The Journal of Supercomputing (2023) 79:814–831
https://doi.org/10.1007/s11227-022-04692-0

Extending the VEF traces framework to model data center network workloads

Francisco J. Andújar[1] · Miguel Sánchez de la Rosa[2] · Jesus Escudero-Sahuquillo[2] · José L. Sánchez[2]

Accepted: 8 March 2022 / Published online: 20 July 2022

Abstract

Data centers are a fundamental infrastructure in the Big-Data era, where applications and services demand a high amount of data and minimum response times. The interconnection network is an essential subsystem in the data center, as it must guarantee high communication bandwidth and low latency to the communication operations of applications, otherwise becoming the system bottleneck. Simulation is widely used to model the network functionality and to evaluate its performance under specific workloads. Apart from the network modeling, it is essential to characterize the end-nodes communication pattern, which will help identify bottlenecks and flaws in the network architecture. In previous works, we proposed the VEF traces framework: a set of tools to capture communication traffic of MPI-based applications and generate traffic traces used to feed network simulator tools. In this paper, we extend the VEF traces framework with new communication workloads such as deep-learning training applications and online data-intensive workloads.

Keywords Data center networks · Modeling and simulation · Data center workloads · Performance evaluation

✉ Francisco J. Andújar
fandujarm@infor.uva.es

Miguel Sánchez de la Rosa
miguel.sanchez@uclm.es

Jesus Escudero-Sahuquillo
jesus.escudero@uclm.es

José L. Sánchez
jose.sgarcia@uclm.es

[1] Computing System Department, Universidad de Valladolid, Campus Miguel Delibes, Paseo de Belén, 47011 Valladolid, Spain

[2] Computing System Department, Universidad de Castilla-La Mancha, Avda. de España s/n, 02071 Albacete, Spain

1 Introduction

Societal challenges in fields such as Climate, Medicine, Energy, Astrophysics, Media content consumption, Social Networks, Security, and Smart Cities are driving the technological evolution of Digital Technology. Data center users worldwide demand instantaneous access to vast amounts of data, generating trillions of requests that must be analyzed and processed, returning valuable information within an acceptable time frame. The International Data Corporation (IDC) forecasts that the global data-sphere will grow from 33 ZB in 2018 to 163 ZB by 2025. In this context, information technologies (IT), such as 5G, Internet of Things (IoT), business and artificial intelligence (BI and AI), or Cloud Computing, need data center infrastructures offering data storage capacity and computing power several times larger than these currently available. In the last few years, the world's largest Cloud platforms (e.g., Microsoft, Google, Facebook, or Amazon) are upgrading their data centers to anticipate the unprecedented growth of required space and power, requiring important innovations in hardware and software to cope with increasingly complex applications and real-time user demands.

The interconnection network (or data center network, DCN) is an essential element in the data center architecture, which communicates thousands of computing and storage nodes. DCNs are composed of Network Interface Cards (NICs), switches, bridges and cables. NICs plug into the server nodes through the PCIe subsystem, which communicates CPU, NIC and memory controller. Externally, NICs communicate end nodes with switches. Moreover, switches and bridges interconnect the data center servers utilizing cables, creating specific network topologies, such as the non-blocking CLOS topology, commonly used in many data centers. In general, the DCN architecture needs to be fault-tolerant, cost-effective, scalable and offers high inter-node communication bandwidth and low response time (latency) in data transmission. Moreover, the design of the DCN must consider the communication requirements of the data center applications, otherwise becoming the system bottleneck.

Nowadays, there are critical use cases [1] that subject the DCN to high communication rates among the end-nodes. Among these critical use cases, we can distinguish four types: online data-intensive applications (e.g., large search websites), deep learning (e.g., instant image analysis), distributed storage and Cloud Computing services. These use cases (or workloads) generate in the DCN a vast amount of user requests that need to be processed immediately, since these users expect to receive a response in an imperceptible fraction of time. Note that, this is a common goal shared with traditional applications in high-performance computing (HPC) clusters. In contrast to HPC applications, which are optimized to minimize data movement in the network and whose results are expected in a predictable time. Unfortunately, the critical use cases for DCNs are more complicated to optimize, since the communication operations on the DCN are unpredictable and variable over time. Hence, the DCNs workloads depend on the number of requests performed by the users interacting with services worldwide, and it may be difficult to predict the amount of traffic generated by these applications

and their communication patterns in the DCNs. As the critical use cases impact negatively on the DCN performance, they need to be thoroughly considered when designing the DCN architecture.

