IIR

Internet
Infrastructure
Review

Sep.2023
Vol. 60

定期観測レポート

ブロードバンドトラフィックレポート ～コロナ禍を経てトラフィックは安定増加傾向～

フォーカス・リサーチ(1)

システムソフトウェアの通信分野における2010年頃からの研究まとめ

フォーカス・リサーチ(2)

IIJとクラウドの変遷 ～30周年特別コンテンツ～

IIJ
Internet Initiative Japan

Internet Infrastructure Review

September 2023 Vol.60

エグゼクティブサマリ

去る7月、政権政党である自由民主党から、政府が保有するNTT株の売却の是非について本格的に検討を始めるとの発表がありました。NTTの株式の3分の1以上は政府が保有することがNTT法で義務付けられていることもあり、NTT法のあり方や日本の情報通信産業全体の国際競争力強化に関して、秋にも提言をまとめたいとしています。

また総務省は、情報通信審議会に対し、「市場環境の変化に対応した通信政策の在り方」について諮問し、来年夏を目途に答申を行うことを要望しています。それを受けて設置された通信政策特別委員会が、「2030年頃に目指すべき情報通信インフラの将来像及び政策の基本的な方向性」とそれに関連する事項について、広く提案・意見を募集しています。

電気通信が国営の事業として営まれていた我が国において、その事業を担ってきたNTTが民営化され、電気通信事業に競争が導入されたのが1985年。当時から現在に至るまでに、電気通信の利用形態、それを通じて提供されるサービスや便益、社会における位置づけや重要性、そこで活躍するプレーヤーなど、情報通信を取り巻く環境は様変わりしました。その要因のひとつがインターネットであることは間違いありません。

NTTは日本の情報通信インフラを支える重要なプレーヤーであることは言うまでもありませんが、日本のみならず世界の情報通信の在り方をも俯瞰した議論が行われるよう期待します。

「IIR」は、IIJで研究・開発している幅広い技術を紹介しており、日々のサービス運用から得られる各種データをまとめた「定期観測レポート」と、特定テーマを掘り下げた「フォーカス・リサーチ」から構成されます。

1章の定期観測レポートは、ブロードバンドトラフィックレポートです。このレポートでは毎年、IIJの固定ブロードバンドサービス及びモバイルサービスのトラフィックを分析しています。新型コロナウイルスの影響により、トラフィックが大きく伸びた2020年以降、目立った変化はないものの、トラフィック量もポート別使用量も着実に変化していることが、数字からも見て取れる結果となっています。

2章のフォーカス・リサーチでは、「システムソフトウェアの通信分野における2010年頃からの研究まとめ」と題して、システムソフトウェアの通信処理の高速化について解説します。ハードウェアの処理能力向上により、ハードウェアを制御するシステムソフトウェアの、特に通信に関する処理の効率性が重要になっています。そこで、システムソフトウェアの通信処理の概要と効率化の研究について説明し、最後にIIJ技術研究所の取り組みを紹介します。

3章のフォーカス・リサーチは、IIJの創業30周年特別コンテンツとして、バックボーンネットワーク、DNSに続いて、IIJのクラウドサービスの変遷を取り上げます。IIJでは創業以来、インターネット接続に付随する様々なサービス提供のためにサービスホストを運用してきました。また、クラウドやIaaSという言葉が一般的になる前から、コンピューティングやストレージのリソースをサービスとして提供してきました。それらに利用されるサーバ、ストレージ、ネットワークが統合されたインフラは、時代の要請によって進化しており、その変遷を紹介します。

IIJは、このような活動を通してインターネットの安定性を維持しながら、日々、改善・発展させていく努力を行っています。今後も企業活動のインフラとして最大限にご活用いただけるよう、様々なサービスやソリューションを提供し続けてまいります。

島上 純一（しまがみ じゅんいち）

IIJ 常務取締役 CTO。インターネットに魅かれて、1996年9月にIIJ入社。IIJが主導したアジア域内ネットワークA-BoneやIIJのバックボーンネットワークの設計、構築に従事した後、IIJのネットワークサービスを統括。2015年よりCTOとしてネットワーク、クラウド、セキュリティなど技術全般を統括。2017年4月にテレコムサービス協会MVNO委員会の委員長に就任し、2023年5月に退任。2021年6月より同協会の副会長に就任。

ブロードバンドトラフィックレポート ～コロナ禍を経てトラフィックは安定増加傾向～

1.1 概要

このレポートでは、毎年IIJが運用しているブロードバンド接続サービスのトラフィックを分析して、その結果を報告しています[*1*2*3*4*5]。今回も、利用者の1日のトラフィック量やポート別使用量などを基に、この1年間のトラフィック傾向の変化を報告します。コロナ禍を経て、トラフィックは安定した成長が続いていて、今のところその傾向に目立った変化は見られません。

図-1は、IIJの固定ブロードバンドサービス及びモバイルサービス全体について、月ごとの平均トラフィック量の推移を示したグラフです。トラフィックのIN/OUTはISPから見た方向を表し、INは利用者からのアップロード、OUTは利用者へのダウンロードとなります。トラフィック量の数値は開示できないため、新型コロナ感染拡大前の2020年1月の両サービスのOUTの値を1として正規化しています。

この1年のブロードバンドトラフィック量は、INは11%の増加、OUTは18%の増加となっています。1年前はそれぞれ13%と17%でした。

ブロードバンドに関しては、IPv6・IPoEのトラフィック量も含めて示しています。IIJのブロードバンドにおけるIPv6は、IPoE方式とPPPoE方式があります。2023年6月時点で、IPoEのブロードバンドトラフィック量の全体に占める割合は、INで42%、OUTで44%と、昨年同月よりそれぞれ3ポイント増えていて、全体の4割強がIPoEとなっています。コロナ禍で顕著になったPPPoEの輻輳を避けて、IPoEへ移行する利用者が増えていて、IPoEの利用拡大が続いています。

モバイルサービスは2021年以降増加傾向が続いています。モバイルはこの1年でINは27%、OUTは31%の増加となっています。

次に、この1年の平日の時間別ブロードバンドトラフィック量の推移を見ていきます。図-2に、昨年5月末の週から約4ヵ月おきに4つの週を選んで、各週の月曜から金曜の各時間の平均トラフィック量を示します。ここ数年学校が休みの時期は平日昼間のトラフィック量が増えるようになったので、学期途中の週を選んでいます。ここでのトラフィック量はPPPoEとIPoEの合計値です。下側の波線はそれぞれの週のアップロード量です

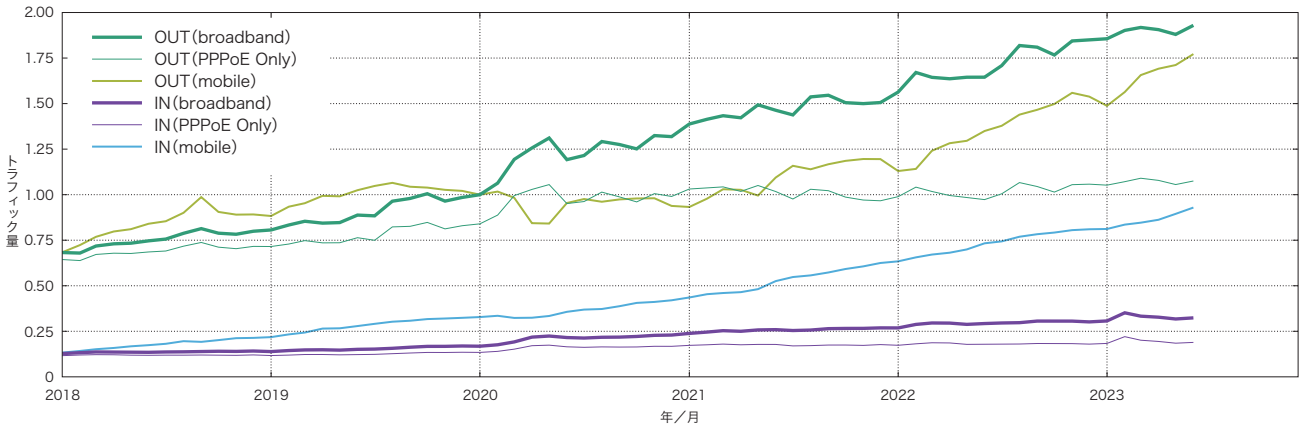

図-1 ブロードバンド及びモバイルの月間トラフィック量の推移

＊1　長健二朗. ブロードバンドトラフィックレポート：コロナ禍3年目のトラフィックは小康状態. Internet Infrastructure Review. vol.56. pp4-11. September 2022.
＊2　長健二朗. ブロードバンドトラフィックレポート：2年目に入ったコロナ禍の影響. Internet Infrastructure Review. vol.52. pp4-11. September 2021.
＊3　長健二朗. ブロードバンドトラフィックレポート：新型コロナウイルス感染拡大の影響. Internet Infrastructure Review. vol.48. pp4-9. September 2020.
＊4　長健二朗. ブロードバンドトラフィックレポート：トラフィック量は緩やかな伸びが継続. Internet Infrastructure Review. vol.44. pp4-9. September 2019.
＊5　長健二朗. ブロードバンドトラフィックレポート：ダウンロードの増加率は2年連続で減少. Internet Infrastructure Review. vol.40. pp4-9. September 2018.

が、今回もダウンロード量に注目すると、どの時間帯においても着実にトラフィック量が増えてきていることが分かります。昼間の増分と夜のピーク時間の増分は量的には同じぐらいなので、増加割合では昼間の方が大きくなっています。

1.2 データについて

今回も前回までと同様に、ブロードバンドに関しては、個人及び法人向けのブロードバンド接続サービスについて、ファイバーとDSLによるブロードバンド顧客を収容するルータで、Sampled NetFlowにより収集した調査データを利用しています。モバイルに関しては、個人及び法人向けのモバイルサービスについて、使用量にはアクセスゲートウェイの課金用情報を、使用ポートにはサービス収容ルータでのSampled NetFlowデータを利用しています。

トラフィックは平日と休日で傾向が異なるため、1週間分のトラフィックを解析します。今回は、2023年5月29日～6月4日の1週間分のデータを解析して、前回解析した2022年5月30日～6月5日の1週間分と比較します。

ブロードバンドの集計は契約ごとに行い、一方モバイルでは複数電話番号の契約があるので電話番号ごとの集計となっています。ブロードバンド各利用者の使用量は、利用者に割り当てられたIPアドレスと、観測されたIPアドレスを照合して求めています。また、NetFlowではパケットをサンプリングして統計情報を取得しています。サンプリングレートは、ルータの性能や負荷を考慮して、1/8192程度に設定されています。観測された使用量に、サンプリングレートの逆数を掛けることで全体の使用量を推定しています。なお、IPoEトラフィックはインターネットマルチフィード社のtransixサービスを利用していて詳細なデータが取得できていないため、ポート別解析の対象にはなっていません。

1.3 利用者の1日の使用量

まずは、ブロードバンド及びモバイル利用者の1日の利用量をいくつかの切り口から見ていきます。ここでの1日の利用量は各利用者の1週間分のデータの1日平均です。

2019年のレポートから、利用者の1日の使用量は個人向けサービス利用者のデータのみを使っています。これは、利用形態が多様な法人向けサービスを含めると分布の歪みが大きくなってしまうため、全体の利用傾向を掴むには個人向けサービス分だけを対象にした方が、より一般性があり分かりやすいと判断したからです。なお、次節のポート別使用量の解析では区別が難しいため法人向けも含めたデータを使っています。また、2021年からブロードバンドにはIPoEの利用者のデータも加えています。前回のレポートでは、PPPoE分とIPoE分を分けて示しましたが、今回から両者を統合してブロードバンドとして示しています[*6]。

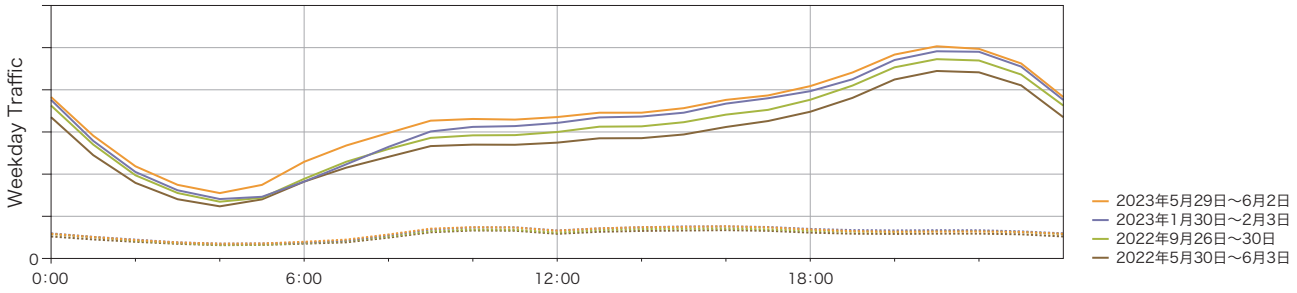

図-2 この1年の平日時間別ブロードバンドトラフィック量の推移

＊6　利用者がPPPoEとIPoEの両方を使っている場合はそれぞれ別の利用者として扱われています。

図-3及び図-4は、ブロードバンドとモバイル利用者の1日の平均利用量の分布（確率密度関数）を示します。アップロード（IN）とダウンロード（OUT）に分け、利用者のトラフィック量をX軸に、その出現確率をY軸に示していて、2022年と2023年を比較しています。X軸はログスケールで、10KB（10^4）から100GB（10^{11}）の範囲を示しています。一部の利用者はグラフの範囲外にありますが、おおむね100GB（10^{11}）までの範囲に分布しています。

図中のINとOUTの各分布は、片対数グラフ上で正規分布となる対数正規分布に近い形をしています。これはリニアなグラフで見ると、左端近くにピークがあり右へなだらかに減少するいわゆるロングテールな分布です。OUTの分布はINの分布より右にずれていて、ダウンロード量がアップロード量より、ひと桁以上大きくなっています。

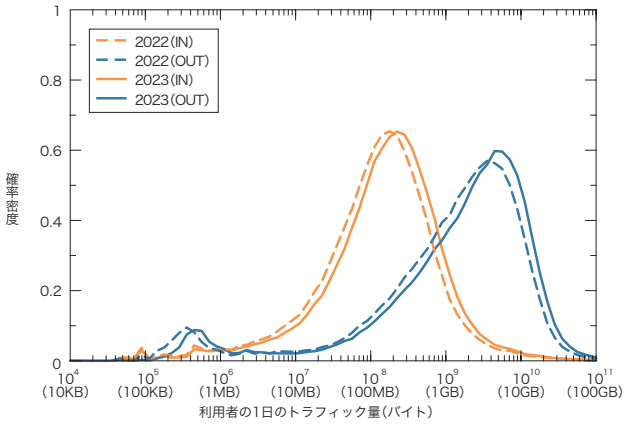

図-3 ブロードバンド利用者の1日のトラフィック量分布 2022年と2023年の比較

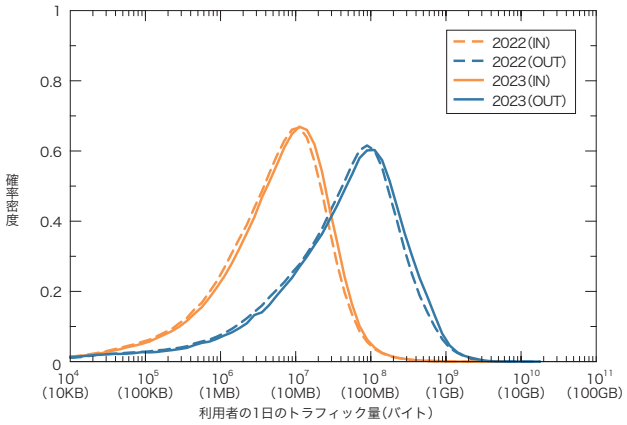

図-4 モバイル利用者の1日のトラフィック量分布 2022年と2023年の比較

まず、図-3のブロードバンドの分布を見ます。2022年と2023年を比較すると、INとOUT共に分布全体がわずかながら右側に移動していて、全体的に利用量が増えていることが分かります。図-4のモバイルの場合も、分布の山が昨年に比べ少し右に移動していて、全体の利用量が増えていることが分かります。モバイルの利用量は、ブロードバンドに比べて大幅に少なく、また、使用量に制限があるため、分布右側のヘビーユーザの割合が少なくなっています。極端なヘビーユーザも存在しません。外出時のみの利用や、使用量の制限のため、各利用者の日ごとの利用量のばらつきはブロードバンドより大きくなります。

表-1は、ブロードバンド利用者の1日のトラフィック量の平均値と中間値、分布の山の頂点にある最頻出値の推移を示します。分布の山に対して頂点が少しずれている場合は、最頻出値は分布の山の中央に来るように補正しています。分布の最頻出値を2022年と2023年で比較すると、INでは178MBから224MBに、OUTでは3981MBから5012MBに増えています。伸び率で見ると、INで1.26倍、OUTも1.26倍となっています。一方、平均値はグラフ右側のヘビーユーザの使用量に左右されるため、2023年には、INの平均は804MB、OUTの平均は

	IN (MB/day)			OUT（MB/day）		
年	平均値	中間値	最頻出値	平均値	中間値	最頻出値
2007	436	5	5	718	59	56
2008	490	6	6	807	75	79
2009	561	6	6	973	91	100
2010	442	7	7	878	111	126
2011	398	9	9	931	144	200
2012	364	11	13	945	176	251
2013	320	13	16	928	208	355
2014	348	21	28	1124	311	501
2015	351	32	45	1399	443	708
2016	361	48	63	1808	726	1000
2017	391	63	79	2285	900	1259
2018	428	66	79	2664	1083	1585
2019	479	75	89	2986	1187	1995
2020	609	122	158	3810	1638	3162
2021	714	143	200	4432	2004	3981
2022	727	142	178	4610	2010	3981
2023	804	166	224	5456	2369	5012

表-1 ブロードバンド個人利用者の1日のトラフィック量の平均値と最頻出値の推移

5456MBと、最頻出値より大きな値になります。2022年には、それぞれ727MBと4610MBでした。なお、前述のように2020年分まではPPPoE利用者のみの数字で、2021年以降はPPPoE利用者とIPoE利用者を統合した数字になっています。

表-2はモバイルの値の推移で、2023年の最頻出値はINで11MB、OUTで100MB、平均値はINで14MB、OUTで129MBです。2022年の最頻出値はINで10MB、OUTで89MB、平均値はINで13MB、OUTで114MBでした。

