URL　https://journal.ntt.co.jp/article/38184
DOI　https://doi.org/10.60249/26025002

DCIの技術開発におけるマルチベンダコンポーザブルサーバ実現の取り組み

NTTソフトウェアイノベーションセンタでは，IOWN（Innovative Optical and Wireless Network）構想実現に向けたDCI（Data-Centric-Infrastructure）の技術開発の取り組みを進めています．本稿では，DCIの構成要素であるコンポーザブルサーバにフォーカスし，マルチベンダ機器を組み合わせたインフラ構築・運用のための仕組み（機器管理インタフェースやフレームワーク）とNTTにおけるマルチベンダコンポーザブルサーバ実現へ向けた課題について紹介します．

キーワード：#DCI，#コンポーザブルサーバ，#マルチベンダコンポーザブルサーバ

幸田（こうだ）　健介（けんすけ）
二ノ方（にのかた）　一生（かずお）
NTT ソフトウェアイノベーションセンタ

はじめに

NTTソフトウェアイノベーションセンタ（SIC）では，IOWN（Innovative Optical and Wireless Network）構想実現に向けたDCI（Data-Centric-Infrastructure）の技術開発の取り組みを進めています．DCIはIOWN Global Forumが定義する全体アーキテクチャにおいて，分散データセンタ環境やヘテロジニアスなコンピューティング環境における高効率なデータ処理を可能とする基盤レイヤとして位置付けられています．IOWNの全体アーキテクチャにおける重要な基盤の1つであり，これまでの特集記事において，IOWN Global ForumでのDCIの機能アーキテクチャやコンピュートクラスタのリファレンス実装モデルの文書化についての取り組み[(1)]や，2025年日本国際博覧会（大阪・関西万博）のNTTパビリオンにおけるDCIを活用したハードウェアリソースの効率的な利用と低消費電力化の取り組み[(2)]について紹介してきました．また，NTTではDCIが複数のコンポーザブルサーバやGPUサーバ等ネットワークを介して接続したハードウェアと，この相互に接続されたCPUやGPU等のリソースを最適に割り当てる「DCIコントローラ」によって構成されるものとしています．

本稿では，デバイスを柔軟に組み合わせて利用が可能なコンポーザブルサーバを用いたDCI構成に注目します（**図1**）．コンポーザブルサーバとはホストサーバと

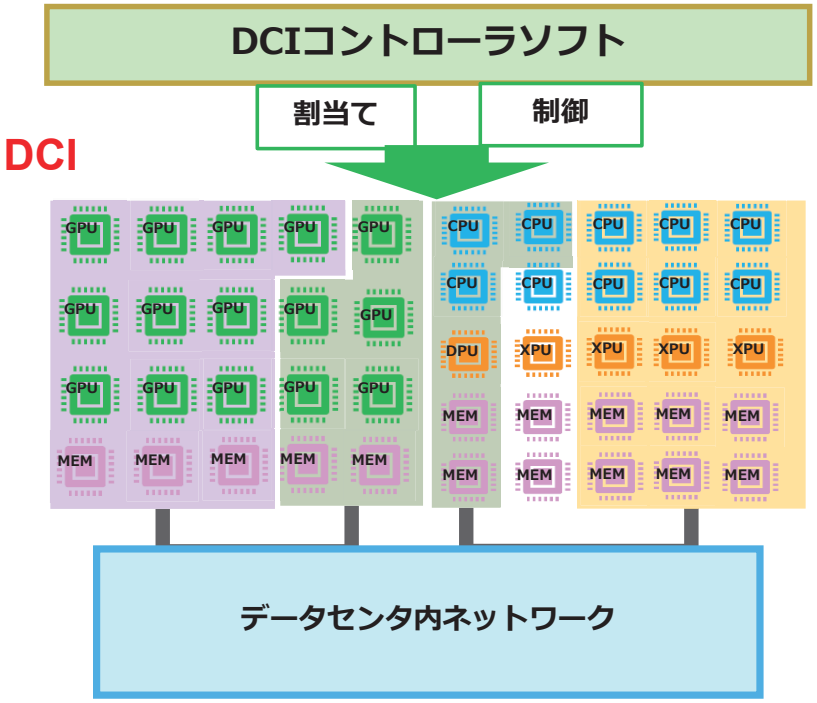


図1　コンポーザブルサーバを用いたDCI構成

PCIe/CXL拡張ボックス（リソースボックス）をPCIe/CXLファブリックスイッチ（ファブリックスイッチ）で接続しCPU，GPU，ストレージ，CXLメモリを自由度高く組み合わせて利用できるものです（**図2**）．

