Vol.2020-OS-148 No.10
2020/2/28

サービス可用性を改善するための異種OS間プロセスマイグレーションの活用

松原 克弥[1,a)]　髙川 雄平[1]

概要：インターネット利用者数の増大に合わせて，比較的小規模な Web サービス基盤に対してもアクセス性能と可用性の両立が求められている．一方，個人などが構築する小規模な Web サービス基盤では，負荷に合わせた計算資源の追加投入や冗長サーバ構成にかかるコストを負担することが難しい場合がある．本研究では，利用できる計算資源が同じ場合でも，動作する OS の内部実装の違いによって性能特性が異なる場合があることに着目する．Web サーバが動作する OS を動的に切り替えることで，計算資源の追加投入や冗長サーバを用いずに，Web サーバのアクセス性能と可用性を両立するシステムの実現を目指す．本稿では，Linux と FreeBSD を対象とした異種 OS 間プロセスマイグレーションの実現手法について述べ，OS を動的に切り替えた際の動作中 Web サーバの性能特性と負荷耐性の変化を実験結果により示す．

キーワード：CRIU，Linux バイナリ互換機能，C10K 問題

1. はじめに

スマートフォンの普及や IoT の活用拡大にともなうインターネットユーザの増大に合わせて，Web サービス基盤に対するアクセス性能と可用性への要求が厳しくなっている．商用サービスでは，負荷に応じてサーバを追加投入するスケールアウトや，より高性能なハードウェアへ置き換えるスケールアップにより対応することが一般的となっている．一方，スケールアウトやスケールアップには，ハードウェアの購入や買い替え，さらに構築というコストがかかり，オンプレミスのサーバや小規模な Web サービス基盤では，これらの対応を行うことが難しい場合がある．

一方，ソフトウェアによる対策も，様々な方策が考えられてきた．高負荷対策におけるソフトウェアの重要性を示す一例として，クライアント 1 万台問題（C10K 問題）が知られている．C10K 問題は，アクセスするクライアント数が 1 万台に達すると，ハードウェア性能の余裕があってもソフトウェア構造上の問題によりパンクすることで可用性が失われてしまうという事例を示している．C10K 問題が注目されたきっかけは，メジャーな Web サーバ実装である Apache Web サーバの構造上の問題であるが，1 万以上のクライアントからのアクセスを想定する必要があるという点で，Web サービスを提供するシステムには，高負荷に対してもパンクせず，安定してサービスを継続提供できる高可用性が求められる．

前述の C10K 問題の例も示すとおり，高負荷時にも停止することなく高可用性を実現するシステムには，ハードウェア性能だけでなく，ソフトウェアの性能特性も重要となる．システムの性能特性へ大きな影響を与えるソフトウェア・コンポーネントとして，オペレーティングシステム（以降，OS）がある．現在も企業やコミュニティにより様々な OS が開発・メンテナンスされており，日々，機能や性能が進化している．たとえば，現在，世界最大のオープンソース・ソフトウェアである Linux は，現在，組み込み機器から大型サーバまで様々な用途の計算機で利用されている．しかし，すべての Windows や Windows Server を置き換えるまでには至っていない．また，NetFlix 社は，サービスを Linux を用いたマイクロサービスアーキテクチャで構成しているが，動画データ配信を担う CDN(Content Delivery Network) システムの OS プラットフォームには FreeBSD を採用している．このように，用途や状況に応じて最適な OS を選択することは，今後も続くと予測する．

実際，文献 [1] や文献 [2] では，Linux や FreeBSD など複数の OS の性能特性をベンチマークテスト結果により評価しており，OS 毎に性能特性が異なる結果を示している．また，文献 [3] では，OS を構成する機能の一つであるスケジューラの実装の違いに着目し，ストレージやネットワークに関するベンチマークの結果から，スケジューラの実装

[1] 公立はこだて未来大学
Future University Hakodate, Hokkaido 041–8655, Japan
[a)] matsu@fun.ac.jp