図-5及び図-6では、利用者5,000人をランダムに抽出し、利用者ごとのIN/OUT使用量をプロットしています。X軸はOUT（ダウンロード量）、Y軸はIN（アップロード量）で、共にログスケールです。利用者のIN/OUTが同量であれば対角線上にプロットされます。

対角線の下側に対角線に沿って広がるクラスタは、ダウンロード量がひと桁多い一般的なユーザです。各利用者の使用量やIN/OUT比率にも大きなばらつきがあり、多様な利用形態が存在することがうかがえます。モバイルでも、OUTがひと桁多い傾向は同じですが、ブロードバンドに比べて利用量は大幅に少なくなっています。ブロードバンド、モバイル共に、2022年との違いはほとんど分かりません。

利用者間のトラフィック使用量の偏りを見ると、使用量には大きな偏りがあり、結果として全体は一部利用者のトラフィックで占められています。例えば、ブロードバンド上位10%の利用者がOUTの49%、INの76%を占めています。更に、上位1%の利用者がOUTの16%、INの49%を占めています。モバイルでは上位10%の利用者がOUTの49%、INの47%を占めていて、上位1%の利用者がOUTの15%、INの15%を占めています。

	IN（MB/day）			OUT（MB/day）		
年	平均値	中間値	最頻出値	平均値	中間値	最頻出値
2015	6.2	3.2	4.5	49.2	23.5	44.7
2016	7.6	4.1	7.1	66.5	32.7	63.1
2017	9.3	4.9	7.9	79.9	41.2	79.4
2018	10.5	5.4	8.9	83.8	44.3	79.4
2019	11.2	5.9	8.9	84.9	46.4	79.4
2020	10.4	4.5	7.1	79.4	35.1	63.1
2021	9.9	4.7	7.9	85.9	37.9	70.8
2022	12.8	6.0	10.0	113.7	49.2	89.1
2023	14.1	6.8	11.2	129.2	56.0	100.0

表-2 モバイル個人利用者の1日のトラフィック量の平均値と最頻出値

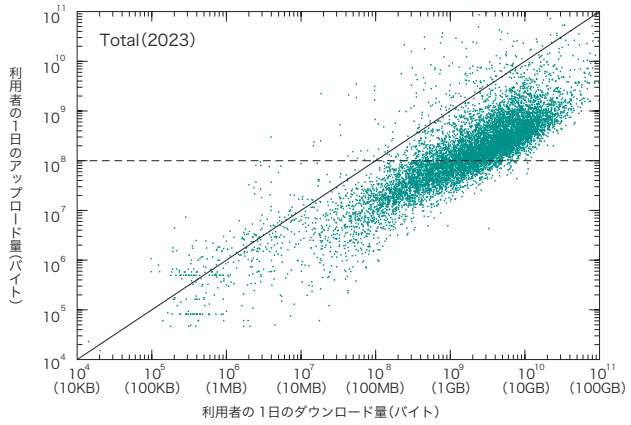

図-5 ブロードバンド利用者ごとのIN/OUT使用量

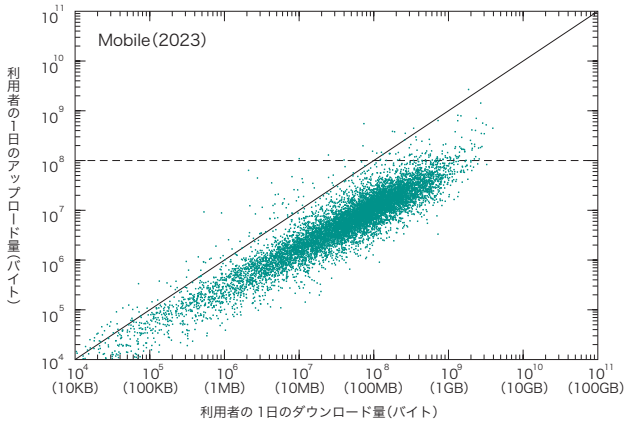

図-6 モバイル利用者ごとのIN/OUT使用量

1.4 ポート別使用量

次に、トラフィックの内訳をポート別の使用量から見ていきます。最近では、ポート番号からアプリケーションを特定することは困難です。P2P系アプリケーションには、双方が動的ポートを使うものが多く、また、多くのクライアント・サーバ型アプリケーションがファイアウォールを回避するため、HTTPが使う80番ポートなどを利用します。大まかに分けると、双方が1024番以上の動的ポートを使っていればP2P系のアプリケーションの可能性が高く、片方が1024番未満のいわゆるウェルノウンポートを使っていれば、クライアント・サーバ型のアプリケーションの可能性が高いと言えます。そこで、TCPとUDPで、ソースとデスティネーションのポート番号の小さい方を取り、ポート番号別の使用量を見てみます。

表-3はブロードバンド利用者のポート使用割合について過去5年間の推移を示します。2023年の全体トラフィックの71%はTCPで、昨年から1ポイント減りました。HTTPSのTCP443番ポートの割合は、57%で前回から1ポイント増加しました。HTTPのTCP80番ポートの割合は9%から7%に減っています。QUICプロトコルで使われるUDP443番ポートは、18%で2ポイント増えました。

TCPの動的ポートは、わずかに減少して6%を切りました。動的ポートでの個別のポート番号の割合はわずかで、最大の31000番でも1.1%となっています。

表-4はモバイル利用者のポート使用割合です。全体的にはブロードバンドの数字に近い値となっています。これは、スマートフォンでもPCと同様のアプリケーションを使うようになってきたことに加え、ブロードバンドにおけるスマートフォンの利用割合が増えているからだと考えられます。

ブロードバンドのポート別データは、PPPoEだけでIPoEを含まないので、固定ブロードバンド全体の傾向を表しているとは限りません。モバイルでのIPv4とIPv6の違いを見ると、IPv6ではTCPもUDPも443番ポートの割合がより大きくなっていて、IPoEでも同様の傾向があると考えられます。

year	2019	2020	2021	2022	2023
protocol port	(%)	(%)	(%)	(%)	(%)
TCP	**81.2**	**77.2**	**71.9**	**71.6**	**70.5**
(< 1024)	73.3	70.5	65.8	65.4	64.8
443(https)	51.9	52.4	53.5	55.7	56.9
80(http)	20.4	17.2	11.6	8.9	7.2
183	0.0	0.0	0.1	0.2	0.2
993(imaps)	0.3	0.2	0.1	0.1	0.1
22(ssh)	0.2	0.2	0.2	0.1	0.1
(>= 1024)	7.9	6.7	6.1	6.2	5.7
31000	0.2	0.4	0.6	0.9	1.1
8080	0.5	0.4	0.4	0.3	0.4
1935(rtmp)	0.3	0.4	0.2	0.2	0.2
UDP	**14.1**	**19.4**	**24.5**	**24.3**	**25.4**
443(https)	7.8	10.5	15.9	16.3	18.2
4500(nat-t)	0.3	0.6	0.8	0.8	1.0
8801	0.0	1.1	0.9	0.6	0.4
ESP	**4.4**	**3.2**	**3.3**	**3.8**	**3.8**
GRE	**0.1**	**0.1**	**0.2**	**0.2**	**0.1**
IP-ENCAP	**0.2**	**0.1**	**0.1**	**0.1**	**0.1**
ICMP	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**

表-3 ブロードバンド利用者のポート別使用量

year	2019	2020	2021	2022	2023
protocol port	(%)	(%)	(%)	(%)	(%)
TCP	**76.9**	**75.5**	**70.3**	**71.6**	**71.0**
443(https)	55.6	50.7	44.4	42.3	42.1
80(http)	10.3	7.4	5.0	4.1	3.5
993(imaps)	0.3	0.2	0.2	0.1	0.1
1935(rtmp)	0.1	0.1	0.1	0.1	0.2
UDP	**17.3**	**18.0**	**23.8**	**24.4**	**26.5**
443(https)	8.3	9.3	16.3	17.9	20.9
4500(nat-t)	3.0	1.8	3.7	2.7	2.5
8801	0.0	1.4	0.7	0.3	0.2
51820	0.0	0.0	0.0	0.1	0.2
53(dns)	0.1	0.1	0.2	0.2	0.2
ESP	**5.8**	**6.4**	**5.8**	**3.9**	**2.4**
GRE	**0.0**	**0.1**	**0.1**	**0.0**	**0.0**
ICMP	**0.0**	**0.0**	**0.0**	**0.0**	**0.1**

表-4 モバイル利用者のポート別使用量

図-7は、ブロードバンド全体トラフィックにおける主要ポート利用の週間推移を、2022年と2023年で比較したものです。TCPポートの80番・443番・1024番以上の動的ポート、UDPポート443番の4つに分けてそれぞれの推移を示しています。グラフでは、ピーク時の総トラフィック量を1として正規化して表しています。全体のピークは19時～23時頃です。2022年と比較して、全体では大きな変化はありませんが、UDPポート443番が少し増えていて、また、図-2のところでも触れたように昼間のトラフィック割合が少し増えています。

図-8のモバイルでは、トラフィックの大半を占めるTCP80番ポートと443番ポート、UDP443番ポートについて推移を示します。2022年と比べると、ブロードバンドと同様にUDPポート443番が少し増えています。また、昼休みのピークが相対的に少し低くなりその分横に広がっていることが確認できます。ブロードバンドに比べると、平日には、朝の通勤時間、昼休み、夕方と3つのピークがあるなど利用時間の違いがあります。

1.5 まとめ

コロナ禍を経てようやく普通の生活が戻ってきましたが、この間にインターネットの利用が生活の中に浸透して、生活インフラとして欠かせないものになりました。ビデオ会議やリモートワークが定着し、子供達も普通に家で動画視聴するようになりました。また、昨年のサッカーワールドカップや今年3月の野球のWBCで日本代表チームが活躍し、ネット中継によるスポーツ視聴も裾野が広がりました。数年前に比べると、SNSにおける動画割合も格段に増えています。一方で、ここ数年の利用者あたりの利用量を見ると、ブロードバンドもモバイルも、新型コロナ感染が始まった2020年は大きく増えましたが、その後は比較的安定した増加にとどまっています。全体のトラフィック量についても堅調な増加傾向が続いています。その要因として、2021年以降は在宅率が下がってきて利用時間が減っていること、これといった新しいサービスや使い方が出てきていないこと、ビデオ圧縮効率向上などの技術進化もあってビデオコンテンツの流通量ほどにはトラフィック量が増加していないことが挙げられます。

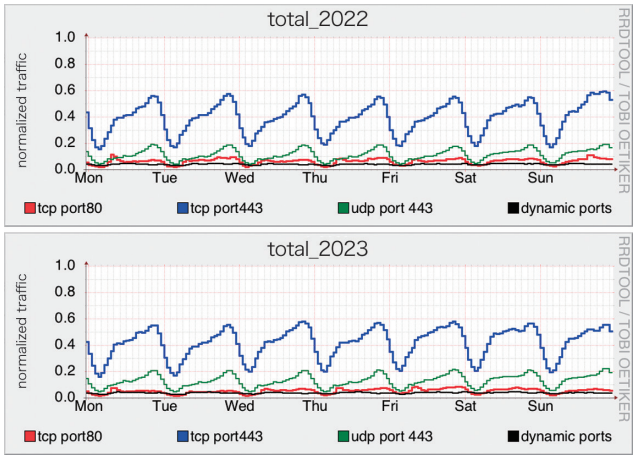

図-7 ブロードバンド利用者のポート利用の週間推移
2022年(上)と2023年(下)

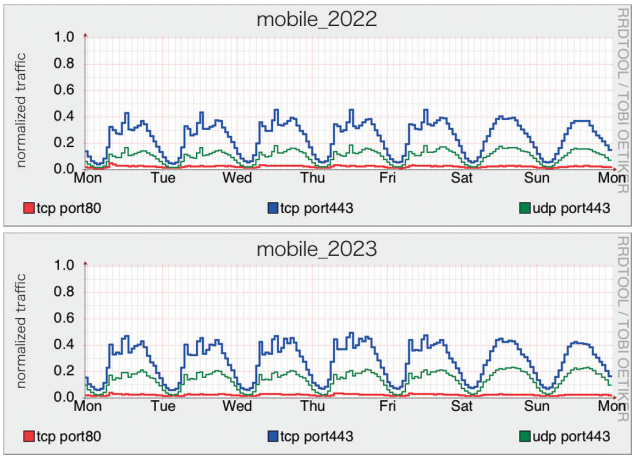

図-8 モバイル利用者のポート利用の週間推移
2022年(上)と2023年(下)

執筆者:
長 健二朗（ちょう けんじろう）
IIJ 技術研究所所長。

システムソフトウェアの通信分野における2010年頃からの研究まとめ

2.1 概要

2010年代初頭より10Gbpsを超えるような速度のネットワークインタフェースカード(NIC)が一般化し、データセンターなどで広く利用されるようになりました。NICの性能向上により、それらハードウェアを制御するシステムソフトウェアと呼ばれるようなソフトウェアの、特に通信に関する処理の効率の重要性が高まり、研究コミュニティにおいては性能の改善を目指した取り組みが数多く行われてきました。

本稿では、まず2.2でシステムソフトウェアにおける通信処理の挙動について触れ、次に2.3では過去の研究がそれらをどのように高速化しようと取り組んできたかについてまとめます。それらをふまえて、2.4で、IIJ技術研究所が近年行っている取り組みについて紹介します。

注意として、2.4で述べている取り組みは研究段階のものであり、IIJサービス・インフラに組み込まれているものではないことを、最初に申し上げておきます。

2.2 通信に関連したプログラムの主な挙動

まず、2.2.1で汎用OS内での通信の関わりについて、次に2.2.2で、データセンターで一般的に利用される仮想マシンが通信を行う場合の処理について述べます。

2.2.1 汎用OS内での通信関連の処理

図-1を元に汎用OS環境での通信処理について見ていきます。

■ 基本的なシステム構成

大まかに、図-1中、上から

(1) ユーザ空間で動作するアプリケーションプログラム
(2) ネットワークスタックとデバイスドライバを実装しているカーネル
(3) パケットの送受信を行うNIC

の3つを主な構成要素とします。

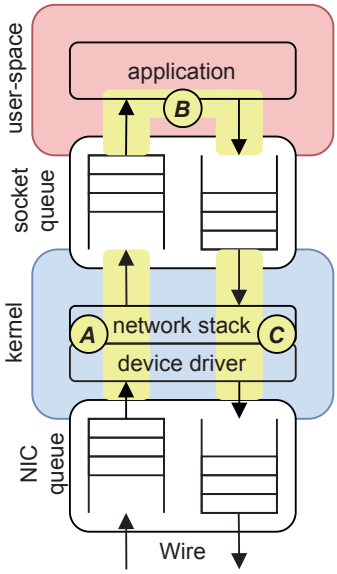

図-1 汎用OS環境での通信関連ソフトウェアの構成

■ 典型的なループ

サーバと呼ばれるような、クライアントのリクエストに対して応答を返すようなプログラムは典型的にA:カーネル空間での受信処理、B:ユーザ空間でのアプリケーション固有の処理、C:カーネル空間での送信処理のループになります。

また、受信と送信処理を実行コンテキストごとにまとめると図-2、図-3のようになります。

■ A:カーネル空間での受信処理

□ STEP1 ハードウェア(NIC)からの通知

NICが新しいパケットを受信すると、これをソフトウェアへ通知するために、CPUに対してハードウェア割り込みを発生させます。これにより、CPUでそれまで実行されていたプログラムは中断され、カーネルが事前に設定していたハードウェア割り込みハンドラへ処理が切り替わります。ハードウェア割り込みハンドラは、実装依存ですが、ある程度一般的な処理として、受信パケットを処理するためのカーネルスレッドを起動します。

□ STEP2 受信パケットの処理

上記ステップ1で起動された受信パケット処理用のカーネルスレッドが、受信パケットのヘッダを読み取り、対応する処理を行います。例えば受信したのがTCPパケットであれば、対応するコネクションについてのTCPのACK番号確認などの状態処理を行い、そのコネクションに紐づけられているソケットのキューに受信パケットを追加します。

□ STEP3 ユーザ空間プロセスへの通知

上記ステップ2において、特定のソケットのキューにデータや新しいコネクションを追加した場合、そのソケットに紐づいているユーザ空間プロセス・スレッドが、新規入力に備えてselectやpoll、epoll_waitもしくはread系列のシステムコール(例:read、recvmsgなど)で待機している(ブロックされた)状態であれば、それを起動(ブロックを解除)します。

■ B:ユーザ空間でのプログラムの処理

□ STEP1 入力イベントの待機と検知

ユーザ空間で動作するサーバプログラムの多くは、select や poll、epoll_wait または read 系列のシステムコールで監視対象のソケット(ファイルデスクリプタ)への新規入力を、実行を停止(ブロック)された状態で待機します。監視対象のソケットに入力があれば、前述Aのカーネル空間での受信処理ステップ3によって、待機状態(ブロック)が解除されます。また、select、poll、epoll_wait のようなシステムコールはブロックの解除・リターンに併せて、どのソケット(ファイルデスクリプタ)について入力イベントがあったかの情報がカーネルから渡されます。

□ STEP2 カーネルからユーザ空間へのデータの受け渡し

ユーザ空間プログラムは、上記ステップ1 で得られた入力イベントが検知されたソケット(ファイルデスクリプタ)へ向けて、

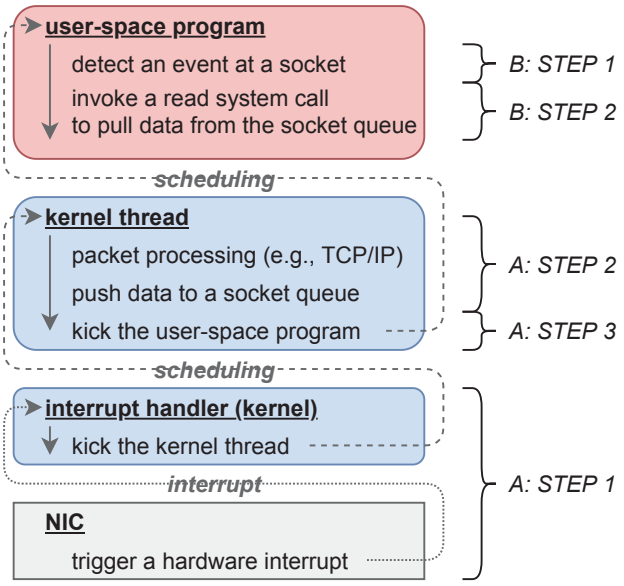

図-2 汎用OSにおける受信処理の流れ:
右側吹き出しは対応する章とステップを示します

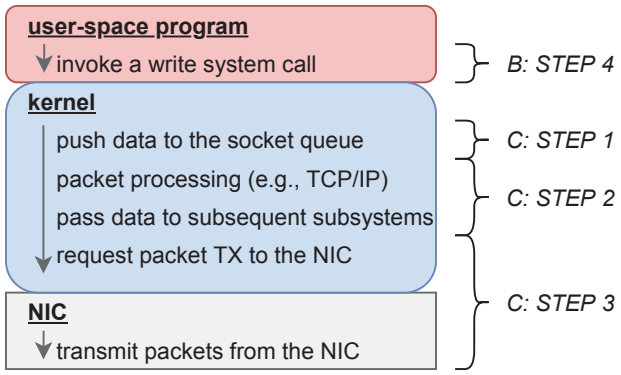

図-3 汎用OSにおける送信処理の流れ:
右側吹き出しは対応する章とステップを示します

read 系列のシステムコールを発行し、前述Aのカーネル空間での受信処理ステップ2によって追加された、ソケットの受信キューに入っているデータを、カーネルからユーザ空間へコピーしてもらいます。