NTTではコンポーザブルサーバの製品ベンダ各社とも連携して，複数ベンダの製品を組み合わせたシステムの構築や運用の検証を行ってきました．また，DCIコントローラソフトを介して，複数ベンダの製品を組み合わせたマルチベンダコンポーザブルサーバの構築およびコンテナ基盤と連携した運用を可能とすることをめざしています．

本稿では特に，DCIコントローラソフトの実現において活用を検討している，マルチベンダコンポーザブルサーバの管理標準やフレームワークについて，その成立の経緯を含めて紹介します．その次に，NTTにおけるマルチベンダコンポーザブルサーバを実現するための課題を紹介します．管理標準としては，データセンタインフラを制御することを目的にDistributed Management Task Force（DMTF）[3]によって仕様策定され，サーバやネットワーク機器に機能実装されてきたRedfish[4]インタフェースについて紹介します．また，Open Fabrics Alliance（OFA）[5]が提唱する，Sunfish Framework[6]について詳述します．Sunfishは，ベンダが異なるコンポーザブルサーバのハードウェア（サーバ，メモリ，アクセラレータ等）を，Redfish等の標準的な管理インタフェースを通じて統一的かつ一元的に管理することができます．また，物理構成に依存しないコンピューティングリソースの管理と動的構成（ライフサイクル管理）を可能とする論理モデルを提供することから，NTTが実現をめざすDCIコントローラソフトでも活用可能なサービスフレームワークと考えています．Sunfishの2025年12月時点のバージョンは0.5版であり，OFAにおいて，今後のバージョン1.0版の仕様リリースに向けた活動が活発に進められています．

マルチベンダコンポーザブルサーバに関する標準化の動向

ここではマルチベンダコンポーザブルサーバの構築・運用を可能にするための仕組みである，標準管理インタフェースやフレームワークの設計について紹介します．

近年にみられるようなクラウドや仮想化，ハイパースケール環境の普及により，大量のサーバの管理を，統一APIを用いて自動化する必要が生じています．一方で，サーバ機器等の製品の管理インタフェースとして，これまでIPMI（Intelligent Platform Management Interface）が広く用いられてきましたが，IPMIは1998年に策定された古い規格であり，大量のサーバをREST APIで扱うことはできず，また拡張性に乏しいといった問題もありました．そのような経緯もあり，2015年，DMTFがRedfishを標準化しました．Redfishは，RESTful API/JSON/HTTPSベース，セキュリティの強化，構造化されたハードウェア構成情報の提供，スケールアウト管理に適した設計といった特徴があります．具体的にはRedfishのデータモデルは階層的なURI構造を通してアクセス可能なツリー構造（リソースツリー）となっています．

大手ベンダもRedfishを採用し，モダンなデータセンタ管理全般に用いられるようになってきています．またコンポーザブルサーバのように，データセンタ内にあるサーバやPCIe/CXL拡張ボックス内のデバイスをプール化（リソースプール）し，動的に再構成しながら使用するといった新たな需要も出てきています．この需要にこたえるべく，DMTFは2017年ごろからComposabilityの概念を標準仕様に追加し始めており，コンポーザブルサーバの制御にRedfishが適用できるようになってきています[7]．

また，Redfishのようにインタフェースを規定するだけでなく，コンポーザブルサーバを運用しやすくするためのフレームワークを策定する動きとして，OFA Sunfishがあります．Sunfishは，コンポーザブルサーバを運用管理するためのオープンアーキテクチャであり，CPU，メモリ，ストレージ，GPUなどのデバイスをネットワーク越しに接続し，それらを柔軟に組み合わせて論理的なサーバ（論理サーバ）を構成するフ