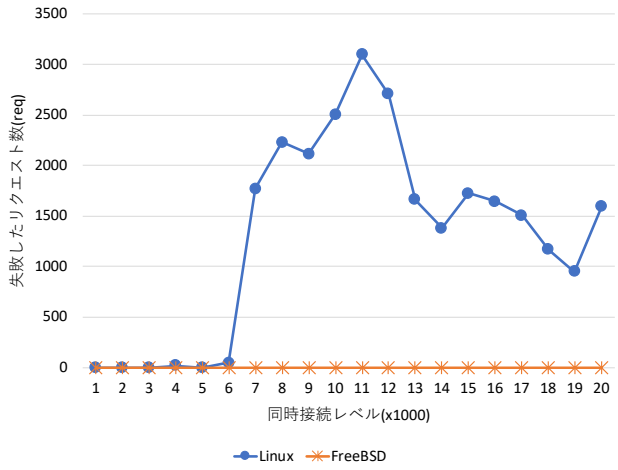

図 1　Linux と FreeBSD 上の Nginx への Apache Benchmark 実行における同時接続レベル別処理失敗リクエスト数

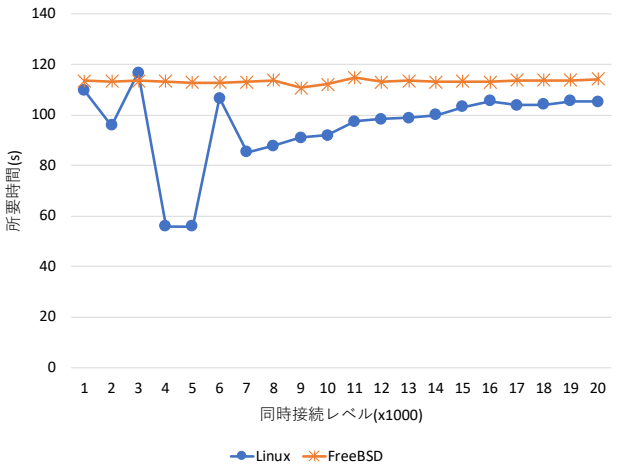

図 2　Linux と FreeBSD 上の Nginx に対する 100 万リクエストの処理に要した時間

の違いにより OS 性能が大きく異なることを示している．

本研究では，サービス要件の高度化によって求められている高可用性を OS の切り替えによるサービス性能特性の変化を利用して実現することを目的とする．そのために，状況に応じた複数 OS 動的切り替えを可能する異種 OS 間プロセスマイグレーション機能の実現と活用を提案する．

2. Linux/FreeBSD におけるネットワーク性能特性

前述のように，同一のハードウェアプラットフォームを用いた場合でも，OS 毎に異なる特性を示す可能性がある．本研究では，Web サーバの処理性能に対する OS 特性の影響を確認する予備実験として，Linux と FreeBSD 上で動作する Web サーバ Nginx に対して，Apache Benchmark を用いて負荷をかけた際の性能を測定した．実験環境として**表 1** に示す PC2 台を用いて，片方の PC で Apache Benchmark を実行し，ネットワーク接続したもう 1 台の PC で FreeBSD もしくは Linux を起動した上で Nginx を動作させた．ネットワーク通信に関する OS カーネルパラメータおよび Nginx のパラメータは，それぞれ**表 2**，**表 3**，**表 4** に示す値で設定した．

図 1 は，横軸が Apache Benchmark に指定した同時接続数の目標値設定の値（以降，同時接続レベル），縦軸が Apache Benchmark から Nginx へ送信されたリクエストのなかでレスポンスが正しく受信できず処理が完了できなかった数を示している．図 1 の Linux の結果は，同時接続レベルが 6,000 よりも大きくなると，送信したリクエストの一部が失敗していることを示している．一方，FreeBSD の値は，Apache Benchmark が送信したリクエストが，同時接続レベルに関わらずすべて処理できていることを示している．

図 2 は，前述と同条件で Apache Benchmark を実行したときに要した時間を示している．100 万リクエストに対する総処理時間を Linux と FreeBSD で比較すると，Linux の一部の結果にバラツキがあるものの，どの同時接続レベルにおいても，FreeBSD よりも Linux が高速に処理できていることがわかる．

これらの予備実験結果から，FreeBSD と Linux では，以下に述べるネットワーク性能特性を持つことを示せた．FreeBSD のネットワーク機能は，同時接続数が増えたような高負荷状態に対して耐性がある．一方，Linux は，高負荷耐性において FreeBSD より劣っているところがあるが，処理性能おいて FreeBSD より勝っている．

3. プロセスマイグレーションによるサービス可用性改善

本稿では，性能特性の異なる Linux と FreeBSD を活用して，負荷やメンテナンスなどの状況に応じて OS を切り替えることによりサービスの高可用性を実現することを提案する．

前章の予備実験により，低負荷時の処理性能は，Linux が優れていることが判明している．一方，FreeBSD は，処理性能で Linux に劣るが，高負荷時もエラーが発生することなくリクエストを処理できる特性を持つ．この性能特性の違いを活用して，低負荷時は Linux 上でサービスを稼働させ，負荷が上昇した際は FreeBSD へ稼働中のサービスを移送することで高負荷アクセスに対応することで，常にサービスへのアクセスが可能な高可用性が実現できる．

また，OS に依存した脆弱性対策によりシステム更新しなければならない場合に，その脆弱性がない異なる OS へ稼働中のサービスを移送することで可用性を維持する対策も可能となる．OS を含めた脆弱性は日々発見されており，本提案手法を活用することで，システム更新などの素早い脆弱性対策とサービス高可用性維持の両立を容易にする．

4. Linux-FreeBSD 間プロセスマイレーションの実現

本提案手法では，異なる性能特性を持つ Linux と FreeeBSD 間でサービスが稼働するプロセスをマイグレーションすることで，サービスの可用性向上を実現する．本提案手法の実現には，Linux-FreeBSD 間でのプロセスマイグレーション機能が必要となる．本研究では，Linux 上ですでに実現しているプロセスマイグレーション・ツールである CRIU を参考にしつつ，CRIU によって保存されたプロセス状態を FreeBSD 上で復元する機能を実現する．

4.1 CRIU

CRIU（Checkpoint/Restore in Userspace）[4], [5] は，Linux 上でプロセスマイグレーションを実現するツールとして開発され，OpenVZ や LXC，Docker など多くの Linux コンテナシステムでマイグレーションで活用されている．CRIU は，プロセスマイグレーションを行うのための，プロセスの状態取得と状態復元の機能を実現している．

