社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

コアペリフェリー構造を有する NFV ソフトウェアシステムの有効性評価

津久井佑樹† 荒川 伸一† 村田 正幸†

† 大阪大学 大学院情報科学研究科 〒565–0871 大阪府吹田市山田丘 1–5
E-mail: †{y-tsukui,arakawa,murata}@ist.osaka-u.ac.jp

あらまし SDN や NFV により、様々なネットワーク化サービスを収容し、収容したサービスに対して柔軟なリソース割り当てが可能になると期待されている。新たなネットワーク化サービスの収容に際し、サービスコンポーネントの追加開発が必要になることが想定され、このときの開発量が少ないほどネットワーク化サービスの収容は容易になる。本稿では、コアサービスコンポーネントを再利用することで開発量を削減する性質を NFV ソフトウェアシステムのアジリティと定義し、アジリティを高める NFV ソフトウェアシステムの設計指針を明らかにする。コアサービスコンポーネントは、再利用を可能にするために汎用化が必要であり、より多くの実装コストを必要とする。そこで、コアサービスコンポーネント数が異なる 4 つの NFV ソフトウェアシステムの設計指針それぞれの、時間経過とともに必要となる実装コストを求めて比較した。数値結果より、コアサービスコンポーネント数を少なくしサービス要求増大とともに追加することで、コアサービスコンポーネントの実装コストとサービス収容に追加するサービスコンポーネントの実装コストの和である総実装コストが抑制され、コアサービスコンポーネントを導入しない設計指針と比べてアジリティが高まることがわかった。
キーワード SDN、NFV、ソフトウェア設計、ソフトウェアアジリティ、コアペリフェリー構造

On the Agility of NFV Software System through a Core-Periphery Analysis

Yuki TSUKUI†, Shin'ichi ARAKAWA†, and Masayuki MURATA†

† Graduate School of Information Science and Technology, Osaka University
1–5 Yamadaoka, Suita,Osaka 565–0871 Japan
E-mail: †{y-tsukui,arakawa,murata}@ist.osaka-u.ac.jp

Abstract SDN/NFV is expected to accommodate various networked services, and resources can be flexibly allocated to the accommodated services. However, when a new networked service emerges, additional service components should be developed in order to accommodate a new networking service. With this case, less amount of development cost is preferred. In this paper, we define the agility of NFV's Software System as an ability to have less amount of developmental cost by reusing 'core' service components Because the core service component itself requires an additional implementation cost to make it reusable, total developmental/implementation cost should be concerned. We compare several design approaches for NFV's Software System by changing design philosophy of core service components, and evaluate agilities of NFV's Software System. Our numerical results show that when the amount of core service components is appropriate, the agility is expected to be higher than the case with no core service components.
Key words SDN, NFV, software design, software agility, core-periphery structure

1. はじめに

近年、世界的なデバイスの普及によりネットワークの利用形態は多様化し、多様なサービス要求への対応がネットワークに求められている [1]。ユーザのサービス要求への対応を容易にするため、SDN (Software Defined Network) や NFV (Network Function Virtualization) が注目を集めている [2–4]。特に NFV では、ソフトウェア化された複数のサービスコンポーネントを仮想化環境で実行し、それらをネットワークを介して連結することで、サービス（以降、ネットワーク化サービス）要求に応じてネットワークに収容可能となる。SDN や NFV に関する既存研究では、サービス要求量が与えられると

し、ユーザが必要とするサービスコンポーネントを物理インフラに最適に配置する手法が広く検討されている [5–7] 。

サービスコンポーネントの配置手法によってサービス要求量の変化に柔軟に対応可能となる一方で、ネットワーク化サービスの収容に用いるサービスコンポーネント集合の設計も重要となる。設計が不適切であれば、新たな種類のネットワーク化サービスの登場に際して多くのサービスコンポーネントの追加開発が必要となり、また、設計が適切であれば最小限のサービスコンポーネントの追加開発で十分となる。本稿では、追加開発の量を少なく抑える性質をアジリティと呼び、ネットワーク化されたソフトウェアシステムのアジリティに着目する。

ネットワーク化ソフトウェアのアジリティを高める手段の一つに、マイクロサービス化がある [8–11]。非マイクロサービス、すなわち、モノリシックなソフトウェア設計では、各サービスの収容のためにそれぞれ専用のサービスコンポーネントが実装され、それらのコンポーネントは特定のコンポーネントとの連結が想定される [8]。一方でマイクロサービスは、サービスコンポーネントを小規模かつ互いに独立する設計となっており、特定のサービスコンポーネントとの連結に留まらず様々なサービスコンポーネントとの連結を可能とする。ネットワーク化ソフトウェアをマイクロサービス化し、既に実装されているサービスコンポーネントを新たなネットワーク化サービスの収容に再利用することで、追加で実装するサービスコンポーネントの数を減らし、アジリティを高めることが期待される。

