情報処理学会論文誌 Vol.1 No.1 1–5 (Jan. 1960)

環境適応アプリケーションの運用中配置再構成の評価

山登 庸次[1,a)]

受付日 2022年7月8日, 採録日 2022年7月8日

概要：著者は，通常コードを，環境に応じて自動変換し，高性能で運用可能とする環境適応ソフトウェアを提案し，GPU 等への自動変換や適正配置での性能向上を検証してきた．本稿では，他ユーザ配置状況を勘案した運用開始後の配置再構成を検証する．

キーワード：環境適応ソフトウェア, 自動オフロード，最適配置，運用中再構成

Study and evaluation of deployment reconfiguration during operation of environment-adaptive applications

Yoji Yamato[1,a)]

Received: July 8, 2022, Accepted: July 8, 2022

Abstract: I have proposed environment-adaptive software that automatically converts normal code according to the environment and enables high-performance operation, and has verified automatic conversion to GPU and FPGA, and performance improvement with proper placement. In this paper, I will verify the relocation after the start of operation in consideration of the placement of other users.

Keywords: Environment Adaptive Software, Automatic Offloading, Optimum Placement, Reconfiguration During Operation

1. はじめに

近年, ムーアの法則の鈍化が予想されている．そのような状況から，CPU だけでなく，FPGA（Field Programmable Gate Array）や GPU（Graphics Processing Unit）等のデバイスの活用が増えている．例えば, Microsoft 社は FPGA を使って検索効率を高め [1]，Amazon 社は，FPGA, GPU をクラウド技術を用いて（例えば，[2]-[5]）提供している．また，IoT 機器利用（例えば，[6]-[13]) も増えている．

しかし，CPU 以外のデバイスを適切に活用するためには，デバイス特性を意識したプログラム作成が必要で，OpenMP（Open Multi-Processing）[14]，OpenCL（Open Computing Language）[15], CUDA（Compute Unified Device Architecture）[16] や組み込み技術といった知識が必要になってくるため，スキルの壁が高い．そこで，私は，一度記述したコードを，配置先の環境に存在する GPU や FPGA，メニーコア CPU 等を利用できるように，変換，配置等を自動で行い，アプリ（アプリケーション）を高性能に動作させることを目的とした，環境適応ソフトウェアを提案した．合わせて，環境適応ソフトウェアの要素として，アプリのループ文及び機能ブロックを，FPGA，GPU に自動オフロードする方式を提案評価している [17]-[21]．さらに，変換アプリをユーザの応答時間や価格要求を満たして，配置先を適正化する手法についても提案してきた．

本稿では，環境適応ソフトウェアの新要素として，変換して配置し動作させているアプリを，別ユーザの配置状況等を踏まえて再配置することで，全体的なユーザ満足度を上げる運用中再構成の手法を検討する．

2. 既存技術

GPU の並列計算能力を一般用途にも使う GPGPU（General Purpose GPU）の環境として CUDA が普及している．CUDA は GPGPU 向けの NVIDIA 社環境だが，FPGA,

[1] 日本電信電話株式会社ネットワークサービスシステム研究所
NTT Corporation, Musashino-shi, Tokyo 180–8585, Japan
[a)] yoji.yamato.wa@hco.ntt.co.jp

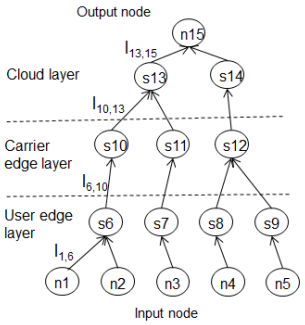

図 1　計算ノードのトポロジー例

メニーコア CPU，GPU 等のヘテロなデバイスを同様に扱うための仕様として OpenCL が出てきている．しかし，CUDA，OpenCL は，C 言語の拡張を行いプログラムを行う形だが，メモリ処理の記述等プログラムの難度は高い．

このため，スキルが乏しいプログラマーが，FPGA や GPU を活用してアプリを高速化することは難しいし，自動並列化技術（例えば，[22] 等）を使う場合も並列処理箇所探索の試行錯誤等の稼働が必要だった．並列処理箇所探索の試行錯誤を指示行ベース仕様（OpenACC[23][24] 等）を用いて自動化する取り組みとして，著者は進化計算手法を用いた GPU 自動オフロードを提案している．