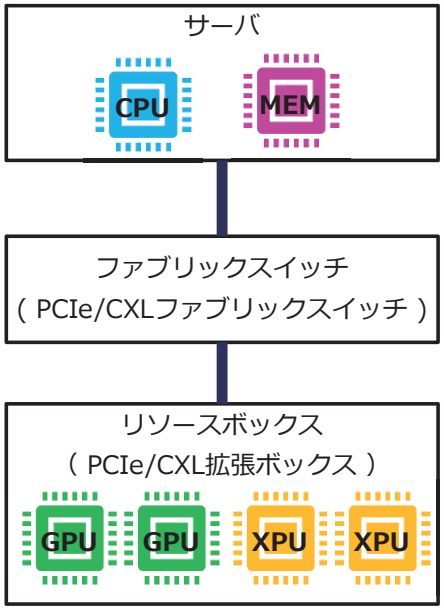


図2　コンポーザブルサーバの概要

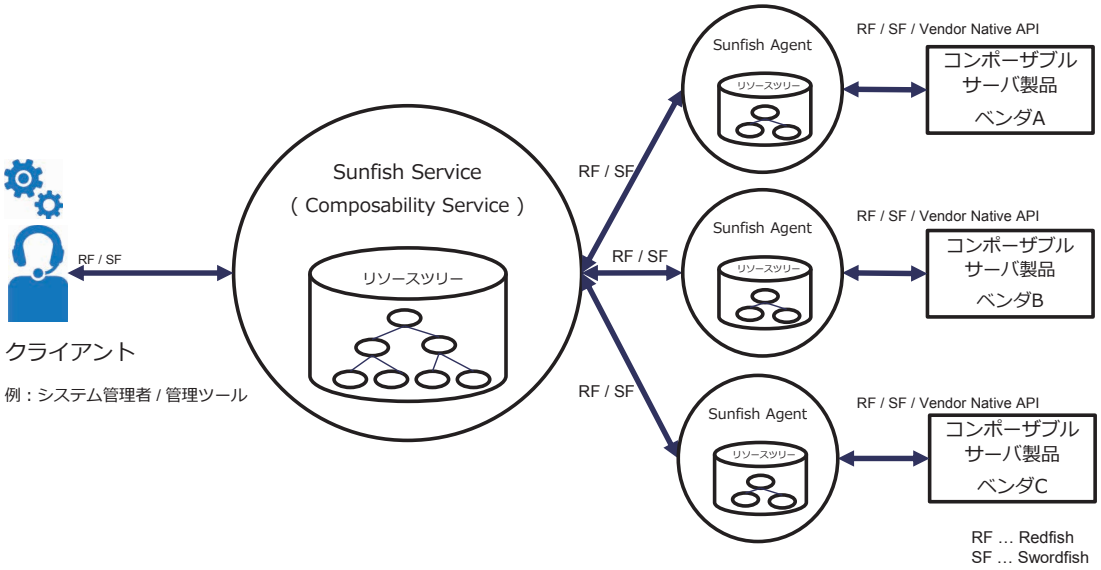


図 3　Sunfishのアーキテクチャ概要

レームワークです．Sunfishのアーキテクチャを**図 3** に示します．

ここで，Sunfishの特徴として以下の 3 点があげられます．

① ベンダニュートラル
② 抽象化されたリソース表現モデル
③ 標準ベースでオープンな管理インタフェース

1 番目の特徴は，ベンダニュートラルなところです．各ベンダ製品がRedfishを提供している場合，それぞれの名前空間が重複する可能性があります．Sunfishでは，それら製品に対してSunfishの管理空間内でユニークなIDを割り当てることでこの問題に対応しています．また，ベンダ独自のAPIやツールを提供している製品に対応するために，インタフェースをRedfishに変換するレイヤ（Sunfish Agent）を設けています．このように複数の異なるベンダ製品を統一的かつ一元的に管理することを可能とするリポジトリ・サービス設計となっています．

2 番目の特徴は，抽象化されたリソース表現モデルであることです．Sunfishは，Sunfish ServiceにおいてSunfish Agentを介して収集されたサーバ，ストレージ，ファブリック構成に関する情報を，Redfishのリソースツリーとして抽象化して管理します．これにより管理者や管理ツールといったクライアントがどのサーバにGPUがあり，どのメモリがどこにつながっているかなど，物理面を意識せずに論理レベルでリソースをプール，割り当て，再構成できるよう設計されています．