Simulation is a popular method to evaluate the behavior and performance of IT systems, such as HPC clusters or data centers, and has been extensively used to model and evaluate new designs for high-performance interconnection networks, such as those used in data centers. There are many simulation tools proposed to model interconnection networks, such as OMNet++-based [2] simulators (e.g., INET model, ib_model or SAURON), NS-3 [3], SST [4] or CODES [5]. In general, these simulators are able to model the network components and their architecture, with different levels of detail or abstraction. Apart from the interconnection network architecture, another important aspect that network simulators need to offer is the ability to reproduce realistic workloads in the interconnection network. In many cases, the simulation frameworks only model ad hoc or synthetic traffic patterns, which do not reproduce the behavior of the communication operations observed in the DCNs of real data centers.

In previous works, we presented the *VEF traces framework* [6, 7], which comprises several tools that allow the simulation tool developers to model network workloads based on MPI applications. One of these tools is the *VEF-Prospector*, which captures the application MPI calls, and gathers them in trace files using a special format (i.e., VEF). These traces are used to reproduce the application behavior in network simulation tools, as long as this simulator uses another VEF framework tool: the *VEF-TraceLib* library. This library is independent of the network simulator and can be integrated with any simulator tool. *VEF-TraceLib* is in charge of reading the VEF trace and generates the corresponding messages that can be inserted in the simulated NICs, so that the simulator injects them into the network. The library is also responsible of collecting the packets received by the end-node NICs and finishes the communication operations. Moreover, *VEF-TraceLib* also permits running several applications (i.e., traces) simultaneously in the same simulation, configuring the task mapping of jobs to nodes or choosing a different implementation of an MPI collective communication algorithm. Most importantly, the *VEF traces framework* allows network simulators to reproduce the application behavior in a completely agnostic way, meaning that the node architecture and timestamps of the system gathering a VEF trace are not stored in the trace, but only the information of communication operations (e.g., source, destination, prior message dependency, etc.).

However, in data center systems, there are different types of network workloads other than those generated by MPI-based applications. As mentioned above, there are mainly four critical use cases that threaten the performance of DCNs, such as online data-intensive (OLDI) services used in applications such as web search, social networks, streaming, or the traffic generated on the computing-intensive training phases of deep learning services.

Therefore, modeling the workloads generated by these critical use cases in the DCN is also a cornerstone for network simulator developers, and it is also a challenge for the *VEF traces framework*. In this paper, we extend the *VEF traces framework* to model the DCN traffic, generated by some of the critical use cases, such as OLDI services and deep-learning training applications. First, we have included

a new trace format, compatible with the previous one, to easily model the communication operations of different online data-intensive workloads. These workloads are modeled based on data published by different companies using large data center facilities, such as Facebook and Google. Next, we have extended *VEF-TraceLib* to run the extended traces and reproduce their behavior in a simulation tool. Note that, we have not made any modification in the library interface with simulators, so that we preserve backward compatibility with the previous version of the library. Therefore, *VEF-TraceLib* users can simply update the library and simulate the new traces modeling the DCN workloads. Note that, all the features of *VEF-TraceLib* are also available for the new traces, thereby the extended VEF traces can be used in combination with the original MPI-based VEF traces.

2 Background

Data centers are the fundamental infrastructure nowadays, where real-time services and applications, such as augmented reality, voice recognition, and contextual searching, demand immediate responses to meet the user demands. In the data centers, the interconnection network must guarantee unprecedented levels of performance, scale, and reliability to meet these real-time demands [8]. As we have mentioned above, we can classify the workloads stressing the data center network (DCN) performance into four critical use cases, such as large-scale On-Line Data Intensive (OLDI) services (e.g., automated recommendation systems for online shopping, social media, and web searches), high-performance deep learning services (e.g., for image processing), distributed storage (e.g., high-speed distributed pools of Non-Volatile Memory Express (NVMe) storage), and Cloud Computing applications.

