これを利用してサーバに集中する SSL 通信の一部を他のサーバに対して再配布することで，サーバ間の負荷バランスの均等化を実現する．さらに，SSL セッションのマイグレーションにより，サーバ間で SSL セッションを共有することが可能となり，その結果，SSL セッションの効率的な再利用によって SSL 通信のコストを最小化する．最後に，処理性能の観点から，既存方式と提案方式の評価を行う．

2. SSL 概 要

ここでは，SSL の概要について示す．SSL は，Handshake プロトコル，Record プロトコル，Alert プロトコル，ChangeCipherSpec プロトコル，Application Data プロトコルの 5 つのプロトコルから構成されている．以下で，それぞれの役割を簡潔に示す☆☆：

Handshake プロトコル：

Handshake プロトコルは，暗号スイートの決定，通信主体の認証，セッション鍵の交換という手続きを経て，クライアントとサーバ間で SSL セッションを確立するためのプロトコルである（**図 1** の実線で示すメッセージ交換に相当）．セッション鍵の交換には，RSA，Diffie-Hellman，Fortezza などのアルゴリズムを，通信主体の認証には，サーバ認証モード，クライアント認証モード，匿名認証モードの 3 つのモードをサポートしており，そのアルゴリズムとして RSA や DSS（DSA）などが利用される．

SSL セッションを確立する方法は 2 通りあり，新規に SSL セッションを確立する方法（以降，Full Handshake と呼ぶ）と過去に確立した SSL セッションを再利用して SSL セッションを確立する方法（図 1 の '*' が添加されたメッセージ交換により実現する方法で，以降，Session Resumption と呼ぶ）がある．両者の処理コストであるが，Full Handshake は複数の暗号処理を経て SSL セッションを確立するため，その処理コストは高いが[9]，一方の Session Resumption は過去に生成した SSL セッションを再利用するため，その処理コストは低い[10]．

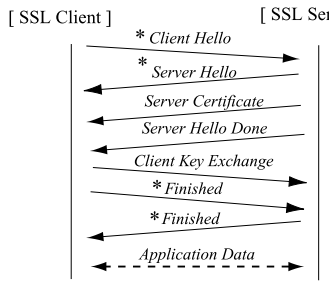

図 1 SSL 通信（RSA 鍵交換，サーバ認証モード）
Fig. 1 An SSL communication (RSA key-exchange in server authentication).

Record と Application Data プロトコル：

Record プロトコルは，Handshake プロトコルによって確立した SSL セッションをもとに，通信データの暗号化，復号，圧縮や MAC の生成および検証を行うプロトコルであり，暗号化には DES，AES，RC4 など，そして，MAC の生成には MD5 や SHA-1 などが利用される．SSL では，Application Data プロトコルがアプリケーションデータを Record プロトコルに渡すことで暗号通信を透過的に実現する．以降，Application Data プロトコルと Record プロトコルをあわせて，バルク暗号通信と呼ぶ（図 1 の破線で示すメッセージ交換に相当）．

3. 既存の負荷分散クラスタ

ここでは既存の負荷分散クラスタリング方式の分類とそれぞれの問題点を示す．

3.1 負荷分散装置

負荷分散装置を利用して SSL-Web サーバに対する HTTPS 通信の分散化を行うことで，システム全体の処理性能を向上させる方式である．このシステム構成における負荷分散の方法は，2 つに大別することができ，以降それぞれ，LB Stateful 方式と LB Stateless 方式と呼ぶ．LB Stateful 方式は，エンドポイント間，すなわち，クライアントと SSL-Web サーバ間の通信

☆ マイグレーションとは，サーバが保持する情報（ここでは SSL セッション）を他のサーバに配布（移動）するという意味である．

☆☆ ChangeCipherSpec プロトコルと Alert プロトコルは，議論する上で本質ではないため省略する．

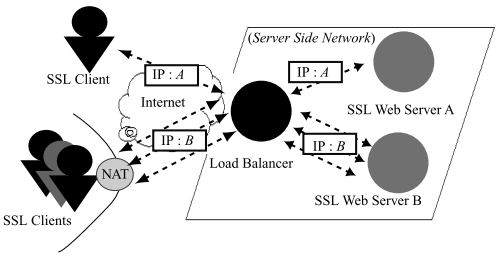

図 2 負荷分散装置を用いたシステム構成例（LB Stateful）
Fig. 2 An example cluster by deploying a load balancer (LB Stateful).

の永続性を確保する場合に利用され，これに属する具体的な負荷分散方式には，Client IP Address 方式や SSL Session ID 方式☆などがある[11]．また，通信データ中の URL や Cookie を参照する方式も存在するが，これらは SSL 通信によって暗号化されているため，ここでは利用できないことに注意されたい．一方，LB Stateless 方式は通信の永続性を確保する必要がない場合に用いられ，ここには，Round Robin 方式，Least Connections 方式，Shortest Response 方式，Weighted Round Robin 方式などが含まれる[11]．

3.1.1 LB Stateful 方式について

LB Stateful 方式を採用したシステムの構成例を**図 2** に示す．ここで，図中の破線は HTTPS 通信を示している．

このようなシステム構成において，HTTPS 通信の分散化に Client IP Address 方式☆☆を利用した場合，特定の SSL-Web サーバに大量の SSL 通信が集中することがある．たとえば，図 2 が示すように，NAT（Network Address Translation）ルータやプロキシを経由してクラスタに到達する大量の SSL 通信には，すべて同一の送信元 IP アドレスが割り振られており，この負荷分散方式に従うと，それらすべての SSL 通信は特定の SSL-Web サーバにのみ転送されることになる．このような状況下では，その処理に膨大な計算機資源を費やすサーバが存在する一方で，SSL 通信が割り振られていないサーバには利用可能な多くの計算機資源が残存することになり，これは SSL 通信の処理にクラスタ内の計算機資源を最大限に活用しているとはいえない．また，各サーバの処理能力に差がある