ネットワーク化ソフトウェアは、複数のネットワーク化サービスの収容に用いるサービスコンポーネントをコア機能、一種類のネットワーク化サービスの収容にのみ用いるサービスコンポーネントをペリフェリー機能とみなすことで、コアペリフェリー構造と解釈することができる。コアペリフェリー構造は生態系やソーシャルネットワーク、インターネットシステムなどでも観測され、効率性を有し変化が少ないコアと、積極的に変化し環境変化を吸収するペリフェリーにより分類されるコアペリフェリー構造によりシステムが構成される [12, 13]。コアペリフェリー構造は、生物システムなどの自己組織的に動作するシステムにおいて、外的要因の急激な変化に対して安定的に機能提供可能であることを説明する構造の一つである。ただし、適切なコアの大きさは、システム依存であることから未だ明らかになっていないのが現状である。

ネットワーク化ソフトウェアにおいては、コアペリフェリー構造はネットワーク化サービス要求の変化に対して安定的にサービスを提供可能であることを説明する構造となり、ソフトウェアシステムであることを考慮した適切なコアの大きさを決定する必要がある。モノリシックなソフトウェア設計におけるサービスコンポーネントは特定のサービスコンポーネントのみとの連結を想定すれば良いため比較的容易に開発できるが、マイクロサービスのサービスコンポーネントの実装コストは、複数のサービスコンポーネントとの連結を可能とする汎用化が必要となるため高くなる。したがって、アジリティを高めることを目的として全てのコンポーネントをマイクロサービス化するのでなく、全体の実装コストを考慮して、複数のネットワーク化サービスの収容に用いるサービスコンポーネント（コア）と、特定のネットワーク化サービスの収容にのみ用いるサービスコンポーネント（ペリフェリー）の両方を有するように、ネットワーク化ソフトウェアを設計しなければならない。そこで本稿では、ネットワーク化ソフトウェアの設計指針を得ることを目的とし、コアペリフェリー構造にもとづいてネットワーク化ソフトウェアの実装コストをモデル化し、コア機能数に依存してネットワーク化ソフトウェアのアジリティがどのように変化するかを明らかにする。時間の経過に伴って多様なユーザ要求が生じ、ネットワーク化ソフトウェアに収容するネットワーク化サービスの種類が増加するシナリオにおいて、コア機能数に関するいくつか設計指針の実装コストを比較し、アジリティを高めることで全体の実装コストが抑制されるネットワーク化ソフトウェアの設計指針を述べる。

本稿の構成は以下の通りである。まず 2 章では関連研究を紹介する。3 章ではコアペリフェリー構造によるネットワーク化ソフトウェアのより詳細な解釈を説明し、続く 4 章ではネットワーク化ソフトウェアの実装コストの定義を行う。5 章は実装コストの数値結果および数値結果をもとにしたネットワーク化ソフトウェアの設計指針の議論を示す。最後に 6 章でまとめと今後の課題を議論する。

2. 関 連 研 究

SDN や NFV の解決すべき 運用上の課題の一つに、物理インフラへのサービスコンポーネントの配置方法の決定があげられる。CPU や帯域などのリソース制約、通信遅延などの最適化を目指して多くの研究がこの課題に取り組んでいる。文献 [5] ではネットワーク化サービスの収容率や収益、プロビジョニングコストを向上するために混合整数線形計画法を用いている。文献 [6] では end-to-end のサービス時間を最小化するため、サービスコンポーネントの利用頻度をジップの法則でモデル化し、物理インフラへの配置を行う。文献 [7] は仮想マシンの借用量やランニングコスト、伝送コスト、end-to-end の遅延などを考慮した最適化問題を解いている。このようにサービスコンポーネントの配置方法の課題が焦点を浴びる一方で、ネットワーク化ソフトウェアの設計指針は十分に議論されていない。