3 番目の特徴は，標準ベースでオープンな管理インタフェースであることです．API は DMTF Redfish および SNIA Swordfish[8] を利用しています．これらが提供するRESTful APIをとおしてリソースの管理，論理サーバの構成を実施できるため，将来的な拡張性や異種ハードウェアの統合にも耐え得るようになっています．

続いてSunfishの管理方法について説明します．各機器のRedfishのリソースツリーを基に全体を管理するRedfishのリソースツリーが統合，構成されます（**図 4**）．各機器はSunfish Serviceへの登録時，Sunfishの管理空間内でユニークなIDがSunfish Serviceから割り振られ，この仕組みによって各機器のRedfishのリソースツリーの名前空間の重複は回避されます．この結果，クライアントは，Sunfish Serviceが割り振ったIDに基づいて，統合されたリソースツリーを操作することが可能になります．また，Sunfish ServiceはIDの対応表を保持し，任意のSunfish Agent配下のコンポーザブルサーバ製品を操作する際には，そのSunfish Agentが管理する名前空間のIDに変換したうえで操作を要求します．

最後に，Sunfishを用いた論理サーバの作成の流れについて説明します．なお，簡単のために論理サーバの構成に必要なリソースの情報は事前に把握できているものとします．

- クライアントは，Sunfish Serviceが管理する名前空間のIDを用いて，Sunfish Serviceに対して論理サーバの作成を要求する．
- Sunfish Serviceはクライアントの要求に従い，記載されているリソースを持つSunfish Agentに対して論理サーバを構成するリソースの確保を要求する．このとき，リソースのIDは，そのSunfish Agentが管理する名前空間のIDに変換される．
- Sunfish Agent は Sunfish Service の要求に従い，コンポーザブルサーバ製品に対するリソース確保を要求する．

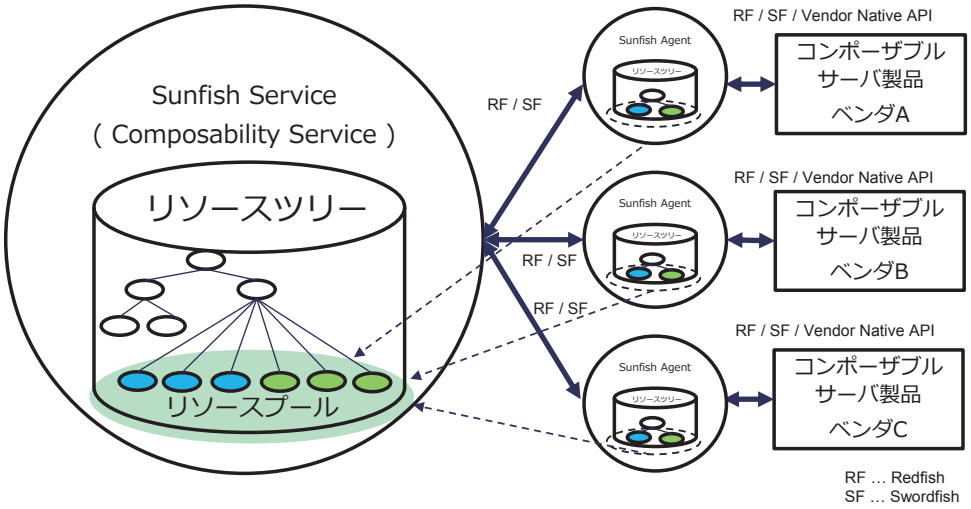


図 4　Sunfish Serviceによるリソースの統合管理

・コンポーザブルサーバ製品はSunfish Agentの要求に従ってリソースを確保する．
・Sunfish Serviceは自身のRedfishのリソースツリーの構成情報を更新する．

Sunfishではリファレンス実装にも取り組まれており，上述した機器登録の機能が実装されています．一方で，上記の論理サーバの作成の流れを実現するには，Sunfish Agentが持つリソースをどのように認識し，Sunfish ServiceのRedfishのリソースツリーへどう組み込むか等を実装する必要があります．また，Sunfish Agentをどの単位で構成するかといったアーキテクチャとしての検討も必要です．