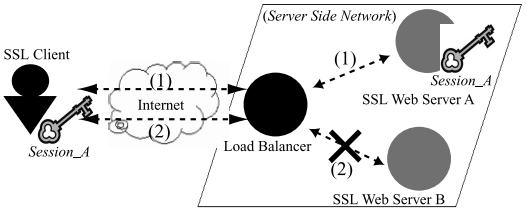

図 3 負荷分散装置を用いたシステム構成例（LB Stateless）
Fig. 3 An example cluster by deploying a load balancer (LB Stateless).

場合，この LB Stateful 方式に分類される負荷分散方式では，必ずしも処理能力の高い，または，優先度の高いサーバに通信を割り振るとは限らないといった問題もある．したがって，これに分類される負荷分散方式では，クラスタ内のサーバに対して SSL 通信の最適な負荷分散が困難であり，負荷の偏向が生じる可能性がある．

3.1.2 LB Stateless 方式について

LB Stateless を用いた際のシステム構成例を**図 3** に示す．ここで，図中の破線は HTTPS 通信を示している．

この負荷分散方式は，各 SSL-Web サーバに対して順番に，またはサーバの稼動状況や処理能力に応じて HTTPS 通信を分散化するため，LB Stateful のそれと比べて，特定の SSL-Web サーバに負荷が集中する可能性が低く，クラスタ内の計算機資源を適切に活用することができる．しかしながら，これらの負荷分散方式では，クライアントと SSL-Web サーバ間で確立した SSL セッションの対応関係を考慮せずに HTTPS 通信を分散化するため，図 3 の HTTPS 通信 (2) が示すように，クライアントの通信がつねに当該 SSL セッションを保持するサーバに割り振られるとは限らない．たとえば，Apache[12] などの Web サーバでは，TCP の KeepAlive 値が設定されており，このタイムアウトを契機に，負荷分散装置は後続の HTTPS 通信の分散先を再決定する．このとき，新たな分散先となった SSL-Web サーバ上に該当する SSL セッションがすでに確立されている場合には，それを再利用することができるが，一方で SSL セッションが確立されていない場合は，Full Handshake を用いて新たに SSL セッションを確立する必要がある．したがって，HTTPS 通信が複数のサーバに分散される可能性のある LB Stateless 方式では，確立済みの SSL セッションを継続して再利用するが困難であり，分散先が変更するたびに，Full Handshake の実行をクライアントとサーバに強いる可能性がある．

☆ Internet Explorer では，定期的に SSL Session ID を更新するように設計されており，通信の永続性を完全に確保できないことがある．そのため本論文では，本方式を議論の対象に含めない．

☆☆ クライアントの IP アドレスを参照して通信の分散先を決定するアルゴリズムである．

3.2 DNS サーバ

これは，DNS（Domain Name System）サーバとSSL-Web サーバを連携し，負荷分散方式として DNS Round Robin を用いることで HTTPS 通信の分散化を図る方式である．しかしながら，実際は，DNS サーバやクライアントのキャッシング機能により，特定のSSL-Web サーバに対して大量の SSL 通信が集中することがある[13]．さらに，本来，DNS サーバは各サーバの稼動状況を確認しないため，結果として，すでに SSL 通信が集中している，または停止状態にあるSSL-Web サーバの IP アドレスをクライアントに返すことさえある．したがって，SSL 通信の負荷分散をDNS サーバに委託するこのクラスタリング方式では，均等な SSL 通信の分散化が困難である．

3.3 SSL リバースプロキシ

これは，SSL リバースプロキシが HTTPS 通信を処理し，さらに，その後方に配置された複数の Web サーバに対して HTTP 通信を分散化することでシステム全体の処理性能を向上させる方式である．**図 4** は，SSL リバースプロキシを 2 台用いた際のシステム構成例を示している．ここで，図中の破線は HTTPS 通信を，実線は HTTP 通信を示している．以降，SSL リバースプロキシを 1 台で構成する場合（以降，SSLRP-S）と複数台で構成する場合（以降，SSLRP-M）とに分け，それぞれの問題点について述べる．

3.3.1 SSLRP-S について

1 台の SSL リバースプロキシを配置したこのシステム構成では，SSL リバースプロキシがシステムのゲートウェイに相当し，ここに処理コストの高い HTTPS 通信が大量に集中することによって，Web サーバの規模や負荷バランスに関係なくシステム全体の性能を低下させる要因，すなわち，SSL リバースプロキシがこのシステムにおける単一障害点となる可能性がある．また，この方式では，バックエンドサーバに対しては，HTTP 通信を分散化できるが，SSL 通信自体の処理の分散化ができないため，SSL のアクセス数の増加に対する対策としては，SSL リバースプロキシ単体での性能向上，いわゆる，スケールアップに頼ることになる．

3.3.2 SSLRP-M について

3.3.1 項で指摘した問題点に対する 1 つの解決策として，SSL リバースプロキシを 2 台以上で構成し，HTTPS 通信の負荷を分散化する方法が考えられる（図 4）．

この場合の論点は，並列に配置された SSL リバースプロキシに対する HTTPS 通信の分散方法である

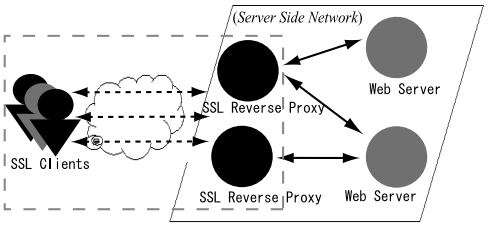

図 4 SSL リバースプロキシを用いたシステム構成例
Fig. 4 An example cluster by deploying two SSL reverse proxies.