ネットワーク化ソフトウェアを含むソフトウェア設計として、複数のコンポーネントが全体で一つのモジュールを形成するモノリシックなソフトウェア設計が広く用いられてきた [8]。これらのコンポーネントは特定のサービスを構成するために実装され、コンポーネント同士が密接に連結されている。モノリシックなソフトウェア設計におけるコンポーネントは特定のコンポーネントと連結することを前提として設計されているため、あるコンポーネントの変更によって受け渡す処理内容が変化すると別のコンポーネントの変更も必要になる場合があり、変更が困難となる技術的負債が課題としてあげられる [14]。加えて密接な連結のため、それぞれのコンポーネントを別のサービスの構成に用いることは困難である。このようにモノリシックなソフトウェア設計の課題は多い。文献 [15, 16] では、Linux や Mozilla といった長期的に開発・運用されているソフトウェア

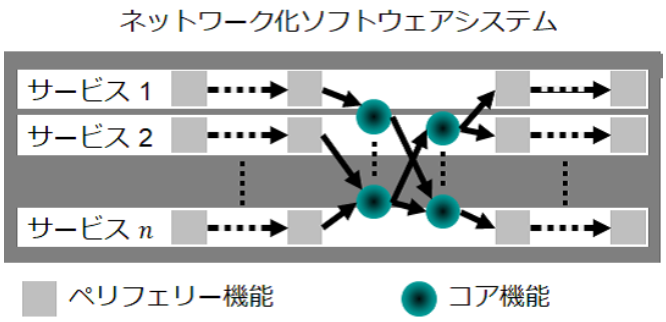

図 1　ネットワーク化ソフトウェアの例

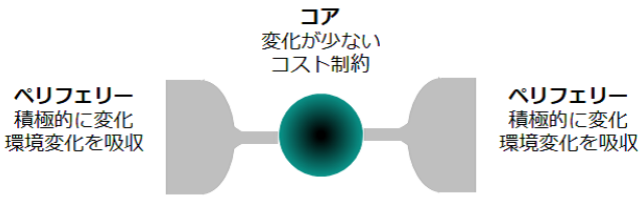

図 2　コアペリフェリー構造の概念図

のコンポーネントがどのように連結されているかを分析し、大規模なリファクタリングによってコンポーネント同士の密接な連結を削減したことが Mozilla の普及に貢献している可能性を述べている。

そのため、アジリティを高めるソフトウェア設計アプローチとして、マイクロサービスが注目されている [8,10]。マイクロサービスでは、コンポーネントが独立しており、それらを組み合わせることで様々なサービスを構成する。実装済みのコンポーネントの再利用が可能なため、新たなサービスを構成する際に追加するコンポーネント数を減らし、アジリティを高めることが期待される。文献 [9] では、本稿がネットワーク化ソフトウェアのマイクロサービス化を考えるのと同様に、MEC (Multi-access Edge Computing) 環境でのアジリティを高めるためにマイクロサービスの導入を推奨している。しかしながら、実装済みのコンポーネントの再利用によるアジリティの向上と、コンポーネントを再利用できるようにするために必要な実装コスト増加のトレードオフについては十分に議論されていない。

本稿ではマイクロサービスの観点を踏まえ、これまで十分に議論されていないネットワーク化ソフトウェアの設計指針に着目する。再利用するサービスコンポーネントの数によって、ネットワーク化ソフトウェアのアジリティがどのように変化するかを調べ、得られた結果をもとに設計指針を議論する。

3. コアペリフェリー構造によるネットワーク化ソフトウェアの解釈

ネットワーク化ソフトウェアは多くのサービスコンポーネント (以後、サービス機能) を有し、それらを適切に組み合わせることで種々のネットワーク化サービスを収容する。ある一つのサービス機能を複数のネットワーク化サービスの収容に用いることが考えられ、そのようなサービス機能は他の多くのサービス機能と連結できるように汎用化されていると想定する。このようなサービス機能をコア機能と定義する。また、その他のサービス機能をペリフェリー機能と定義する。ペリフェリー機能は、ただ一種類のネットワーク化サービスの収容のために用いられる。他のサービス機能の処理結果を入力として受け取り、他のサービス機能に処理結果を出力として受け渡すためにそれぞれ一種類ずつ、最大二種類のサービス機能のみとの連結が想定される。図 1 はコア機能とペリフェリー機能を用いてネットワーク化サービスを収容するネットワーク化ソフトウェアの例を示している。

コア機能とペリフェリー機能の分類はコアペリフェリー構造の解釈に基づいている。コアペリフェリー構造は生態系やソーシャルネットワーク、インターネットシステムなどの振る舞いを解釈するために用いられており、システムをコアとペリフェリーに分類する [12,13]。コアは変化が少なく効率性を有する。ペリフェリーは積極的に変化し環境の変化を吸収する。図 2 はこれらのコアペリフェリー構造の基本的な概念を示している。コア機能は、他の多くのサービス機能と連結が可能であり複数のネットワーク化サービスの収容に用いることができるため効率性を有し、汎用化されているため頻繁な変更は望ましくない。ペリフェリー機能は、コア機能だけでは収容できない新たなネットワーク化サービスを収容するために用いられることから、新しいユーザ要求の発生に対応し変化を吸収する。