メモリやソケットバッファ以外の各状態は，Google Protocol Buffer[6] 形式で保存されている．Google Protocol Buffer は，異なるプログラム言語でもデータの受け渡しを容易にすることを目的としたデータ形式である．なお，メモリやソケットバッファは RAW データをファイルに書き込んでいる．

CRIU がプロセスの状態を保存する際には，ptrace システムコールや procfs 仮想ファイルシステムを用いて，対象プロセスが持つ計算資源の情報を取得する．CPU レジスタの情報は，ptrace システムコールの GETREGS を用いて取得できる．復元では，割り込み処理が呼ばれた際に自動的にセットされる rt_sigreturn システムコールを利用し，取得した値を全てのレジスタに一斉にセットしている．メモリマップの情報は，procfs の maps ファイルから取得することができる．メモリの内容は，vmsplice システムコールや splice システムコールを用いて，メモリ内容をパイプにコピーし，パイプからファイルに書き込んでいる．復元では，mmap システムコールを利用し，メモリを保存したファイルをマップすることで復元している．

Parasite code という他のプロセスに任意のプログラムを埋め込み実行させる手段を用いることで，プロセスが持つファイルアクセス情報やネットワークの情報の取得と復元を実現する．Parasite code によって埋め込まれたプログラムから Unix Domain Socket 経由で送信されてくるファイルディスクリプタを受け取り，fctl システムコールや fstat システムコールを利用することでプロセスが持つファイルアクセス情報を取得している．復元したネットワークアクセス状態を持つファイルディスクリプタを Unix Domain Socket 経由で送信することで，任意のプロセスがネットワーク通信を再開できるようにしている．

表 1 実験環境

CPU	Intel Core-i3 2.4GHz
RAM	4GB
Disk	SATA 5400RPM HDD
NIC	Intel 82579V Gigabit Ethernet
OS	CentOS 7.7-1908 (Linux kernel: 3.10) FreeBSD 13.0-CURRENT
Web Server	Nginx v1.16
Client App.	ApacheBench v2.3

表 2 Linux カーネルのパラメータ設定値

`ulimit open files`	113958
`net.core.somaxconn`	20000
`net.ipv4.tcp_keepalive_time`	7200
`net.ipv4.tcp_tw_reuse`	1
`net.ipv4.ip_local_port_range`	10000 65535
`net.ipv4.tcp_max_syn_backlog`	20000

表 3 FreeBSD カーネルのパラメータ設定値

`ulimit open files`	113958
`kern.ipc.maxsockets`	126627
`kern.ipc.soacceptqueue`	20000

4.2 FreeBSD におけるプロセスマイグレーションの実現

FreeBSD におけるプロセスマイグレーションの実現は，著者らが文献 [7], [8] で報告した実装技術を用いる．CRIU が生成する Google Protocol Buffer 形式に沿って保存されたプロセスの状態をファイルから読み出し，ptrace や Parasite code インジェクションにより状態を復元する．Linux プロセスの実行に必要なシステムコールのエミュレーションなどは，FreeBSD が持つ Linux 互換レイヤを活用する．procfs などの，FreeBSD の Linux 互換レイヤが提供する機能が不十分なものは，代替機能を用いて実現する手法を確立する．ネットワーク状態の保存と復元を実現する TCP Repair 機能は，FreeBSD カーネル内のネットワークスタック実装を改造することで実現する．

5. 評価

本章では，本研究で実現した異種 OS 間プロセス・マイグレーションを評価するための実験とその評価について述べる．なお，全ての実験で同一の PC を用いており，その PC のスペックを表 1 に示す．また，Nginx とネットワーク通信に関わる各 OS のカーネルパラメータは表 4，表 2，表 3 に示す．

5.1 プロセスマイグレーションによるサービス可用性への影響

稼働中の Nginx プロセスを Linux-FreeBSD へマイグレーションした際に，移送中にリクエストが受け付けられない

表 4　Nginx の設定

worker_processes	1
worker_connections	20000
sendfile	off
keepalive_timeout	65
backlog	65535

表 5　作成されたプロセス保存ファイル

レジスタ	4.00 KB
メモリ	9.56 MB
その他	1.61 KB
合計	9.57 MB

表 6　マイグレーションの所要時間

プロセス状態保存	44 ms
ネットワーク転送	429 ms
プロセス状態復元	1,150 ms
合計	1,623 ms

期間（ダウンタイム）を計測する．マイグレーションによるダウンタイムは，Linux 上でのプロセスの状態保存にかかる時間，保存した情報をネットワーク転送する時間，および，FreeBSD 上でプロセス状態の復元にかかる時間の合計となる．本計測では，前述のそれぞれの時間を計測し，合計を算出することでサービスのダウンタイムとした．

本実験で作成されたプロセス状態保存ファイルのサイズを**表 5** に示す．プロセスの状態は，3 つのファイルに分割されて保存される．本表の項目「レジスタ」は，CPU に関する情報を保存しているファイルのサイズである．「メモリ」は，メモリのローデータとメモリマップに関する情報を保存しているファイルのサイズである．「その他」は，ファイルやプロセス ID などの上記に該当しない情報を保存しているファイルのサイズである．