（図 4 の破線枠の関係）．HTTPS 通信の分散化に負荷分散装置を利用した場合，3.1 節で示したとおりの問題，また，DNS サーバを利用した負荷分散方式では，3.2 節で指摘した問題に帰着する．したがって，このようなシステム構成でも，前述の議論と同様に，その問題点として負荷バランスの均等化の困難さや過剰な SSL セッションの生成があげられる．

3.4 既存方式の再考と提案システムの目的

ここでは，既存方式の問題点を簡潔にまとめ，さらに関連研究について示す．また，それらをふまえ，提案するクラスタリング方式の目的を定める．

前述のとおり，LB Stateful 方式における問題点は，大量の SSL 通信がクラスタを構成する一部のサーバに集中する可能性があることであり，同様に，SSLRP-S 方式の場合も，SSL 通信を一元的に管理するサーバが存在し，そこが負荷の集中点となるため，システム全体の性能を低下させる単一障害点になる可能性がある．次に，LB Stateless 方式では，サーバの稼動状況に応じて SSL 通信が複数のサーバに分散化されるため，クライアントとサーバ間で新規 SSL セッションを不要に確立するといった問題がある．

関連研究として，IP，TCP，および，アプリケーションレベルでの負荷分散クラスタリング方式[14)~19)]がある．しかしながら，いずれの方式も SSL のクラスタリングには直接適用することはできない．TCP および SSL に対するクラスタリング方式の提案[20)]があるが，これは，SSL サーバのフェイルオーバを目的とした提案であり，クラスタの処理性能の向上を目的としたものではない．さらに，SSL サーバが確立した SSL セッションをクラスタ内に配置されたデータベース上で管理することで，サーバ間での SSL セッションの共有を実現する方式[21),22)]があるが，システムの提案のみであり，実装や評価については記述されていない．

以上，既存方式と関連研究をふまえて，本論文では，セッションマイグレーション（詳細は，4 章）を用い

ることで複数のサーバ間での SSL セッションの共有化を実現し，これにより負荷の再配布と均等化を行うことを目的とする．

4. 提案システム

4.1 セッションマイグレーションの概要

本来，SSL のセッションは，それを生成したクライアントとサーバ間のみで共有されるため，後続のバルク暗号通信や Session Resumption は，当該主体間でのみ実行される．それに対して，SSL セッションを他のサーバにマイグレーションすることで，Handshake プロトコルとバルク暗号通信を分離し，それぞれの独立性を高めることで，各フェーズを異なるサーバ上で透過的に実行することが可能となる．ここで SSL セッションのマイグレーションと SSL 通信の細分化の概要を**図 5** に示す．これは SSL Server A と確立した Client C の SSL セッション *Sess_C* を SSL Server B にマイグレーションしたことを意味しており，また，図中の粗い破線は Full Handshake を，実線はバルク暗号通信を示している．

このように，セッションマイグレーションによってサーバ上で実行できる SSL の処理単位を細分化することで，あるサーバに割り振られた SSL の処理を他のサーバに改めて再分配することが可能となり，複数のバックエンドサーバの一部だけにアクセスが集中するような負荷バランスの偏向が発生した場合でも，負荷の再配置が可能となる．さらに，セッションマイグレーションによって，複数のサーバ間で SSL セッションを共有することで，SSL のアクセスが複数のサーバに分散化されるような場合でも，どのサーバ上でも当該 SSL セッションを再利用することができる．

そこで，本論文では，上述した SSL セッションのマイグレーションを利用した負荷分散クラスタリング方式を提案する．

4.2 提案システムの主体構成と役割

提案システムは，以下の主体によりクラスタを構成する．各主体の役割を示し，その構成例を**図 6** に示す．ここで図中の「RAgent/LB」は，RAgent または LB を利用することを意味しており，その詳細は 4.4 節に示す．また，SSLAgent は，アクセスの規模に応じて，拡張することができることに注意されたい：

RAgent

RedHat Linux-9（kernel 2.4.20）上で gcc-3.2.2-5（C 言語）を用いて実装した RAgent（Routing Agent）は，クライアントと SSLAgent との間に存在するディスパッチャである．RAgent は，クライアントからの

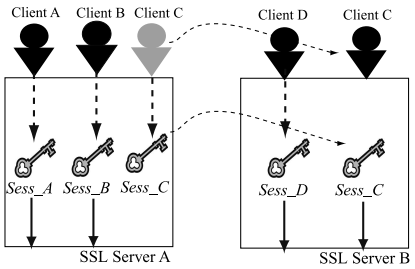

図 5 SSL セッションマイグレーションと SSL の細分化の概観
Fig. 5 Concept of SSL session migration.