ネットワーク化ソフトウェアのコア機能数の変化により、アジリティや全体の実装コストが変化する。コア機能数が増加すると、より多くのコア機能を再利用して新たなネットワーク化サービスを収容することが可能となり、追加するサービス機能数を削減できる。加えて、ペリフェリー機能の役割の一部をコア機能で代用可能なため、ペリフェリー機能の実装コストが低下する。なお実装コストとは、サービス機能を開発しネットワーク化ソフトウェアにデプロイして正常に動作するか確認するために必要な費用とする。このようなサービス機能数の削減およびペリフェリー機能の実装コストの低下はアジリティを増加させる。しかしながら、コア機能は多くのサービス機能との連結を想定した汎用化が必要であり、実装コストが高い。したがって、コア機能数の増加によるアジリティの向上だけでなく、それらのコア機能の実装コストが多く必要なことも考慮してネットワーク化ソフトウェアを設計しなければならない。次章では、追加のサービス機能数の削減およびペリフェリー機能の実装コスト低下と、コア機能の実装コスト増加とのトレードオフを理解するために、ネットワーク化ソフトウェアの実装コストを定義する。

4. ネットワーク化ソフトウェアの実装コスト

ある時点において、ネットワーク化ソフトウェアが n 種類のネットワーク化サービスを収容しており、各ネットワーク化サービスの収容に平均 k 個のサービス機能が用いられているとする。このとき、ネットワーク化ソフトウェアが有するサービス機能数 $f_{all}(n)$ をコア機能数 $f_c(n)$ とペリフェリー機能数 $f_p(n)$ の和で定義する。

$$f_{all}(n) = f_c(n) + f_p(n) \tag{1}$$

ネットワーク化ソフトウェア全体の実装コスト $c_{all}(n)$ を式 (2) で定義する。

$$c_{all}(n) = \sum_{i=1}^{f_c(n)} c_c(i) + \sum_{j=1}^{n} k_p(j) c_p(j) \tag{2}$$

ここで $c_c(i)$ は i 種類目のコア機能の実装コストであり、第一項は $f_c(n)$ 個のコア機能の実装コストの和である。$k_p(j)$ は j 種類目のネットワーク化サービスの収容に用いるペリフェリー機能数であり、$c_p(j)$ は j 種類目のネットワーク化サービスの収容に用いるペリフェリー機能あたりの実装コストである。ペリフェリー機能はただ一種類のネットワーク化サービスの収容にのみ用いられるサービス機能と定義されているため、$f_p(n) = \sum_{j=1}^{n} k_p(j)$ が成立し、式 (2) の第二項は n 種類のネットワーク化サービスの収容に用いるペリフェリー機能の実装コストの和を表す。

コア機能の実装コストは、より多くのサービス機能との連結を想定するために高くなる。そこで、i 種類目のコア機能の実装コスト $c_c(i)$ を式 (3) で定義する。

$$c_c(i) = \alpha i \tag{3}$$

式 (3) は i の一次関数であり、実装済みのコア機能との連結に比例して、i 種類目のコア機能の実装コスト $c_c(i)$ が増加することを表している。また、α はより多くのコア機能との連結が想定されるときの、コア機能の実装コストの増加度を決定するパラメータである。

役割の一部をコア機能で代用することによりペリフェリー機能の開発が容易になる。j 種類目に収容するネットワーク化サービスが用いるペリフェリー機能あたりの実装コストを式 (4) で定義する。

$$c_p(j) = \exp(-\beta f_c(j)) \tag{4}$$

式 (4) は、コア機能数が増加するほどペリフェリー機能の実装コストは低下するが、過度にコア機能数を増加させてもペリフェリー機能の実装コストはあまり低下しないことを表している。例えば標準関数などを含むライブラリがコア機能として実装済みであるとき、一部をコア機能で代用することによりペリフェリー機能の実装コストは低下すると考えられる。コア機能数が増加するほど代用可能なライブラリは増加するが、実際に代用されるのは一定のライブラリに限定されるため、過度なコア機能数の増加はペリフェリー機能の実装コスト削減にあまり貢献しない。なお、β はコア機能数の増加に伴うペリフェリー機能の実装コストの低下度を決定するパラメータである。

