MOBO-NFV: Automated Tuning of a Network Function Virtualization System using Multi-Objective Bayesian Optimization

Pietro Mercati
Intel Corporation
pietro.mercati@intel.com

Bin Li
Intel Corporation
bin.li@intel.com

Mesut Ergin
Intel Corporation
mesut.a.ergin@intel.com

Charlie Tai
Intel Corporation
charlie.tai@intel.com

Michael Kishinevsky
Intel Corporation
michael.kishinevsky@intel.com

Boris Serafimov
Intel Corporation
boris.k.serafimov@intel.com

Subhiksha Ravisundar
Intel Corporation
subhiksha.ravisunda@intel.com

Eoin Walsh
Intel Corporation
eoin.walsh@intel.com

Thomas Long
Intel Corporation
thomas.long@intel.com

***Abstract*—With today's dramatic rise of Internet traffic, telecommunication providers increasingly rely on Network Function Virtualization (NFV) for the fast deployment and flexible orchestration of infrastructure workloads. Characterizing and tuning to run at maximum performance and/or minimum power is a critical business goal for companies.**

In this paper, we implement and characterize the Virtual Broadband Network Gateway (vBNG), an important NFV workload, running on a real Intel® Xeon® server. Then, we present a fully automated approach based on Multi-Objective Bayesian Optimization (MOBO-NFV) for tuning real systems running NFV workloads. We demonstrate improvements in power and performance by up to 18% and 17% respectively. We envision that the proposed methodology and results will benefit service providers for resource planning and orchestration.

***Index Terms*—NFV, vBNG, resource management, workload optimization, network, Bayesian optimization.**

I. Introduction

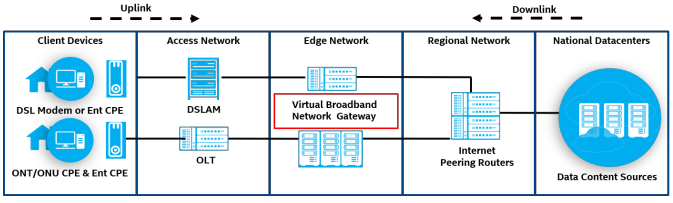


Fig. 1. Operating context of Edge Network and Virtual Broadband Network Gateway

Internet traffic today is seeing a dramatic rise [1], [2]. According to forecasts [3], in 2023 the total number of Internet users would reach 5.3 billion, with the consumer segment having nearly three-fourths share of total devices and connections. Therefore, building and operating an efficient network infrastructure is urgent and vital. In the software space, a recent novelty is Network Function Virtualization (NFV) [5], a computing paradigm that implements infrastructure workloads on Virtual Machines (VMs) or portable containers for seamless execution on different hardware platforms [25]. This enables a fast and scalable approach for the deployment of NFV.

978-3-903176-32-4 © 2021 IFIP

The primary goal of telecommunication providers is to achieve maximum execution efficiency of NFV workloads and of the servers they run on. This is important because it translates into better bandwidth utilization, lower communication latency, and lower power consumption, which directly impact the returns on investments and the total cost of ownership (TCO). Therefore, it is fundamental to develop practical techniques that can automatically determine server system settings for improved efficiency, while relying only on outputs observable at a system-level. In fact, while a more detailed knowledge of the NFV software would be helpful to refine the optimization, given the VM or containerized model, such internal information is generally not exposed. Resource management should be also portable and therefore hardware-agnostic. Finally, it is important to optimize simultaneously for multiple conflicting objectives, such as power and performance, since they have non-trivial tradeoffs and both impact the TCO and network efficiency.

An ubiquitous and representative example of a network function is the Broadband Network Gateway (BNG), which is the aggregation point where subscribers are authenticated and given access to services and data (VoIP, storage, TV, etc.). The BNG is a large fixed-function appliance based on ASICs, which is efficient but has low flexibility. The Virtual Broadband Network Gateway (vBNG) [7], depicted in Figure 1, is a virtual software instantiation of BNG's appliance functions and thus it is a very important NFV workload. In Figure 2 we show the vBNG workload performance (system throughput) and power efficiency (expressed as performance per Watt) measured on our system under test, described in Table I (more details in Section IV). In the diagram on the left, we fix the core frequency to the maximum and sweep the uncore frequency. On the right, we fix the uncore

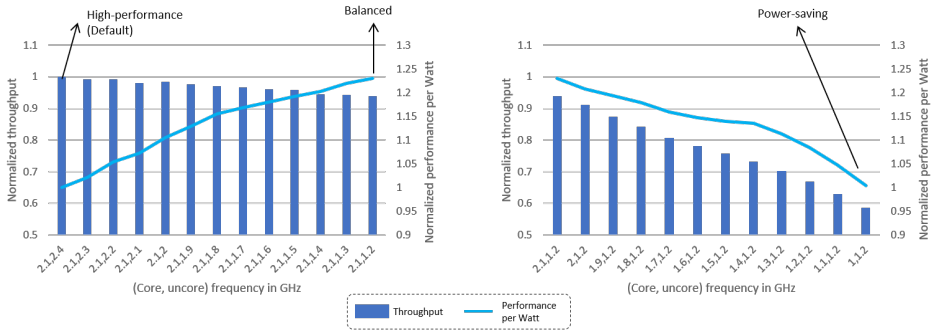


Fig. 2. Impact of changing uncore frequency (left) and core frequency (right) on vBNG throughput and performance per Watt

frequency to the minimum and sweep the core frequency[1]. We observe that different settings for this vBNG workload could have highly non-trivial interactions and affect different objectives in different ways. **Therefore, it is fundamental to develop automated techniques for the tuning of critical system parameters that can capture interaction between parameters and optimize for multiple objectives**.