配置に関して，ネットワークリソースの最適利用として，ネットワーク上にあるサーバ 群に対して VN（Virtual Network）の埋め込み位置を最適化する研究がある [25]．[25] では，通信トラヒックを考慮した VN の最適配置を決定する．しかし，単一リソースの仮想ネットワークが対象で，キャリアの設備コストや全体的応答時間の削減が目的で，個々に異なるアプリの処理時間や，個々のユーザの価格や応答時間要求等の条件は考慮されていなかった．

ヘテロなデバイスに対するオフロードは手動での取組みが主流である．著者は環境適応ソフトウェアのコンセプトを提案し，自動オフロードを検討してきたが，自動変換した後のアプリの配置については，個別ユーザの要件に応じた個別最適の配置検討のみで，他ユーザの配置状況を勘案した全体最適の配置検討はされていない．そのため，本稿では，運用開始後に配置を再構成することで再構成対象の複数のユーザの満足度を向上させることを検討する．

3.　アプリケーション配置再構成

3.1　既存方式と再構成必要性

計算ノードリンクとしては，ユーザエッジ，キャリアエッジ，クラウドの 3 層のトポロジーを想定する（図 1 等）．計算ノードは CPU，GPU，FPGA の 3 種に分けられる．GPU や FPGA を備えるノードは，仮想化技術（例えば，[26]）により，GPU，FPGA インスタンスとして，CPU リソースも含む形で提供される．

従来は [25] 等の検討の様に，例えば仮想ネットワークを収容するサーバを配置する場所を，トラフィック増加等の長期的傾向を見て，計画的に設計していた．それに対して，環境適応ソフトウェアでは 2 つ特徴がある．一つ目は，配置アプリは静的に定まっているのでなく，GPU や FPGA 向けに自動変換され，遺伝的アルゴリズム [27] 等を通じ利用形態に適したパターンが実測を通じて抽出されるため，アプリコードや性能は動的に変わり得る．二つ目は，キャリア設備コストや全体的応答時間だけ低減すれば良いのでなく，応答時間や価格に対する個々のユーザ要求を満たす必要があり，アプリ配置ポリシーも動的に変わり得る．

この二つの特徴も踏まえ，著者は以前に，ユーザの応答時間要求や価格要求を満たす配置計算のため，線形計画手法により，ユーザがアプリを配置依頼した際の，変換後アプリの応答時間と価格を定式化して，応答時間か価格の片方を目的関数に最小化する方法を提案している．提案方式を用いて，複数種のアプリを想定して，1000 アプリの配置最適化を GLPK（Gnu Linear Programming Kit）5.0 にてシミュレーションし，提案方式の有効性を確認している．

しかし，提案方式は，個別ユーザの応答時間や価格要求に従い，アプリを配置していくため，基本的に早い者勝ちとなり，安さ優先要求ばかりの場合クラウドが，速さ優先要求ばかりの場合エッジが埋まってしまい，埋まった後は別サーバへ配置が必要になってくる．そこで，早い者勝ちの配置を緩和するため，運用開始前だけでなく運用開始後の配置再構成が必要と考える．

3.2　再構成方式

本サブ節では，他ユーザ配置状況も踏まえ，アプリを適切な配置場所に再構成するための再構成方式を，線形計画手法の定式化も含めて提案する．

再構成では，一定数アプリの配置毎（100 アプリ等）に，複数ユーザの当初要求条件を満たす形で，一定数アプリ（全アプリ or100 アプリ等）の配置再構成を試行計算して，応答時間や価格の変化により定まるユーザ群の満足度を向上させる．再構成の試行計算の結果，満足度変化が一定の閾値を超える等，再構成効果が高い場合のみ，再構成が行われる．実際の再構成は，サーバ変更が必要となるため，ライブマイグレーション等を用いて，影響を抑えて行う．

再構成のための線形計画式を (1)-(5) に，パラメータを図 2 に示す．(1) が満足度評価のための目的関数，(2)(3) がアプリ毎にユーザが指定する応答時間要求，価格要求の制約条件，(4)(5) がサーバリソース上限の制約条件である．