計測された各マイグレーション処理の時間を**表 6** に示す．プロセスマイグレーションの各処理に要した時間の合計 1,623 ミリ秒がサービスのダウンタイムとなる．文献 [9] には，Web ページにアクセスしたユーザが，その Web ページを閲覧せず離脱してしまう割合が約 3 秒をすぎると大きく上昇するという統計がある．つまり，Web サーバの負荷やダウンが原因で 3 秒以上レスポンスがない場合は，サービス可用性に影響しているという判断ができる．本計測結果で得られたダウンタイムは，3 秒と比較して十分に小さいため，ダウンタイムの要因であるマイグレーションのオーバヘッドは十分小さいと考える．

5.2　OS 切り替えによる性能特性の変化

異種 OS 間プロセスマイグレーションによるサービス性能特性の変化を計測するために，Apache Benchmark 実行中に Nginx プロセスのマイグレーションを行った場合の処理時間やエラー発生率を計測する．

本実験では，Linux 上で動作する Nginx プロセスの状態保存を行い，FreeBSD 上で状態保存した Nginx プロセスを復元する．プロセスマイグレーションの前後で Apache Benchmark による当該 Nginx へのリクエスト送信を継続させておき，処理時間と正しくレスポンスが返送されなかったリクエストの数を計測した．Apache Benchmark の設定は，総リクエスト数を 100 万とし，同時接続レベルを 1,000 から 20,000 まで変化させて，計測を繰り返した．なお，Apache Benchmark のオプションには接続エラーが発生しても実行を中断しないオプションを設定することで，ダウンタイムからの復帰後もリクエスト送信が継続されるようにした．また，プロセスマイグレーションの際に移送対象となる接続済みコネクションの数を最小化するために，Apache Benchmark では HTTP KeepAlive を利用するオプションを指定せず，加えて，プロセス状態保存処理を開始する直前に SYN パケットをドロップさせるパケットフィルタを設定して，プロセスマイグレーション時に新規接続を開始しないよう抑制した．プロセスマイグレーションを行うタイミングは，Linux 上の Nginx に対して Apache Benchmark の実行を開始してから約 10 秒経過した時点とした．

正しくレスポンスを受信できず失敗したリクエスト数の計測結果を**図 3** に示す．第 2 章で述べた通り，Linux では，同時接続レベルが 6,000 を超えると失敗するリクエスト数が増加するのに対して，FreeBSD では同時接続レベルに関わらずすべてのリクエストを正しく処理できた．本提案手法である，開始 10 秒後に Linux から FreeBSD へ Nginx をマイグレーションをした場合，どの同時接続レベルでも Linux で実行を継続するのと比較して失敗リクエスト数が約 10 分の 1 以下に減少した．

100 万リクエストの処理に要した時間の測定結果を**図 4** に示す．Linux では，同時接続レベルが 4,000 と 5,000 のときに所要時間が減っているが，同時接続レベル 6,000 で再び増加し，同時接続レベルが 7,000 からはレベルが大きくなるのに比例して所要時間も増加している．一方，FreeBSD では，常に Linux より時間を要しているが，すべての同時接続レベルで一定の所要時間となった．本提案手法によるマイグレーションを実行した場合，同時接続レベル 7,000 までは FreeBSD よりも低い状態であるが，同時接続レベル 8,000 を超えると FreeBSD と変わらない時間ですべてのリクエストを処理できている．

Apache Benchmark 行う各リクエスト送信時に，Apache Benchmark が Nginx とのコネクションを確立するのに要した時間を**図 5** に示す．コネクション確立に要する時間は，Apache Benchmark 内の connect システムコールの実行時間である．この所要時間，接続要求に対するサーバの性能を表した指標として用いられ，主にサーバの accept シ

ステムコールの性能に依存する．計測結果から，Linux では，同時接続レベル 5,000 以降でレベルに比例して増加していることがわかった．一方，FreeBSD では，同時接続レベル 8,000 以上で線形的に増加しているが，多くの同時接続レベルで Linux よりも 200 ミリ秒以上小さい値となっている．本提案手法によりプロセスマイグレーションを実行した場合，Linux と FreeBSD の中間を取るような結果となった．

コネクション確立後，リクエストの送信開始からレスポンスの受信完了までに要した時間を**図 6** に示す．リクエスト送信開始からレスポンス受信完了までの時間は，Apache Benchmark 内の connect システムコールの完了後からコネクションを切断する close システムコールの実行直前までの時間を示し，特に，read システムコールを用いてレスポンスデータを受信する時間が多くの割合を占める．この値は，データ操作と転送に関するサーバの性能を表した指標として用いられる．計測結果から，Linux では，同時接続レベル 6,000 のときに一時的に減少するが，それ以外では同時接続レベルに比例して線形的に増加している．一方，FreeBSD でも線形的に増加し，多くの同時接続レベルで Linux よりも約 400 ミリ秒大きい結果となった．提案方式によるプロセスマイグレーションを実行した場合，Linux と FreeBSD の中間を取るような結果となった．