□ STEP3 アプリケーション固有の処理

上記ステップ2にて受信したデータに対してアプリケーションプログラム固有の処理を行います。例えば、Web サーバであれば、受信したデータをパースしリクエスト内容を判別した上で、そのリクエストに対応する応答のデータを生成します。

□ STEP4 カーネルへデータの送信をリクエスト

ソケット（ファイルデスクリプタ）に対してwrite 系列のシステムコール（例：write、sendmsg など）を発行し、上の処理ステップ3によって用意されたデータの送信をカーネルに対してリクエストします。

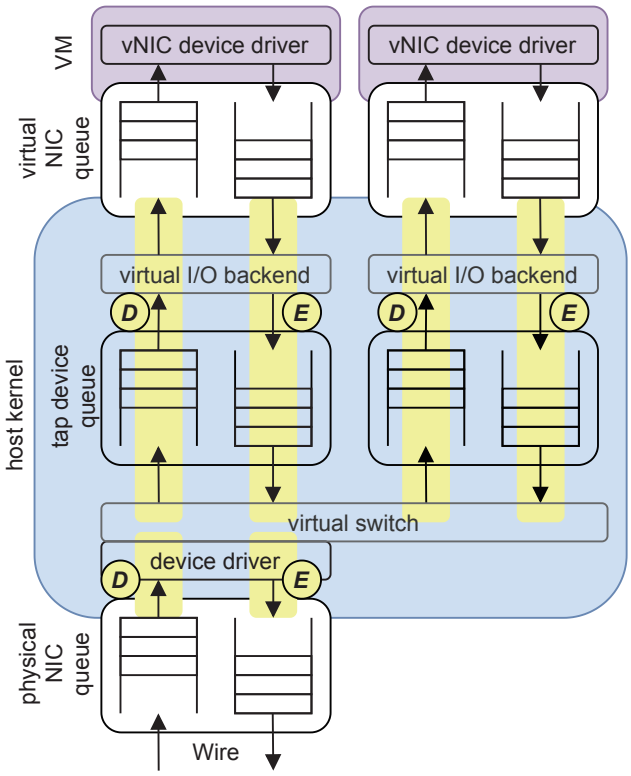

図-4 仮想マシンの通信機構

■ C：カーネル空間での送信処理

□ STEP1 ユーザ空間からカーネルへのデータの受け渡し

Bのステップ4で呼び出されたwrite 系列のシステムコールによって、処理がカーネル空間に切り替わります。その後、カーネルはユーザ空間プログラムが用意したデータをカーネル空間にコピーし、それをユーザ空間プログラムが指定したソケットに紐づいている送信キューに追加します。

□ STEP2 プロトコルに応じたヘッダ処理

ステップ1と同じカーネルのコンテキストで、必要であれば転送対象のデータに対してパケットヘッダ付与の処理などを行います。その後、パケットとして準備が完了し、かつ、カーネル内部のサブシステムがそのデータを転送して良いと思ったタイミングで、次のサブシステムに引き渡されます[*1]。

□ STEP3 NICからデータの転送

ヘッダが付与されたデータは最終的にNICのデバイスドライバに引き渡され、デバイスドライバはNICに対してそのデータの送信をリクエストします。

2.2.2 仮想マシンのネットワークI/O

図-4を元に、仮想マシン環境における通信処理の内容について見ていきます。

■ 基本的なシステム構成

構成要素は大まかに図-2上から、

(1) 仮想マシン
(2) 仮想マシンに割り当てられる仮想NIC
(3) 仮想I/Oを行うバックエンド、tapデバイス、仮想スイッチ、デバイスドライバを実装するホストカーネル
(4) 物理NIC

＊1 TCPソケット（ファイルデスクリプタ）に対するwriteシステムコール呼び出し時には、ユーザ空間プログラムが指定したデータがNICから発信されるまでに至らない場合があります。例えば、その理由として、TCP実装の輻輳制御や、性能向上のために特定サイズまで送信バッファが埋まるのを待つNagleアルゴリズム、また、NICの帯域制御を担当するqdiscのようなサブシステムがデータの送信を遅延させる場合が考えられます。

の4点とします。仮想マシンの通信機能の実装形式は何通りか存在しますが、今回はホストカーネル内部のスレッドが仮想I/Oのバックエンドとして機能するLinuxのvhost-netのような形式を想定しています。

■ 典型的なループ

仮想マシン上で、前節で述べたような、受信したリクエストに対して返信するといったプログラムが動作している場合、仮想マシン環境では典型的に、D:ホスト内での仮想マシン通信の受信処理、前述A〜C:仮想マシン上で、汎用OSによる通信関連の処理[*2]、E:ホスト内で仮想マシン通信の送信処理のループが実行されます。

受信と送信処理を実行コンテキストごとにまとめると次の図-5、図-6のようになります。

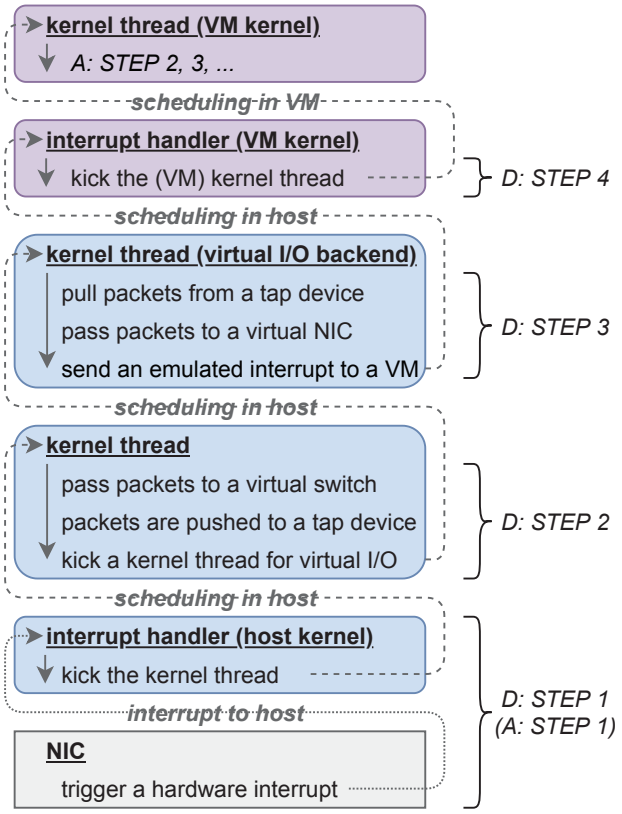

図-5 仮想マシン環境での受信処理の流れ:
右側吹き出しは対応する章とステップを示します

■ D:仮想マシン通信の受信処理

□ STEP1 ハードウェア(物理NIC)からの通知

物理NICにパケットが到着したときの最初の処理は、前述Aのステップ1と同じで、ホストカーネル内でハードウェア割り込みハンドラが起動し、受信パケットを処理するためのカーネルスレッドが起動されます。

□ STEP2 受信パケットを仮想スイッチに渡す

前述Aのステップ2と同じく、ステップ1で起動されたカーネルスレッドが受信パケットを処理しますが、その処理内容は、前述Aの時とは異なります。まず、受信パケットは仮想スイッチへ渡されます。仮想スイッチは受信パケットのEthernetヘッダを読み取り、そのパケットの適切な宛先のインタフェースを見つけ、そのインタフェースの受信キューにパケットを追加します。ここで、宛先インタフェースがtapデバイスであった場合、そのtapデバイスに紐づけられた、仮想I/Oを担当するバックエンドのカーネルスレッドを起動します。

□ STEP3 仮想NICへ受信データを渡す

上述ステップ2で起動された仮想I/Oを担当するバックエンドのカーネルスレッドが、tapデバイスからデータを取り出し、仮想NICの受信キューに渡します。その後、仮想NICでパケットが受信されたことを仮想マシンに通知するために、仮想マシンに対して割り込みを送ります。

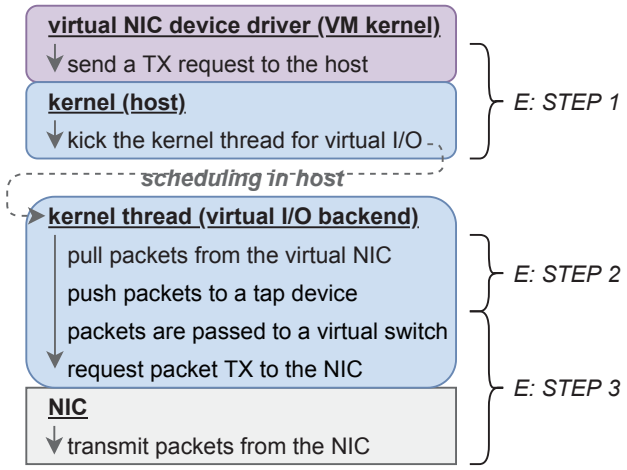

図-6 仮想マシン環境での送信処理の流れ:
右側吹き出しは対応する章とステップを示します

＊2　仮想マシン上で汎用OSを実行している場合、仮想マシン内の通信プログラムの挙動は基本的に2.2.1で述べられたものと同じになります。

□ **STEP4 仮想マシン内で受信処理**

仮想マシンはステップ3でホストから送られた割り込みを受け取り、仮想マシン内のカーネルが事前に設定していた割り込みハンドラへ処理が切り替わります。以後、仮想マシン内での処理は前述Aのステップ1以降と同じになります。

■ **E:仮想マシン通信の送信処理**

□ **STEP1 仮想マシンからのデータ送信リクエスト**

仮想マシンが前述Cのステップ3を実行して仮想NICデバイスドライバを通して、送信したいデータを仮想NICの送信キューへ追加した後、仮想NICに対してパケット送信をリクエストすると、実行のコンテキストが仮想マシンからホストのカーネルへ切り替わり、その中で仮想I/Oを担当するカーネルスレッドが起動されます。

□ **STEP2 仮想NICからtapデバイスへデータの受け渡し**

上記ステップ1で起動された仮想I/Oを担当するカーネルスレッドは、仮想NICの送信キューに入っているデータを取り出し、tapデバイスに引き渡します。

□ **STEP3 tapデバイスから仮想スイッチへデータの受け渡し**

上述ステップ2の後、tapデバイスを通じてパケットは仮想スイッチに渡され、パケットの宛先に応じたインタフェースから送信されます。

2.3 研究コミュニティでの取り組み

ここでは、前節で見てきたようなワークロードを高速化するために研究コミュニティが行ってきた取り組みについてまとめます。

2.3.1 システムコール呼び出しコストの削減

NICのI/Oが高速であればあるほど、CPUリソースが追いつく限り前節Bにて前述したようなユーザ空間でのプログラムの処理のループが頻繁に繰り返されることになります。ここでポイントなのが、システムコールはユーザ空間とカーネル空間でコンテキストを切り替える処理が含まれるため、CPUにとって負荷の高い操作であるという点です。具体的に、前節Bで触れたワークロードではステップ1でselect、poll、epoll_waitシステムコール、ステップ2でread系列のシステムコール、ステップ4でwrite系列のシステムコールというようにシステムコールを頻繁に繰り返し呼び出すため、全体のプログラム実行時間の中で、コンテキスト切り替えのために費やされる時間の割合が大きくなってしまうという課題があります。

■ **複数のシステムコールをまとめて発行**

2010年にFlexSC[*3]という、複数のカーネルに対する処理のリクエストをひとまとめにして(バッチして)カーネルに対して送ることができるシステムが提案されました。仕組みとしては、ユーザ空間とカーネル空間で共有メモリを作成し、ユーザ空間から実行したいシステムコールとその引数を共有メモリに書き込むと、非同期にカーネル内のスレッドがそのシステムコールを実行して結果を返す、というようになっています。この仕組みにより、システムコールごとにコンテキスト切り替えを発生させなくて済むようになります。このようなアプローチ[*4]は、詳細の実装方法は異なりますが、2.3.3で述べるようなネットワークスタック実装の最適化において広く採用されていきます。

---

*3 Livio Soares and Michael Stumm. 2010. FlexSC: Flexible System Call Scheduling with Exception-Less System Calls. In 9th USENIX Symposium on Operating Systems Design and Implementation (OSDI 10). (https://www.usenix.org/conference/osdi10/flexsc-flexible-system-call-scheduling-exception-less-system-calls)。

*4 リクエストを複数をまとめて発行することでドメイン間のコンテキスト切り替えを削減するというアイデアはFlexSC[*3]以前にはコンパイラ[*5]やハイパーバイザ[*6]においてmulti-callと呼ばれるような仕組みで模索されていたそうです。

*5 Mohan Rajagopalan, Saumya K. Debray, Matti A. Hiltunen, and Richard D. Schlichting. 2003. Cassyopia: Compiler Assisted System Optimization. In Proceedings of the 9th Conference on Hot Topics in Operating Systems - Volume 9 (HotOS '03), 18.

*6 Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. 2003. Xen and the Art of Virtualization. In Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles (SOSP '03), 164–177. (https://doi.org/10.1145/945445.945462)。

■ アプリとカーネルの境界をなくす

また、アプリケーションプログラムとOSカーネルを同じアドレス空間で動かす、つまり、アプリケーションとカーネルの境界を取り払ってしまうことで、アプリケーションプログラムからカーネルに実装されている機能を、システムコールではなく、通常の関数呼び出しを通して利用できるようにする、というアプローチも存在します。このような構成は、アプリケーションとカーネルを含むすべてのプログラムを同一のアドレス空間で動かすUnikernels[*7]や、カーネル機能をユーザ空間で動作可能なライブラリとして実装するライブラリOSと呼ばれる仕組みによって実現されます。また、UnikernelsやライブラリOSは、システムコールのコンテキスト切り替えコスト削減による性能の向上に加え、データセンター環境において需要のあるOS機能の起動時間の短縮、メモリ使用量低減、セキュリティ向上などの利点が謳われており、OSv[*8]、IncludeOS[*9]、LightVM[*10]、HermiTux[*11]、Lupin Linux[*12]、Unikraft[*13]、Unikernel Linux[*14]のようなUnikernelsやVirtuOS[*15]、Graphene[*16]、EbbRT[*17]、KylinX[*18]、Demikernel[*19]のようなライブラリOSなど、数多くの研究と実装が発表されています。

*7 Anil Madhavapeddy, Richard Mortier, Charalampos Rotsos, David Scott, Balraj Singh, Thomas Gazagnaire, Steven Smith, Steven Hand, and Jon Crowcroft. 2013. Unikernels: Library Operating Systems for the Cloud. In Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '13), 461–472.(https://doi.org/10.1145/2451116.2451167)。

*8 Avi Kivity, Dor Laor, Glauber Costa, Pekka Enberg, Nadav Har'El, Don Marti, and Vlad Zolotarov. 2014. OSv - Optimizing the Operating System for Virtual Machines. In 2014 USENIX Annual Technical Conference (USENIX ATC 14), 61–72. (https://www.usenix.org/conference/atc14/technical-sessions/presentation/kivity)。

*9 Alfred Bratterud, Alf-Andre Walla, Hårek Haugerud, Paal E. Engelstad, and Kyrre Begnum. 2015. IncludeOS: A Minimal, Resource Efficient Unikernel for Cloud Services. In 2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom), 250–257.(https://doi.org/10.1109/CloudCom.2015.89)。

*10 Filipe Manco, Costin Lupu, Florian Schmidt, Jose Mendes, Simon Kuenzer, Sumit Sati, Kenichi Yasukata, Costin Raiciu, and Felipe Huici. 2017. My Vm Is Lighter (and Safer) Than Your Container. In Proceedings of the 26th Symposium on Operating Systems Principles (SOSP '17), 218–233.(https://doi.org/10.1145/3132747.3132763)。

*11 Pierre Olivier, Daniel Chiba, Stefan Lankes, Changwoo Min, and Binoy Ravindran. 2019. A Binary-Compatible Unikernel. In Proceedings of the 15th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE 2019), 59–73. (https://doi.org/10.1145/3313808.3313817)。

*12 Hsuan-Chi Kuo, Dan Williams, Ricardo Koller, and Sibin Mohan. 2020. A Linux in Unikernel Clothing. In Proceedings of the Fifteenth European Conference on Computer Systems (EuroSys ' 20).(https://doi.org/10.1145/3342195.3387526)。

*13 Simon Kuenzer, Vlad-Andrei Bădoiu, Hugo Lefeuvre, Sharan Santhanam, Alexander Jung, Gaulthier Gain, Cyril Soldani, Costin Lupu, Ştefan Teodorescu, Costi Răducanu, Cristian Banu, Laurent Mathy, Răzvan Deaconescu, Costin Raiciu, and Felipe Huici. 2021. Unikraft: Fast, Specialized Unikernels the Easy Way. In Proceedings of the Sixteenth European Conference on Computer Systems (EuroSys '21), 376–394.(https://doi.org/10.1145/3447786.3456248)。

*14 Ali Raza, Thomas Unger, Matthew Boyd, Eric B Munson, Parul Sohal, Ulrich Drepper, Richard Jones, Daniel Bristot De Oliveira, Larry Woodman, Renato Mancuso, Jonathan Appavoo, and Orran Krieger. 2023. Unikernel Linux (UKL). In Proceedings of the Eighteenth European Conference on Computer Systems (EuroSys '23), 590–605.(https://doi.org/10.1145/3552326.3587458)。

*15 Ruslan Nikolaev and Godmar Back. 2013. VirtuOS: An Operating System with Kernel Virtualization. In Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles (SOSP '13), 116–132.(https://doi.org/10.1145/2517349.2522719)。

*16 Chia-Che Tsai, Kumar Saurabh Arora, Nehal Bandi, Bhushan Jain, William Jannen, Jitin John, Harry A. Kalodner, Vrushali Kulkarni, Daniela Oliveira, and Donald E. Porter. 2014. Cooperation and Security Isolation of Library OSes for Multi-Process Applications. In Proceedings of the Ninth European Conference on Computer Systems (EuroSys '14).(https://doi.org/10.1145/2592798.2592812)。