まず，新規配置は (2)-(5) の式に応じて配置がされる．ユーザは配置の際に，応答時間，価格の片方か両方の要求条件を付ける．応答時間要求 R_k^{upper} は何秒以内に応答が必要か，価格要求 P_k^{upper} は一月幾ら以下かを指定する．(2)(3) の片方だけ指定の場合は，その逆の最小化が目的関数となり，両方指定の場合は目的関数はどちらかの最小化

a_i: Device usage cost
b_j: Link usage cost
C_i^d: Device calculation resource limit of #i
C_j^l: Link bandwidth limit of #j
C_k: Data size of #k application
$A_{i,k}^d$: Whether to use of #k application on #i device
$A_{j,k}^l$: Whether to use of #k application on #j link
B_k^d: Calculation resource of #k application
B_k^l: Bandwidth usage of #k application
$B_{i,k}^p$: Processing time of #k application on #i device

図 2　線形計画式パラメータ

をユーザが指定する．(4)(5) は，計算リソース及び通信帯域の上限を設定する制約条件であり，他ユーザ配置アプリ含めて計算され，新規ユーザのアプリ配置によるリソース上限超過を防ぐ．新規配置は，ユーザの配置依頼に対して，順次 (2)-(5) の計算を行っていくことで行われる．

次に再構成を考える．再構成は (1)-(5) 式に応じて計算がされるが，特にユーザ満足度に応じる値 S の計算が新たに加わる．個別ユーザの満足度として，再構成前の配置で応答時間 1 点，価格 1 点を基本の値として，再構成前応答時間 R_k^{before} が再構成後 R_k^{after} で X 倍になったら X が応答時間満足度に関連した値，再構成前価格 P_k^{before} が再構成後 P_k^{after} で Y 倍になったら Y が価格満足度に関連した値とする．再構成の試行計算の目的関数は，再構成対象の一定数アプリのユーザ群満足度に関連した値であり，複数アプリに対して（X+Y）の総和を最小化する配置を計算する．目的関数の具体的内容は (1) である．また，新規配置時に，ユーザが片方しか制約条件を指定していない場合は，そのアプリでは (2)(3) は片方だけ指定される．

(1)-(5) の線形計画の式に基づき，GLPK や CPLEX 等の線形計画ソルバで解を導出することで，再構成の効果を計算する．再構成対象のアプリ数は一定値であり，全アプリでない事もある．ソルバ計算時間は，再構成を計算するアプリの数が増えると増大する．そのため，アプリ一定数の設定は可変とし，100 配置毎に 100 アプリを最適化や，一度に全アプリを最適化等は，ソルバの計算時間に応じてサイズを調整して決定する．

$$S = \sum_{k \in App} \left(\frac{R_k^{after}}{R_k^{before}} + \frac{P_k^{after}}{P_k^{before}} \right) \tag{1}$$

$$\sum_{i \in Device} (A_{i,k}^d \cdot B_{i,k}^p) + \sum_{j \in Link} (A_{j,k}^l \cdot \frac{C_k}{B_k^l}) = R_k^{after} \leq R_k^{upper} \tag{2}$$

$$\sum_{i \in Device} a_i \left(\frac{A_{i,k}^d \cdot B_k^d}{C_i^d} \right) + \sum_{j \in Link} b_j \left(\frac{A_{j,k}^l \cdot B_k^l}{C_j^l} \right) = P_k^{after} \leq P_k^{upper} \tag{3}$$

$$\sum_{k \in App} (A_{i,k}^d \cdot B_k^d) \leq C_i^d \tag{4}$$

$$\sum_{k \in App} (A_{j,k}^l \cdot B_k^l) \leq C_j^l \tag{5}$$

4. 評価

4.1 評価条件

4.1.1 対象アプリケーション

配置アプリは，多くのユーザが利用すると想定されるフーリエ変換と画像処理とする．

フーリエ変換処理 FFT（Fast Fourier Transform）は，振動周波数の分析等，IoT でのモニタリングの様々な場面で利用されている．NAS.FT[28] は，FFT 処理のオープンソースアプリの一つである．備え付けのサンプルテストの 2048*2048 サイズの計算を行う．

MRI-Q[29] は，非デカルト空間の 3 次元 MRI 再構成アルゴリズムで使用されるキャリブレーション用のスキャナー構成を表す行列 Q を計算する．MRI-Q は，性能測定中に 3 次元 MRI 画像処理を行い，Large の 64*64*64 サイズのサンプルデータを使用して処理時間を測定する．

GPU，FPGA 自動オフロード技術 [21][20] により，NAS.FT は GPU で，MRI-Q は FPGA で高速化でき，CPU に比べて 5 倍，7 倍の性能なことが分かっている．