For ease of discussion, we identify three modes of operation for vBNG with different configurations in the system: *high-performance* (which we assume as the default configuration), *balanced* and *power-saving*. These deliver respectively the best performance, performance per Watt and power. We will refer to these in our experimental results.

Pushing the boundary of vBNG performance is of primary importance for providers, and requires tuning system configurations for optimized behavior. System tuning is traditionally a manual activity carried out by experts. This approach lacks scalability, is time-consuming and delivers sub-optimal solutions. Determining such configurations in an automated fashion is thus desirable, but also very challenging, since vBNG is designed to run seamlessly on different hardware platforms.

Improving the performance and the power efficiency of a NFV system can be treated as a "black-box" optimization problem, for which the objective function (such as power or performance) has no known analytical formulation, but can be evaluated by changing input configurations and measuring the output optimization objectives. Bayesian Optimization (BO) is a theoretically-grounded and practical approach for the solution of black-box optimization problems [8]. It has been successfully adopted in different optimization tasks [9]. BO first samples the system to obtain a few initial training points and uses probabilistic machine learning to build a surrogate model of an objective function. Using the surrogate model's prediction and uncertainty, BO selects a system configuration to evaluate. Each selected configuration is likely to either improve the best result in the vicinity of already evaluated promising solutions, or to reduce uncertainty in unexplored regions of the solution space. The selected configurations are evaluated on the system and the result is used to update the surrogate model. The process then iterates.

BO has been successfully used in a recently proposed framework called CLITE [10], which applies it to the problem of co-locating a mix of latency-critical and background jobs in a multi-node system. Another recent publication successfully used BO for optimizing cloud workloads [11]. In this work we consider a different class of workloads and focus on the tuning of hardware parameters on a real system. Also, while both [11] and [10] only consider a more traditional single-objective setting, in this work we apply multi-objective optimization and show that our technique can improve different metrics simultaneously.

Contribution: In this work, we analyse and characterize the vBNG workload on a real Intel® Xeon® server. Then, we present MOBO-NFV, an approach based on Multi-Objective Bayesian Optimization (MOBO) for improving power and performance of server systems running Network Function Virtualization (NFV) workloads. Our solution automatically discovers configurations that improve power and performance by up to 18% and 17% respectively. In addition, we demonstrate the superiority of MOBO compared to random sampling and a multi-objective Genetic Algorithm.

II. vBNG Overview

In this section, we describe the structure and functionality of Virtual Broadband Network Gateway (vBNG). For further details, we refer to [7].

vBNG consists of a control plane and a data plane. The latter is further divided into downlink (DL) and uplink (UL). The control plane handles authentication and management of subscribers, while the data plane manages bidirectional traffic of data packets between the Internet and the subscriber's home. Figure 3 represents the reference pipeline that implements the processing stages of control and UL/DL data planes respectively.

The uplink data plane handles the user's traffic from a home location to the provider's network, while the downlink data plane manages the traffic in the opposite direction, scheduling it to the different users connected to the BNG. The average traffic flow through uplink is generally eight to five times smaller than the traffic through downlink. This is because

[1]Recent processor architectures provide the capability to set each processor core frequency and the uncore frequency separately [26].

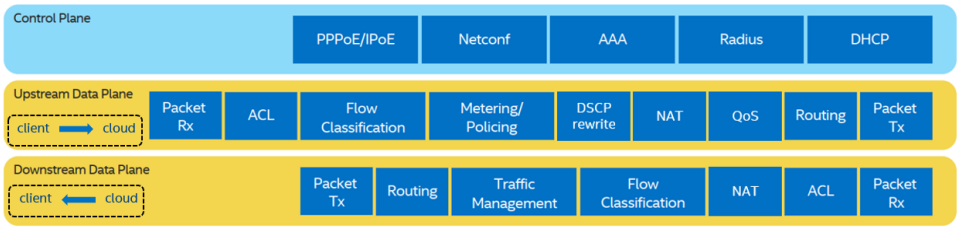


Fig. 3. vBNG control plane and uplink & downlink data planes

today, despite applications such as Instagram and Snapchat are encouraging uploads, users are still predominantly consumers of data.

When a packet is ready for upload, it enters the first stage of the uplink data plane *Packet Rx*. Here a dedicated driver (the poll mode driver of the data plane development kit [27]) receives bursts of frames from the network interface controller (NIC) and sends them to an uplink thread for the remaining processing stages. Once completed, the same driver is used to send bursts of frames back to the NIC port.

Downstream has a similar pipeline, where a major difference is represented by the *Traffic Management* stage. Here, each packet goes through a QoS prioritization mechanism for transmission to the access network. Prioritization follows a hierarchical scheme, where each leaf corresponds to a pipe assigned to a single subscriber. This is important to enforce priority and policy since subscribers cannot get more bandwidth than they paid for.

Both uplink and dowlink include an *Access Control List* library to apply a predefined list of filters to each packet to verify its properties and approve or reject.

A single vBNG instance is implemented in a dedicated container, and it has two separate pipeline applications for UL and DL. This separation helps with scheduling and CPU resource usage. Since DL handles more traffic, it is usually assigned to a full physical core, with two logical cores on the Intel® Xeon® system used for experiments, corresponding to two hardware threads with Intel® Hyper-Threading Technology [31], while UL is assigned to a single logical core. Both pipelines can report telemetry data, which are logged separately. The "container" used in our experiment for virtualization is a Docker container, but other forms of virtualization, such as VMs, can be used as well.