本提案手法におけるプロセスマイグレーションにより起きる変化の詳細を分析するために，同時接続レベルが 6,000 と 20,000 のそれぞれの場合で，コネクション確立にかかる時間，および，リクエスト送信開始からレスポンス受信完了にかかる時間を 1 秒ごとに平均を求め，時間経過による変化を可視化した．同時接続レベル 6,000 における Linux, FreeBSD, 提案手法のそれぞれの計測結果を**図 7**，**図 8**，**図 9** に示す．同時接続レベルが 6,000 の場合，Linux 上の Nginx では，コネクション確立に要する時間と，その後のリクエスト処理に要する時間が，どちらも 200 ミリ秒前後と五分五分の割合となっている．一方，FreeBSD 上の Nginx では，コネクション確立に要する時間が非常に小さく，逆に，その後のリクエスト処理に要する時間が Linux と比べて大きいことがわかる．本提案手法によるプロセスマイグレーションを用いることで，マイグレーションを開始する 10 秒経過までは Linux と同じ特性を示し，プロセスマイグレーションを開始した直後は，サービスダウンタイムが発生するためにコネクション確立時間が一時的に非常に大きくなる．しかし，プロセスマイグレーション完了後は，FreeBSD の性能特性に切り替わり，コネクション確立に要する時間が大幅に小さくなり，リクエスト処理時間が延びていることが確認できる．同時接続レベル 20,000 における Linux, FreeBSD, 提案手法のそれぞれの計測結果を**図 10**，**図 11**，**図 12** に示す．同時接続レベルが 20,000 の

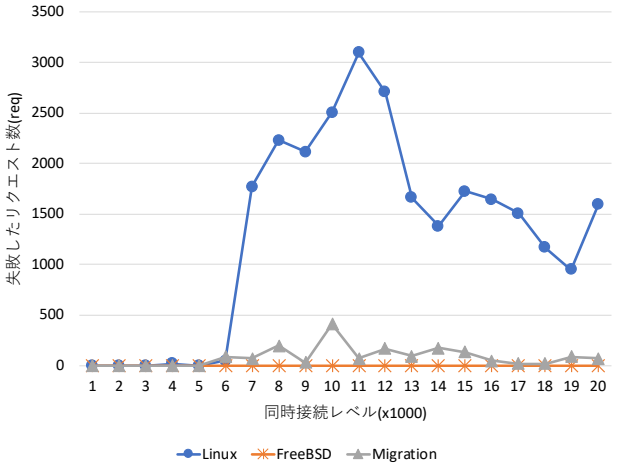

図 3　正しくレスポンスを受信できず失敗したリクエスト数

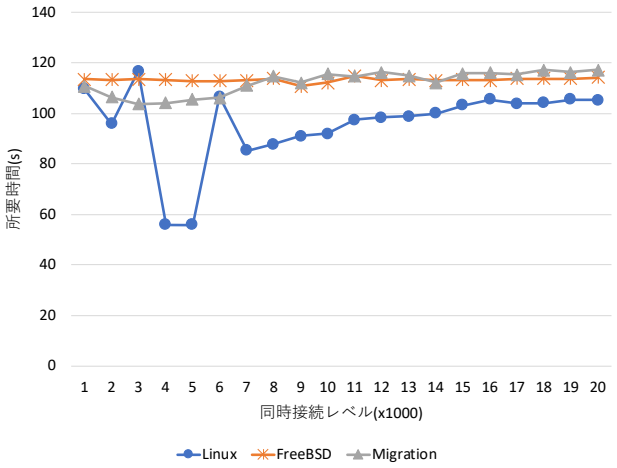

図 4　100 万リクエストを処理するのに要した時間

場合，FreeBSD 上の Nginx も Linux の場合と同様に，コネクション確立に要する時間が大きくなる．本提案手法によるプロセスマイグレーションを用いた場合，同時接続レベルが 6,000 の場合と同様に，マイグレーションを開始する 10 秒経過までは Linux と同じ特性を示し，プロセスマイグレーションを開始した直後は，サービスダウンタイムが発生するためにコネクション確立時間が一時的に非常に大きくなる．そして，プロセスマイグレーション完了後は，FreeBSD の性能特性に切り替わる．しかし，同時接続数 20,000 の場合は，Linux と FreeBSD で性能特性にそれほど大きな差がない．そのため，100 万リクエストの処理時間に対するプロセスマイグレーション後の所要時間が大きく，プロセスマイグレーション前の Linux 性能特性であるリクエスト処理時間の削減効果が小さい．

本実験結果により，本提案手法のプロセスマイグレーションによる OS 切り替えにより，切り替え前後の OS 性能特性を活かして処理時間を削減できることが確認できた．このことより，コネクション確立とリクエスト処理のそれぞれの時間を削減するために，OS を切り替えること