コア機能数 $f_c(n)$ の変化によりネットワーク化ソフトウェアのアジリティや実装コストがどのように変化するかを調べるため、式 (2) を $f_c(n)$ の関数で表す必要がある。このために、$k_p(j)$ を $f_c(j)$ で解く。j 種類目のネットワーク化サービスの収容に用いるコア機能数を $k_c(j)$ としたとき、$k = k_c(j) + k_p(j)$ が成立する。$f_c(j)$ 個のコア機能がネットワーク化サービスの収容に用いられる頻度を表すパラメータを γ とし、$k_c(j) = k\gamma f_c(j)$ であるとすると、$k_p(j)$ は式 (5) によって得られる。

$$k_p(j) = k - k\gamma f_c(j) \tag{5}$$

ただし、$0 < \gamma < 1/f_c(j)$ である。$k_c(j)$ が k の一次関数であることは、各ネットワーク化サービスを収容するために平均してより多くのサービス機能が用いられるとき、収容にコア機能が用いられる機会が増加することを表現している。

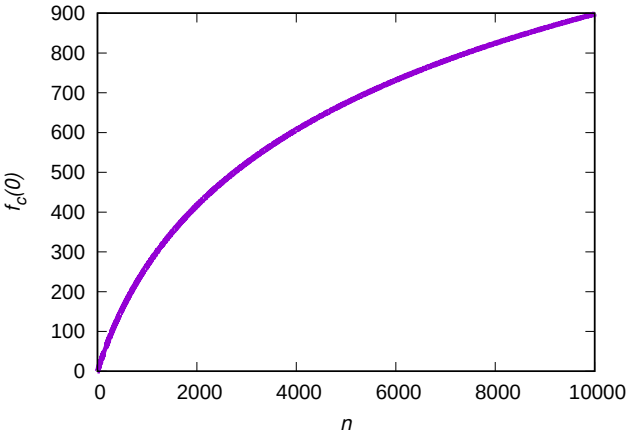

図 3　ネットワーク化サービスの増加に伴う実装コストを最小化するコア機能数の変化

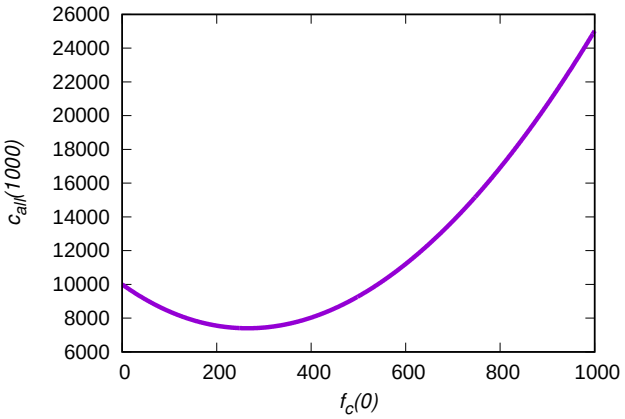

図 4　コア機能数増加に伴うネットワーク化ソフトウェアの実装コストの変化 ($n = 600$)

5. 数値結果

ネットワーク化ソフトウェアが収容するネットワーク化サービスの種類 n が増加するシナリオを想定する。n の増加は、新たな種類のユーザ要求が時間の経過に伴って動的に生じることを表している。また、各ネットワーク化サービスの収容に用いる平均サービス機能数を $k = 10$ とし、パラメータを $\alpha = 5.0e-2, \beta = 1.0e-3, \gamma = 1.0e-3$ とする。