*17 Dan Schatzberg, James Cadden, Han Dong, Orran Krieger, and Jonathan Appavoo. 2016. EbbRT: A Framework for Building Per-Application Library Operating Systems. In 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), 671–688.(https://www.usenix.org/conference/osdi16/technical-sessions/presentation/schatzberg)。

*18 Yiming Zhang, Jon Crowcroft, Dongsheng Li, Chengfen Zhang, Huiba Li, Yaozheng Wang, Kai Yu, Yongqiang Xiong, and Guihai Chen. 2018. KylinX: A Dynamic Library Operating System for Simplified and Efficient Cloud Virtualization. In 2018 USENIX Annual Technical Conference (USENIX ATC 18), 173–186.(https://www.usenix.org/conference/atc18/presentation/zhang-yiming)。

*19 Irene Zhang, Amanda Raybuck, Pratyush Patel, Kirk Olynyk, Jacob Nelson, Omar S. Navarro Leija, Ashlie Martinez, Jing Liu, Anna Kornfeld Simpson, Sujay Jayakar, Pedro Henrique Penna, Max Demoulin, Piali Choudhury, and Anirudh Badam. 2021. The Demikernel Datapath OS Architecture for Microsecond-Scale Datacenter Systems. In Proceedings of the ACM Sigops 28th Symposium on Operating Systems Principles (SOSP '21), 195–211.(https://doi.org/10.1145/3477132.3483569)。

2.3.2 ユーザ空間とNIC間のパケット受け渡しの効率化

10Gbps NICが広く使われるようになって、図-1にあるような構成では十分な性能、特に小さいパケットサイズにおいてワイヤーレートを達成するのが難しいということが言われるようになりました。

■ パケットI/Oフレームワーク

このような課題に対して、2010年代初め頃に、Data Plane Development Kit（DPDK）[*20]やnetmap[*21]のようなパケットI/Oフレームワークと呼ばれる、ユーザ空間とNICの間で効率的にデータの受け渡しを可能にする仕組みが提案されました。パケットI/Oフレームワークの基本的な機能としては次の2点があります。

(1) NICに登録している送受信用のパケットバッファを直接ユーザ空間プログラムに貼り付けます。
(2) ユーザ空間プログラムが
  a)新しくNICが受信したパケットの検知、
  b)NICへパケットの転送をリクエストすることを可能にする軽量なインタフェースを提供します。

■ プログラムの挙動

ユーザ空間プログラムが上のようなパケットI/Oフレームワークの基本機能を使って前節Bのような、受信したデータに合わせて生成したデータを送信するといった処理を行う場合、以下のような挙動になります。

□ STEP1 受信パケットの検知

先述のパケットI/Oフレームワークから提供されているインタフェースを使って「新しくNICが受信したパケットの検知」します[*22]。

□ STEP2 アプリケーション固有の処理

前節Bのステップ3と同じように受信したデータに応じてアプリケーション固有の処理を行います。

□ STEP3 NICからのデータ転送

アプリケーション固有の処理がデータの転送を必要とする場合は、まずユーザ空間に貼り付けられている送信用パケットバッファに送信したいデータを詰め、その後、先述のパケットI/Oフレームワークから提供されているインタフェースを使って「NICへパケット転送をリクエスト」します。

□ 注意点

上のプログラムの挙動全体は、前節で述べたA・B・Cのすべての処理を置き換え、大幅に簡略化するため、ユーザ空間プログラムとNICの間で非常に高速にデータのやり取りを行うことを可能にしています。ただし、上の挙動にはAのステップ2やCのステップ2にあるようなプロトコルの処理が含まれていないため、そのままではTCP接続経由でデータを配送するようなWebサーバなどを動かすことはできないという点に注意が必要です。

□ 削減できるコスト

詳細はパケットI/Oフレームワークの実装に依存しますが、例えばDPDK[*20]であれば2.2.1で述べた汎用OS環境と比較して、上の「注意点」にあるようにプロトコル処理が介在しないだけでなく、前節Aのステップ1を起点としたカーネルスレッドのスケジューリング、ステップ3を起点としたユーザ空間プログラム起動に伴うスケジューリング、前節Bのステップ1、2、4に含まれるシステムコール呼び出しとそれに伴うユーザ空間とカーネルの間でのメモリコピー、等々に由来するコストを削減可能です。

＊20 Intel. 2010. Data Plane Development Kit.（https://www.dpdk.org/）。
＊21 Luigi Rizzo. 2012. Netmap: A Novel Framework for Fast Packet I/O. In 2012 USENIX Annual Technical Conference (USENIX ATC 12), 101–112.（https://www.usenix.org/conference/atc12/technical-sessions/presentation/rizzo）。
＊22 新しい受信パケットが検知された場合、そのデータは既にユーザ空間に貼り付けられたパケットバッファ上に存在するため、前節Bのステップ2にあるような処理は不要です。

□ **主な用途**

「注意点」として先ほど述べたとおり、ユーザ空間プログラムがNICから受け取るデータはTCPのようなプロトコルの処理がされていないものですが、この特性は、ルータのようなネットワーク機能をソフトウェアで実装する場合に都合が良く、Network Function Virtualization(NFV)[*23]というようなコンテキストにおいて、パケットI/Oフレームワークは広く採用されていきました。例を挙げると、研究コミュニティでは、FastClick[*24]、E2[*25]、NetBricks[*26]、Metron[*27]、のようなパケットI/Oフレームワークを利用したNFV基盤が開発されました。また、次節で後述するように、WebサーバなどのアプリケーションプログラムをパケットI/Oフレームワークを組み合わせて利用できるように、パケットI/Oフレームワークの上で動作するプロトコルスタックが開発されています。更に、2.3.4で述べるような、仮想マシンの通信高速化にも利用されていきます。

2.3.3 ネットワークスタック設計の再考

■ **マルチコア環境でスケールさせる**

多くのNICは、マルチコア環境で性能をスケールさせることができるように、複数のパケット送受信キューを作成することが可能で、それぞれのCPUコアごとに利用する送受信キューを分けることで、それらキューへアクセスするときのロックの取り合いを避けることを可能にしています。また、多くの高性能なNICはReceive Side Scaling(RSS)と呼ばれるハードウェア機能を実装しており、受信したパケットをTCPのコネクションごとや、IPアドレスを元に特定の受信キューへパケットを振り分け、処理を分散することができるようになっています。ただ、パケットの送受信キューを分けるだけでは不十分で、新しく接続が確立されたTCPコネクションのキューが各ソケットあたり1つずつしかなく、マルチコア環境において並列で同じソケットに対してacceptシステムコールを発行すると性能がスケールしない、という課題がありました。これに対して、Affinity-Accept[*28]、MegaPipe[*29]、Fastsocket[*30]というようなシステムにおいて、TCPコネクションのキューもコアごとに用意する方法が提案され、accept処理のマルチコア環境での性能がスケールできるようになることが示されています。また、MegaPipe[*29]では、2.3.1で触れたFlexSC[*3]を参考に、複数の処理をバッチできるようにしています。

■ **パケットI/Oフレームワークの利用**

2.3.2のパケットI/Oフレームワークの「主な用途」として述べたような、パケットI/OフレームワークをWebサーバのようなプログラムの高速化に利用する研究が行われました。具体的には、2.3.2の「プログラムの挙動」のステップ2の中に組み

*23 NFVは、それまでネットワーク機能ごとに高価な専用ハードウェアを購入する必要があったところを、汎用コンピュータを使ってソフトウェアでネットワーク機能を実装するようにするもので、単一のコンピュータを複数の用途で利用可能にし、かつ、機能の追加・変更が専用ハードウェアを用いるより容易であるとされています。特に、高速なNICが安価で入手可能になったことはNFVの適用について追い風であったと考えられます。

*24 Tom Barbette, Cyril Soldani, and Laurent Mathy. 2015. Fast Userspace Packet Processing. In 2015 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS), 5–16.(https://doi.org/10.1109/ANCS.2015.7110116)。

*25 Shoumik Palkar, Chang Lan, Sangjin Han, Keon Jang, Aurojit Panda, Sylvia Ratnasamy, Luigi Rizzo, and Scott Shenker. 2015. E2: A Framework for NFV Applications. In Proceedings of the 25th Symposium on Operating Systems Principles (SOSP '15), 121–136.(https://doi.org/10.1145/2815400.2815423)。

*26 Aurojit Panda, Sangjin Han, Keon Jang, Melvin Walls, Sylvia Ratnasamy, and Scott Shenker. 2016. NetBricks: Taking the V Out of NFV. In 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), 203–216. (https://www.usenix.org/conference/osdi16/technical-sessions/presentation/panda)。

*27 Georgios P. Katsikas, Tom Barbette, Dejan Kostić, Rebecca Steinert, and Gerald Q. Maguire Jr. 2018. Metron: NFV Service Chains at the True Speed of the Underlying Hardware. In 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18), 171–186.(https://www.usenix.org/conference/nsdi18/presentation/katsikas)。

*28 Aleksey Pesterev, Jacob Strauss, Nickolai Zeldovich, and Robert T. Morris. 2012. Improving Network Connection Locality on Multicore Systems. In Proceedings of the 7th ACM European Conference on Computer Systems (EuroSys '12), 337–350. (https://doi.org/10.1145/2168836.2168870)。

*29 Sangjin Han, Scott Marshall, Byung-Gon Chun, and Sylvia Ratnasamy. 2012. MegaPipe: A New Programming Interface for Scalable Network I/O. In 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12), 135–148. (https://www.usenix.org/conference/osdi12/technical-sessions/presentation/han)。

*30 Xiaofeng Lin, Yu Chen, Xiaodong Li, Junjie Mao, Jiaquan He, Wei Xu, and Yuanchun Shi. 2016. Scalable Kernel TCP Design and Implementation for Short-Lived Connections. In Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '16), 339–352.(https://doi.org/10.1145/2872362.2872391)。

込むことができるTCP/IPのようなプロトコルの実装が行われ、結果として、同じく2.3.2で「削減できるコスト」として述べられたような処理のコストを排除することができるようになり、大幅な高速化が実現しました。研究としては、2014年にSandstorm[*31]とmTCP[*32]というユーザ空間ネットワークスタックが発表されました。mTCP[*32]では上述のFlexSC[*3]で提案されているようなリクエストのバッチに加え、3.3.1にて触れたAffinity-Accept[*28]やMegaPipe[*29]にあるようなTCPコネクションのキューをコアごとに分ける最適化が含まれています。また、同じく2014年にArrakis[*33]とIX[*34]というデバイスを高速に扱えるようにすることを目指した新しいOSが発表されました。これらは共にlwIP[*35]という実装を元にしたネットワークスタックがNICに対して直接I/Oを発行できるようにしています。また、2016年には、パケットの送受信はパケットI/Oフレームワークを使い2.3.2の「プログラムの挙動」を採用しつつ、TCP/IPのプロトコルの処理は前述Aのステップ2やCのステップ2にあるようなカーネルの実装を利用することで成熟したカーネルのTCP/IP実装の恩恵を受けられるようにするStackMap[*36]というシステムが提案されています。2019年にはTAS[*37]という、こちらもDPDK[*20]を利用したユーザ空間で動作するTCPスタック実装が発表されています。2022年にはTAS[*37]とStrata[*38]というファイルシステムを拡張して、既存のアプリケーションに対して変更を加えることなく、I/Oのためのメモリコピーを取り除くことができるようにしたzIO[*39]というシステムが提案されています。また、データセンター内で必要とされる低遅延通信実現のためにZygOS[*40]、Shenango[*41]、Shinjuku[*42]、Caladan[*43]というような、タスクへのCPUコア割り当ての最適化についての研究も行われましたが、これらもDPDK[*20]の上で動作するTCP/IPスタックを採用しています。

*31 Ilias Marinos, Robert N. M. Watson, and Mark Handley. 2014. Network Stack Specialization for Performance. In Proceedings of the 2014 ACM Conference on SIGCOMM (SIGCOMM '14), 175–186.(https://doi.org/10.1145/2619239.2626311)。

*32 EunYoung Jeong, Shinae Wood, Muhammad Jamshed, Haewon Jeong, Sunghwan Ihm, Dongsu Han, and KyoungSoo Park. 2014. mTCP: A Highly Scalable User-Level TCP Stack for Multicore Systems. In 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14), 489–502.(https://www.usenix.org/conference/nsdi14/technical-sessions/presentation/jeong)。

*33 Simon Peter, Jialin Li, Irene Zhang, Dan R. K. Ports, Doug Woos, Arvind Krishnamurthy, Thomas Anderson, and Timothy Roscoe. 2014. Arrakis: The Operating System Is the Control Plane. In 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14), 1–16.(https://www.usenix.org/conference/osdi14/technical-sessions/presentation/peter)。

*34 Adam Belay, George Prekas, Ana Klimovic, Samuel Grossman, Christos Kozyrakis, and Edouard Bugnion. 2014. IX: A Protected Dataplane Operating System for High Throughput and Low Latency. In 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14), 49–65.(https://www.usenix.org/conference/osdi14/technical-sessions/presentation/belay)。

*35 Adam Dunkels. 2001. Design and Implementation of the lwIP TCP/IP Stack. Swedish Institute of Computer Science 2, 77.

*36 Kenichi Yasukata, Michio Honda, Douglas Santry, and Lars Eggert. 2016. StackMap: Low-Latency Networking with the OS Stack and Dedicated NICs. In 2016 USENIX Annual Technical Conference (USENIX ATC 16), 43–56. (https://www.usenix.org/conference/atc16/technical-sessions/presentation/yasukata)。

*37 Antoine Kaufmann, Tim Stamler, Simon Peter, Naveen Kr. Sharma, Arvind Krishnamurthy, and Thomas Anderson. 2019. TAS: TCP Acceleration as an OS Service. In Proceedings of the Fourteenth EuroSys Conference 2019 (EuroSys '19).(https://doi.org/10.1145/3302424.3303985)。

*38 Youngjin Kwon, Henrique Fingler, Tyler Hunt, Simon Peter, Emmett Witchel, and Thomas Anderson. 2017. Strata: A Cross Media File System. In Proceedings of the 26th Symposium on Operating Systems Principles (SOSP '17), 460–477. (https://doi.org/10.1145/3132747.3132770)。

*39 Timothy Stamler, Deukyeon Hwang, Amanda Raybuck, Wei Zhang, and Simon Peter. 2022. zIO: Accelerating IO-Intensive Applications with Transparent Zero-Copy IO. In 16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22), 431–445. (https://www.usenix.org/conference/osdi22/presentation/stamler)。

*40 George Prekas, Marios Kogias, and Edouard Bugnion. 2017. ZygOS: Achieving Low Tail Latency for Microsecond-Scale Networked Tasks. In Proceedings of the 26th Symposium on Operating Systems Principles (SOSP '17), 325–341. (https://doi.org/10.1145/3132747.3132780)。

*41 Amy Ousterhout, Joshua Fried, Jonathan Behrens, Adam Belay, and Hari Balakrishnan. 2019. Shenango: Achieving High CPU Efficiency for Latency-Sensitive Datacenter Workloads. In 16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19), 361–378.(https://www.usenix.org/conference/nsdi19/presentation/ousterhout)。

*42 Kostis Kaffes, Timothy Chong, Jack Tigar Humphries, Adam Belay, David Mazières, and Christos Kozyrakis. 2019. Shinjuku: Preemptive Scheduling for μsecond-scale Tail Latency. In 16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19), 345–360.(https://www.usenix.org/conference/nsdi19/presentation/kaffes)。

*43 Joshua Fried, Zhenyuan Ruan, Amy Ousterhout, and Adam Belay. 2020. Caladan: Mitigating Interference at Microsecond Timescales. In 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20), 281–297. (https://www.usenix.org/conference/osdi20/presentation/fried)。

■ ハードウェアへの処理のオフロード

TCPのような処理はコネクションのステートの管理など、比較的複雑な処理が多くCPUにとって負荷が高いため、TCP Offload Engine(TOE)と呼ばれる、NICのようなハードウェアに処理をオフロード可能にするアプローチも模索されてきました。研究コミュニティにおいては、2020年に発表されたAccelTCP[*44]というシステムで、TCPの接続の確立など、特定の状態に関する処理をNICにオフロード可能にすることで、2つのコネクションをつなぎ合わせるspliceのような処理が高速化でき、主にL7ロードバランサーの性能向上に貢献できることを示しています。また、同じく2020年に、Tonic[*45]というNICの中にトランスポート層プロトコルを実装するのに適したハードウェアの設計がなされています。2022年にはFlexTOE[*46]というスマートNICの上で動作するTOE実装が発表され、2023年にはIO-TCP[*47]という、コンテンツ配送ワークロード効率化のために、NICがTCPの処理に加えて直接ストレージデバイスへアクセス可能なようにしたシステムが提案されています。

2.3.4 仮想マシン通信の高速化

前掲の図-4にあるように、仮想マシンの通信においては、物理NICにおけるパケット入出力を多重化する仮想スイッチと、仮想NICのエミュレーションを担当するバックエンドが主なソフトウェアの構成要素となっています。本節では、これら2つの最適化に向けた取り組みについて見ていきます。

■ 仮想スイッチの高速化

2.3.2で触れたパケットI/Oフレームワークは仮想スイッチの高速化において非常にインパクトがあり、過去の研究では、パケットI/Oフレームワークを仮想スイッチの足回りとして適用することで、既存の仕組みと比べて大幅に性能を向上できることが示されました。仮想スイッチは、2.2.2で述べた仮想マシンのI/Oにおいて、Dのステップ2とEのステップ3で実行されるため、仮想スイッチの性能向上は仮想マシンの通信性能の向上に大きく貢献します。研究としては、2012年に、VALE[*48]という2.3.2で触れたnetmap[*21]のAPIの上で動作可能な仮想スイッチ、また、2013年にCuckooSwitch[*49]とい

*44 YoungGyoun Moon, SeungEon Lee, Muhammad Asim Jamshed, and KyoungSoo Park. 2020. AccelTCP: Accelerating Network Applications with Stateful TCP Offloading. In 17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20), 77–92.(https://www.usenix.org/conference/nsdi20/presentation/moon)。

*45 Mina Tahmasbi Arashloo, Alexey Lavrov, Manya Ghobadi, Jennifer Rexford, David Walker, and David Wentzlaff. 2020. Enabling Programmable Transport Protocols in High-Speed NICs. In 17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20), 93–109.(https://www.usenix.org/conference/nsdi20/presentation/arashloo)。