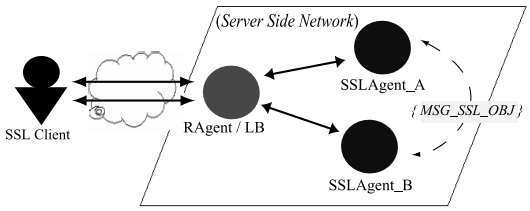

図 6 提案システムの構成例
Fig. 6 An example configuration of proposed system.

表 1 SSLv3 と TLS プロトコルのレコード識別値
Table 1 SSLv3 and TLS record type.

Protocol Type	Hexadecimal
ChangeCipherSpec	0x14
Alert	0x15
Handshake	0x16
Application Data	0x17

Handshake 要求に対しては，Round Robin 方式☆で SSLAgent への分散化を，また，バルク暗号通信に対しては，通信の永続性を確保するために，Client IP Address 方式で適切な転送を SSLAgent に対して行う．このように，RAgent は，SSL レコードごとに異なる 2 つの負荷分散方式で SSL 通信の処理の最適な負荷分散を実現する．ここで，RAgent は，受信した SSL レコードの種類を**表 1** に示す SSL のレコードごとに定義された識別値をもとに解析し，上述の転送処理を行う（4.4 節参照）．

LB

LB（Load Balancer）は,「クライアント」と「SSLAgent」との間に存在する負荷分散装置であり，今回は Array Networks 社の TMX3000 [6] を利用し，その負荷分散方式として Round Robin 方式を採用する．ただし，LB は，TMX3000 に限らず，他のアプライアンス型またはソフトウェア型の負荷分散装置やツー

☆ ただし，SSLAgent が稼動するマシンの性能が異なる場合などには，Least Connections 方式，Shortest Response 方式といった他の負荷分散方式を採用することもできる．

```
Struct{
        protocol_version;
        cipher_suite;
        session_id;
        read_sequence;
        write_sequence;
        Struct{
                server_random;
                client_random;
                pre_master_secret;
        }RandomValues;
        Struct{
                server_write_key;
                client_write_key;
                server_MAC_secret;
                client_MAC_secret;
                server_write_iv;
                client_write_iv;
        }SessionKeys;
}SESS_OBJ;
```

図 7 *SESS_OBJ* 構造体
Fig. 7 Structure *SESS_OBJ*.

ルでも代替できる．

SSLAgent

RedHat Linux-9（kernel 2.4.20）上で gcc-3.2.2-5（C 言語）と OpenSSL-0.9.7g [23] を用いて実装した SSLAgent は，主に Full Handshake，Session Resumption，そして，バルク暗号通信を実行するサーバである．また，後述する *MSG_SSL_OBJ* メッセージを他の SSLAgent に対して配布することでセッションマイグレーションを実現する．

4.3 メッセージとパラメータ

ここでは，提案方式においてセッションマイグレーションを実現するために独自に定義した 2 種類のメッセージとそれに含まれるパラメータの解説を行う：

(1) *MSG_SSL_ID* {*Session_id*}：
MSG_SSL_ID は，*Server Hello*（図 1）に含まれているセッション識別子 *Session_id* を通知する．

(2) *MSG_SSL_OBJ* {*SESS_OBJ*}：
MSG_SSL_OBJ は，SSL セッションのマイグレーションを実現するために必要なパラメータを格納した構造体 *SESS_OBJ*（**図 7**）を通知する．
SESS_OBJ 構造体は，クライアントとサーバ間で合意した SSL プロトコルのバージョン（*protocol_version*），暗号スイート（*cipher_suite*），セッション識別子（*session_id*）やシーケンス番号（*read_sequence*，*write_sequence*）などを保持する．また，通信の暗号化や HMAC（Keyed-Hashing for MAC）の生成と検証に利用するセキュリティパラメータを格納した *SessionKeys* 構造体もこれに含まれる．

4.4 提案システムの詳細

提案システムは，CSSL-SF と CSSL-SL と呼ばれる 2 つの動作モードをサポートしている．CSSL-SF は，エンドポイント間の通信の永続性を確保する場合に，一方で CSSL-SL は，通信の永続性を確保しない場合に利用する．

4.4.1 CSSL-SF について

CSSL-SF は，一連の SSL 通信を Full Hanshake とバルク暗号通信の 2 つのフェーズに細分化し，前者のフェーズを他のサーバに再配布することでサーバ間の負荷バランスを均等化する．ここで CSSL-SF の具体的な利用例として，3.1.1 項と図 2 で示した状況に対して適用したケースを考える．このような状況では，SSL Web Server A 上にかかる負荷が低いことは明らかであり，その計算機資源を SSL Web Server B の処理の一部に割り当てることができると考えられる．そこで，CSSL-SF を利用して，SSL セッションは SSL Web Server A 上で確立し，その後，その SSL セッションを SSL Web Server B にマイグレーションすることで，通信の永続性の確保，および，負荷の分散化が可能となる．すなわち，CSSL-SF は，集中する SSL 通信の処理の一部（Handshake プロトコル）を他のサーバに再配布することでサーバ間の負荷バランスを適切に均等化し，クラスタ内の計算機資源を最大限に活用する．

CSSL-SF の処理の流れ：

CSSL-SF における SSL 通信およびセッションマイグレーションの一連の流れについて示す．ここでは，説明上の例題として，クライアントは，図 6 の SSLAgent_A 上で Full Handshake を行い，その後，SSLAgent_B はマイグレートされた SSL セッションを利用してバルク暗号通信を行うというケースをもとに説明を行う．ただし，このケースは，説明上の一例であり，つねに，Full Handshake は SSLAgent_A 上で，バルク暗号通信は SSLAgent_B 上で実行されるわけではない．以下に示す説明文に割り当てた番号は，**図 8** のそれと対応している：