From a network designer perspective, the characterization of DCN workloads is crucial to identifying network bottlenecks generated by communication operations. It consists of analyzing a set of properties of communication operations, such as size, source/destination distribution, and frequency of communications over time. The data obtained from traffic characterization aid the network designers in developing traffic models used to feed network simulator tools. These traffic patterns must generate the communication operations happening in a real DCN inside the simulator, so that the simulation infrastructure can reproduce a realistic environment, which will be very helpful for the network design process. In this regard, it is very typical to analyze the behavior of the new network designs using the mentioned simulation frameworks.

On the one hand, there are proposals for modeling DCN workloads which use information publicly released by some data center owners. For instance, recent proposals model the network traffic observed in some of Facebook's data centers [9]. They assume a workload is a set of traffic flows from different applications and services generated within a given time fraction. A traffic flow is a bunch of information transferred from a given source host to another throughout the network. As each application or service may generate multiple traffic flows, it is common to group these flows in separate traffic classes, and each traffic class may generate flows of different sizes at different time instants. Besides, source and destination end-nodes

of traffic flows may be different, depending on the application or service (i.e., the traffic class). Therefore, each traffic class has a different communication pattern, which may correlate with a specific distribution of destinations. This approach is efficacious, as it allows to configure a set of parameters to vary the DCN workloads.

Other proposed traffic models [10], which follow a similar idea to that described above, are based on a collection of data center applications at Google and Facebook data centers. The workloads characterization focuses on the measurements of applications based on request-response protocols, which generate tiny messages of a few hundred bytes or less. For instance, this happens in typical remote procedure call (RPC) use cases, where the request or the response is a short traffic flow.

Apart from the OLDI workloads, other DCN workloads are worth mentioning, such as those based on Deep Learning services. Disciplines such as computer vision, speech recognition, or social network filtering, among others, apply Deep Learning algorithms. A current trend for processing large sets of data is the use of Deep Neural Networks (DNNs) [11] to learn, via a process called "training" to handle each new problem, instead of coding a different custom program to solve that problem in a specific domain.

Training a DNN is a highly parallel procedure [1] requiring specialized computing devices to supply high throughput and low latency. However, the communication overhead in the training process can offset the gains of using these devices. In parallel training, massive data sets are split into chunks and shared among the worker nodes in the data center. These servers process these chunks and send the temporary results to the aggregator nodes, and this process repeats in each DNN layer until it computes the correct output and returns the exact result. The communication overhead introduced by this algorithm is the bottleneck of the training process, so that characterization and modeling of deep learning training traffic are required to analyze its behavior in the DCN.

The traffic patterns mentioned above can be modeled in simulation tools in several ways, such as using instrumentation tools to capture the communication calls of the applications running in the data center, or using mathematical models to reproduce the distribution and parameters of the generated traffic flows. The first approach is followed by several frameworks such as Vampir [12], Score-P [13], Scalasca [14], DUMPI [15] or Extrae [16]. However, these frameworks leverage in general the hardware performance counters (e.g., provided by the PAPI library) and several distributed/shared memory profiling tools to generate enormous traffic traces, which are difficult to handle by network simulation tools. Note that, apart from the traces size, the network simulation tools only need that the traces store the communication operations calls in order to generate network traffic. The second approach to model network traffic patterns is based on using mathematical expressions to generate traffic patterns. For instance, there are numerous studies which use these mathematical expressions [10] to model realistic workloads. Actually, there are numerous micro-benchmarks, such as OSU [17] or Intel MPI [18], which measure the performance of single communication operations, such as the message passing interface (MPI) calls.

The VEF traces framework [6, 7] was conceived as an easy way to capture the network traffic generated by MPI calls and record this information in traffic traces smaller than those generated by other traces frameworks (e.g., Extrae or Scalasca),

since VEF traces only store the information related to communication operations, while preserving the dependencies of these operations. However, the VEF traces framework does not support the approach of using mathematical expressions to reproduce the communication operations generated data center applications. In this paper, we extend the VEF traces framework to model several data center workloads, such as those described above. In addition, a brief description of the framework has been included in Sect. 1 of the Supplementary Information.

3 Extended VEF-traces model description

In this section, we describe the extensions included in the *VEF traces framework* to model DCN workloads. We have modeled three new traffic patterns: generic and configurable workloads based on publicly available information, workloads modeling traffic flows grouped into different traffic classes, and workloads based on the Horovod deep learning training framework [19, 20]. Note that, the Horovod framework uses MPI so that it would be possible to run Horovod-based applications and generate VEF traces. However, these traces would have a fixed number of MPI tasks, not being possible to scale the number of MPI tasks. For this reason, we have analyzed the communication operations generated by the Horovod framework in order to reproduce its behavior, regardless the network size, based on a set of parameters.