4.1.2 評価手法

配置トポロジーは図 1 の様に 3 層で構成され，クラウドレイヤー拠点数は 5，キャリアエッジレイヤーは 20，ユーザエッジレイヤーは 60，インプットノードは 300 とする．

各拠点に，クラウドでは，サーバは CPU 8 台，GPU 16GB RAM 4 台，FPGA 2 台，キャリアエッジでは，CPU 4 台，GPU 8GB RAM 2 台，FPGA 1 台，ユーザエッジでは，CPU 2 台，GPU 4GB RAM 1 台とする．サーバ 1 台の全リソース使用（GPU では 16GB RAM 利用時）月額はクラウドでは 5 万，10 万，12 万とした．集約効果のため，キャリアエッジ，ユーザエッジは割高になり，クラウドの 1.25 倍，1.5 倍月額とした．

リンクについては，クラウド-キャリアエッジ間は 100Mbps，キャリアエッジ-ユーザエッジ間は 10Mbps の帯域とする．リンクコストについては，100Mbps のリンクは月額 8000 円，10Mbps のリンクは月額 3000 円とした．

アプリが利用するリソースとして，処理時間等は，実際に GPU，FPGA にオフロードした際の値を用いる．NAS.FT は，利用リソース量は GPU 1GB RAM，利用帯域 2Mbps，転送データ量 0.2MB，処理時間 5.8 秒である．MRI-Q は，利用リソース量は FPGA サーバの 10%（FlipFlop, LookUpTable 等の利用数が FPGA の利用リソースとなる），利用帯域 1Mbps，転送データ量 0.15MB，処理時間 2.0 秒である．

まず，900 個のアプリを新規配置する．アプリは，インプットノードから生じるデータを分析する想定で，300 のインプットノードから配置依頼をランダムに生じさせる．

配置依頼数として，NAS.FT：MRI-Q=3:1 の割合で 900 回アプリの配置依頼をする．

ユーザ要求条件として，配置依頼する際に価格条件か応答時間条件かその両方が選ばれる．NAS.FT の場合，価格に関しては月 7500 円 (a) か 8500 円 (b) か 10000 円 (c) 上限か，応答時間に関しては 6 秒 (A) か 7 秒 (B) か 10 秒 (C) 上限かが選択される．MRI-Q の場合，価格に関しては月 12500 円 (x) か 20000 円 (y) 上限か，応答時間に関しては，4 秒 (X) か 8 秒 (Y) 上限が選択される．ユーザ要求として，NAS.FT では a, b, c, A, B, C, aC, bB, bC, cA, cB, cC をそれぞれ 1/12 ずつ，MRI-Q では x, y, X, Y, xY, yX, yY をそれぞれ 1/7 ずつの確率で選択する．ユーザ上限要求が 1 指標の際は別指標の最小化が目的関数に，2 指標の際は一方がランダムに選択されその最小化が目的関数になる．

900 個の配置後は，100 配置毎に，一定数アプリの再構成を行う．配置周期の配置アプリ数は 100 で固定だが再構成対象とするアプリ数は，100，200，400，700, 1000 アプリと可変にし，ユーザ満足度を計算する．

4.2　結果

実験は，GLPK5.0 でのシミュレーションにより行った．図 3(a) は，再構成対象アプリ数を横軸に，実際に再構成したアプリ数を縦軸に取ったグラフであり，図 3(b) は，実際に再構成したアプリの $R_k^{after}/R_k^{before}+P_k^{after}/P_k^{before}$ の平均値を縦軸に取ったグラフである．

計算時間について，新規配置では総計 1000 個を順に計算して配置するので，2 分以内の短時間で終わる．一方，再構成時間は，再構成対象のアプリが増えるにつれ，線形計画の条件式が増えることになるが，100 アプリでは 10 秒以内だったが，1000 アプリでも 2 分以内で終わっている．

図 3(a) に関して，若干ばらつきはあるが，再構成対象アプリ数の約 1 割弱が，実際に再構成されていることが分かる．新規配置では，1 アプリ毎に個別最適な配置をしているが，再構成では複数アプリ一括で適正配置を計算することで，再構成可能なアプリがある程度見つかることが分かる．図 3(b) に関して，再構成を行ったアプリは $R_k^{after}/R_k^{before}+P_k^{after}/P_k^{before}$ の平均が 1.96 程度に改善されていることが分かる．この値は 2 から大きく改善される値ではないが，例えば，NAS.FT をキャリアエッジからクラウドに配置を変えた際は応答時間が 6.6 秒から 7.4 秒になるが，価格が約 8400 円から約 7000 円となるため，値は 2 から 1.954 になる．図 3(b) より，この値は再構成対象アプリ数によらずほぼ一定となっており，再構成対象は全アプリを対象にしなくても良いことが分かる．