5.1 ネットワーク化ソフトウェアの実装コストを削減するコア機能数

$n = 0$ の時点でコア機能を用意し、その後の $0 < n$ においてコア機能数が変化しないシナリオを想定する。このとき、ネッ

トワーク化ソフトウェアの実装コスト $c_{all}(n)$ を削減するコア機能数 $f_c(0)$ を議論する。

$n = 0$ の時点で n 種類のネットワーク化サービスの収容が予測されるとき、予測に基づいてあらかじめコア機能を用意することで、追加のサービス機能数やペリフェリー機能の実装コストを削減し、アジリティを高めることができると考えられる。図 3 に $c_{all}(n)$ を最小化する $f_c(0)$ を示す。この図に従い $f_c(0)$ 個のコア機能を用意することで、コア機能の実装コストを考慮しつつアジリティを高めた設計が得られる。図より n の増加に伴って、$c_{all}(n)$ を最小化する $f_c(0)$ は単調に増加することが確認される。したがって、より多くのネットワーク化サービスの収容が予測される場合に、ネットワーク化ソフトウェア全体の実装コストを最小化するためには、より多くのコア機能を用意する必要がある。しかしながら、過剰にコア機能を用意することは望ましくない。例えば、$n = 1000$ 種類のネットワーク化サービスの収容が予測されるとき $f_c(0) = 266$ であれば $c_{all}(1000)$ を最小化することが可能であり、$n = 4000$ 種類のネットワーク化サービスの収容が予測されるとき $f_c(0)$ を約 2.3 倍の 607 にすると $c_{all}(4000)$ を最小化することが可能である。しかし、$n = 10000$ 種類のネットワーク化サービスの収容が予測されるとき $c_{all}(10000)$ を最小化するのは $f_c(0) = 898$ のときであり、$n = 1000$ のときの約 3.4 倍に留まる。

また 1000 種類のネットワーク化サービスの収容が予測され、予測に基づいてあらかじめコア機能を用意するとき、$c_{all}(1000)$ は $f_c(0)$ に依存して図 4 のように変化する。$c_{all}(1000)$ は下に凸となり、$f_c(0) = 266$ まで減少し、それ以後は増加する。$266 < f_c(0)$ の範囲における $c_{all}(1000)$ の増加は、コア機能の実装コストの増加が、追加のサービス機能数の削減やペリフェリー機能の実装コストの減少を上回ったことに起因している。この結果も同様に、コア機能を過剰に用意することが望ましくないことを示している。

図 3 および図 4 の数値結果は、新たなネットワーク化サービスの収容のためにコア機能を追加しないことが前提にある。しかしながら、より現実的な環境では将来生じるユーザ要求が既存のものと大きく異なり、どのようなネットワーク化サービスを収容するか予測が困難な状況が想定される。このように、ユーザ要求が動的に激しく変化し予測が困難な状況下では、$n = 0$ の時点であらかじめコア機能を用意するだけでなく、n の増加に伴いコア機能の追加が必要になる。次節では、コア機能の追加を踏まえた異なる設計指針の数値結果を示す。数値結果を比較することでネットワーク化ソフトウェア全体の実装コストを抑えつつ、アジリティを高める設計指針を議論する。

5.2 異なるコア機能設計指針と追加の実装コスト

コア機能数 $f_c(n)$ が異なる四つの設計指針の実装コストを示し比較を行う。全ての設計指針に共通して、収容するネットワーク化サービスは $0 \leq n \leq 1000$ の範囲で増加する。

比較に用いる設計指針は *no core*, *forecast*, *adaptive* および *unadaptive* の四つである。*no core* はコア機能を用いず、ネットワーク化サービスの収容にペリフェリー機能のみを用いる設計指針である。この設計指針では、n に依らず $f_c(n) = 0$ となる。*forecast* は前節同様、$n = 0$ の時点で $n = 1000$ 時点のネットワーク化サービスの収容を予測し、ネットワーク化ソフトウェアの実装コスト $c_{all}(1000)$ を最小化するために、あらかじめコア機能を用意する設計指針である。*forecast* は参考用の設計指針で、コア機能の追加が行われないと仮定しており、$f_c(n) = f_c(0) = 266$ となる。*adaptive* は新たなネットワーク化サービスの収容に伴い、コア機能が追加されることを想定した設計指針である。この設計指針では $f_c(0) = 50$ とし $n = 500$ までの間、n が 100 増加する度にコア機能数が 50 増加する。*unadaptive* は *forecast* と同様に $n = 0$ の時点で $n = 1000$ 時点のネットワーク化サービスの収容を予測し、あらかじめ 266 個のコア機能を用意する。加えて *adaptive* と同様に、新たなネットワーク化サービスの収容に伴い $n = 500$ までの間、n が 100 増加する度にコア機能が 50 ずつ追加される設計指針である。図 5 は収容するネットワーク化サービスの種類が n のときの各設計指針のコア機能数 $f_c(n)$ を示している。