マルチベンダコンポーザブルサーバの実現へ向けた課題

マルチベンダコンポーザブルサーバを実現するためには，私たちは大きく以下の2つの課題に取り組む必要があると考えています．

■統合管理の実現へ向けた課題

前述のとおり，OFAにおけるSunfishのようにマルチベンダコンポーザブルサーバの統合管理に向けた標準の策定も進んでいますが，実運用までを考慮すると，製品とのギャップやアーキテクチャ上の課題がいくつか残っていることが，これまでのSICによる検証を通じて分かってきています．ここではその中でも2つの課題について紹介します．

⑴　リソースの抽出，管理方法

Sunfishでは各製品のリソースを，Sunfish ServiceのRedfishのリソースツリーで統合管理します．ここでは，Sunfish Agent配下のリソースの情報をどのようにしてSunfish Serviceに通知，統合させるかという問題があります．物理的な接続構成により依存関係にあるリソース群も存在するため，その考慮も必要となります．私たちはRedfishのComposability機能を備えるファブリックスイッチ製品を対象に現在Sunfishを介した構成に必要な機能の実装および構成操作を試行しています．しかし，リソースボックス内のデバイスをサーバに割り当てる際に，製品仕様によってはデバイス単位での割り当てではなく，それらを取りまとめるポートに対してデバイスを割り当てる手続きとなっています．またポートがデバイスと一対一対応していないこともあり，こうした製品仕様を考慮に入れたリソースの抽出機構が必要であることが分かってきています（**図5**）．

⑵　Sunfish Agentの構成単位

Sunfish Agentの構成単位も検討が必要な課題の1つです．Sunfish Agentの構成単位は**図6**に示すとおり，大きく以下の2パターンが存在します．

・パターン1　ホストサーバ，ファブリックスイッチ，リソースボックスごとにSunfish Agentを構成：この場合の利点は，ホストサーバ，ファブリックスイッチ，リソースボックスをそれぞれ異なるベンダ製品を用いてコンポーザブルサーバを構成できる点です．欠点はリソースの抽出，管理方法の課題と同じく，機器間の物理的な接続構成といった依存関係をSunfish Service側で保持する必要があるためSunfish Serviceへそのような機能を実装する必要がある点です．
・パターン2　依存関係のある機器をまとめて1つのSunfish Agentを構成：この場合の利点は，Sunfish Service側で物理構成を把握する必要がない点も利点の1つです．また，依存関係ごとにSunfish Agentが存在しているため，障害発生時の製品の影響範囲を特定しやすい点です．欠点はSunfish Agentに障害が発生した際の復旧作業が複雑になる点です．Sunfish Agent