*46 Rajath Shashidhara, Tim Stamler, Antoine Kaufmann, and Simon Peter. 2022. FlexTOE: Flexible TCP Offload with Fine-Grained Parallelism. In 19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22), 87–102.(https://www.usenix.org/conference/nsdi22/presentation/shashidhara)。

*47 Taehyun Kim, Deondre Martin Ng, Junzhi Gong, Youngjin Kwon, Minlan Yu, and KyoungSoo Park. 2023. Rearchitecting the TCP Stack for I/O-Offloaded Content Delivery. In 20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23), 275–292.(https://www.usenix.org/conference/nsdi23/presentation/kim-taehyun)。

*48 Luigi Rizzo and Giuseppe Lettieri. 2012. VALE, a Switched Ethernet for Virtual Machines. In Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies (CoNEXT '12), 61–72.(https://doi.org/10.1145/2413176.2413185)。

*49 Dong Zhou, Bin Fan, Hyeontaek Lim, Michael Kaminsky, and David G. Andersen. 2013. Scalable, High Performance Ethernet Forwarding with CuckooSwitch. In Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies (CoNEXT '13), 97–108.(https://doi.org/10.1145/2535372.2535379)。

うDPDK[*20]を足回りに利用した仮想スイッチが提案されています。また、2015年にVALE[*48]の拡張として、mSwitch[*50]が発表されました。広く利用されている仮想スイッチ実装であるOpen vSwitch[*51]でも、この辺りの時期にDPDK[*20]のサポートを追加する取り組みが行われていました。

■ 仮想I/Oバックエンドの改善

研究コミュニティでは、2013年頃に、上述の「仮想スイッチの高速化」にて述べたような仮想スイッチを仮想マシン通信のバックエンドに適用する試みが行われました。その中の1つは、上に述べたVALE[*48]をQEMU[*52]で利用可能なネットワークバックエンドに使うもので、具体的には図-2にある既存のOSカーネルに実装される仮想スイッチを、VALE[*48]スイッチに置き換えることで、仮想マシンのI/O性能を向上できることを示しました[*53]。ですが、この最適化では仮想マシンに付与される仮想NICは既存のものを利用していたため、VALE[*48]との親和性が低く、2.2.2で述べたDのステップ2やEのステップ2において、仮想スイッチと仮想NIC間でのパケットデータのメモリコピーを排除できないなどの欠点があったため、性能についてはまだ向上の余地がありました。この性能向上の余地を埋めるべく、2015年にptnetmap[*54][*55]というnetmap[*21]インタフェースを直接仮想マシンに対して割り当てる手法がQEMU[*52]/KVM[*56]用に実装され、仮想マシンからでも10Gbps NICの最小パケットサイズでのワイヤーレートである14.88Mppsが達成可能であることが報告されています。また、2014年にはClickOS[*57]という仮想マシンを基本としたNFV基盤の通信を高速化するために、Xen[*6]の仮想マシン通信機構であるnetfront/netbackをVALE[*48]とnetmap[*21]APIを元にした通信機構に置き換える実装が提案されています。更に、同じく2014年にNetVM[*58]という仮想マシンを元にしたNFV基盤が提案されており、こちらはDPDK[*20]をQEMU[*52]/KVM[*56]に適用することで仮想マシンの通信を高速化しています。2017年にはHyperNF[*59]という仕組みにおいて、VALE[*48]を適用した仮想マシン環境でも、前述Eのステップ2にあるようなホスト側で動作するカーネルスレッドと、仮想マシンの仮想CPUを実行するスレッドを別々に分けてしまうことでCPU利用効率が最大化できなくなってしまうという課題に対して、VALE[*48]のような仮想スイッチの処理を、仮想CPUの実行コンテキス

*50 Michio Honda, Felipe Huici, Giuseppe Lettieri, and Luigi Rizzo. 2015. mSwitch: A Highly-Scalable, Modular Software Switch. In Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research (SOSR '15). (https://doi.org/10.1145/2774993.2775065)。

*51 Ben Pfaff, Justin Pettit, Teemu Koponen, Ethan Jackson, Andy Zhou, Jarno Rajahalme, Jesse Gross, Alex Wang, Joe Stringer, Pravin Shelar, Keith Amidon, and Martin Casado. 2015. The Design and Implementation of Open vSwitch. In 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15), 117–130. (https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/pfaff)。

*52 Fabrice Bellard. 2005. QEMU, a Fast and Portable Dynamic Translator. In 2005 USENIX Annual Technical Conference (USENIX ATC 05), 41–46. (https://www.usenix.org/conference/2005-usenix-annual-technical-conference/qemu-fast-and-portable-dynamic-translator)。

*53 Luigi Rizzo, Giuseppe Lettieri, and Vincenzo Maffione. 2013. Speeding up Packet I/O in Virtual Machines. In Architectures for Networking and Communications Systems, 47–58. (https://doi.org/10.1109/ANCS.2013.6665175)。

*54 Stefano Garzarella, Giuseppe Lettieri, and Luigi Rizzo. 2015. Virtual Device Passthrough for High Speed Vm Networking. In 2015 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS), 99–110. (https://doi.org/10.1109/ANCS.2015.7110124)。

*55 Vincenzo Maffione, Luigi Rizzo, and Giuseppe Lettieri. 2016. Flexible Virtual Machine Networking Using Netmap Passthrough. In 2016 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN), 1–6. (https://doi.org/10.1109/LANMAN.2016.7548852)。

*56 Avi Kivity, Yaniv Kamay, Dor Laor, Uri Lublin, and Anthony Liguori. 2007. KVM: the Linux Virtual Machine Monitor. In Proceedings of the 2007 Ottawa Linux Symposium (OLS '07).

*57 Joao Martins, Mohamed Ahmed, Costin Raiciu, Vladimir Olteanu, Michio Honda, Roberto Bifulco, and Felipe Huici. 2014. ClickOS and the Art of Network Function Virtualization. In 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14), 459–473. (https://www.usenix.org/conference/nsdi14/technical-sessions/presentation/martins)。

*58 Jinho Hwang, K. K. Ramakrishnan, and Timothy Wood. 2014. NetVM: High Performance and Flexible Networking Using Virtualization on Commodity Platforms. In 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14), 445–458. (https://www.usenix.org/conference/nsdi14/technical-sessions/presentation/hwang)。

*59 Kenichi Yasukata, Felipe Huici, Vincenzo Maffione, Giuseppe Lettieri, and Michio Honda. 2017. HyperNF: Building a High Performance, High Utilization and Fair NFV Platform. In Proceedings of the 2017 Symposium on Cloud Computing (SoCC '17), 157–169. (https://doi.org/10.1145/3127479.3127489)。

トに含まれるハイパーコールの中で行うことで、CPU利用効率が向上し、結果として高い仮想マシンの通信性能につながることが示されました。

■ ハードウェアへの処理のオフロード

多くのNICはSingle Root I/O Virtualization(SR-IOV)[*60]のようなハードウェアでパケットスイッチングを行う機能を実装しており、この機能を利用すると、ソフトウェアによる仮想スイッチ実装と比較して高い性能を発揮できる場合が多いです。一方、SR-IOV[*60]は、ソフトウェアから物理・仮想インタフェース間でのパケット転送について限られた挙動の制御しかできず、データセンターのような複雑な制御が必要な場面では利用が困難な場合がありました。このような課題に対して、2018年にはAccelNet[*61]というスマートNICを使うことでネットワークの制御の柔軟性を高めるシステムの論文が公開されました(AccelNetの商用環境デプロイ自体は2015年から行われていたそうです)。

2.4 IIJ技術研究所における近年の取り組み

本節では、ここまで見てきた過去の研究をふまえ、IIJ技術研究所がどのような取り組みを行っているかについてご紹介します。

2.4.1 新規OS機能と既存のプログラムとの統合方法

前節で見てきたように、10年以上前から研究コミュニティでは、既存の仕組みを置き換えるような、新しいOS機能の設計と実装が発表されてきました。

■ 問題

新しいOS機能を、既存のアプリケーションプログラムに対して「透過的に」適用するためには、システムコールをフックする機構を利用するのが一般的ですが、既存のシステムコールフックの仕組みは、アプリケーションに大幅な性能劣化を引き起こしたり、一部システムコールをフックしそこねる場合があるなどの欠点がありました。これら既存の仕組みの欠点により、特に前節で述べたようなUnikernels・ライブラリOSや新しい

*60 PCI-SIG. 2010. Single Root I/O Virtualization and Sharing Specification.(https://pcisig.com/specifications/iov/single_root/)。
*61 Daniel Firestone, Andrew Putnam, Sambhrama Mundkur, Derek Chiou, Alireza Dabagh, Mike Andrewartha, Hari Angepat, Vivek Bhanu, Adrian Caulfield, Eric Chung, Harish Kumar Chandrappa, Somesh Chaturmohta, Matt Humphrey, Jack Lavier, Norman Lam, Fengfen Liu, Kalin Ovtcharov, Jitu Padhye, Gautham Popuri, Shachar Raindel, Tejas Sapre, Mark Shaw, Gabriel Silva, Madhan Sivakumar, Nisheeth Srivastava, Anshuman Verma, Qasim Zuhair, Deepak Bansal, Doug Burger, Kushagra Vaid, David A. Maltz, and Albert Greenberg. 2018. Azure Accelerated Networking: SmartNICs in the Public Cloud. In 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18), 51–66.(https://www.usenix.org/conference/nsdi18/presentation/firestone)。

ネットワークスタック実装の適用範囲が制限されており、結果として、多くの人が研究成果の恩恵を受ける妨げになっているという問題があります。この問題は、既に存在する、ソフトウェアの実行効率を大きく向上し、必要なサーバ数や消費電力を削減できるはずの技術を利用することを困難にしているという側面があります。

■ 解決策

我々は、この問題を解決するために、zpoline[*62][*63]という、既存の仕組みの欠点を補った、新しいシステムコールフックの仕組みを考案しました。zpoline[*62][*63]では、`syscall`と`sysenter`というそれぞれ2バイトのシステムコールを発行する命令を`callq *%rax`という同じく2バイトのcall系列の命令と置き換えると共に、仮想メモリアドレス0番地にトランポリンコードを用意することで、プログラム中の`syscall`/`sysenter`を特定のフック処理へのジャンプに置き換えます。この仕組みの負荷は、ptraceやint3命令を用いたバイナリ書き換えテクニック、またSyscall User Dispatch[*64]のような既存の仕組みと比較して、28〜700倍以上小さいという結果になりました。また、これらを使って、広く利用されているキー・バリューストアであるRedis[*65]に対してlwIP[*35]とDPDK[*20]を適用した場合、フックの負荷がほぼ存在しない場合と比べ、既存の仕組みが72.3〜98.8%の性能劣化を引き起こすのに対し、提案手法が引き起こす性能劣化は5.2%に留めることができました。

2.4.2 仮想マシンのI/O高速化

この10年の間に仮想マシンの通信性能は大幅に向上しました。一方で、課題もあります。

■ 問題

仮想マシン環境において性能劣化の原因の1つとされる、仮想マシンコンテキストから出る（exitする）コストは、まだ取り除

＊62 Kenichi Yasukata. 2021. システムコールを速く漏れなくフックする方法. IIJ Engineers Blog.（https://eng-blog.iij.ad.jp/archives/11169）。
＊63 Kenichi Yasukata, Hajime Tazaki, Pierre-Louis Aublin, and Kenta Ishiguro. 2023. zpoline: a system call hook mechanism based on binary rewriting. In 2023 USENIX Annual Technical Conference (USENIX ATC 23), 293–300. (https://www.usenix.org/conference/atc23/presentation/yasukata)。
＊64 Gabriel Krisman Bertazi. 2021. Syscall User Dispatch.（https://www.kernel.org/doc/html/latest/admin-guide/syscall-user-dispatch.html）。
＊65 Salvatore Sanfilippo. 2009. Redis - Remote Dictionary Server.（https://redis.io/）。

かれておらず、結果として、具体的には2.2.2に述べたEのステップ1で発生するexitによりI/O性能が低下し、仮想マシン上で動作するワークロードの性能を制限してしまう、という問題があります。仮想マシンのI/O性能自体が低い場合、2.3.1や2.3.3で述べられたような仕組みを使って仮想マシン上で動作するネットワークに関わる処理を効率化しても、達成可能な最大の性能が制限されてしまうため、多くの計算がデータセンター内の仮想マシンで実行されている現状においてこの問題は重要です。

■ 解決策

この問題に対して、我々は、仮想マシンが、仮想マシンコンテキストから出る(exitする)ことなく、仮想マシン間で共有されているNICへアクセスすることを可能にするExit-Less、Isolated、and Shared Access(ELISA)[*66][*67]という仕組みを提案しました。提案手法では、VMFUNCというCPU命令を用いることで、仮想マシン内で、ホストが許可した挙動のみを行うことができる新しいコンテキストを作ることができ、その新しいコンテキストの中でのみNICにアクセスできるようにすることで、仮想マシンがNICを通じて悪意のある挙動を行うことを防ぎます。また、提案手法ではソフトウェアで仮想マシンがデバイスにアクセスする方法を実装するため、SR-IOV[*60]より高い挙動の柔軟性を提供できます。提案手法を用いて仮想マシン通信機能を実装することで、仮想マシンのI/Oリクエストごとに仮想マシンコンテキストからexitするHyperNF[*59]と同様の仕組みと比較して、最大163%性能を向上できました。

2.5 まとめ

システムソフトウェアの通信機能についてある程度一般的な挙動について触れた上で、2010年代初頭からのシステムソフトウェアの通信分野の過去の研究がどのようにそれらを改善してきたかについて振り返りました。また、それらをふまえた近年のIIJ技術研究所での取り組みをご紹介しました。

執筆者:
安形 憲一(やすかた けんいち)
IIJ 技術研究所 技術研究室。
システムソフトウェアの研究に取り組んでいます。

*66 Kenichi Yasukata. 2023.【国際学会 ASPLOS 2023】論文採択までの道のり〜仮想マシン間のメモリ領域共有に関する課題と解決策〜 . IIJ Engineers Blog. (https://eng-blog.iij.ad.jp/archives/18819)。
*67 Kenichi Yasukata, Hajime Tazaki, and Pierre-Louis Aublin. 2023. Exit-Less, Isolated, and Shared Access for Virtual Machines. In Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3 (ASPLOS 2023), 224–237.(https://doi.org/10.1145/3582016.3582042)。

IIJとクラウドの変遷〜30周年特別コンテンツ〜

3.1 はじめに

IIJは、インターネット接続をはじめとした通信サービスを中核に様々な関連サービスを提供してきました。これらのサービスを提供するための「サービスホスト」は30年間に渡り継続的に拡張されており、現在はサーバ数千台にも及んでいます。また、一方でクラウドサービス「IIJ GIO」としてお客様にコンピューティングリソースを提供することも行っており、こちらもサービス開始以降10年間でサーバ数万台規模のインフラへと成長しました。

今回はIIJがこれまで歩んできた30年を振り返り、前半ではIIJのサービスホストが時代背景を踏まえながらどのように変化してきたのか、またその過程においてどのような工夫を行ってきたのか紹介していきます。後半では、世代を経るごとに大規模なサービスインフラへと進化してきた顧客向けクラウド基盤「IIJ GIO」の変遷を紹介します。

3.2 1990年代:サービスホストの始まり

■ 個別構築の時代

IIJが創業した1992年以降、メールやWebサービス提供のためのサービスホストは急速に数を増やしました。1990年代末期にはIIJサービス向けに東京都内の複数のデータセンターに分散して、200ラック以上、数千台規模のサーバを利用するようになりました。サービスごとにラックを確保し、個別に設備調達して構築していましたので、構成変更の度に現地作業が発生していました。そのような状況だったため、物理環境へのアクセスの容易さから、東京近郊の立地の良いデータセンターを利用しサーバ運用を最適化していました。

最初期にはUNIXワークステーションをサーバとして利用していましたが、コストパフォーマンスの観点からPC/AT互換機や、PC/ATベースの産業用PCの利用へと移行しました。まだ

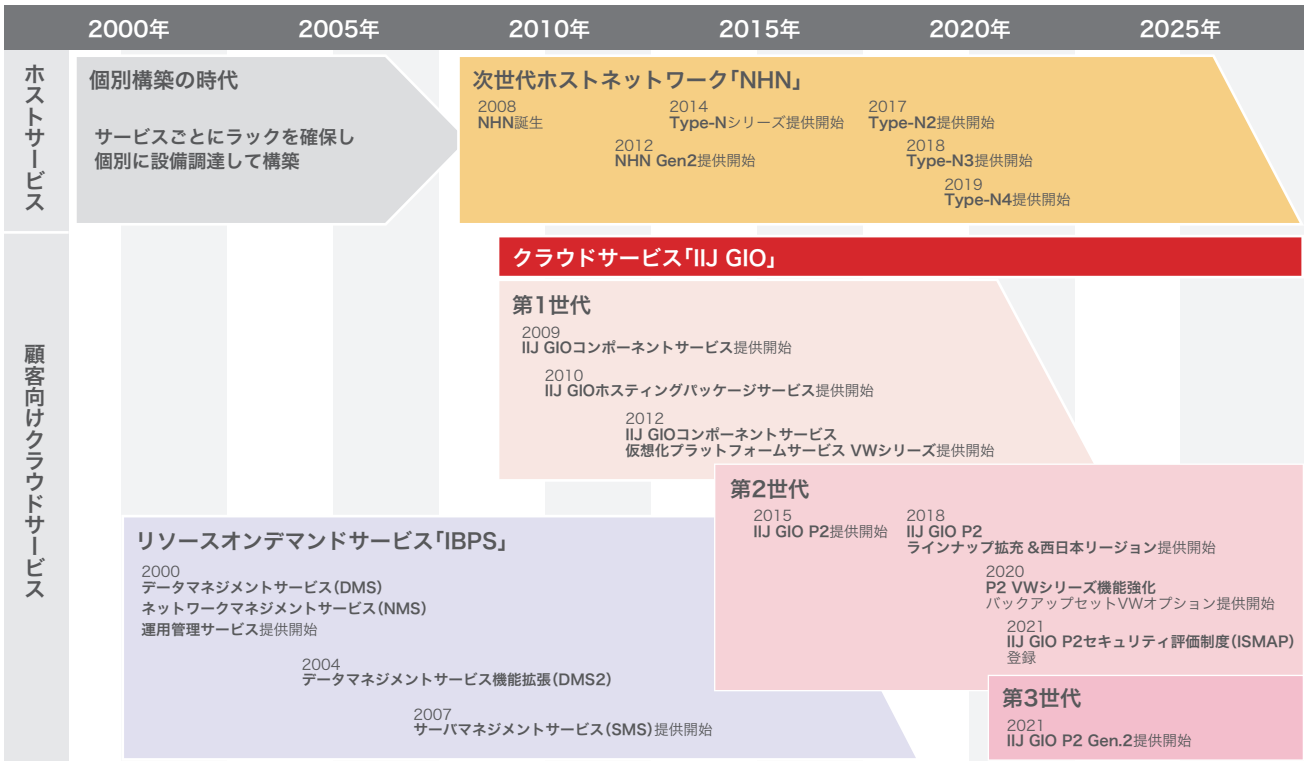

図-1 クラウドの変遷

ラックマウントタイプのPCサーバは市販されておらず、市販の部品を利用してIIJのスタッフが自らサーバを組み立てるようなこともしていました。マザーボードやCPUを入れ替えて使ったり、IIJのスタッフが自分でRAIDを組んだり、マシンが故障したらすぐに交換できるように体制を整えたりと、大手町のデータセンターの中で様々な作業を行いました。低予算で設備や機材を用意することはできましたが、ハードウェアの故障の原因である電源や冷却ファンの故障といった物理的トラブルも増えました。