First, we have modified the VEF trace format to include the proposed extensions. We have called this new traces as *extended VEF traces*. Specifically, the new trace format has at least two records: the *trace header* and the *extended VEF header*. The following records depend on the traffic model selected. Code listing 1 shows the extended VEF trace format.

Listing 1: Extended VEF trace format.

```
VEF3 #Tasks #Msgs #COMM gGComm lGComm oSFlag clock
EXT  type
```

The first line in the trace is the header, which has not varied with respect to the previous format version. The fields of the trace header are the following:

- `VEF3` is the start of the trace header.
- `#Task` is the number of tasks that the application runs in parallel.
- `#Msgs` is used to limit the amount of generated messages, depending on the type of extended trace.
- The `#COMM`, `gGComm`, `lGCom` and `oSFlag` are for communicators and collective communication definition in MPI-based VEF traces. These are not used in the models we propose in this paper.
- `clock` is the clock resolution measured in picoseconds. If this field is omitted, the clock resolution is 1000 ps.

The second line in the trace file shown in code listing 1 is the extended VEF header, which indicates the traffic model. The fields of the extended header are the following: `EXT` indicates that the trace is an extended trace and `type` indicates the traffic model. Currently, `type` can take these possible values:

- `DCNW`: DCN workloads based on the traffic model using information publicly available (see Sect. 3.1).
- `CDCNW`: The traffic model is the same as the previous one, but in this case, the trace format allows the user to specify the message size distribution, instead of using a predefined message size distribution of `DCNW` model.
- `FLOWS`: DCN workloads using traffic flows grouped into traffic classes (see Sect. 3.2).
- `HOROVOD`: Traffic patterns from the Horovod deep learning training framework (see Sect. 3.3).

In the following subsections, we describe the new traffic models included in the *VEF traces framework*.

3.1 DCNW traffic model

This traffic pattern models different DCN workloads based on publicly available information of real workloads measured. We have modeled a selection of traffic patterns based on workloads from real application-level logs. Most of them were measured in real data centers from applications at Google and Facebook. These workloads have also been used to evaluate other transport protocols for data center networks, such as HOMA [10]. Code listing 2 shows the simple third record to be added for configuring the DCNW traffic model.

Listing 2: VEF trace DCNW record format.

```
model destination_dist LOAD l
```

In this trace record, the `model`, `destination_dist` (i.e., destination distribution) and `LOAD l` are used to adjust the generation rate of the messages. Both `destination_dist` and `LOAD l` are optional parameters, while `model` is a mandatory parameter. These parameters can take the following values:

- `model` specifies the message size distribution based on a set of different applications data publicly available [10]. Currently, there are available the following traffic models: Web search workload based on DCTCP work [21]; a collection of memcached servers at Facebook for Key-Value stores [22]; Facebook Cache Follower, Facebook Web Server and Hadoop nodes at Facebook cluster [23]; aggregated workload from applications running in a Google data center [10]; and search application at Google cluster. [10].

- `destination_dist` indicates the distribution of the destination task of the messages. Currently, there are available a uniform distribution, a Poisson distribution and a Gaussian distribution.
- `LOAD l` can be used to adjust the generation rate of the messages. `l` is a floating number that indicates the Gbps generated per each trace task. If `LOAD` is not set, the generation rate is set to 1 Gbps per task.

Code listing 3 shows an example of a VEF trace configuring the DCNW trace model. This trace configures a 128-task application based on the aggregated workload of Google data center, generating 2,000,000 messages, the Gaussian destination distribution and generating 0.5 Gbps per task. The clock resolution is set to 1 nanosecond (i.e., 1000 picoseconds).

Listing 3: VEF trace example configuring the *VEF-TraceLib*.

```
VEF3 128 2000000 0 0 0 0 1000
EXT DCNW
GOOGLE_ALL GAUSS LOAD 0.5
```

Finally, we have also implemented a variation in the DCNW traffic model, called *Custom DCNW traffic*. This trace format allows to specify a message distribution file, which can be defined by the user.

Due to the lack of space, a detailed descriptions of the destination distributions, the message size distribution of each `model`, and the arrival distribution, are included in Sect. 2 of the Supplementary Information. Moreover, in Sect. 3 of the Supplementary Information, we also included a the complete description of how to configure the traces for using the *DCNW* and *Custom DCNW* models.