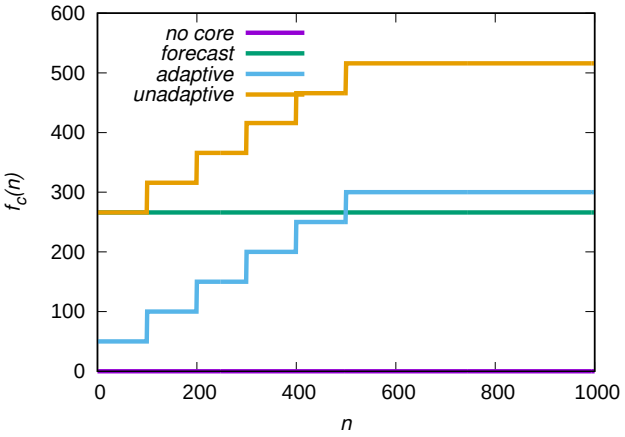

図 5　各設計指針のコア機能数

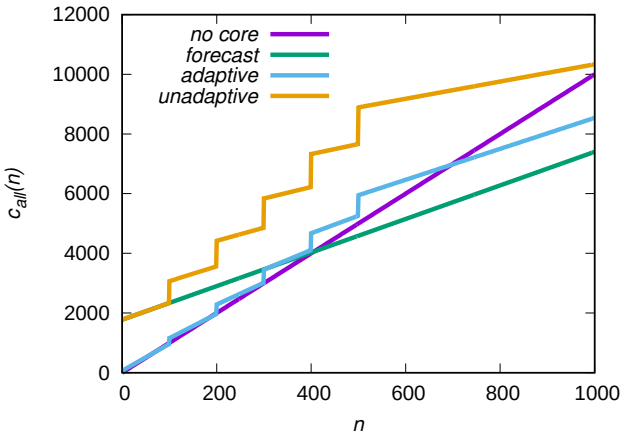

図 6　各設計指針のネットワーク化ソフトウェアの実装コスト

図 6 は収容するネットワーク化サービスの種類が n のときのネットワーク化ソフトウェアの実装コスト $c_{all}(n)$ を示している。想定通り、$c_{all}(1000)$ が最小となる設計指針は *forecast* であることが確認される。*no core* と比較すると、*forecast* は $c_{all}(1000)$ を約 26% 削減する。しかしながら、コア機能の追加を考慮していないため *forecast* より他の設計指針の方が現実的である。こ

のことを踏まえた上で *adaptive* に着目すると、$n = 1000$ 種類のネットワーク化サービスを収容したときのネットワーク化ソフトウェアの実装コスト $c_{all}(1000)$ が、*no core* や *unadaptive* と比較して、より小さいことが確認できる。この結果は、ペリフェリー機能のみを用いてネットワーク化サービスの収容を行う設計指針より、コア機能とペリフェリー機能の両方を用いるコアペリフェリー構造に基づいた設計指針が、ネットワーク化ソフトウェアのアジリティを向上させ得ることを示している。*no core* と $c_{all}(100), c_{all}(200), c_{all}(300), c_{all}(400), c_{all}(500)$ を比べたとき、*adaptive* ではそれぞれ約 15%, 14%, 15%, 17%, 19% 増加する。しかし *adaptive* の $c_{all}(1000)$ は、*no core* と比較して約 15% 削減される。*unadaptive* に着目すると、コア機能が追加される $100 \leq n \leq 1000$ において、他のどの設計指針よりネットワーク化ソフトウェアの実装コスト $c_{all}(n)$ が高くなることが確認される。ユーザ要求が動的に激しく変化し、予測が困難な状況が想定される場合、*unadaptive* のように事前の予測に基づいてあらかじめコア機能を用意するのではなく、*adaptive* のようにコア機能数を少量に留め、新たなユーザ要求の発生に伴いコア機能を設計し追加することで、より現実的なネットワーク化ソフトウェアのアジリティの向上が見込める。

6. まとめと今後の課題

本稿では、複数のネットワーク化サービスの収容に用いられるサービス機能の数に依存して、ネットワーク化ソフトウェアのアジリティがどのように変化するかに着目し、アジリティを高める設計指針の議論を行った。ここでネットワーク化ソフトウェアは、複数のネットワーク化サービスの収容に用いられるサービス機能をコア機能、一種類のネットワーク化サービスの収容にのみ用いられるサービス機能をペリフェリー機能とみなすことで、コアペリフェリー構造と解釈することができる。他の多くのサービス機能との連結を想定した汎用化が必要となるため、コア機能の実装コストが多く必要となる一方、より多くのコア機能を実装し収容に用いることで、新たなネットワーク化サービスの収容のために追加するサービス機能数を減らし、ペリフェリー機能の開発が容易になるため、アジリティの向上が期待される。ネットワーク化ソフトウェアの設計においては、これらのトレードオフを考慮しなければならない。数値結果より、後に収容するネットワーク化サービス予測しコア機能をあらかじめ用意することで、ネットワーク化ソフトウェアの実装コストを抑えつつ、アジリティを高めることができると確認された。しかしながら、ユーザ要求が動的に激しく変化する環境では、新たなネットワーク化サービスを収容するためにコア機能を追加することも想定される。このような状況下でコア機能追加時のアジリティを高めるためには、事前の予測に基づいてコア機能を用意するのでなく、あらかじ用意するコア機能は少量に抑え、新たなユーザ要求の発生に応じてコア機能を設計・追加する指針がより現実的で望ましい。