当時の商用UNIXワークステーションに搭載されていたOSは、価格の割に機能が十分ではなく、自分たちでの改良も困難でした。また、当時はまだLinuxが登場した直後で、実用的に使えるような状態ではありませんでした。そこで、IIJではBSD(カリフォルニア大学バークレー校の開発者グループによって生み出されたソフトウェア群)に由来するPC-BSDのライセンスを取得し、自分たちでサーバ用に再構成することにしました。これにより、格段にサーバの自由度が上がりました。ソフトウェアもハードウェアも手作りでした。こんなにサーバのコストと品質にこだわっていたのは、日本ではIIJぐらいだったと思います。

当時のインターネットでは深夜23時以降に電話料金が定額になる「テレホーダイ」が広く利用されていたため、個人向けWebサイトのアクセスピークは深夜23時でした。このため、日中は問題なく動作していたのに、深夜にWebサーバへのアクセスが集中しサーバの負荷が上がり、消費電力が増えすぎてブレーカーを落としてしまい、ラックごと電力供給が止まることもありました。

この頃のストレージは、サーバ内蔵ストレージを利用する場合と、大容量が必要な場合はサーバとSCSIケーブルでストレージを接続するDASが主流でした。1990〜2000年代はまだ物理的に必要な容量とサーバ1台あたりの容量がどれくらいで、何台並べてどう構成したら費用がどうなるか、予算内に収めるためにはどこのハードウェアベンダーのサーバを使えば良くて、データセンターはどこに置くのか、そんな計画と計算が必要でした。ストレージの大障害を経験したこともあり、バックアップや余裕を持ったサイジングは絶対に必要だ、と身をもって学習しました。

3.3 2000年代:サービスホスト個別構築からの脱却

■ 次世代ホストネットワーク「Next Host Network」誕生

Next Host Network(以下、NHN)は2008年度にサービスホスト運用の効率化を目的に、サーバ構成やサーバ運用ネットワークを再設計し開発されたIIJサービス向け基盤です。これ以前は、前述の通りサービスごとに設備調達を行いシステムを構築していたため、全体的に見て基盤にかかる費用が増加の一途でした。構成変更のたびに個々の拠点で現地作業が発生したり、運用も個々に定義され煩雑になってしまいました。また、それぞれのサービスの需要増に備え余裕のあるキャパシティを確保していましたが、需要の予測が外れれば設備転用が難しく、採用した機器も原則保守契約を結ぶため保守費用も増大しました。ファシリティについても、当時はシステムに問題があればすぐに駆けつけられるように立地の良い都市型データセンターを利用していたため、費用は高騰する一方でした。

このような慢性的な高コスト構造から脱却するには、基盤の根本的な在り方を再検討する必要がありました。数々の問題を解決に導くべく、個別に構築していたシステムを集約し、遠隔でも同等のサービスレベルで提供可能な運用が適用できる基盤として実装されたのがNHNです。国内に分散するデータセンター内のサーバ数千台を4年計画で順次リプレースし、これまでDASに格納していたデータはiSCSIストレージに統合、新たに導入するディスクレスサーバと併せてサービス需要に応じてIPネットワーク上で柔軟に組み替えることで、フレキシブルなサービスホストの提供を基本コンセプトにしました。

基盤設計方針としては、サーバは故障を前提に、ストレージは信頼性の高いもの、ネットワークは複雑なことをせずシンプルに再設計しました。現地作業を削減し、基本はリモート対応する運用方式への転換です。サーバ機器の故障ポイントは多岐に

渡り、故障は避けられず故障率の低減には限界があるため、故障しても影響の少ない構成にしました。サービスごとのラック割を廃止しサーバプール方式を採用して収容効率を向上させ、現地作業は集約し外部業者へ委託できるようになりました。サーバ構成を画一化し構成管理負荷を低減しつつ、仮想化技術との組み合わせにより自由度も向上し、機器設置以降の作業をリモートで実行可能になりました。ストレージ機器のHDD単体の故障は避けられないですが、サービス停止に繋がる故障は回避できるよう、これまでより信頼性の高い構成にする必要がありました。そこでPXE Bootと、コントローラとパスを二重化したiSCSIストレージを利用し、安価で信頼性の高いディスクレスサーバ環境を構成し、故障時にはサーバ機器を切り替えるだけで復旧可能になりました。ホスト収容エッジスイッチの故障は、これまでの経験上さほど多くありませんでしたので、NIC冗長化設定などは必要な箇所のみ実施するシンプルな構成にしました。構成情報に基づき自動的にスイッチのVLAN設定を書き換える仕組みを導入し、物理配線の変更なしにネットワークを構成することで、設定ミスと運用コストの削減を図りました。NHNでは主にコストパフォーマンスに優れるハードウェアベンダー製のx86サーバを採用し、OSはCentOSなどのLinux OSのみでしたが、後にWindows ServerやVMware基盤も提供。ストレージはローエンドクラスながら保守性の高いストレージを採用し、それまで専門家任せだったストレージの保守作業が簡略化されたことで、IIJのスタッフでも自営保守しやすく、年間保守コストの削減に貢献しました。以下からNHNの各世代の特徴を紹介します。

NHN Gen1：2008年に提供開始したNHN第1世代インフラです。DASの排除を目的にiSCSIストレージを採用したのが特徴で、ストレージとして基本領域(60GB)と拡張領域(160GB)の2領域を提供していました。NHN Gen1ではサーバ4ラックとストレージ4ラックのセット構成を1ユニットとしていましたが、在庫リソースがなくなると同一L2領域での拡張ができず、ユニットごとの在庫調整が困難でした。ユニット間のサーバインスタンスの移動もできません。アップリンクが1Gbpsだったため、トラフィック増により上流帯域の逼迫も問題でした。

NHN Gen2：NHN Gen1での問題点を解消すべく設計され、2012年に提供開始した第2世代インフラです。ユニット構成はNHN Gen1と同様ですが、ラック収容効率が2倍に向上(1ラック40台)。JuniperのVirtual Chassis(以下、VC)を採用し、アップリンクの10Gbps化、各サービス固有のNHN標準以外の機器を受け入れるラックの確保や冗長電源、NIC Bondingに対応したのが特徴です。しかし、NHN Gen2ではToR用として採用したスイッチの不具合によりネットワークが全断するという問題も起きました。VCを組むユニット全体に障害が波及し、復旧に時間を要する事態となりました。

NHN Gen2コンテナ：松江データセンターパークに導入された、コンテナ型DCモジュールIZmo/Sに最適化されたNHNです。IZmo/Sの搭載ラック数に合わせ、1ユニット当たり8ラックサーバ288台構成(後にサーバを16台抜いてストレージを4台投入)としたのですが、NHN Gen2コンテナはスリムコンテナ仕様で空間効率が高かった半面、コンテナ内の通路が狭く、メンテナンス性が著しく悪いことが運用上の問題となりました。

3.4 2010年代～現在：次世代サービス基盤「Type-N」登場

■ サービスホストはType-Nへ

2014年10月、NHN第3世代インフラとして新たにType-Nシリーズを提供開始しました。これまでサーバはベアメタルサーバ単位で提供していましたが、Type-Nシリーズから仮想マシン単位の提供も開始しました。ベアメタルサーバーまでは必要のない、より小スペックなニーズにも応えらえるようになりました。以下からType-Nの各世代の特徴を紹介します。

Type-N：2014年に提供開始したNHN第3世代インフラです。ルータにL2スイッチを収容するそれまでの形態から、L2コアスイッチへL2スイッチを収容する形式に変えることで、同一L2面をユニット間で提供可能となり、持込ラックと

の接続性も向上し、ユニットごとの在庫調整も不要となりました。通常のサーバの他、ハイメモリサーバやHDDを大量搭載したストレージサーバも利用可能になりました。しかし、Type-Nではサーバの1GbE NIC向けに採用したスイッチの性能が乏しく、接続可能な機器台数の限界値が低いことが問題となりました。

Type-N100:Type-Nをベースに、コンテンツ配信サービス向けに開発された広帯域提供インフラです。名前は100Gbps提供インフラに由来し、ロードバランサー専用ユニットやMPLSを使用し直接IX接続を担うIIJ BBのルータとも接続可能なことが特徴です。メモリブートが必須(iSCSIストレージは提供しない)で、一時ログやキャッシュ用途として1TB SSDを搭載しました。

Type-N2:2017年に提供開始したNHN第4世代インフラです。1Gネットワークを廃止し、エッジまでフル10G化したのが特徴です。ハイメモリサーバはVMware基盤を前提としたクラスタ型アプリケーション向けにユニット追加の対応を行い、ハードウェアの性能向上により、システムの集約率も向上しました。しかし、Type-N2ではストレージ性能が一部アプリケーションの要求水準を満たせないことが問題となりました。

Type-N3:2018年に提供開始したNHN第5世代インフラです。CPUに当時最新のIntel CPUを採用し、フラッシュストレージ(NVMe SSD)搭載によるIO性能の大幅向上に合わせ、ネットワーク性能も大幅な向上を狙いました。Type-N3ではストレージとして提供していた基本領域と拡張領域は廃止し、全領域を1つのPV(フィジカルボリューム)として提供しました。

Type-N4:2019年に提供開始したNHN第6世代インフラです。引き続きCPUに当時最新のIntel CPUを搭載しつつもOCP準拠サーバ(後述)を新たに採用し、調達コストと消費電力の削減を図りました。ネットワークはエッジ〜コア間をこれまでの10〜20Gbpsから40Gbpsに増強し、コアスイッチにシャーシ型製品を採用することで、将来の設備増強に耐えうる構成としました。

ストレージは、2019年からミッドレンジクラスのストレージをメインストレージとして採用。2021年には性能や信頼性は当然として、これまで築き上げてきた様々な手順や枠組みをそのまま適用できる運用面を大きく評価して、初代のNHNで採用したものと同じメーカの後継機種を採用しています。また、自社構築、自社運用を基本としているNHNにとって、OSやファームウェアのセルフアップデートを含めた自営保守が可能なストレージ製品は、コスト面でも大きなメリットがあります。

■ OCPサーバの導入

OCP(Open Computing Project)は2011年にFacebookをはじめとしたハイパースケーラーが提唱した新しいサーバの規格です。メーカ主導で開発・製造が行われる従来型のサーバと比較して、OCPではサーバを利用するクラウド事業者が部品の選定や調達・製造に深く関与するといったスキームの違いがあります。

IIJとしてもOCPサーバに関心はあったものの、2017年頃までは当時調達していた一般的なサーバメーカに比べて調達価格が高過ぎて割に合わない状況が続いていました。ところが、2018年に入り円高の進行や半導体業界の過剰在庫により、DIMMやSSDなどの価格が下落傾向に転じます。こうした市況を踏まえ、本格的な導入検証に着手しました。為替の影響を直接受けたり、ハードウェア面では新たにOCP専用ラックや集中電源が必要なことやBIOSやBMCのテストは発注側が実施しなければならないこと、運用面では保守はパーツ交換(自営保守)のみ、ODMベンダーとのやり取りは英語等々、様々な課題や懸念事項が山積したものの、それを上回る導入メリットがあると判断し、OCPサーバを導入することを決断しました。実際にOCPサーバを導入した結果、調達費用は最大35%減、電力消費量は最大30%減の削減効果を達成できました。特に調達費用は当初見込んでいた10%減を大幅に上回る削減効果を得

ることができ、その後のOCPサーバ導入拡大の契機となりました。ただし、現状ではエンタープライズ向けでは明らかに適さない用途・領域が存在することも分かっています。2019年夏には既存設備同様OCPサーバにおいてもマルチベンダー体制を確立し、技術的リスクと調達リスクの分散を図っています。

続いて、顧客向けクラウド基盤「IIJ GIO」の変遷を紹介します。

3.5 2000年代：IaaS（Infrastructure as a Service）の先駆け

■ リソースオンデマンドサービス「IBPS」提供開始

1990年代後半にはServer Side JavaやASPが使えるようになりインターネット決済のインフラが実現されると、動的サイトを利用したECサイトが多数登場するようになりました。ECサイトのシステムアーキテクチャは、どのお客様サイトの構成でも似たようなものであったため、予め機器を用意して部品化しておき、ユーザからオーダーがあったら必要な部品を組み合わせて提供すれば効率が良くなる、そう考えて作ったのが、現在のクラウドサービス「IIJ GIO」の前身である、リソースオンデマンドサービス「IBPS（Integration & Business Platform Service）[*1]」です。

当時グループ会社だったアイアイジェイテクノロジー（2010年4月にIIJに吸収合併）から2000年3月に提供開始されたIBPSは、インターネットビジネスで必要となるサーバ機器からソフトウェア、決済/物流コンポーネント、監視・運用管理に至るまで、企業ニーズに合わせて必要な部品を組み合わせ、システム全体をアウトソーシング・サービスとして提供するサービスです。インターネットビジネスの早期展開を必要とする企業からの要望に応え、開発期間と初期投資を従来の1/2以下（当社比）で提供することを可能としました。

■ IBPSが提供する主な４つのサービス

- データマネジメントサービス（DMS）：IIJが保有するストレージリソースをオンデマンド提供。お客様のサービスレベル、利用形態に応じて選択可能なストレージ接続方式（NAS・SAN）・容量・性能のストレージ領域を提供することで、ストレージコストの最適化を実現。別ディスクによる高速バックアップも提供することで、これまでの手間と時間のかかるテープによるバックアップ業務からお客様を解放
- ネットワークマネジメントサービス（NMS）：IIJが保有するネットワークリソースをオンデマンド提供。ロードバランサーやファイアウォールなどをプール化し機能として提供すると共に、典型的な構成はセットメニュー化することで低コストを実現
- サーバマネジメントサービス（SMS）：IIJが保有するサーバリソースをオンデマンド提供。お客様専用のサーバ構成情報をプロビジョニングツールで管理することで構築作業を半自動化し、お客様固有ニーズに応じたシステムを短時間で利用可能
- 運用管理サービス：IBPS上に構築されたお客様システムの運用・監視を代行することで、システムに係るTCOの削減に貢献

必要なときに必要なリソースを使い、いらなくなったら解約できる。ユーザは一切資産保有リスクを持たなくて良い。IBPSは今でいうIaaSに相当するサービスでした。フロントエンドのサーバは当初UNIXサーバ（SPARC Solaris）でしたが、後にx86サーバに移行したものの、ミドル/バックエンドはDB用途でUNIXサーバが残りました。従来はラックマウント型サーバを利用していましたが、2007年8月に国内初のリソースオンデマンド型のサーバ提供を開始したSMSでは国内の大規模サーバ基盤で初めてブレードサーバを導入。スペースを節約できる上に、設定済みのシャーシにCPUブレードを挿すだけで利用できる点を評価しました。

2003年10月にはIBPSサービス開始当初から提供していたデータマネジメントサービスを機能拡張し、総容量40TBの大規模ストレージサービスを提供開始。ミッドレンジストレージでは柔軟なリソース管理とコスト削減を狙って、ストレージ仮想化ソフトウェアを採用しました。2007年にはデータマネ

＊1　サービス開始当初の名称はiBPSとしていたが、2002年にIBPSへとリブランド。

ジメントサービスを設備更改し、ストレージGB単価を1/2[*2]に削減しました。データマネジメントサービスではSAN品目とNAS品目をそれぞれ提供していましたが、適度な価格で高品質なFiber Channel(以下、FC)を利用できるサービスが提供されるとFC-SANのストレージ利用が拡大していきました。NASについては、当初は汎用サーバとクラスタソフトウェアで組み合わせたNASやNAS専用のストレージアプライアンスを利用していました。特に2000年代前半のNAS製品は1TBを超えるボリュームをサポートしている製品が少なく、大容量のNASを利用したい場合などは汎用サーバ+FC-SANのストレージ+クラスタソフトウェアを組み合わせたNASを構築していました。2000年代後半になると、NAS製品が大容量のボリュームをサポートし始めたため、NAS専用のストレージアプライアンスにシフトしていきます。

■ L2リングプロトコル構成を採用した2008年以降

サービス基盤のネットワーク技術においては、2008年以降はL2リングプロトコル構成を採用しました。多段リング物理構成が日本国内のDC収容規模に適しており、効率の良いシステムを構築できるメリットを持ちながら、システム増強もCore/Edgeに関わらず容易に拡張可能でした。ただし、収容ノードのインタフェースが10GbE以上になると1GbE以下の機器構成に比べ選択できる機種が減少するため、収容ノード数に制約が発生するのがデメリットでした。L2リング構成はメーカ独自仕様が多く、異なるメーカ間での互換性がありません。そのため、ベンダーロックインとなってしまう可能性が高く、利用可能な機器の選択肢が少なくなるというのも問題でした。