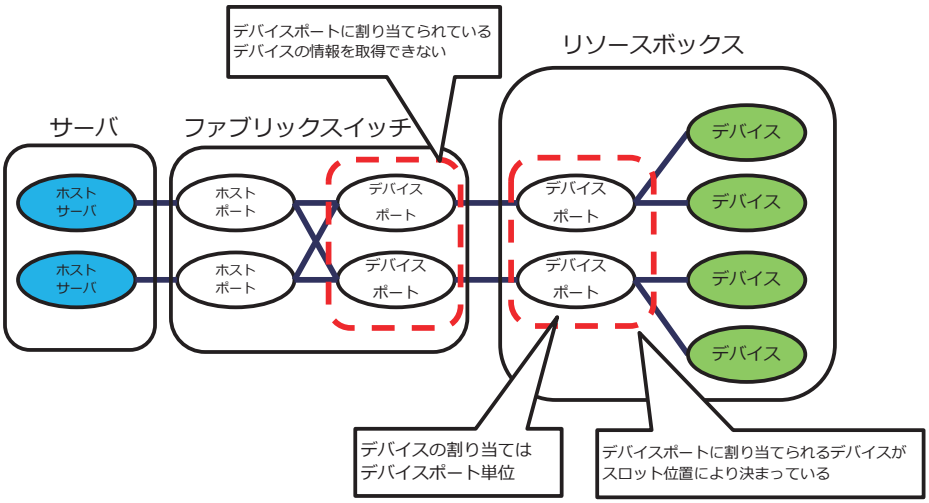
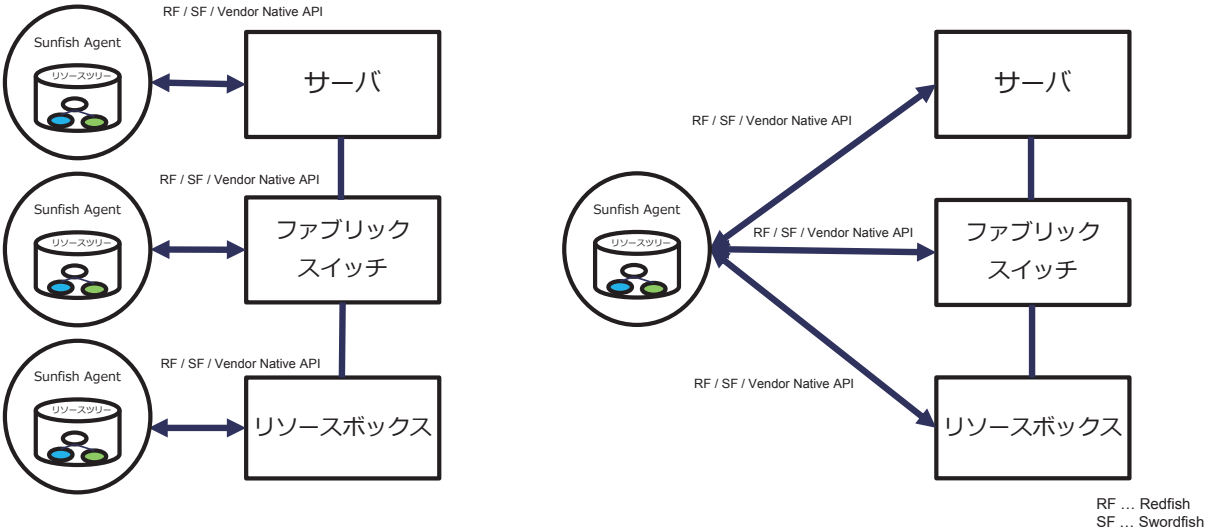


図 5　製品仕様によるリソース管理の難しさ



パターン１：機器ごとにSunfish Agentを構成　　パターン２：依存する機器をまとめてSunfish Agentを構成

図 6　Sunfish Agentの構成単位

に障害が発生した際，それが管理しているすべての機器の状態を依存関係に基づき整合性のあるかたちで復旧する必要があります．

このように，Sunfish Agentの構成単位には複数パターンが存在し，それぞれに利点，欠点が存在します．Sunfish ServiceとSunfish Agentの独立性が保たれることから，私たちはパターン２の実現をめざしています．

■コンポーザブルサーバの実運用に向けて

最後にコンポーザブルサーバの機能検証等で得られた実運用に向けた課題について共有します．

⑴　論理サーバの作成，構成変更に要する時間

論理サーバを作成，再構成する際にはそれに要する時間が運用上課題となり得ます．サーバの再起動には時間を要するため，リソースの割り当て・解除といった操作では，可能な限りサーバを再起動することなく操作が完了できる必要があります．論理サーバの作成，再構成時にサーバの再起動が不要な動的構成変更機能は一部ベンダで提供されています．

⑵　物理構成情報の管理，更新機構

サーバとリソースボックスの物理的な接続情報は，Sunfishによるリソースを抽象化するうえで必要な情報です．そのため，このような物理的な接続情報をコンポーザ

ブルサーバから取得できることはSunfishにおいて重要です．そうでなければ物理的な接続情報を手動で管理することになり，インフラの運用コストが増大するほか，オペレーションミスにより障害を招く可能性も考えられます．コンポーザブルサーバによりシステム全体の複雑さが増していることを考慮しても，物理構成情報の管理，更新機構は必要です．マルチベンダコンポーザブルサーバを実現するために，本機能を具備するよう各コンポーザブルサーバベンダへの働きかけをしていくことを検討しています．