(1)〜(4)：クライアントは，TCP コネクションを RAgent と確立し，その後，Full Handshake を開始する．RAgent は，表 1 の識別値（0x16）をもとに，これが Handshake 要求であると判断し，SSLAgent_A と TCP コネクションを確立し，そのうえで，受信したメッ

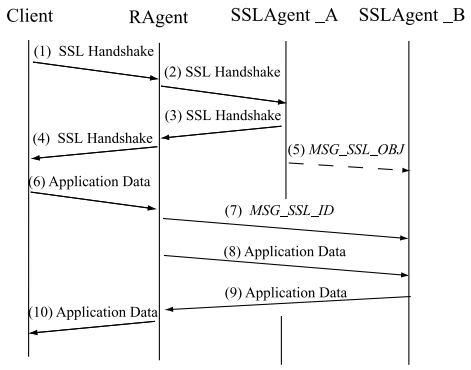

図 8 CSSL-SF における SSL 通信例
Fig. 8 An example of SSL communication in CSSL-SF.

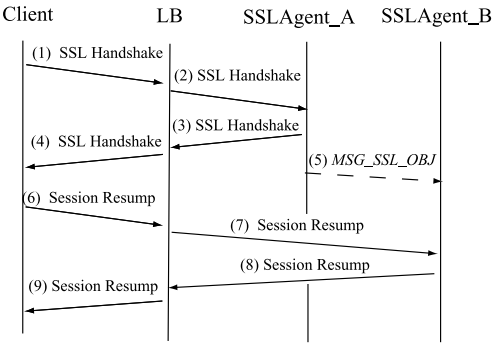

図 9 CSSL-SL における SSL 通信例
Fig. 9 An example of SSL communication in CSSL-SL.

セージを転送する．以降，クライアントと SSLAgent_A は残りの Full Handshake のメッセージを RAgent 経由で交換し，SSL セッションを確立する．また，このメッセージ交換において RAgent は，*Server Hello* に含まれる *Session_id* を取得し，MSG_SSL_ID を作成する．

(5)：次に SSLAgent_A は，SSL セッションのマイグレーションを実行するために SSLAgent_B と TCP コネクションを確立し，ステップ (1)〜(4) で確立した SSL セッションをもとに MSG_SSL_OBJ を生成し，これを送信する．

(6)，(7)： Full Handshake を終えたクライアントは，バルク暗号通信を開始し，このメッセージを RAgent に送信する．これを受信した RAgent は，表 1 の識別値（0x17）をもとにこれがバルク暗号通信であると判断する．次に RAgent は，MSG_SSL_ID を SSLAgent_B に送信し，これを受け取った SSLAgent_B は，このメッセージに含まれる *Session_id* を検索鍵として，該当する SSL_OBJ を取得する．

(8)〜(10)：RAgent は，処理 (6) で受信したメッセージを SSLAgent_B に転送し，以降クライアントと SSLAgent_B は，RAgent を経由しながらバルク暗号通信を行う．

4.4.2 CSSL-SL について

CSSL-SL は，SSL セッションのマイグレーションを利用してサーバ間での SSL セッションの共有化を実現し，後続する SSL セッションの確立要求に対する処理コストを最小化する．ここで，CSSL-SL の具体的な利用例として 3.1.2 項と図 3 で示した状況に対して適用したケースを考える．この場合，SSL Web Server A で確立した SSL セッションを他のサーバでも再利用することができれば，クライアントの通信の宛先が変わるたびに Full Handshake を実行する必要性がない，すなわち，以降は Session Resumption によって SSL セッションの再確立が可能となる．そこで，CSSL-SL では，SSL Web Server A で確立した SSL セッションを事前に SSL Web Server B にマイグレートしておくことで，これを実現する．

CSSL-SL の処理の流れ：

CSSL-SL における SSL 通信およびセッションマイグレーションの一連の流れについて示す．説明の例題として，クライアントは図 6 の SSLAgent_A と Full Handshake とバルク暗号通信を行った後，LB によって後続の SSL 通信の宛先が SSLAgent_B へと変更されたというケースに基づいて説明する．以下に示す説明文に割り当てた番号は，**図 9** のメッセージに割り当てたそれと対応している．なお，バルク暗号通信の処理は省略する：

(1)〜(4)：クライアントは，LB 経由で SSLAgent_A と Full Handshake メッセージを交換し，その後，同様にバルク暗号通信を実行する．

(5)：SSLAgent_A は，当該 SSL セッションをもとに MSG_SSL_OBJ を作成し，これを UDP を用いてクラスタ内のすべての SSLAgent に送信する．

(6)，(7)：LB は，採用する負荷分散方式に従いクライアントからのメッセージの分散先を変更するために，クライアントと確立した TCP コネクションを切断する．ここでクライアントは，再度 SSL 通信を開始するために処理 (1)〜(4) で確立した SSL セッションパラメータをもとに Session Resumption を試みる．LB は，このメッセージを SSLAgent_B に転送する．

(8)，(9)：SSLAgent_B は，保持している複数の SSL_OBJ から該当するパラメータを取得するために，*Client Hello* に含まれる *Session_id* を検索鍵として利用する．該当する SSL_OBJ を取得した SSLAgent_B は，Session Resumption を経て SSL セッションの再確立を行い，その後，クライアントと LB