4.3　考察

今までの環境適応ソフトウェアの研究は，運用開始前の変換や配置が前提であり，運用開始後に再構成することは

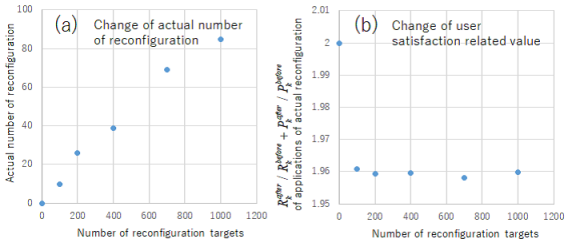

図 3　(a) 実再構成アプリ数の変化，(b) 実再構成アプリの $R_k^{after}/R_k^{before}+P_k^{after}/P_k^{before}$ の変化

想定外だった．他ユーザの配置状況を勘案して，全体最適な配置に再構成可能とする手法の提案により，運用中再構成の有効性を確認できた．運用中再構成の対象として，配置に加え，GPU や FPGA へのオフロードロジックの変更，リソースバランスの変更等も考えており，運用変化に応じた適応と言う環境適応の範囲を広げられる．

また，提案手法により，新規配置時に個別最適に配置したアプリでも再構成することで，約 1 割弱のアプリは $R_k^{after}/R_k^{before}+P_k^{after}/P_k^{before}$ を 2 以下にしており，応答時間や価格等の満足度に直結する指標を運用開始後にも改善できる．この値は再構成対象数によらずほぼ一定なこと，再構成対象数が増えても計算時間はあまり増えないことから，再構成するアプリのユーザに再構成提案や許可を得るビジネス的タイミングに応じて再構成計算数等を設定すればよいと考える．

新規配置時は，ナイーブにクラウドやエッジに配置することに比べ，ユーザ要望に応じた配置が出来る．開始後再構成により，更に高いユーザ満足度での運用が可能となる．

5.　まとめ

本稿では，著者が提案している環境適応ソフトウェアの新たな要素として，運用開始後に他ユーザのアプリ配置状況を勘案して，再配置を行うことで，対象ユーザの満足度を向上させる再構成方式を提案した．

提案方式は，線形計画手法を用いて，再構成対象アプリのユーザ群満足度の向上を目的関数に，試行計算する．具体的には，ユーザの応答時間，価格の要求条件を満たした上で，再構成した際の応答時間，価格から定まるユーザ満足度を計算し，線形計画ソルバで再構成対象の複数アプリに対する解を求める．

複数種のアプリを想定して，シミュレーション実験を行い，提案方式で再構成した際のユーザ満足度向上を確認し，提案方式の有効性を示した．今後は，配置だけでなく，オフロードロジック変更等，再構成対象の拡張を検討する．

参考文献

[1]　A. Putnam, et al., ”A reconfigurable fabric for accelerating large-scale datacenter services,” Proceedings of the 41th Annual International Symposium on Computer Ar-