今後の課題として、コア機能とペリフェリー機能それぞれの特性を踏まえた物理インフラへの機能配置方法の考察があげられる。本稿では実装コストを重視したが、物理インフラのリソース制約のため、サービスコンポーネント数の削減がより重要になる可能性がある。加えて、複数箇所にサービスコンポーネントを複製して配置するシナリオが想定されるため、複製の容易さなども議論する必要がある。

謝辞 本研究の一部は NICT 委託研究「未来を創る新たなネットワーク基盤技術に関する研究開発」によるものである。ここに記して謝意を示す。

文　献

[1] Cisco, VNI, "Cisco visual networking index: Forecast and trends, 2017–2022," *White Paper*, Nov. 2018.

[2] J. H. Cox, J. Chung, S. Donovan, J. Ivey, R. J. Clark, G. Riley, and H. L. Owen, "Advancing software-defined networks: A survey," *IEEE Access*, vol. 5, pp. 25487–25526, Oct. 2017.

[3] G. Kandiraju, H. Franke, M. Williams, M. Steinder, and S. Black, "Software defined infrastructures," *IBM Journal of Research and Development*, vol. 58, pp. 2–1, Mar. 2014.

[4] Y. Jararweh, M. Al-Ayyoub, E. Benkhelifa, M. Vouk, A. Rindos, *et al.*, "Software defined cloud: Survey, system and evaluation," *Future Generation Computer Systems*, vol. 58, pp. 56–74, May 2016.

[5] N. M. K. Chowdhury, M. R. Rahman, and R. Boutaba, "Virtual network embedding with coordinated node and link mapping," in *Proceedings of IEEE INFOCOM 2009*, pp. 783–791, Apr. 2009.

[6] Y. Nam, S. Song, and J.-M. Chung, "Clustered NFV service chaining optimization in mobile edge clouds," *IEEE Communications Letters*, vol. 21, pp. 350–353, Oct. 2017.

[7] Y. Jia, C. Wu, Z. Li, F. Le, A. Liu, Z. Li, Y. Jia, C. Wu, F. Le, and A. Liu, "Online scaling of nfv service chains across geo-distributed datacenters," *IEEE/ACM Transactions on Networking (TON)*, vol. 26, pp. 699–710, Apr. 2018.

[8] Mark Richards, *Software Architecture Patterns*. O'Reilly Media, Inc., Feb. 2015.

[9] A. Reznik, R. Arora, M. Cannon, L. Cominardi, W. Featherstone, R. Frazao, F. Giust, S. Kekki, A. Li, D. Sabella, *et al.*, "Developing software for multi-access edge computing," *ETSI, White Paper*, vol. 20, Sept. 2017.

[10] W. Hasselbring, "Microservices for scalability: keynote talk abstract," in *Proceedings of the 7th ACM/SPEC on International Conference on Performance Engineering*, pp. 133–134, ACM, Mar. 2016.

[11] C. Richardson, "Microservice architecture patterns," *microservices. io.–URL: http://microservices.io/patterns/microservices.html (19.12. 2017)*, Dec. 2017.

[12] P. Csermely, A. London, L.-Y. Wu, and B. Uzzi, "Structure and dynamics of core/periphery networks," *Journal of Complex Networks*, vol. 1, pp. 93–123, Oct. 2013.

[13] V. Miele, R. Ramos-Jiliberto, and D. P. Vázquez, "Core–periphery dynamics in a plant–pollinator network," *bioRxiv*, p. 543637, Jul. 2019.

[14] A. MacCormack and D. J. Sturtevant, "Technical debt and system architecture: The impact of coupling on defect-related activity," *Journal of Systems and Software*, vol. 120, pp. 170–182, Oct. 2016.

[15] A. MacCormack, J. Rusnak, and C. Y. Baldwin, "Exploring the structure of complex software designs: An empirical study of open source and proprietary code," *Management Science*, vol. 52, pp. 1015–1030, Jul. 2006.

[16] A. MacCormack, "The architecture of complex systems: do "core-periphery" structures dominate?," *Academy of Management Proceedings*, vol. 2010, pp. 1–6, Nov. 2010.