経由でバルク暗号通信を行う．

5. 評　　価

ここでは，SSL-Web システムに対する既存のクラスタリング方式と提案方式の比較を処理性能の観点から行う．

5.1 実 験 環 境

ここでは，実験環境（**図 10**）として，ネットワーク環境，クライアント，および，クラスタの構成について説明する．ここで図 10 の Server Side Network 内の a)，b)，c) は，今回の実験の計測対象となる 3 つの SSL-Web システムを構成する主体の組合せを示しており，たとえば，SSL-Web システム a) は Load Balancer と 2 台の Apache/mod_ssl サーバから構成，SSL-Web システム b) は Pound と 2 台の Apache から構成，同様に，SSL-Web システム c) は Load Balancer，または，RAgent と 2 台の SSLAgent から構成されることを示している（詳細は後述）：

1) **ネットワーク環境**：
本実験で利用するすべてのマシンが 1,000 Mbps の NIC を装備し，互いに 1,000 Mbps のレイヤ 2 スイッチで接続されている．また，ネットワーク上の通信遅延をエミュレートするためのツールである NIST Net [24] が稼動しているルータをクライアントとクラスタ間に配置することで，Round Trip あたり 30 msec の通信遅延を挿入した．

2) **クライアント（HTTPS 負荷生成ツール）**：
Pentium 4（3.20 GHz），512 MB の RAM を搭載したマシンであり，RedHat Linux 9（kernel 2.4.20）上に OpenSSL-0.9.7g と POSIX Threads を用いて今回の実験用に実装した HTTPS の負荷生成ツールを稼動させる．この負荷生成ツールでは，擬似クライアント（スレッド）が，GET メソッドを利用した一定の HTTPS リクエスト（たとえば，`GET sample.html HTTP/1.1`）をサーバに対して送信し，サーバからのレスポンスを受信した時点で，次の HTTPS リクエストの発行を指定したリクエスト数に達するまで，複数の擬似クライアントで繰り返すものである．本実験では，並列稼動することができる擬似クライアント数の上限を 100 としている．

3) **クラスタ（サーバサイド）**：
クラスタに利用される各サーバは Xeon（3.20 GHz），1,024 MB の RAM を搭載しており，RedHat Linux 9（kernel 2.4.20）が稼動している．計測を行う SSL-Web システムは，以下の 3 つである（図 10 と対応）：

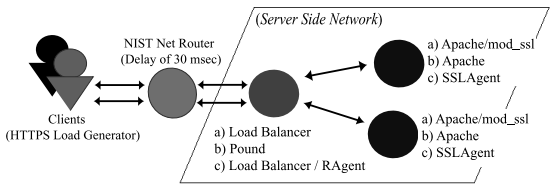

図 10 実験環境
Fig. 10 Experimental environment.

a) **Load Balancer + Apache/mod_ssl**：
3.1 節の SSL-Web システムを構築するため，負荷分散装置として TMX3000 を，SSL-Web サーバとして Apache-2.0.54/mod_ssl が稼動するサーバマシンを 2 台用いてシステムを構成した．ここで，負荷分散方式として，Client IP Address 方式を利用する構成を以降 LB-IP と表記し，Round Robin 方式を利用する構成を LB-RR と表記する．

b) **Pound + Apache**：
3.3 節の SSL-Web システムを構築するために，HTTP 通信の負荷分散機能やバックエンドサーバの障害検知機能付きの SSL リバースプロキシである Pound [8] が稼動するサーバ 1 台と Web サーバとして Apache-2.0.54 が稼動するサーバマシン 2 台を用いてシステムを構成した．また上述の規則に従い，このシステムも Pound-IP と Pound-RR の 2 つの構成がある．

c) **CSSL**：
提案システムであり，RAgent または TMX3000 を 1 台と SSLAgent を 2 台用いて構成した．前述のとおり，CSSL もその構成として CSSL-SF と CSSL-SL の 2 つがある．

5.2 処理性能の計測

提案システムの評価を行うために 2 つの実験シナリオを用意した．Scenario A は，SSL-Web システムの運用ポリシとして通信の永続性の確保を必要とする場合であり，Scenario B はそれを必要としない場合である：

Scenario A：
これはエンドポイント間の通信の永続性を確保するための負荷分散方式を採用したシステムである LB-IP，Pound-IP，CSSL-SF の性能を比較するためのシナリオである．3 台のクライアントから各 SSL-Web システムに対して計 2,100 個の Full Handshake（1,024 bit

表 2 処理性能の比較（Scenario A）
Table 2 Comparison of total processing (Scenario A).

クラスタの種類	リクエスト/秒
LB-IP	179.5
Pound-IP	108.2
CSSL-SF（Disable）	184.7
CSSL-SF（Enable）	223.4
参考値：Apache/mod_ssl サーバ 1 台	100.5

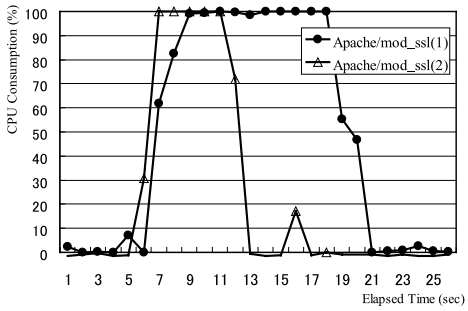

図 11 CPU 使用率（LB-IP）
Fig. 11 CPU consumption (LB-IP)