In Section IV we will describe in more detail the specific implementation of vBNG used in our evaluation.

III. MOBO-NFV Framework

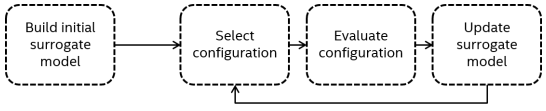


Fig. 4. MOBO algorithm

In this section we describe the MOBO-NFV framework, which has two components, the MOBO algorithm and the target NFV system. For a more general introduction to Bayesian Optimization, we refer the reader to [8]. We assume that the scenario is static, i.e. the input traffic and system parameters other than those changed by the optimization are approximately constant during the execution (within some noise margins).

Figure 4 illustrates the proposed MOBO algorithm. An initialization stage uses a previously obtained dataset to train the initial surrogate model. Such initial dataset is usually collected by randomly sampling the input space.

Then, MOBO starts a loop of iterations. First, it leverages the surrogate model to select the next candidate configuration to evaluate. Second, the candidate configuration is applied to the NFV target system and evaluated. Finally, the measurement provides a new data point, which is used to update the surrogate model. For simplicity we assume that the algorithm generates and evaluates one configuration at a time, but MOBO in general can generate a batch of candidate configurations at each iteration, which can be evaluated in parallel when multiple systems are available.

The use of a probabilistic surrogate model over a traditional one is what makes the optimization "Bayesian". A traditional machine learning model, such as linear regression or neural network regression, only returns a predicted value for each input. A probabilistic model, instead, returns both a prediction and an estimate of uncertainty of that prediction. This information is used in MOBO at each iteration, guiding the search of candidate configurations to evaluate on the target system. Predictions and uncertainties are used to compute an "acquisition function", which takes large values in the vicinity of previous promising configurations or in regions with large uncertainty. In this way, it automatically balances local and global search. The configuration selected for the next evaluation is the one that maximizes the acquisition function.

We use a Gaussian Process (GP) as a surrogate model for each objective [12]. The GP is highly "data-efficient", meaning that it achieves good accuracy with a relatively small number of training points, and it can be easily customized to model different classes of functions by changing its "kernel". Finally, it has convenient analytical properties which allow to train it with a closed-form approach. On the downside, GP training time has a cubic time complexity from the size of the training set. This is not a problem for us, since most datasets for our application have at most a few hundred points.

For clarity, we briefly introduce here concepts that are

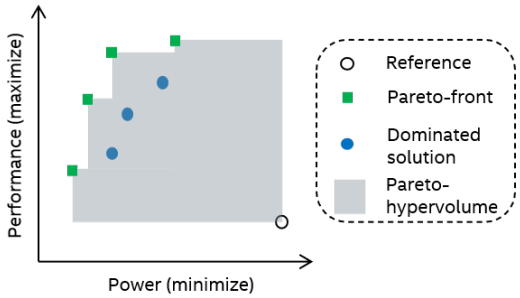


Fig. 5. An example of Pareto-hypervolume

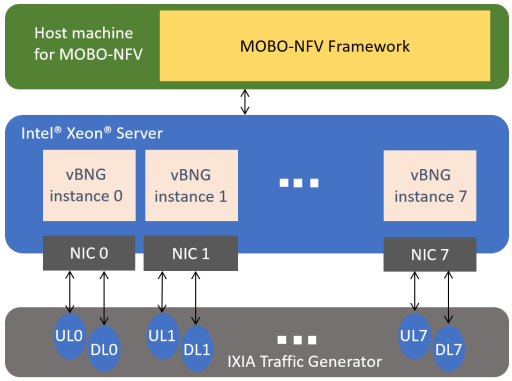


Fig. 6. A diagram showing the system setup.

useful to interpret our results. Figure 5 reports an example of a multi-objective setting (power and performance). Pareto-optimal solutions lie in the upper-left region of the figure. This is because we want to reduce power and improve performance at the same time. All other solutions are dominated by the Pareto-optimal ones. The gray area delimited by an arbitrary reference point and the Pareto-front is the so-called Pareto-hypervolume. To compare two sets of solutions (for example MOBO vs. GA) we will use the concepts of Pareto-front and Pareto-hypervolume.

We adopt "SMSego" [13] as acquisition function, because it enables multi-objective optimization and it is fast to compute, while delivering optimization performance comparable to more advanced but more complex functions, such as those based on entropy. For each point in the solution space, this function takes as input the corresponding prediction and uncertainty from surrogate models of each objectives. For each solution it estimates how likely it is that evaluating it would extend the Pareto-hypervolume of solutions.

Finally, to fully automate this process we include a "software management" component which handles the iterative interaction between the optimization algorithm and the system. This component automatically applies the selected settings to the NFV target system, triggers and stops the measurement of power and performance, returns the result to the optimization algorithm and activates it to select the next configuration for evaluation. Automation is critical to achieve higher productivity, because it removes the latency of human intervention.

Note that the proposed solution naturally extends to other scenarios that include heterogeneous workloads with diverse priorities: one just needs to reformulate the objective functions such that performance metrics of all workloads are included. For example, if there are two workloads with different priorities, we can interpret their performance metrics as separate objectives. MOBO is then able to prioritize the search based on the "weight" of each objective. To avoid explosion of objectives, an alternative implemented technique aggregates multiple workloads in a single objectives with priority weights if needed.