⑶　機器間のケーブル接続作業の増大

PCIe拡張ボックス，PCIeファブリックスイッチはレーン数にはよるものの，基本的に多くのPCIeケーブルを要します．また，世代によりケーブルの太さも大きく変わり，例えば第5世代の場合は，太いうえにケーブル長も短いため，ケーブルの取り回しが容易でないほか，ケーブル長の制約により接続するサーバ機器などを含め，物理的なラッキング位置も考慮が必要でした．さらにPCIe拡張ボックスに関しては，例えばGPUの補助電源ケーブルはベンダ提供の製品でないと使用ができず，市販品等，ベンダ提供ではない製品を使用するとボックスが故障するといった問題もみられるなど，作業時に考慮すべき点が多々ありました．

⑷　可用性の向上

コンポーザブルサーバは大きく，サーバ，ファブリックスイッチ，リソースボックスの3つで構成されます．そのため，ファブリックスイッチやリソースボックスに対して可用性の向上が必要ですが，どのようにそれを実現するかも重要です．例えば，リソースボックスを，その中に搭載されているデバイスを含めて，リソースボックス単位で単純にアクティブ・スタンバイ構成をとると，待機系のリソースを持て余すことになり，リソースの効率活用というコンポーザブルサーバの本来の目的に反する結果となります．

⑸　低レイヤまで見据えた構成の必要性

論理サーバの構成時，単にリソースプールから構成可能なリソースを選択するだけでなく，可能な限り，その上で動作するワークロードを考慮した，より低レイヤを意識した構成が望ましいと考えます．例えば，NVIDIAのGPUDirect RDMAといったオフロード機能を使用する際には，PCIeスイッチやルートコンプレックスを意識したリソースの選択がワークロードの性能やデータ処理の効率性に影響を与えます．

⑹　構成オペレーションの安全性確保

コンポーザブルサーバはその状態管理に不整合が起きるとシステムが停止するおそれがあります．マルチベンダコンポーザブルサーバを実現するうえでは，これらトランザクション管理の徹底が非常に重要です．

今後の展開

本稿では，DCIの技術開発におけるマルチベンダコンポーザブルサーバ実現の取り組みとして，統合管理として現時点で有力と考えるSunfishや，その設計およびコンポーザブルサーバの実運用上の課題を紹介しました．

今後は，2026年度の商用化をめざすDCI-2[9]の実現に向けた研究開発を強化していきます．具体的には，Sunfishの設計思想を取り込みながらマルチベンダコンポーザブルサーバをDCI-2のシステムに統合していきます．さらに，現状のSunfishの仕様やそのリファレンス実装に不足している機能等をOFA Sunfishに提案し，統合管理の標準化と普及を推進していきます．そのうえで，マルチベンダコンポーザブルサーバとコンテナ基盤との連携した運用を可能とするための取り組みも行っていきます．具体的には，コンテナ基盤のデファクトスタンダードとなっているKubernetesでは，動的リソース割り当て機能（Dynamic Resource Allocation）が提供され始めていますが，本機能と連携して，例えばコンポーザブルサーバのリソースボックス内のデバイスをワーカーノードに動的に割り当てるような取り組み[10]も始まっており，NTTも参画しています．また，DCIの各種オペレーション機能の拡充に向けて，DCIコントローラソフトの技術開発やIOWN Global Forumへのシステムリファレンスの提案を行いつつ，AI・映像処理を必要とするモビリティユースケースや，そのほかのユースケースへの技術の展開も図ります．

■参考文献

(1) https://journal.ntt.co.jp/article/33807
(2) https://journal.ntt.co.jp/article/35374
(3) https://www.dmtf.org/
(4) https://www.dmtf.org/standards/redfish
(5) https://www.openfabrics.org/
(6) https://www.openfabrics.org/openfabrics-management-framework/
(7) https://www.dmtf.org/dsp/DSP2050
(8) https://www.snia.org/forums/smi/swordfish
(9) https://www.rd.ntt/forum/2024/keynote_2.html
(10) https://github.com/CoHDI

（左から）　幸田　健介／　二ノ方　一生

IOWNはネットワークだけでなく，コンピューティング基盤に変革をもたらす構想です．その構想の実現に向けて，IOWN Global Forum等でのアーキテクチャ議論やパートナ企業との連携を行い，技術開発の取り組みを推進していきます．

◆問い合わせ先

NTTソフトウェアイノベーションセンタ