RSA，サーバ認証モード）とバルク暗号通信（RC4）として PHP[25] によって生成された 5,120 Byte のファイルの取得要求を行った際の 1 秒間あたりに処理したリクエスト数（リクエスト/秒）を計測した．ただし，CSSL-SF ついては，セッションマイグレーションを有効にした場合（Enable と表記）と無効にした場合（Disable と表記）についてそれぞれ計測する．また，各サーバ上で cyclesoak[26] を利用して，CPU 使用率を取得することで，サーバ間での SSL の処理の分散具合を確認する．**表 2** に各クラスタの計測結果（5 回計測した平均値）と参考値として Apache-2.0.54/mod_ssl が稼動するサーバ 1 台でこの処理をした場合の結果を，また，**図 11** に LB-IP の計測中の各 Apache/mod_ssl サーバの CPU 使用率を，同様に **図 12** に CSSL-SF（Enable）における各 SSLAgent の CPU 使用率を示す．

まず，Pound-IP では，大量の SSL 通信が SSL リバースプロキシである Pound に集中したため，表 2 のとおり，その処理速度は低いことが分かる．次に，LB-IP であるが，図 11 が示すように，負荷分散にクライアントの IP アドレスを参照するこの方式では，一方のサーバにクライアント 2 台分の処理が集中するため，サーバ間の CPU 使用率に大差がある．つまり，サーバ間で処理する SSL 通信の量に偏りがあるといえる．一方，CSSL-SF では，同実験においても，セッションマイグレーションを利用した Full Handshake の再配布が効果的に機能しているため，SSLAgent 間

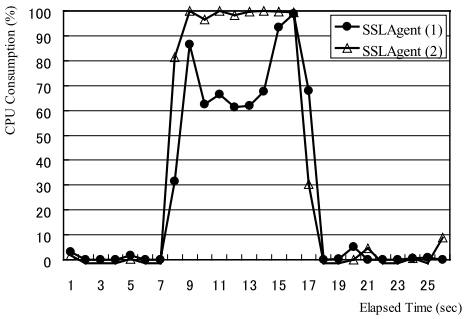

図 12 CPU 使用率（CSSL-SF）
Fig. 12 CPU consumption (CSSL-SF)

表 3 処理性能の比較（Scenario B）
Table 3 Comparison of total processing (Scenario B)

クラスタの種類	リクエスト/秒
LB-RR	441.2
Pound-RR	142.3
CSSL-SL (Disable)	389.6
CSSL-SL (Enable)	638.3
参考値：Apache/mod_ssl，1 台	319.1

の負荷バランスが適切に均等化されている．結果として，CSSL-SF は既存のクラスタリング方式よりも処理性能が向上した．

Scenario B：

これはエンドポイント間の通信の永続性を確保しない負荷分散方式を採用した LB-RR，Pound-RR，CSSL-SL の処理性能を計測するためのシナリオである．クライアントを 2 台を利用して，各 SSL-Web システムに対して 1,000 回の Full Handshake（1,024 bit RSA，サーバ認証モード），2,000 回の Session Resumption を発生させ，バルク暗号通信（RC4）として 1,024 Byte のファイルの取得要求を行った際の 1 秒間あたりに処理したリクエスト数（リクエスト/秒）を計測した．また，ここでも Scenario A と同様に，CSSL は，セッションマイグレーションの有無による比較を行い，さらに，計測中の各サーバの CPU 使用率を取得する．**表 3** に各クラスタの計測結果（5 回計測した平均値）と参考値として Apache-2.0.54/mod_ssl が稼動するサーバ 1 台でこの処理をした場合の結果を，また，**図 13** に LB-RR を計測した際の各 Apache/mod_ssl サーバの CPU 使用率を，同様に**図 14** に CSSL-SL（Enable）を計測した際の各 SSLAgent の CPU 使用率を示す．

LB-RR では，クライアントの Session Resumption 要求に対して確実に応答できない場合があり，計 3,000 個の SSL セッションを確立した際，平均で約 1,893 個

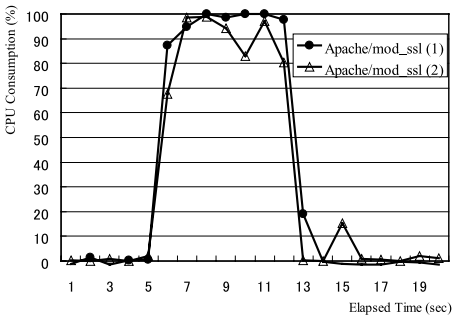

図 13 CPU 使用率（LB-RR）
Fig. 13 CPU consumption (LB-RR)

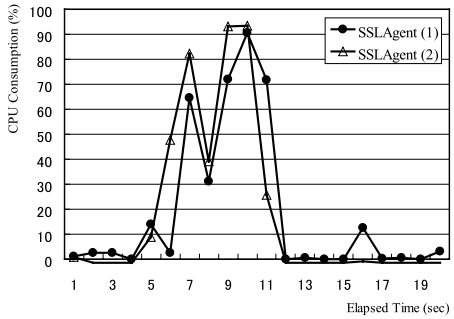

図 14 CPU 使用率（CSSL-SL）
Fig. 14 CPU consumption (CSSL-SL)