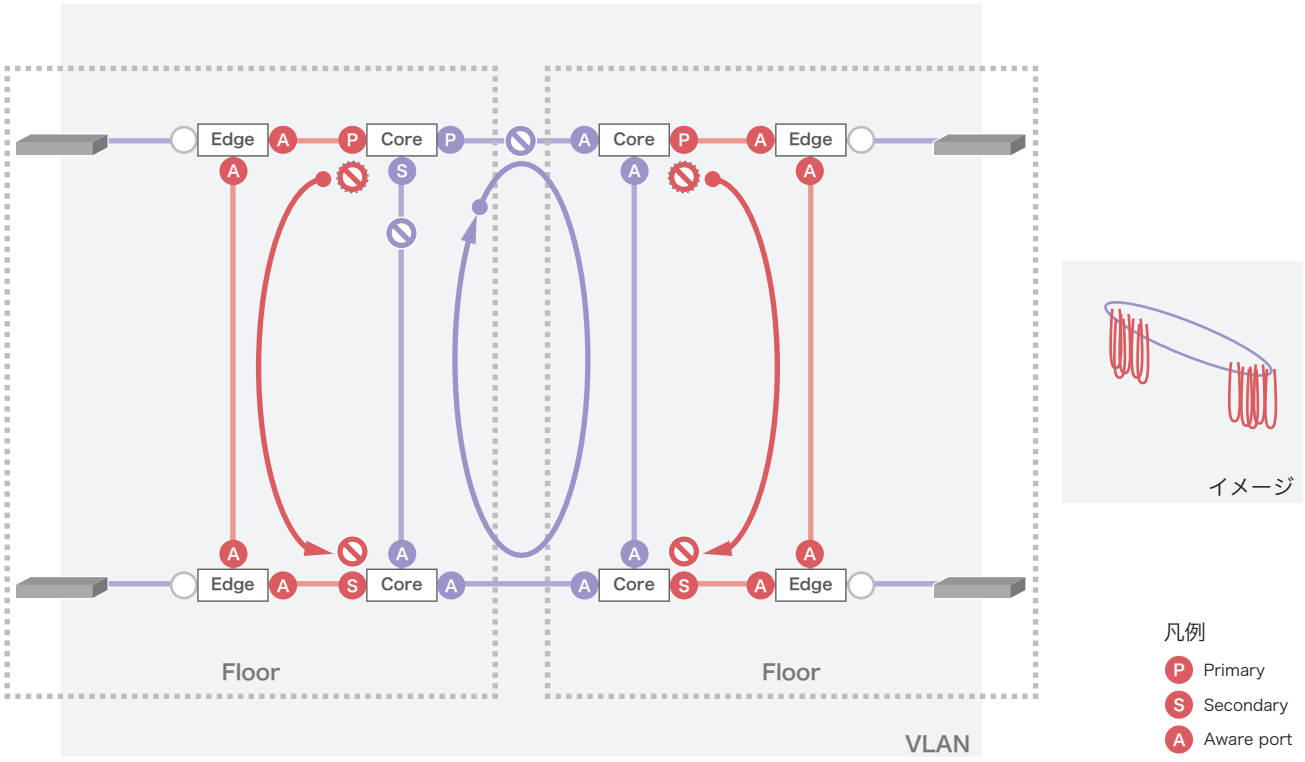

図-2 L2 リングプロトコル構成(2008年〜)

*2　当時はムーアの法則がよく効いていた時代でもあったのです。

■ 様々な基盤に採用されるストレージアレイシステム

サービス基盤のストレージ技術においては、2000年にIBPSを開始した当時からサービス基盤にはストレージアレイシステムを採用しています。主にハイエンド～ミッドレンジクラスのストレージを性能要件に合わせて数種類、調達や技術的なリスク分散も兼ねて採用していました。そして現在もIIJ GIOとNHNにおいては、それぞれの基盤にストレージアレイシステムを採用しており、ストレージの容量は日々増え続けています。

ストレージ運用は、導入当初はストレージに関する知識が乏しいこともあり、ストレージの構成変更や監視はすべて各製品ベンダーに運用を依頼して行っていました。しかし、この運用方法だと例えばLUNを1個作成するだけでもそれなりのキャッシュアウトとリードタイムが発生してしまうため、IIJのスタッフがストレージの操作について学び、自社でストレージの構成変更ができるようになることで、運用コストの削減を図りました。監視については当初からIIJの標準監視サービスを利用していましたが、順調にサービスが成長していく中で、障害が発生してしまった際に標準監視サービスが圧迫され、ストレージ運用チームが障害を認知するのに遅延が生じるようになったため、ストレージ設備専用の監視システムを構築しました。

3.6 2010年代：本格的なクラウド時代へ

■ クラウドサービス「IIJ GIO」誕生

IIJは10年以上に渡るIaaSビジネスの経験とサービス基盤の進化の過程で、IBPSサービスにおける仮想化の適用やプロビジョニングシステム、監視システムの内製、サーバ・ネットワーク・ストレージ設備の大規模更改といった、様々な課題を乗り越えてきました。そのノウハウを結集させて開発したのがクラウドサービス「IIJ GIO」です。ユーザが自社でIT資産を所有することなく、企業のビジネスインフラとして耐えうる高い可用性とセキュリティレベルを維持したサービスレベルの高いシステム環境を、より安価に利用できるという、クラウドのメリットを訴求したサービスでした。

第一弾として、企業の多様なニーズにきめ細かく対応するプライベート型クラウドサービス「IIJ GIOコンポーネントサービス」を2009年11月より提供開始し、続いて、パッケージ化された安価なパブリック型クラウドサービス「IIJ GIOホスティングパッケージサービス」を2010年6月に提供開始しました。

■ IIJ GIO ホスティングパッケージサービス

IIJ GIOホスティングパッケージサービスは、用途に合わせてあらかじめパッケージングされたプランをユーザ自身がオンライン上で選択するだけで、ECサイトや高機能なネットビジネスの情報システム基盤を手軽に構築することができるのが特徴でした。仮想化技術を駆使したサービスアーキテクチャにより、システム要件に応じてサーバリソースを柔軟に選択できるほか、IIJ独自開発の制御機能によりレイヤー2ネットワーク上で複数台のサーバリソースの自動的な割り当ても可能にしました。また、オープンソースソフトウェア（OSS）を採用し、リソースの割り当てを制御するプロビジョニングツールや複雑な運用、監視を自動化する管理ツールを自社開発することで、運用・コスト両面での大幅な効率化を図りました。更に、基盤を集約することによりハードウェア調達コストを抑え、安価な価格設定を可能することで、仮想サーバ1台当たり月額8000円からの低価格を実現しました。2013年9月にはIaaSを操作するAPIを公開し、多数の仮想サーバを一度にデプロイするなどの定型操作をユーザ独自のプログラムを作って自動化したいというニーズに対応しました。

■ IIJ GIOコンポーネントサービス

IIJ GIOコンポーネントサービスは、サーバやストレージ、ネットワークといった各システム構成要素を”コンポーネント（部品）”として提供することで、様々なメニューの中からユーザの要望に合った最適な構成を組み合わせ可能な、柔軟性の高いエンタープライズ向けIaaS型クラウドサービスです。オンプレミス環境と直接広域ネットワークで接続することで、プライベートクラウドとして利用することも可能です。IIJ GIOコンポーネントサービスの主要なコンポーネントは「ベース

サーバ」と2012年8月に提供開始した「仮想化プラットフォームVWシリーズ(以下、VWシリーズ)」です。ベースサーバは、他の顧客と物理サーバを共用する仮想サーバ「Vシリーズ」、1台丸ごと専有できる物理サーバ「Xシリーズ」の2種類を提供しました。VWシリーズは、物理サーバに米VMware社の仮想化ソフト「VMware vSphere ESXi」を搭載した顧客専有環境を、管理サーバであるVMware vCenter Serverと共に管理者権限付きで提供しました。それゆえに、システム構成の自由度ではオンプレミス環境と遜色なかったため、新規にサーバ統合やクラウド構築を検討しているユーザだけでなく、既にVMwareで仮想インフラを構築・運用しているユーザも安心して利用することができました。

IaaSとして、サーバリソース以外の機能も提供しました。1つめは、ベースサーバやVWシリーズで標準提供されるネットワーク機能を強化する「ネットワークアドオン」です。共用のインターネット接続回線を専用回線(プライベート接続)に変更することで、インターネットVPNや閉域網(広域ネットワーク)を使って安全にオンプレミス環境と接続することができます。また、WAN回線のキャリアを分けるマルチキャリア構成にすることもできます。2つめは、「ストレージアドオン」です。金融機関でも利用されるハイエンドなストレージを提供する「スタンダード」、一般的なWebシステムなどのデータ管理に適したミッドレンジの「ベーシック」の大きく分けて2種類を用意し、ネットワーク経由でNASストレージ、FC-SANストレージ、またはiSCSI-SANストレージとして提供しました。3つめは、Oracle Database、MySQLをDBaaS(DataBase as a Service)として提供する「データベースアドオン」です。2012年7月に提供開始したデータベースアドオンは、国内のデータベース市場で圧倒的なシェアを持つOracle Databaseを国内事業者としては初めて月額料金で提供。ユーザのデータベースライセンスへの初期投資や保守費用負担の軽減、投資リスクの回避を実現しました。本サービスでは、多くのリレーショナルデータベースの導入・運用経験を基に、IIJがデータベースのインスタンスの設計・運用を行い、IIJ GIOの仮想サーバ群から利用できるようにしています。更に、閉域網またはインターネットVPN経由で既存のオンプレミス環境との接続も可能としました。2014年5月には価格改定を実施し月額料金を最大56%値下げし、同年10月にはMicrosoft SQL Serverをラインナップに追加するなど積極的なサービス開発を展開しました。4つめは、クラウド上で利用するソフトウェアライセンスを月額料金で提供する「ライセンスアドオン」です。Microsoft SPLAをはじめとして、利用ニーズの高いRed Hat、VMware、Arcserve、Trend Micro各社の製品、サービス等を月額料金で提供しています。

2014年1月にはIIJ GIOコンポーネントサービスのラインアップを強化し、既存の「ベースサーバ Vシリーズ」の機能を拡充した後継サービス「ベースサーバ VシリーズG2」(以下、Vシリーズ G2)を追加しています。Vシリーズ G2では、当時最新だったWindows Server 2012 R2に対応するほか、CPU、メモリとディスク容量を増強し、旧シリーズのWindowsラインアップと比べ、CPUは3倍の最大24ICU[*3]、メモリは6倍の最大48GBを搭載したラインアップを提供しました。また、サーバ設備は東日本及び西日本のサイトで提供することとし、ディザスタリカバリ用途にも利用可能となりました。

■ 第2世代クラウドサービス「IIJ GIO インフラストラクチャー P2」登場

2015年10月、「IIJ GIO」で提供しているIaaSを刷新し、新たなラインアップとして「IIJ GIOインフラストラクチャーP2」(以下、IIJ GIO P2)を提供開始しました。これまでIaaSのラインアップとして、オンラインで手軽に導入できるパブリッククラウド「IIJ GIOホスティングパッケージサービス」と、多様なITリソースを組み合わせてシステムを構成できるオーダーメイド型の「IIJ GIOコンポーネントサービス」を提供してきましたが、IIJ GIO P2はより信頼性・処理性能を高めたパブリッククラウドと、オンライン申し込みで即時利用を可能にしたプライベートクラウドを1つに融合し、幅広いユーザのニーズをすべてカバーするサービスを目指しました。IIJ GIO P2は、仮想サーバを中心とした共有リソースを提供する「パブリックリソース」

＊3　ICUはCPU性能指標です。24ICU=6Core×2相当。

と、VMware仮想化環境と物理サーバを専有リソースとして提供する「プライベートリソース」、パブリックリソースとプライベートリソースの両方のサーバで利用できる「ストレージリソース」から構成され、ユーザは最適なリソースを組み合わせてシステムを構築できます。更にIIJ GIO P2はマルチキャリア対応やプライベートセグメントの延伸など外部接続性にも優れ、オンプレミスや他社クラウドサービス環境とシームレスな連携が可能です。

パブリックリソース：仮想サーバを中心とした共有リソースを提供し、開発環境、簡易なWebサービスから高いI/O性能が求められるオンラインゲームやECサイトのプラットフォームまで、幅広い用途に対応できるパブリッククラウドで、ユーザは用途に応じて特徴の異なる3つのタイプから自分に最適なサーバリソースを選択できます。

- 「性能保証タイプ」は、安定した処理性能を必要とするユーザ向けに、CPUが確実に割り当てられる仮想サーバを提供。月額固定料金で安心して利用できます。
- 「ベストエフォートタイプ」は、CPUを分配利用することで低コストを実現した仮想サーバを提供し、1時間単位の従量課金で、コストの最適化を図ることが可能です。
- 「専有タイプ」は、高負荷に耐えうる高いI/O性能を求めるユーザ向けに、仮想化された専有サーバを提供します。物理的に他のサーバから切り離されたセキュアな環境で、SSDまたはサンディスク社製高速フラッシュストレージを搭載したサーバを利用できます。

3つのサーバタイプの組み合わせやタイプ変更はオンラインで自由に行うことができ、更に稼働中の仮想サーバからOSイメージを専用領域に保管することで、それを元にした迅速なサーバ構築が可能です。運用負荷が軽減し、高負荷時のスケールアウトやパッチの適用にもスピーディーに対応することができます。カスタムOSイメージを保管する専用領域は保管容量に応じた従量課金のため、無駄なコストも発生しません。

プライベートリソース：オンプレミス環境で構築されたシステムをそのまま移設でき、企業の基幹システムにも利用できる信頼性の高いプライベートクラウドです。VMware仮想化環境と物理サーバを中心としたラインアップを提供し、ユーザ専用のリソースとして利用することができます。また、これまでのIIJ GIOコンポーネントサービスでは対応していなかったオンライン申し込みが可能になり、ユーザはコントロールパネル経由でサービスの即時利用（標準モデルの場合）とサーバリソースのセルフ管理が可能となり、1日単位で必要なリソースを増減できます。最大24コア対応のCPU、192GBのメモリ、帯域10Gbpsのネットワーク対応など、これまでのIIJ GIOコンポーネントサービスに比べて、選択できるサーバの性能が向上しています。更にディスクなどのサーバスペックはオンライン上でカスタマイズすることが可能で、より大容量なメモリの搭載により、システムの集約率を高め、ユーザ自身でシステム要件に沿ったサーバを設計・構築することができます。

■ IIJ GIO P2ラインナップ拡充と西日本リージョン提供開始

2018年6月にはIIJ GIO P2で提供している「仮想化プラットフォームVWシリーズ（以下、P2 VWシリーズ）」のサーバ性能を強化し、従来提供しているCPUコア数の倍となる48コアのCPUリソースを搭載した「VW48-1024-FC-10G」、及び96コアを搭載した「VW96-1024-FC-10G」を追加し、それぞれ2018年6月1日、2018年10月以降より提供開始しました。追加する品目は、AIの情報処理やSAP S/4 HANAなど、取り扱うデータの処理にハイコア・ハイメモリが求められるアプリケーションのインフラ需要を想定したものです。

同じく2018年6月、IIJ GIO P2において西日本リージョンを開設し「パブリックリソース」を提供開始し、同年10月より「プライベートリソース」「ストレージリソース」を提供開始しました。東日本リージョン、西日本リージョン間はIIJの提供するプライベートバックボーンサービスで結ばれており、広帯域なリージョン間ネットワークを無償で利用することができます。また、地理的に十分に離れたリージョン間でサービスを利用することで、ユーザはBCP(事業継続計画)に基づいたシステム構成をとることができるなど、利用用途の拡大を図りました。

2020年7月にはP2 VWシリーズの機能を強化し、仮想マシンを容易かつ低コストでバックアップできる「バックアップセット/VWオプション」を提供開始しました。バックアップセット/VWオプションでは、米ルーブリック社のバックアップ製品「RCDM(Rubrik Cloud Data Management)」を活用し、仮想マシンのバックアップに必要なコンポーネントを一括で提供します。本オプションを利用することにより、ユーザはバックアップ設定メニューから必要なプランを選択するだけで容易にバックアップ及びリストアが行えます。取得したデータは暗号化され、IIJのクラウド上にあるバックアップサーバに保存するため、ユーザ側でのバックアップシステムの構築や運用が不要になり、コスト削減、運用負荷軽減にもつながります。

■ L2 MLAG構成からL2 over L3構成へ

サービス基盤のネットワーク技術においては、2014年以降はL2 MLAG構成を採用しました。収容ノードのインタフェースが10GbE以上でも一定のスケールを実現できるのがメリットです。MLAGを実装しているのはデータセンタースイッチメーカのため、ハードウェア、ソフトウェア共にデータセンター向けに最適化されていますが、どのようなファシリティにも適用できる設計ではない点に注意が必要です。どちらかといえば広大な面積を持ったフロア向けの設計であるため、機器を設置するフロア面積や、給電・冷却の能力によってはスイッチの収容ポートを使い切るだけの機器が設置できず、シャーシを採用する前提(収容ノード数を確定した前提)でフロア構成が設計できる場合でないと、効率良く利用できないといった状況に陥りやすいと考えられます。

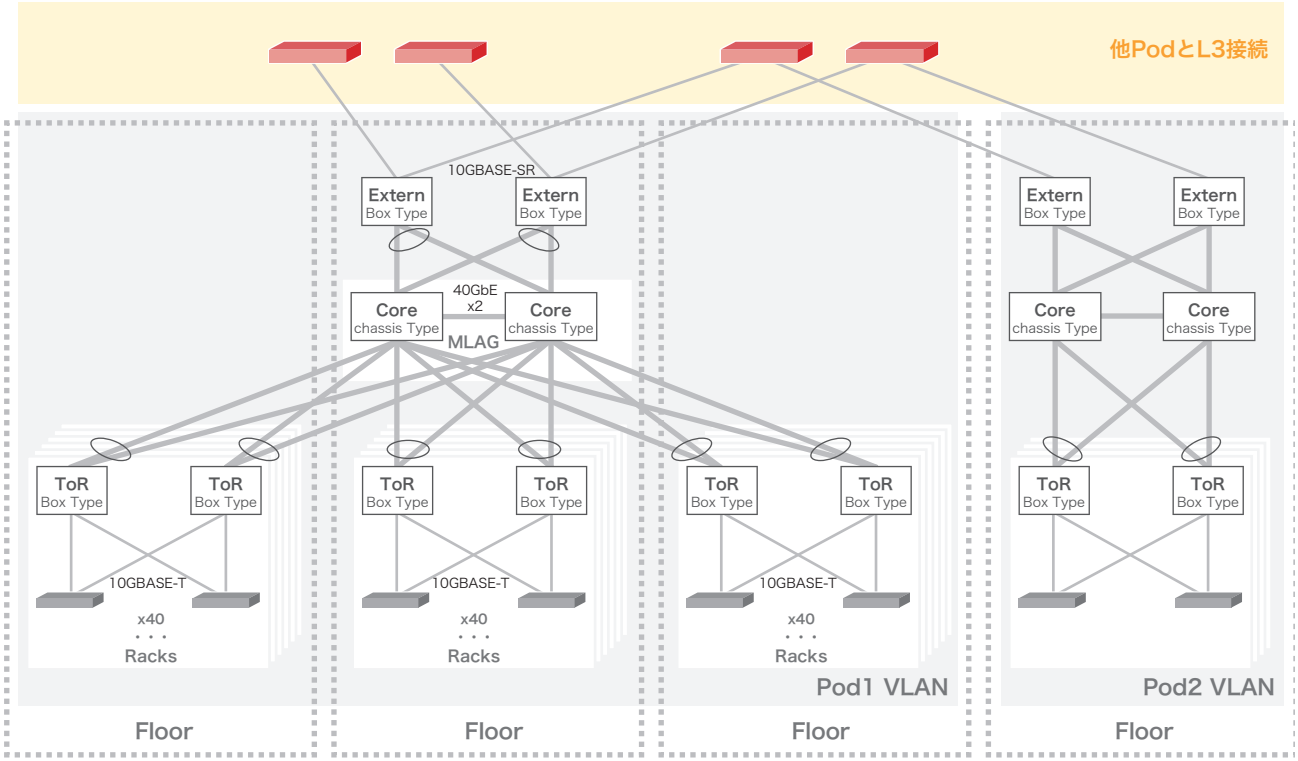

図-3 L2 MLAG構成(2014年〜)