IV. Experimental Setup

Our experimental setup, shown in Figure 6, has three components: the target NFV system that needs tuning, a traffic generator to send and receive packets from the server, and a host machine to run the MOBO algorithm.

In our experiments, we run the vBNG workload on a two-socket server with Intel® Xeon® Gold 6252 CPU [20] (Cascade Lake). Table I reports details of this setup. The CPU has a 35.75MB last-level cache (LLC) with eleven ways that can be partitioned among applications through Intel® Resource Director Technology (RDT) Cache Allocation Technology (CAT) [28]. RDT also allows to allocate memory bandwidth per application via Memory Bandwidth Allocation (MBA) [29]. Intel® Data Direct I/O (DDIO) [30] subsystem is allocated two LLC ways for I/O traffic, while the vBNG application can use up to eleven LLC ways (including the two shared with the DDIO). Core frequency is configured at fixed rates specified per experiment, as opposed to OS or platform P-state governors. The server has eight Intel® XXV710 25GbE NICs [21] installed on the second socket of the dual socket server that are connected to the IXIA XGS2 and NOVUS 100G traffic generator [22].

TABLE I
Evaluation Configuration

Server	Description
Processor	Intel® Xeon® Gold 6252 CPU @ 2.10GHz nominal
NIC	Eight Intel® XXV710
BIOS settings	As specified in vBNG white paper [7]
Core frequency	Fixed per experiment configuration
Traffic generator	IXIA XGS2 and NOVUS 100G

TABLE II
Workload Configuration

	Uplink	Downlink
Frame Size	128B	512B
Percentage of overall traffic	11%	89%
Subscribers	4K	4K

We run the base vBNG pipeline configuration from Figure 3, which is the typical configuration for vBNG. The uplink pipeline stages are: Packet Rx, ACL, Flow Classification, Metering/Policing, Routing, and Packet Tx. The downlink pipeline stages are: Packet Rx, ACL, Flow Classification, Traffic Management, Routing, and Packet Tx.

The number of vBNG instances in the experiment is eight and the downlink:uplink traffic ratio is 8:1. Table II summarizes the vBNG workload configuration for each vBNG instance. Each uplink pipeline uses one logical core (one hardware thread of the two-threaded core), and each downlink pipeline uses two logical cores (full physical CPU core). The same traffic profile is applied to all the eight vBNG instances.

TABLE III
TUNING RANGE OF PARAMETERS

Parameter	Range
Core (UL, DL) frequency	1.0 – 2.1 GHz with step 0.1 GHz
Uncore frequency	1.2 – 2.4GHz. with step 0.1 GHz
RDT CAT	1 – 11 ways with step 1
RDT MBA	10% – 100% with step 10%

In our exploration, we focus on tuning four hardware parameters: core frequency, uncore frequency, RDT CAT and RDT MBA. For core frequency, we experiment both with same core frequency settings between UL and DL cores and with separate settings, which delivers further improvements. The tuning range for each parameter is summarized in Table III. The default configuration in our experiment has core frequency at 2.1GHz, uncore frequency at 2.4GHz, 11 ways for RDT CAT and 100% RDT MBA.

We use collectd [23] to collect vBNG application throughput data, and the Processor Counter Monitor tool [24] to collect platform power consumption. The throughput and power consumption of the system are the optimization objectives for our MOBO algorithm, which issues configurations of parameters to the server platform running vBNG workload for performance/power optimization. For each configuration proposed by MOBO, we apply it to the system and run the vBNG workload for a fixed interval of time (60 seconds, determined empirically). We collect the average throughput and power consumption over such interval, and send the results back to the MOBO algorithm. As mentioned, MOBO is deployed on a different machine from the one running vBNG, and the interaction is implemented via *ssh*. In this way, MOBO does not interfere with workload execution. An alternative could be to deploy MOBO on a dedicated core, but if the memory is shared among all cores it would require the optimization engine to limit its memory usage. The MOBO algorithm is implemented mainly in Python. We use the *GPy* library to implement Gaussian Processes. Model training and acquisition function optimization rely on the *numpy* and *scipy* libraries. Interaction via *ssh* is implemented using the *paramiko* library.

We implemented two algorithms for comparison: random search and multi-objective Genetic Algorithm (GA). This is an in-house Python implementation in flavor of NSGA-II [35]. The GA takes the history of evaluations as input and orders the input-output pairs based on a fitness scoring function. Then, it picks the inputs of the two fittest pairs, called "parents", and generate a new input by copying part of the input components from the first parent and the other from the second parent (crossover). Then, it also randomly changes one or more components (mutation). Random sampling and GA leverage the same framework implementation of MOBO, but use a different strategy to obtain the next configuration to evaluate.

V. RESULTS

In this section we first present the analysis and characterization of the vBNG workload, then we show the results of tuning the system using MOBO-NFV.

A. Analysis and Characterization

In this section, we perform basic vBNG workload characterization and highlight two aspects. First, we investigate the scalability of performance with respect to the number of active vBNG instances. For this, we sweep the number of vBNG instances from 1 to 8 and measure the corresponding throughput. Figure 7 reports these measurements, where throughput is normalized over the maximum achieved throughput with 8 instances at default configuration. The results show that vBNG scales almost linearly.