の SSL セッションを Full Handshake で確立した．そのため，2 台の Apache/mod_ssl サーバの CPU 使用率がつねに高い状態にあることが図 13 から分かる．次に，Pound-RR では，Pound がすべての SSL 通信を一元的に処理しているため，Full Handshake の実行回数は，つねに 1,000 回であった．つまり，すべての Session Resumption 要求に正確に応答したことになる．しかしながら，SSL 通信の処理が Pound に集中するため，その処理性能は低い．それに比べて，CSSL-SL（Enable）では，セッションマイグレーションによる SSL セッションの共有化を実現しているため，Session Resumption 要求に対する応答回数が LB-RR のそれよりも大幅に向上し，Full Handshake の実行回数は，平均して 1,124 回であった．これにより，表 3 に示すとおり，その処理性能が既存の 2 つの方式に比べて明らかに向上した．なお，参考値として，CSSL-SL（Disable）が実行した Full Handshake の平均回数は，2,883 回であった．

6. ま と め

本論文では，SSL セッションのマイグレーションを利用した負荷分散クラスタリング方式の提案を行った．提案方式では，クラスタを構成するサーバ間の負荷バランスの均等化，および，SSL の Handshake プロトコルが備えるセッション再開機能を効果的に利用して，SSL セッションを確立するための処理コストの最小化を図った．また，処理性能の計測により，提案システムを利用することで SSL の処理をサーバ間で適切に分散化できること，そして，SSL-Web システムの処理性能が既存の方式に比べて明らかに向上することを確認した．

参 考 文 献

1) Freier, A.O., Kocher, P. and Kaltorn, P.C.: The SSL Protocol Version 3.0 draft (Mar.1996).
2) Dierks, T. and Allen, C.: RFC2246: The TLS Protocol Version 1.0 (Jan. 1999).
3) Coarfa, C., Druschel, P. and Walach, D.S.: Performance Analysis of TLS Servers, *ISOC NDSS*, San Diego, California (Feb. 2002).
4) Apostolopoulos, G., Peris, V. and Saha, D.: TransportLayer Security: How much does it really cost?, *The Conference on Computer Communications, joint conference of the IEEE Computer and Communications Societies* (1999).
5) Lin, X., Wong, J.W. and Kou, W.: Performance Analysis of Secure Web Server Based on SSL, Lecture Notes in Computer Science, Vol.1975, *ISW 2000*, p.249 (Jan. 2000).
6) ArrayNetworks, TMX Series. http://www.arraynetworks.net/
7) F5 Networks. http://www.f5.com/products/bigip/
8) Pound. http://www.apsis.ch/pound/
9) Zhao, Li, Iyer, R., Makineni, S. and Bhuyan, L.: Anatomy and Performance of SSL Processing, *Performance Analysis of Systems and Software, 2005, ISPASS* (2005).
10) Rescorla, E.: *SSL and TLS, Designing and Building Secure Systems*, Addison Wesley (2000).
11) Syme, M. and Goldie, P.: *OPTIMIZING NETWORK PERFORMANCE with CONTENT SWITCHING, Server, Firewall and Cache Load Balancing*, Prentice Hall (2003).
12) The Apache Software Foundation. http://www.apache.org/
13) Bourke, T.: *Server Load Balancing*, Oreilly & Associates Inc. (Aug. 2001).
14) Alvisi, L., Bressoud, T.C., EL-Khashab, A., Marzullo, K. and Zagorodnov, D.: Wrapping Server-Side TCP to Mask Connection Ffailures, *Proc. IEEE INFOCOM 2001*, pp.329–337

(2001).
15) Sultan, F., Srinivasan, K., Ilyer, D. and Iftode, L.: Migratory TCP: Connection Migration for Service Continuity in the Internet, *IEEE International Conference on Distributed Computing Systems*, Vienna, Austria, pp.189–191 (Mar. 2004).
16) Marwah, M., Mishra, S. and Fetzer, C.: TCP Server Fault Tolerance Using Connection Migration to a Backup Server, *Proc. IEEE Intl. Conf. on Dependable Systems and Networks 2003*, pp.373–382 (June 2003).
17) Shoa, Z., Jin, H., Chen, B., Xu, J. and Yue, J.: HARTs: High Availability Cluster Architecture with Redundant TCP Statcks, *Performance, Computing, and Communications Conference, 2003*, pp.253–260 (Apr. 2003).
18) Sit, Y.-F., Wang, C.-L. and Lau, F.: Socket Cloning for Clustered-Based Web Server, *International Conference on Cluster Computing* (Sep. 2002).
19) NOKIA: White Paper: The IP Clustering Power of Nokia IP VPN Keeping Customers Connected (Jan. 2005).
20) Rescorla, E., Cain, A. and Korver, B.: SSLACC: Aclusterd SSL Accelerator, *11th USENIX Security Symposium* (Aug. 2001).
21) Cox, M.J. and Thorpe, G.: *High Scalability for SSL and Apache*, OREILLY (2000).
22) Cox, M.J and Thorpe, G.: Apache e-Commerce Solutions, *ApacheCon2000*, Florida (Jan. 2000).
23) The OpenSSL Project. http://www.openssl.org/
24) NIST Net Home Page. http://www-x.antd.nist.gov/nistnet/
25) The PHP Group. http://www.php.net/
26) http://www.zipworld.com.au/˜akpm/linux/#zc

(平成 18 年 11 月 24 日受付)
(平成 19 年 6 月 5 日採録)

初谷 良輔
2007 年明治大学大学院理工学研究科基礎理工学専攻修士課程修了．同年株式会社野村総合研究所入社（NRI セキュアテクノロジーズ株式会社出向）．

齋藤 孝道（正会員）
明治大学理工学部情報科学科准教授．博士（工学）．