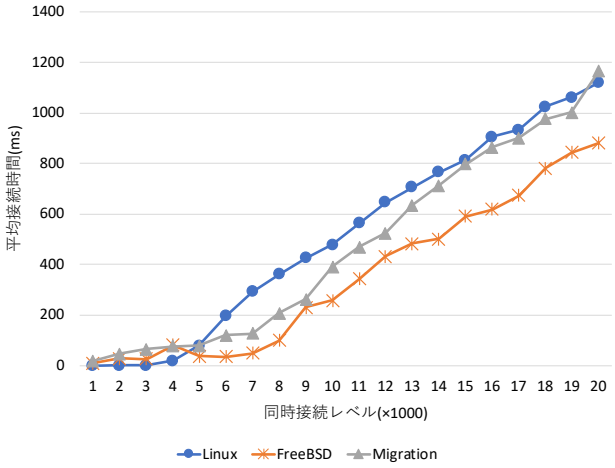

図 5　サーバとのコネクション確立に要した時間

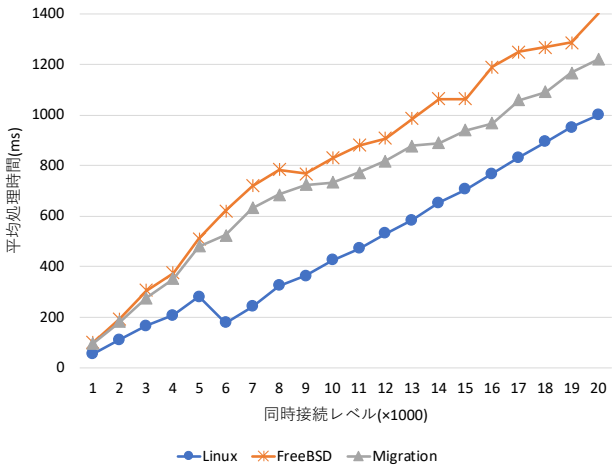

図 6　リクエスト送信開始からレスポンス受信完了までに要した時間

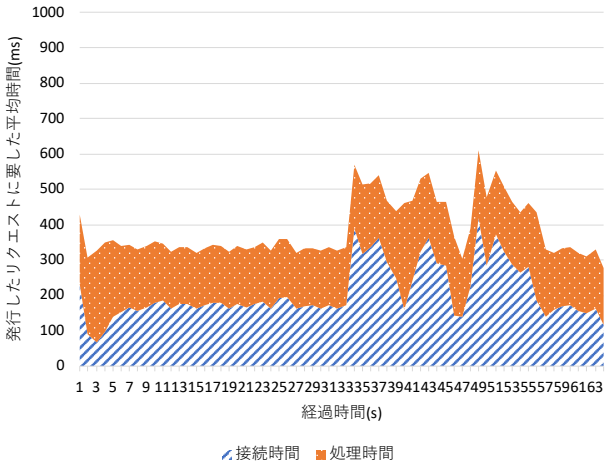

図 7　Linux に対するコネクション確立とリクエスト送信から受信完了までの平均時間内訳の経緯（同時接続レベル 6,000）

は有効であると考える．さらに，同時リクエスト数が多い高負荷時に Linux 上の Nginx で発生していたリクエスト失敗のエラーは，FreeBSD へマイグレーションすることによりその発生を解消できていることが示せた．したがって，