Second, we evaluate the impact of LLC cache way allocation to vBNG in the configuration with 8 instances running DL traffic only. Since 89% of the traffic is DL, we focus on DL traffic for cache sensitivity analysis here. The system has 11 LLC ways in total, and by default two of them are allocated for DDIO and are shared with the vBNG application. It is also possible to restrict cache ways for vBNG workload to the 9 cache ways not used by DDIO. In Figure 8 we report measurements for both cases, the results are normalized over the maximum achieved throughput with 8 instances (both UL and DL) to have a consistent normalization point throughout the paper. From Figure 8, we can see that in the case vBNG cache ways are overlapping with DDIO ways, vBNG has similar throughput when allocating 8 to 11 ways to vBNG. vBNG performance starts to be affected with less than 7 ways allocated. This indicates we can potentially take away a few LLC cache ways from vBNG without impacting vBNG performance. And those spare LLC cache ways could potentially be used by other applications running on the server platform and improve server utilization. Further, if we allocate LLC cache ways for vBNG in a fashion that is non-overlapping with DDIO ways, we found that vBNG performance are further improved by 6.5%. This is because such LLC way allocation reduces the cache contention between vBNG and DDIO at LLC, leading to further performance improvement for vBNG. In the following experiments, we will use the second configuration where DDIO and vBNG does not share two ways when applicable.

B. Optimization

In the following experiments we take average measurements of power and throughput over an interval of 60 seconds. The time it takes for MOBO to update the GP model and suggest a new configuration to evaluate depends on the number of input parameters, size of training set, and on the implementation details. In our cases this is in the order of a few seconds.

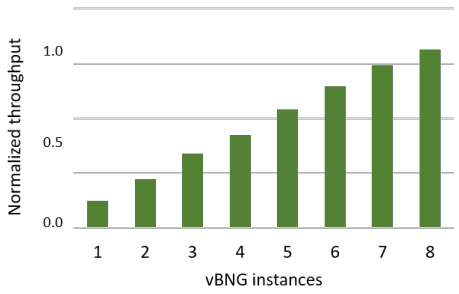


Fig. 7. Aggregated throughput for IPoE scaling of vBNG instances

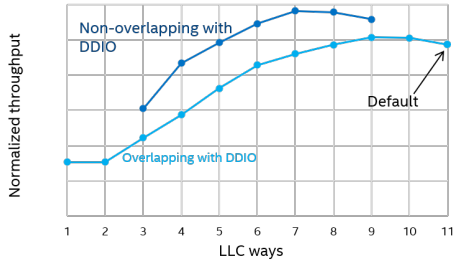


Fig. 8. vBNG throughput vs. number of allocated LLC ways

For each configuration we run 3 repetitions and consider only the median result to mitigate noise, so an automated experiment with 250 iterations takes about 12 hours to complete. This runtime can be shortened considerably by improving the BO algorithm and its implementation, by developing a faster evaluation strategy and by avoiding repetitions. This however falls out of the scope of this paper, and we will address it in our future work.

All reported measurements of the objectives are normalized dividing by power or throughput of the default configuration. In all the MOBO experiments, the first 20 configuration are sampled randomly to initialize the GP model. All these experimental parameters have been chosen empirically.

In the first experiment, we consider core and uncore frequency only as input parameters, while fixing CAT and MBA to the maximum. We then run 100 iterations of MOBO. For each iteration, we measure average throughput and average power consumption (also true for the following experiments), reported respectively in Figure 9(a) and 9(b). In Figure 9(c), we report the number of dominators of each solution, where we limit the maximum at 5 for easier plotting. Recall that solutions with zero dominators are Pareto-optimal and belong to the Pareto-front. We observe that MOBO quickly and systematically discovers Pareto-optimal solutions, many of them right after the random initialization stage (at iteration 20). This represents a clear advantage over manual exploration.

We then report this Pareto-front in the "objective space" of power and throughput, in Figure 10(a). Thanks to this we can identify configurations for different tradeoffs. We consider three of them as an example and report the values of input parameters and objectives in Figure 10(b). The default configuration is high-performance, it has maximum core and uncore frequency, and delivers the highest throughput. For clarity, we report also the values for CAT and MBA, which are fixed for this experiment. The balanced configuration has maximum core frequency and minimum uncore frequency, and it is the one with the highest performance per Watt (an indication of this is that it lies at the "knee" of the Pareto-front). The Power-saving configuration has minimum core and uncore frequency and delivers the lowest power consumption at the cost of performance degradation. We now use this result to show that the benefits of MOBO are more evident when we increase the number of input parameters from two to four.

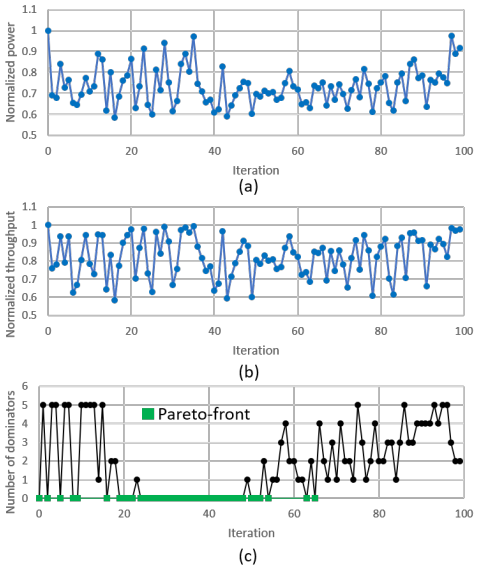
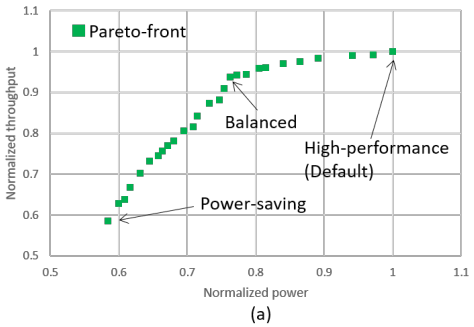


Fig. 9. Bayesian optimization of vBNG system with two input parameters (core and uncore frequency) showing (a) Power, (b) throughput and (c) number of dominators per each iteration.