chitecture (ISCA'14), pp.13-24, June 2014.
[2] O. Sefraoui, et al., "OpenStack: toward an open-source solution for cloud computing," International Journal of Computer Applications, Vol.55, No.3, 2012.
[3] Y. Yamato, "Automatic system test technology of virtual machine software patch on IaaS cloud," IEEJ Transactions on Electrical and Electronic Engineering, Vol.10, Issue.S1, pp.165-167, Oct. 2015.
[4] Y. Yamato, "Proposal of Optimum Application Deployment Technology for Heterogeneous IaaS Cloud," 2016 6th International Workshop on Computer Science and Engineering (WCSE 2016), pp.34-37, June 2016.
[5] Y. Yamato, et al., "Fast and Reliable Restoration Method of Virtual Resources on OpenStack," IEEE Transactions on Cloud Computing, DOI: 10.1109/TCC.2015.2481392, Sep. 2015.
[6] M. Hermann, et al., "Design Principles for Industrie 4.0 Scenarios," Rechnische Universitat Dortmund. 2015.
[7] Y. Yamato, "Proposal of Vital Data Analysis Platform using Wearable Sensor," 5th IIAE International Conference on Industrial Application Engineering 2017 (ICIAE2017), pp.138-143, Mar. 2017.
[8] Y. Yamato, et al., "Proposal of Real Time Predictive Maintenance Platform with 3D Printer for Business Vehicles," International Journal of Information and Electronics Engineering, Vol.6, No.5, pp.289-293, Sep. 2016.
[9] Y. Yamato, et al., "Security Camera Movie and ERP Data Matching System to Prevent Theft," IEEE Consumer Communications and Networking Conference (CCNC 2017), pp.1021-1022, Jan. 2017.
[10] Y. Yamato, et al., "Proposal of Shoplifting Prevention Service Using Image Analysis and ERP Check," IEEJ Transactions on Electrical and Electronic Engineering, Vol.12, Issue.S1, pp.141-145, June 2017.
[11] Y. Yamato, et al., "Analyzing Machine Noise for Real Time Maintenance," 2016 8th International Conference on Graphic and Image Processing (ICGIP 2016), Oct. 2016.
[12] Y. Yamato, "Experiments of posture estimation on vehicles using wearable acceleration sensors," The 3rd IEEE International Conference on Big Data Security on Cloud (BigDataSecurity 2017), pp.14-17, May 2017.
[13] P. C. Evans and M. Annunziata, "Industrial Internet: Pushing the Boundaries of Minds and Machines," Technical report of General Electric (GE), Nov. 2012.
[14] T. Sterling, et al., "High performance computing : modern systems and practices," Cambridge, MA : Morgan Kaufmann, ISBN 9780124202153, 2018.
[15] J. E. Stone, et al., "OpenCL: A parallel programming standard for heterogeneous computing systems," Computing in science & engineering, Vol.12, No.3, pp.66-73, 2010.
[16] J. Sanders and E. Kandrot, "CUDA by example : an introduction to general-purpose GPU programming," Addison-Wesley, 2011.
[17] Y. Yamato, et al., "Automatic GPU Offloading Technology for Open IoT Environment," IEEE Internet of Things Journal, DOI: 10.1109/JIOT.2018.2872545, Sep. 2018.
[18] Y. Yamato, "Study and Evaluation of Automatic GPU Offloading Method from Various Language Applications," International Journal of Parallel, Emergent and Distributed Systems, Taylor and Francis, DOI: 10.1080/17445760.2021.1971666, Sep. 2021.
[19] Y. Yamato, "Study and Evaluation of Improved Automatic GPU Offloading Method," International Journal of Parallel, Emergent and Distributed Systems, Taylor and Francis, DOI: 10.1080/17445760.2021.1941010, June 2021.
[20] Y. Yamato, "Automatic Offloading Method of Loop Statements of Software to FPGA," International Journal of Parallel, Emergent and Distributed Systems, Taylor and Francis, DOI: 10.1080/17445760.2021.1916020, Apr. 2021.
[21] Y. Yamato, "Study of parallel processing area extraction and data transfer number reduction for automatic GPU offloading of IoT applications," Journal of Intelligent Information Systems, Springer, DOI:10.1007/s10844-019-00575-8, 2019.
[22] E. Su, et al., "Compiler support of the workqueuing execution model for Intel SMP architectures," In Fourth European Workshop on OpenMP, Sep. 2002.
[23] S. Wienke, et al., "OpenACC-first experiences with real-world applications," Euro-Par 2012 Parallel Processing, pp.859-870, 2012.
[24] M. Wolfe, "Implementing the PGI accelerator model," ACM the 3rd Workshop on General-Purpose Computation on Graphics Processing Units, pp.43-50, Mar. 2010.
[25] C. C. Wang, et al., "Toward optimal resource allocation of virtualized network functions for hierarchical datacenters," IEEE Transactions on Network and Service Management, Vol.15, No.4, pp.1532-1544, 2018.
[26] NVIDIA vGPU software web site, https://docs.nvidia.com/grid/index.html
[27] J. H. Holland, "Genetic algorithms," Scientific american, Vol.267, No.1, pp.66-73, 1992.
[28] NAS.FT website, https://www.nas.nasa.gov/publications/npb.html
[29] MRI-Q website, http://impact.crhc.illinois.edu/parboil/