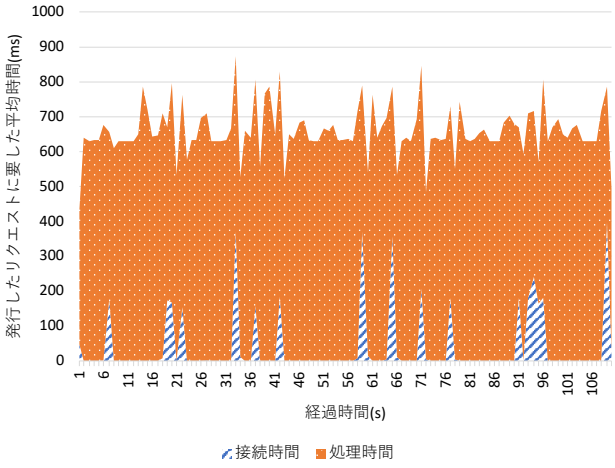

図 8　FreeBSD に対するコネクション確立とリクエスト送信から受信完了までの平均時間内訳の経緯（同時接続レベル 6,000）

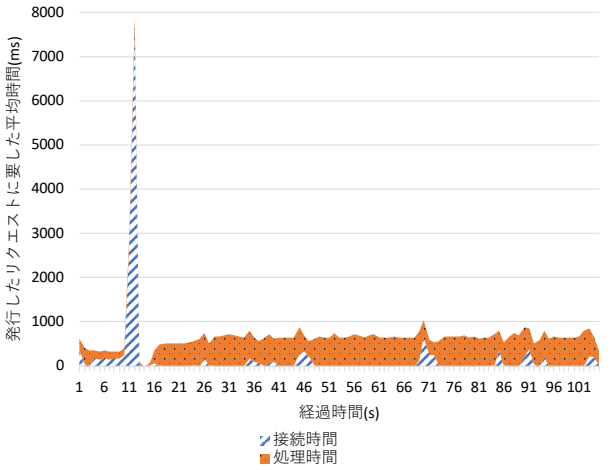

図 9　提案手法におけるコネクション確立とリクエスト送信から受信完了までの平均時間内訳の経緯（同時接続レベル 6,000）

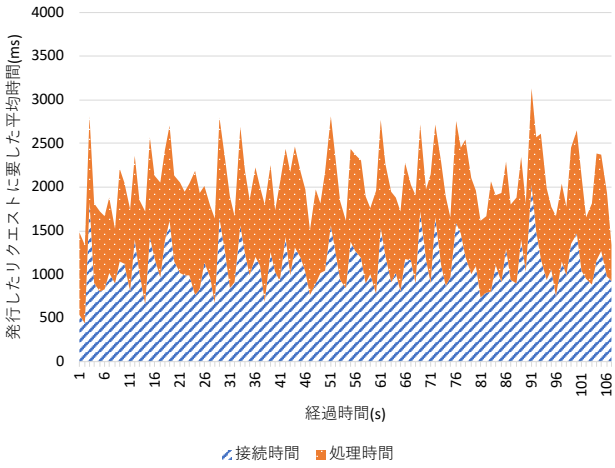

図 10　Linux に対するコネクション確立とリクエスト送信から受信完了までの平均時間内訳の経緯（同時接続レベル 20,000）

プロセスマイグレーションによりサービス可用性を向上する本提案手法は，有効に働くと判断する．今回は，プロセスマイグレーションのタイミングを 10 秒経過時に固定し

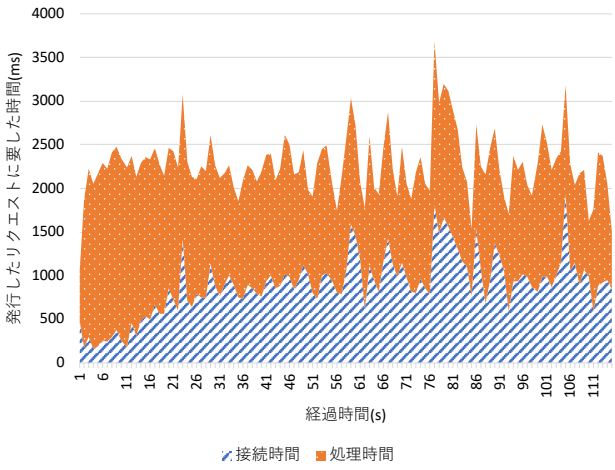

図 11　FreeBSD に対するコネクション確立とリクエスト送信から受信完了までの平均時間内訳の経緯（同時接続レベル 20,000）

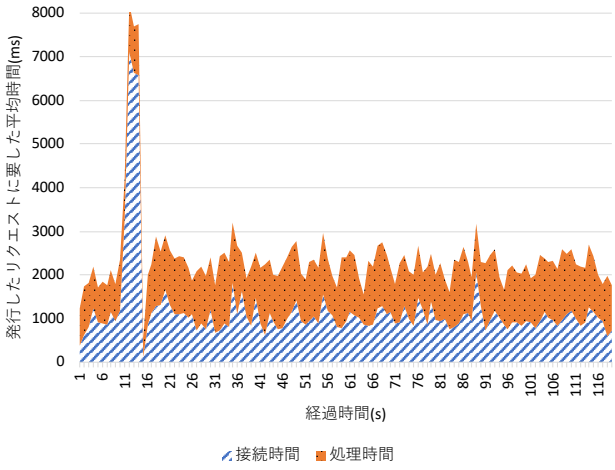

図 12　提案手法におけるコネクション確立とリクエスト送信から受信完了までの平均時間内訳の経緯（同時接続レベル 20,000）

たが，OS 切り替えによる効果を最大限享受するためには，負荷状況の変化を契機にプロセスマイグレーションを行うべきかを判断すべきと考える．今後，負荷状況の変化を予測してプロセスマイグレーションの契機を決定するためのアルゴリズムを考える必要がある．

6. 関連研究

本研究の対象である異種環境間のプロセスマイグレーションは，これまでも様々な試みが行われてきた．

文献 [10], [11] では，コンパイラを改造することで，CPU アーキテクチャが異なる環境間でプロセスをマイグレーションする技術について提案している．コンパイラが生成した各 CPU アーキテクチャ向けコード間で各関数の配置アドレスを管理することで，マイグレーション時に実行再開位置などが含まれるレジスタ値を補正している．文献 [12] は，コンパイラや OS カーネルを改造することで，実行中のプロセスを含むコンテナのマイグレーションを可能にしている．前述の先行研究では，コンパイラや OS を改造しているため，既存システムでマイグレーションを有効にするために，ライブラリやアプリケーションの再コンパイルが必要となる．本研究の提案方式では，OS カーネルやコンパイラに改造を加えないため，Nginx のような既存アプリケーションをマイグレーション対象とすることが比較的容易である．

上村らの研究 [13] では，Windows 上の Linux プログラム実行環境 BEE[14] の開発とその機能を拡張して，Windows と Linux 間でのプロセスマイグレーションを実現する手法を提案している．BEE のプログラムローダは，Linux ELF 形式のバイナリを解析して，Windows のプロセス上に実行可能な形式でプログラムを展開する．さらに，BEE では，Linux プログラムに含まれるシステムコールをエミュレーションする機能が実装されている．上村らは，この BEE の実装を使って，Linux 上で実行されているプロセス，および，Window プロセス内で実行されている Linux プロセスの実行状態の保存と復元方法を提案している．異種 OS 間でプロセスをマイグレーションする手法として本研究と関連が深いが，文献 [13] では，Linux 上でのプロセス実行状態保存と復元のみを実現しており，異種 OS 間でのプロセスマイグレーションの実装や評価が行われていない．

FreeBSD VPS（Virtual Private System）[15] は，FreeBSD Jail の機能を拡張して，Linux におけるコンテナのような隔離したプロセス実行環境の実現とその環境を含むプロセスのマイグレーションを FreeBSD 上で実現している．ただし，本機能は FreeBSD カーネルを改造して実現されており，API や仕様も独自に設計されたものであるため，Linux などの異種 OS 間でのプロセスマイグレーションは考慮されていない．