In the second experiment, we also include CAT and MBA in the optimization, for a total of four parameters. This gives now more than 15000 possible configurations, which is unreasonable to approach with manual tuning. In this case and in the following ones, we run for 250 iterations (the initial 20 are random). In Figure 11(a), we report the number of dominators, and we observe that BO can discover numerous Pareto-optimal solutions quickly. In Figure 11(b), we report all the solutions in the objective space, highlighting the Pareto-front. We then indicate five specific configurations. We report their detail in Figure 11(c) and discuss them in the following.

As the results show, in this case the default configuration does not belong to the Pareto-front since a few solutions dominate it. For illustration let us examine two Pareto optimal configurations labelled in Figure 11(b) as Perf1 and Perf2.

Perf1 has both larger throughput and lower power compared to the default. Perf1 can achieve larger throughput thanks to less LLC ways allocated to the vBNG application, and can reduce the contention with DDIO traffic at LLC, leading to improved performance. With uncore frequency at 2.3GHz, it helps reducing power consumption while having minimal

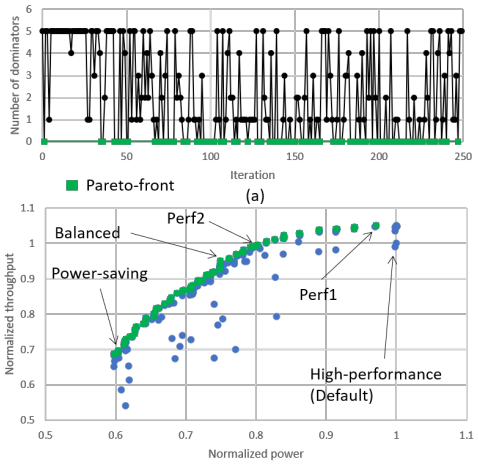


(a)

Configuration	Core frequency [Ghz]	Uncore frequency [Ghz]	CAT	MBA [%]	Norm. power	Norm. throughput
High-performance (Default)	2.1	2.4	11	100	1	1
Balanced	2.1	1.2	11	100	0.76	0.93
Power-saving	1.0	1.2	11	100	0.58	0.58

(b)

Fig. 10. Bayesian optimization of vBNG system with two input parameters (core and uncore frequency) showing (a) power and throughput of all solutions, (b) selected configurations.



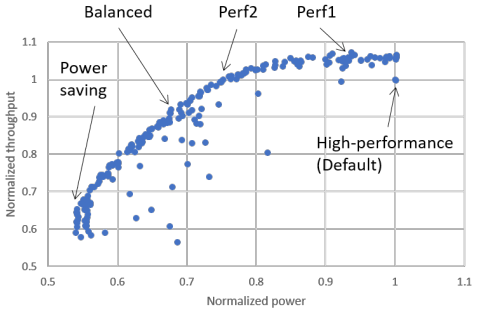
(a)

(b)

Configuration	Core frequency [Ghz]	Uncore frequency [Ghz]	CAT	MBA [%]	Norm. power	Norm. throughput
High-performance (Default)	2.1	2.4	11	100	1.00	1.00
Perf1	2.1	2.3	7	80	0.97	1.05
Perf2	2.0	1.8	7	60	0.81	1.00
Balanced	1.8	1.6	7	100	0.74	0.95
Power-saving	1.1	1.2	7	50	0.59	0.68

(c)

Fig. 11. Bayesian optimization of vBNG system with four input parameters (core frequency, uncore frequency, CAT, MBA) showing (a) power and throughput of all solutions, (b) selected configurations.



(a)

Configuration	DL Core frequency [Ghz]	UL Core frequency [Ghz]	Uncore frequency [Ghz]	CAT	MBA [%]	Norm. power	Norm. throughput
High-performance (Default)	2.1	2.1	2.4	11	100	1.00	1.00
Perf1	2.1	1.2	2.4	7	100	0.91	1.06
Perf2	2.0	1.2	1.8	7	100	0.76	1.00
Balanced	1.8	1.1	1.5	7	100	0.67	0.91
Power-saving	1.3	1.0	1.2	7	40	0.56	0.70

(b)

Fig. 12. Bayesian optimization of vBNG system with five input parameters (DL core frequency, UL core frequency, uncore frequency, CAT, MBA) showing (a) power and throughput of all solutions, (b) selected configurations.

impact on performance. Also the remaining LLC ways and memory bandwidth may be potentially allocated to other applications to improve server utilization. Perf2 instead achieves a throughput similar to default, while reducing power consumption by 18%. Compared to the balanced configuration with only two parameters obtained in Figure 10(b), the balanced configuration here has 4% better performance per Watt, while the power-saving can deliver 17% more throughput with only a very small increase in power. This does not require human intervention, it is completely automated and it can be run overnight in the worst case. Therefore, including additional parameters (CAT and MBA in this case) enables MOBO to explore a vast space of configurations and automatically discover significant improvements.

To investigate further, in the third experiment we consider five input parameters instead of four, by running DL and UL instances of vBNG on different physical cores, thus controlling the DL and UL core frequency separately. Figure 12 reports the results.