2017年以降はL2 over L3構成を採用しました。MLAG同様のメリットを踏まえつつ、インターネットで長く利用され実績のあるL3技術で柔軟な物理トポロジを実現できるのが強みです。ただし、当時のVxLANの実装では設計に制約があり、どのVTEPにどのVLANを設定するかなど、動的な制御を実現するには標準実装が成熟しておらず、静的な設定で管理できる現実的な位置にVTEPを置いた設計としました。

■ ストレージ設定の自動化

「IIJ GIO」では、ユーザからストレージの申し込みがあった場合、2010年頃まではストレージを運用しているエンジニアが手動でストレージとFCスイッチの設定変更を行っていました。この頃まではストレージの設定変更を行う業務は多くても週に1回程度と頻度が少なかったため、エンジニアによる手動の設定変更でもサービス仕様として設定されているリードタイムに十分に間に合いました。しかし、2010年を過ぎると週に1回の業務が週に数回、更には1日に数回発生するようになり、いずれ人手を介した作業が困難になると予想されたため、ストレージの設定変更をすべて自動化するようなシステムを設計・構築しました。

ストレージ設定の自動化を行うためには、市販のアプリケーションなどを利用すれば簡単なのですが、市販のアプリケーションはストレージ設定の自動化だけでなく、それ以上の機能を有するものが多く（かつ高価なため）、ストレージの制御部分は自作をしました。ストレージ制御はPythonやRubyなどのスクリプト言語を使って記述しています。本来、各種言語からストレージの構成変更を行う場合、ストレージ管理の仕様であるStorage Management Initiative - Specification（SMI-S）や独自APIを経由して構成変更を行うことが望ましいです。当時使用していたストレージやFCスイッチそれぞれ単体ではSMI-SやAPIに対応しておらず、別途専用の（高価な）アプリ

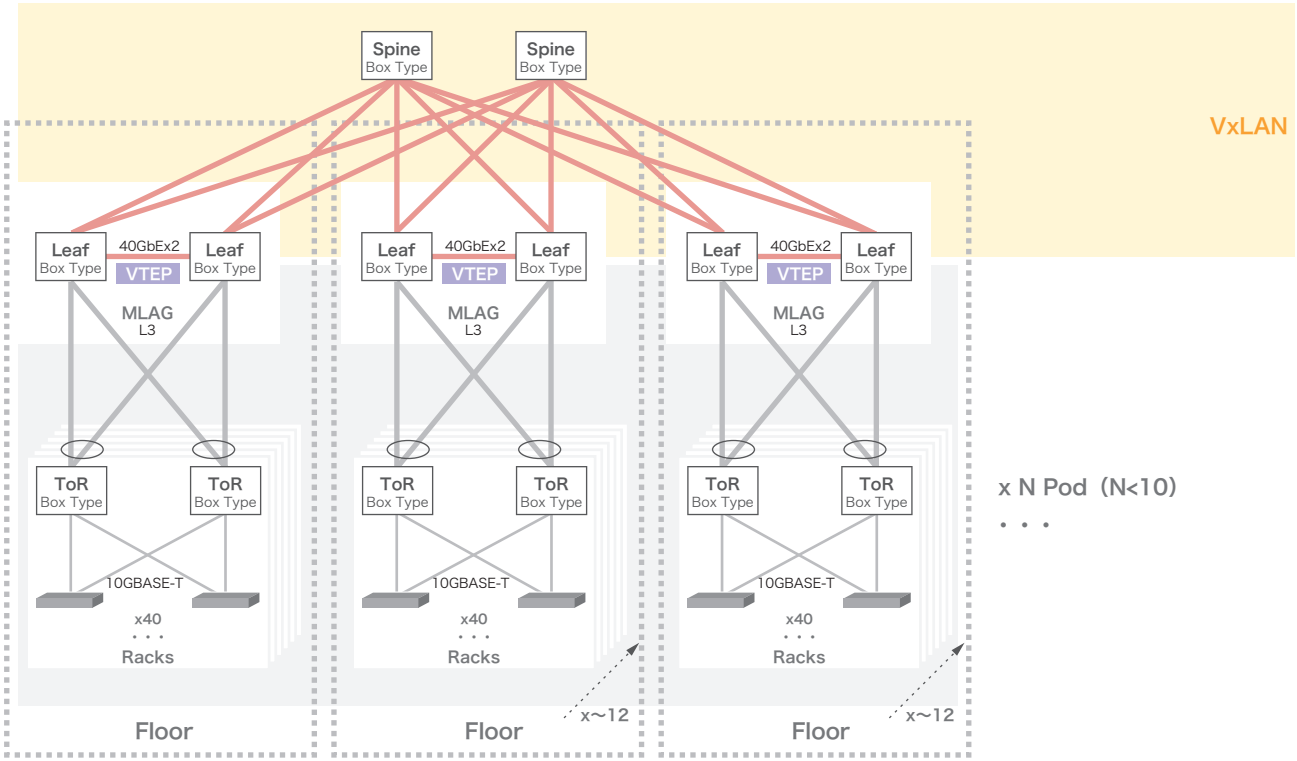

図-4 L2 over L3構成（2017年～）

ケーションが必要であったため、標準で提供されているストレージを制御するコマンドラインアプリケーションを使用しました。コマンドラインアプリケーションの入出力は人間が操作することを想定して作られておらず、とりわけ出力はスクリプト言語で制御するには非常に扱いにくい形式であることが多く、作成したプログラムも相当面倒な文字列処理を行う必要がありました。また、使用していた機器の一部は、コマンド及びパラメータに誤りがあってもリターンコードが0(正常終了)を返すため、独自のエラー処理を追加する必要もありました。

最も手間がかかるのは、ストレージアレイシステムやFCスイッチのファームウェアをバージョンアップするときです。バージョンアップ後のコマンドライン出力結果がバージョンアップ前と比較して変更が発生する製品もあり、本番でバージョンアップする前に開発環境で事前のテストを行う必要がありました。最近のストレージやFCスイッチは、構成管理を行うソフトウェアであるAnsibleなどへの対応が進んできていることから機器単体でのAPI実装が進み、かつPythonなどでストレージ制御に使えるモジュールを各ストレージベンダーが提供するようになったため、当時と比べると非常に簡単にプログラムを組むことができます。2010年以降のストレージ選定においては引き続き前世代からの調達方針を踏襲していましたが、性能要件の拡大と我々の採用条件をクリアするストレージアレイ製品の増加により、採用するストレージ製品も増えてきています。

3.7 2020年代～現在:更なる進化を目指して

■「IIJ GIOインフラストラクチャーP2 Gen.2」提供開始

IIJ GIOブランドで開発・提供してきたパブリック型IaaSとプライベート型IaaSを完全統合し、オンプレミスからの移行を容易にする次世代モデルのIaaSを「IIJ GIOインフラストラクチャーP2 Gen.2(ピーツー ジェンツー)」(以下、GIO P2 Gen.2)として2021年10月1日に提供開始しました。

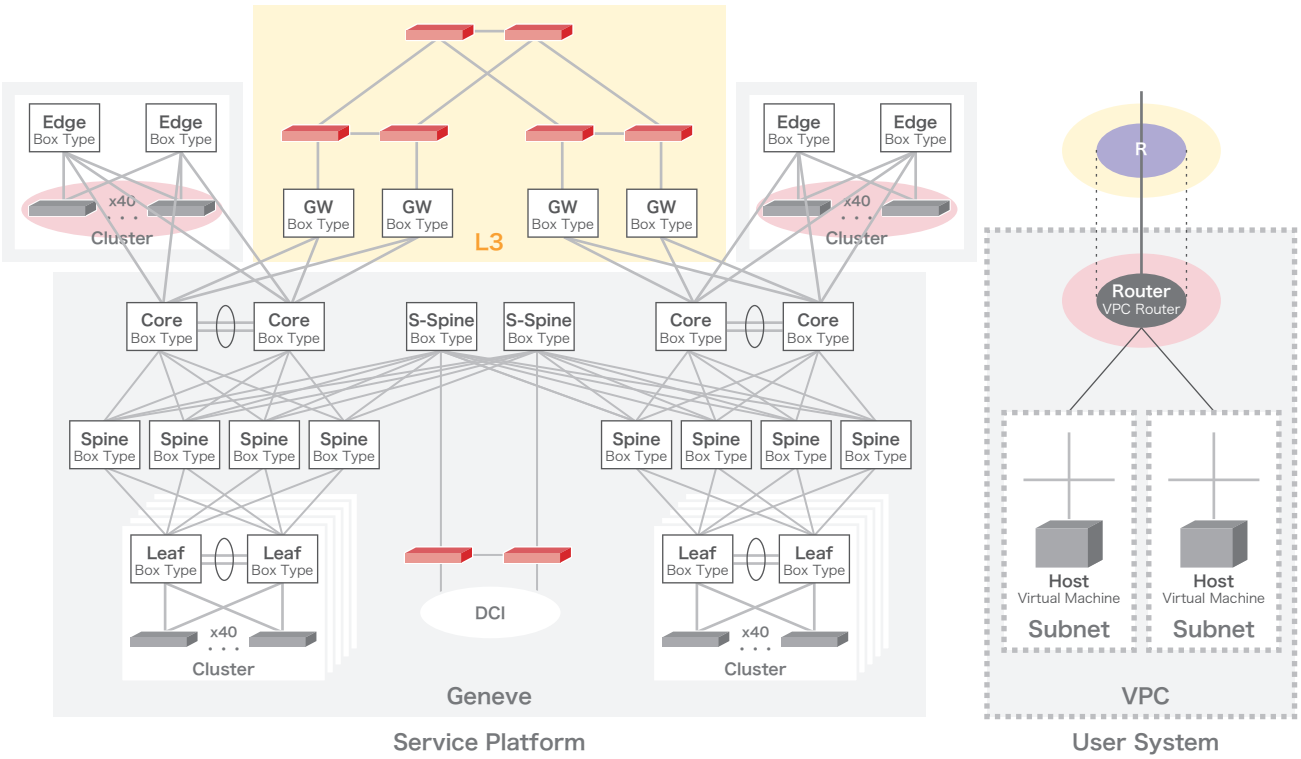

図-5 コントロールプレーン／データプレーン完全分離構成(2021年～)

GIO P2 Gen.2の特徴は、仮想基盤にVMwareを採用しオンプレミス環境の設計思想や運用体制をそのまま持ち込み可能とすることで、GIO P2から引き続きオンプレミスのVMware環境からのクラウド移行需要を狙ったものになります。もちろん、後継サービスとして既存ユーザの新たな受け皿としての役割も担っています。また、GIO P2 Gen.2は一般的なプライベートクラウドのような「サーバ単位」ではなく、「仮想リソースプール」から最小「1vCPU/4GBメモリ」というリソース単位で自由に仮想マシンを作成可能です。つまり、現在ユーザの環境で動いているマシンスペックをそのまま移行することができます。この形式であればイメージを持ち込んだり、P2VやV2Vを実施することで既存環境の物理マシン・仮想マシンを移行することも可能です。ファイルサーバやActive Directory、データベースなど、企業のIT環境に必要なコンポーネントを備えた各種マネージドサービスなども提供される上、ハイパーバイザや、サーバ、ストレージ、ネットワークなどのハードウェアは抽象化されるため、ユーザは機器の差異を意識する必要はありません。これは、GIO仮想化プラットフォーム VWシリーズが抱えていた、ユーザがハイパーバイザの管理を行うことで生じる脆弱性対応のためのソフトウェアアップデートの作業や、ハードウェア更改のための移行作業をユーザ自身が行う負担を大幅に軽減します。他社パブリッククラウドとの閉域接続を提供するネットワークサービスなど、IIJの各種サービスと連携することにより、パブリッククラウドへの将来的な移行を見据えた利用も可能です。もちろん、プライベートクラウドと同様に自由にリソースを選んでシステム環境を構成できる「フレキシブルサーバリソース(FSR)」に加えて、VMware vCenter Serverをフル権限で利用できる従来のP2 VWシリーズと同等の「デディケイテッドサーバリソース(DSR)」も継続して提供しています。

GIO P2 Gen.2「フレキシブルサーバリソース(FSR)」では、米VMware社が提供するサービスプロバイダ向け製品であるVMware Cloud Director(VCD)を採用し、ハイパーバイザ(vSphere)層を利用者から隠蔽することで、vSphere並みに柔軟なリソース制御権限をユーザに提供しつつ、ハイパーバイザやハードウェアのライフサイクル管理をサービスプロバイダとしてIIJが受け持つという、新しい共同責任モデルを定義することでハイパーバイザのネットワークをIIJが管理・運用できるようになりました。また、サービスの機能として移行機能を提供し、お客様は最小限の停止時間、操作でクラウド移行を実現できます。併せて、ハイパーバイザ層のネットワークを効率的に運用するため、vSphereと統合的なインテグレーションが行えるVMware NSX-T DataCenter(以下、NSX-T)を採用し、IaaSのネットワークを大幅に改善させました。GIO P2 Gen.2では、レイヤー3で構成したIPファブリックのアンダーレイネットワーク上にNSX-Tでオーバーレイネットワークを構成し、テナントごとのネットワークを完全に分割しVPC(Virtual Private Cloud)として提供できるよう設計しています。これまでIIJ GIOで培った大規模サーバプールの運用ノウハウと組み合わせることで、物理的なコンピューティングリソース(CPU、メモリ、ストレージ)の配置に縛られることなく、利用者にリソースを割り当てることを可能にしています。

■ SDN技術を活用したオーバーレイネットワーク

複数のユーザを収容するIaaSにおいて、リソースを効率的に提供するため物理リソースを共有すると共に、ユーザ間のセキュリティを保証することは、基盤を設計する上で最低限求められてきたことです。ユーザのコンピューティングリソースであるサーバの仮想化技術は十分に成熟した製品が市場にありまし

たが、ネットワークの仮想化は既存のプロトコルとの相互接続性も担保する必要もあり、実装は慎重に進められてきました。近年、ハードウェアの性能向上とSDN技術の発達によりネットワークの仮想化技術も大規模なネットワークで採用できる技術になってきています。GIOの基盤も早くからネットワークの仮想化技術の1つであるオーバーレイネットワークを評価し、それぞれの世代で信頼できる技術を採用してきました。

GIO P2 Gen.2では、これまで性能を確保するためネットワークハードウェアの機能で終端していたオーバーレイネットワークを、ホスト内の仮想スイッチで終端する方式に変更したことで、ユーザ間の分割はもとより、ユーザシステムと物理基盤システムの完全な分離を実現しています。これにより、基盤システムの変更によるユーザシステムへの影響はより小さくなり、新しい機能の追加が容易になりました。

また、物理層と疎結合であるということは、IaaSをマルチサイトで展開する上でも有利に働きます。既存のGIOと同様に、東日本・西日本といった少数の拠点に大規模に設備を展開し、その間をシームレスに接続することができるだけでなく、災害に備えより小さな規模で各地に分散配置することも容易になることが期待できます。

反面、これまでハードウェアの制限で個々に分割・管理していた各デバイスの設定は、ソフトウェアでの処理に変わることで制限がなくなり非常に膨大な情報量になります。人が把握できる量を超えるため、これまでのように人で変更して回る運用は品質を担保する上で変えていかなくてはいけません。同時に、ユーザにタイムリーに利用していただくため、これまで静的に設定していた各デバイスも、動的に変更可能にする必要があります。これらを実現するためオーケストレーター(商用プロダクトをベースに不足している機能や追加機能を自社開発)を採用し、設定情報を一元管理しています。設定情報が一元化されることで、連携サービスもAPIを介してのシステム間での処理が可能になり、関連サービスも安全でスピーディーに利用を開始できるようになっています。

3.8 まとめ

IIJの30年の歴史をサービス基盤の観点から振り返りました。様々なIIJサービスを支えるサービスホスト基盤は、安定性と効率性のバランスを求めながら日々生まれてくる技術革新を見極め、時流に合わせて取り込みながら進化を続けています。今年で24年目を迎える「IIJ GIO」は、今もなおビジネスインフラに求められる多様なニーズに応えるため、サービスラインアップを拡充し続けています。近年では、AI技術をベースとしたデジタル社会基盤の需要が高まってきており、それを実現する超高密度AI計算基盤の提供に向けて取り組みを進めているところです。IIJでは現在開発中のサービスや基盤技術について、今後も市場のニーズに沿う形で皆様に提供し続けてまいります。

執筆者:
木村 真理(きむら しんり)
IIJ クラウド本部クラウドサービス2部 部長 兼 クラウド本部技術開発室長。
2001年2月より、大規模サービス基盤、クラウド基盤の開発、運用業務、クラウドサービスの提供業務に従事。併せて、技術開発、人材開発等の役割をもつ部署を担当。

September 2023 Vol.60

IIJ
Internet Initiative Japan

株式会社インターネットイニシアティブ(IIJ)について

IIJは、1992年、インターネットの研究開発活動に関わっていた技術者が中心となり、日本でインターネットを本格的に普及させようという構想を持って設立されました。
現在は、国内最大級のインターネットバックボーンを運用し、インターネットの基盤を担うと共に、官公庁や金融機関をはじめとしたハイエンドのビジネスユーザに、インターネット接続やシステムインテグレーション、アウトソーシングサービスなど、高品質なシステム環境をトータルに提供しています。
また、サービス開発やインターネットバックボーンの運用を通して蓄積した知見を積極的に発信し、社会基盤としてのインターネットの発展に尽力しています。

本冊子の情報は2023年9月時点のものです。

IIJ-MKTG019-0060

株式会社インターネットイニシアティブ
〒102−0071 東京都千代田区富士見2-10-2　飯田橋グラン・ブルーム
E-mail：info@iij.ad.jp　URL：https://www.iij.ad.jp