7. おわりに

本研究では，OS の性能特性の差異に着目し，プロセスマイグレーション技術を用いて稼働中のサービスを異なる OS へ移動させることにより，動的にサービス性能特性を変化させることでサービスの可用性を向上する手法を提案した．CRIU を用いて Linux 上で動作するプロセスの状態を保存し，その保存した状態を FreeBSD 上で復元するための機構を実現した．FreeBSD におけるプロセス状態の復元では，Linux のプロセス状態の復元する実装を参考に，類似機能による代替実現や Parasite Code 技術などを用いて，可能な限りユーザ空間での実現を目指し，ネットワーク通信状態の復元などのカーネル内情報への制御のためだけに留めて FreeBSD カーネルを変更した．異種 OS 間でプロセスマイグレーションを行うための差異は，プロセス初期化処理の差異，メモリ領域の差異，ネットワークの差異があり，初期化処理のタイミングの調整やメモリ空間の

各領域の再配置，フロー制御の条件式を満たすように復元することなどによって解決した．

提案手法にもどづいたプロトタイプ実装を用いて，プロセスマイグレーションによるサービス可用性向上の評価実験を行った．実験結果から，プロセスマイグレーション前後でサービスの性能特性が変化することが確認でき，提案手法を用いたサービス可用性向上の可能性を示した．

今後の展望として，サービスの状況と各 OS の性能特性を考慮したプロセスマイグレーションのタイミングを決めるアルゴリズムについて検討を進める必要がある．サービスの状況に応じて，最適な OS を選択するためには，先の状況変化を見越してマイグレーションの契機とその移送先を決定する必要がある．また，単一プロセスだけなく，複数のプロセスを含むコンテナの異種 OS 間マイグレーションを実現したい．近年，Web サーバなどクラウド上で動作するサービスをコンテナを使ってデプロイする手法が普及しつつある．Linux と FreeBSD 間でコンテナ設定を含めたマイグレーションを実現することで，実稼働システムを対象としたより実践的な評価を行いたい．

参考文献

[1] Michael Larabel: FreeBSD 12.0 vs. DragonFlyBSD 5.4 vs. TrueOS 18.12 vs. Linux On A Tyan EPYC Server, `https://www.phoronix.com/scan.php?page=article\&item=dfly-freebsd-tyanamd`.

[2] Michael Larabel: Windows Server 2019 vs. Linux vs. FreeBSD Gigabit & 10GbE Networking Performance (accessed 2019-02-04).

[3] Justinien Bouron, Chevalley, S., Lepers, B., Zwaenepoel, W., Gouicem, R., Lawall, J., Muller, G. and Sopena, J.: The Battle of the Schedulers: FreeBSD ULE vs. Linux CFS, *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, Boston, MA, USENIX Association, pp. 85–96 (online), available from ⟨https://www.usenix.org/conference/atc18/presentation/bouron⟩ (2018).

[4] CRIU Project: CRIU Main page (accessed 2019-01-30).

[5] A. Mirkin, A. Kuznetsov and K. Kolyshkin: Containers checkpointing and live migration, *In Proceedings of the 2008 Ottawa Linux Symposium*, Vol. 2, pp. 85–90 (2008).

[6] Google Inc.: Protocol Buffers, `https://developers.google.com/protocol-buffers/` (accessed January 8, 2020).

[7] 高川雄平，松原克弥：FreeBSD における Linux 互換コンテナを対象としたマイグレーション機構の実現，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（OS），No. 13, pp. 1–8 (2019).

[8] Yuhei Takagawa and Katsuya Matsubara: Yet Another Container Migration on FreeBSD, *AsiaBSDCon 2019 Proceedings*, pp. 97–102 (2019).

[9] Pingdom AB All Rights Reserved: Does Page Load Time Really Affect Bounce Rate?, `https://royal.pingdom.com/page-load-time-really-affect-bounce-rate/` (2018 (accessed January 8, 2020)).

[10] Hai Jiang and Chaudhary, V.: Process/thread migration and checkpointing in heterogeneous distributed systems, *System Sciences, 2004. Proceedings of the 37th Annual Hawaii International Conference on*, IEEE, pp. 10–pp (2004).

[11] Peter Smith and Hutchinson, N. C.: Heterogeneous process migration: the Tui system, *Software: Practice and Experience*, Vol. 28, No. 6, pp. 611–639 (1998).

[12] Barbalace, A., Lyerly, R., Jelesnianski, C., Carno, A., Chuang, H.-R., Legout, V. and Ravindran, B.: Breaking the Boundaries in Heterogeneous-ISA Datacenters, *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '17, New York, NY, USA, ACM Press, pp. 645–659 (2017).

[13] 中島佳宏，上村佳史，相田祥昭，佐藤三久：チェックポインティングとシステムコールエミュレーションを用いたヘテロ OS 環境のホスト間でのプロセスマイグレーションの実現，2006 年度 CS テクニカルレポート・システム開発型研究プロジェクト特集号，筑波大学 (2006).

[14] 上村佳史，中島佳宏，佐藤三久：Windows PC をグリッド環境で利用するための軽量 Linux バイナリ実行システム，情報処理学会論文誌コンピューティングシステム（ACS），Vol. 49, No. SIG2(ACS21), pp. 88–97 (2008).

[15] Ohrhallinger, K. P.: Virtual Private System for FreeBSD, *EuroBSDCon 2010* (2010).