Thanks to this additional degree of freedom, compared to those of the second experiment, Perf1 and Perf2 are both better. Balanced has lower throughput, but also a lower power which results in a better performance per Watt. Power-saving is comparable.

In the above experiments, we just showed some representative configurations as an example. In general the Pareto-front results allow operators to select optimal configurations at different operating conditions, e.g. optimized throughput, balanced power and performance, minimized power. By running additional experiments at various traffic load levels, the network operator can construct a configuration table from the optimization results and utilize the table to dynamically adopt the best settings based on observed load at runtime.

To better highlight the advantages of MOBO, we next compare the results of Figure 11(b) against two other techniques: random sampling and Genetic Algorithm. For clarity, we make a qualitative comparison.

In Figure 13 we compare MOBO against random sampling. For this we use a different seed, to avoid replicating the random configurations used to initialize MOBO. A visual

comparison shows that the solutions found by MOBO are more dense and closer to the front. The front itself is more extended in the central region, which is of higher interest. Instead, the random front is more extended in the lower left corner. This happens because random sampling is doing exploration only. The drawback is that it wastes majority of iterations evaluating very poor solutions. MOBO, instead, balances exploration with exploitation to expand the front in more promising regions.

Finally, in Figure 14 we compare MOBO against GA. In this case we observe that the Pareto-hypervolume of MOBO is larger than the one obtained by GA. This indicates that MOBO, thanks to the use of its internal surrogate model, achieves a faster convergence than GA.

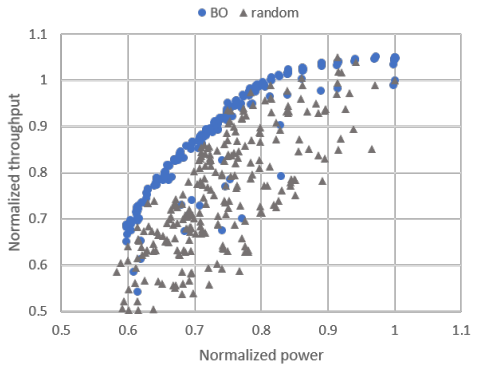


Fig. 13. Comparison of MOBO and random sampling

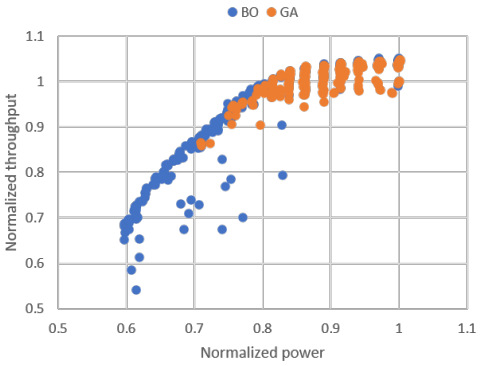


Fig. 14. Comparison of MOBO and GA

VI. Related Work

Network Function Virtualization was proposed in 2012 [6] as a new way to design and deploy networking services more quickly, by leveraging virtualization technologies. A comprehensive survey on NFV from 2016 [5] highlights resource management as a fundamental aspect of NFV deployment. It also presents system energy efficiency and performance improvement among the most important research challenges and outlines related work in the area. Reference [7] provides a description of the vBNG workload.

Recent work on NFV management and orchestration mainly focuses on improved function splitting and allocation. Work in [14] presents a cost-efficient architecture for the cloud radio access network, formulates the functional split problem as an integer linear problem and solves it using an evolutionary algorithm. Work in [15] describes an approach based on a mixed-integer linear programming formulation for NFV placement and traffic routing, and proposes a fast heuristic for the solution. These works consider workload allocation, but they do not consider tuning system settings such as operating frequency and cache ways allocation, and are thus orthogonal to us.

Resource management for server systems and data centers is a broad and active research area [16]. For example, Heracles [33] develop heuristics to dynamically manage resources to isolate latency-sensitive jobs while maximizing resources for best effort jobs. The RLDRM [34] work presents a deep reinforcement learning approach to dynamically tune LLC ways between high-priority NFV workloads and best-effort workloads on server platforms to meet high-priority NFV workload's service level objective (SLO), while improving server utilization and reducing TCO cost.

Bayesian Optimization today is commonly used for tuning parameters of complex machine learning models, and has also been applied to a vast spectrum of non-traditional applications. We refer to the excellent surveys [8] and [9] for an overview of past BO uses. More recently, BO has been used for design space exploration of hardware accelerators [17], automated optimization of analog circuits [18], assigning cloud VM resources [19]. In this work, we showed the benefits of BO also for the case of multi-objective optimization of NFV systems.

VII. Conclusion

In this paper, we proposed MOBO-NFV, an approach for automated multi-objective Bayesian Optimization of systems running NFV workloads. We implemented and evaluated it on a real server system running the vBNG workload. With the proposed approach, we automatically obtained configurations that improve vBNG power and performance by up to 18% and 17% respectively.

The Pareto-front solutions discovered by MOBO can be used by network operators to either statically tune the system with targeted performance while saving power consumption, or can be populated into tables to dynamically tune system configurations based on various traffic load levels to reduce TCO. While we focused on vBNG due to its importance, the proposed approach can be easily extended for tuning and optimization of other NFV workloads as well.

Acknowledgments

We thank Prof. José Miguel Hernández-Lobato and Marton Havasi (University of Cambridge) for useful discussions and collaboration on Bayesian Optimization. We also thank Yipeng Wang and Sankar Chokkalingam (Intel Corporation) for precious help with setup and experiments.

REFERENCES

[1] [online] https://discover.plume.com/wfh-dashboard

[2] [online] https://www.cisco.com/c/dam/m/en_us/solutions/service-provider/vni-forecast-highlights/pdf/Global_2020_Forecast_Highlights.pdf

[3] [online] https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html.

[4] [online] https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/.

[5] R. Mijumbi, J. Serrat, J. Gorricho, N. Bouten, F. De Turck and R. Boutaba, "Network Function Virtualization: State-of-the-Art and Research Challenges", IEEE Communications Surveys & Tutorials, 2016.

[6] R. Guerzoni, "Network functions virtualisation: An introduction benefits enablers challenges and call for action. Introductory white paper", 2012.

[7] [online] www.intel.com/content/dam/www/public/us/en/documents/platform-briefs/broadband-network-gateway-architecture-study.pdf

[8] Brochu, Eric and Cora, Vlad M. and de Freitas, Nando, "A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning", 2010.

[9] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams and N. de Freitas, "Taking the Human Out of the Loop: A Review of Bayesian Optimization", 2016.

[10] T. Patel and D. Tiwari, "CLITE: Efficient and QoS-Aware Co-Location of Multiple Latency-Critical Jobs for Warehouse Scale Computers", HPCA 2020.

[11] Y. Shi, Z. Peng, R. Wang and Z. Bian, "Adaptive Cloud Application Tuning with Enhanced Structural Bayesian Optimization", CloudCom 2019.

[12] Rasmussen, C.E., Williams, C.K.I.. "Gaussian Processes for Machine Learning", MIT Press, 2006.

[13] D. Hernández-Lobato, J. M. Hernández-Lobato, A. Shah, and R. P. Adams. "Predictive entropy search for multi-objective bayesian optimization", ICML, 2016

[14] S. Matoussi, I. Fajjari, S. Costanzo, N. Aitsaadi and R. Langar, "5G RAN: Functional Split Orchestration Optimization", IEEE Journal on Selected Areas in Communications, 2020.

[15] D. Johansson, A. Kassler and J. Taheri, "On the Energy Cost of Robustness and Resiliency for Virtual Network Function Placement", NFV-SDN, 2018.

[16] Ali Hammadi, Lotfi Mhamdi, "A survey on architectures and energy efficiency in Data Center Networks", Computer Communications, 2014.

[17] B. Reagen et al., "A case for efficient accelerator design space exploration via Bayesian optimization", ISLPED, 2017.

[18] W. Lyu et al., "An Efficient Bayesian Optimization Approach for Automated Optimization of Analog Circuits", IEEE Transactions on Circuits and Systems, 2018.

[19] C. Hsu, V. Nair, V. W. Freeh and T. Menzies, "Arrow: Low-Level Augmented Bayesian Optimization for Finding the Best Cloud VM", ICDCS, 2018.

[20] [online] https://ark.intel.com/content/www/us/en/ark/products/192447/intel-xeon-gold-6252-processor-35-75m-cache-2-10-ghz.html.

[21] [online] https://www.intel.com/content/www/us/en/products/docs/network-io/ethernet/network-adapters/ethernet-xxv710-brief.html.

[22] [online] https://www.ixiacom.com/products/novus-qsfp28-10050402510 ge-load-modules.

[23] [online] https://collectd.org

[24] [online] https://github.com/opcm/pcm

[25] [online] https://cntt-n.github.io/CNTT/

[26] D.L. Hill, T. Huff, S. Kulick, and R. Safranek, "The uncore: A modular approach to feeding the high-performance cores", Intel Technology Journal, 2010.

[27] Intel Corporation,"Data Plane Development Kit (DPDK)", https://www.dpdk.org

[28] A. Herdrich, E. Verplanke, P. Autee, R. Illikkal, C. Gianos, R. Singhal, R. Iyer, "Cache QoS: From concept to reality in the Intel® Xeon® processor E5-2600 v3 product family", HPCA, 2016.

[29] "Intel 64 and IA-32 Architectures Software Developer's Manual"

[30] Intel Corporation, "Intel® data direct I/O (DDIO) Technology", https://www.intel.com/content/www/us/en/io/data-direct-i-o-technology.html

[31] Intel Corporation, "Intel® Hyper-Threading Technology", https://www.intel.com/content/www/us/en/architecture-and-technology/hyper-threading/hyper-threading-technology.html

[32] Intel Corporation, "Intel® Turbo Boost Technology 2.0", https://www.intel.com/content/www/us/en/architecture-and-technology/turbo-boost/turbo-boost-technology.html

[33] David Lo, Liqun Cheng, Rama Govindaraju, Parthasarathy Ranganathan, and Christos Kozyrakis, "Heracles: Improving Resource Efficiency at Scale", In Proceedings of the 42nd Annual International Symposium on Computer Architecture, 2015.

[34] Bin Li, Yipeng Wang, Ren Wang, Charlie Tai,Ravi Iyer, Zhu Zhou, Andrew Herdrich, Tong Zhang, Ameer Haj-Ali, Ion Stoica, Krste Asanović, "RLDRM: Closed Loop Dynamic Cache Allocation with Deep Reinforcement Learning for Network Function Virtualization", IEEE Conference on Network Softwarization, 2020.

[35] K. Deb, A. Pratap, S. Agarwal and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II", IEEE Transactions on Evolutionary Computation, 2002.