3.2 FLOWS traffic model

This model, called FLOWS, allows us to model different traffic classes, each of them grouping a variety of traffic flows according to the model described in [9]. To select this model, we need to add the `FLOWS` value to the second trace record (i.e., `EXT FLOWS`). In this model, the trace header param `#Msgs` indicates the number of traffic flows. After that, the third line of the trace includes a record to specify the number of traffic classes, the generation time and the default destination distribution. Code listing 4 shows the format of this register. The meaning of each of the fields required to configure this traffic model is the following:

Listing 4: VEF trace record for the configuration of the FLOWS model.

```
CLASSES c TIME t destination_dist
```

- `CLASSES c` indicates the number of traffic classes. The parameter `c` must be an integer greater than 0. This parameter is mandatory.
- `TIME t` indicates the maximum generation time. All the flows will be generated between cycle 0 and cycle `t`. The time between two consecutive flows is modeled using a exponential distribution with λ=`t`/`#Msgs`. The parameter `t` must be an integer greater than 0. This parameter is mandatory.
- `destination_dist` indicates the default destination task distribution. The option is the same as in the DCNW traffic model. This parameter is optional. If it is not set, the default distribution is set to uniform distribution.

After this common configuration, a new line must be added to configure each traffic class. Code listing 5 shows the class configuration format. All the parameters are mandatory except `destination_dist`. The traffic class configuration must start with the class identifier (`id`), but the remaining parameters can be specified in any order. The meaning of these fields is the following:

Listing 5: VEF Trace record for the configuration of each FLOWS traffic class.

```
id MAX max MIN min PERC p destination_dist
```

- `id` is the traffic class identifier. It must be an integer in the range [0, c-1]. The line must start with the traffic class identifier.
- `MAX max` is the maximum flow size measured in Kbytes. The flow sizes are generated using an uniform distribution between `min` and `max`.
- `MIN min` is the minimum flow size measured in Kbytes.
- `PERC p` is the percentage of flows generated by this traffic class. The parameter `p` must be a float between 0 and 100.
- `destination_dist`: destination distribution for this traffic class.

Code listing 6 shows a VEF trace example of the FLOWS model. This VEF trace models a 128-task application, with 2M flows (i.e., `#Msgs` parameter in the VEF header), two traffic classes, a generation time of 200M cycles and the uniform distribution set as default distribution. The first traffic flow generates 80% of the flows and its messages have a size between 1 and 10 KB. The second traffic flow generates 20% of the flows, its messages have a size between 1 and 10 MB, and it uses the gaussian distribution instead of the default distribution.

Listing 6: VEF trace example configuring a 2-class FLOWS model.

```
VEF3 128 2000000 0 0 0 0 1000
EXT FLOWS
CLASSES 2 TIME 200000000 UNIFORM
0 MAX 10 MIN 1 PERC 80.0
1 MAX 10000 MIN 1000 PERC 20 GAUSS
```

3.3 HOROVOD traffic model

Currently, it is possible to leverage large and Deep Neural Networks (DNNs) to solve complex problems in multiple disciplines. These artificial neural networks can handle large amounts of data and perform a high number of operations on top of them. Thus, DNN-based applications require a vast amount of computing power and a large storage capacity for the training, validation, and inference processes. Hence, as data centers meet these computational and storage requirements, they have become highly demanded to run Machine- and Deep-Learning applications. Indeed, the reliance on data centers for DNN-based applications has been growing significantly in recent years.

The Horovod library [19] for distributed training has grown rapidly in popularity, since it requires minimal changes to the training program to make it distributed. Horovod is a Python package that implements versions of different Deep Learning frameworks for execution with MPI. From the publicly available Horovod source code [20], all the processes run an "endless" loop, called "negotiation phase," and the main actions in each iteration are the following:

1. The workers send `MPI_Request` for tensors to master.
2. The master gets message lengths from every worker (`MPI_Gather`), computes offsets and collects the tensor indexes from every worker (`MPI_Gatherv`).
3. The master creates a vector of tensors ready to be reduced and performs an optimization (Tensor Fusion), assembling a single response for tensors with the same response type.
4. The indexes of the tensors reduced are broadcast to all workers by using `MPI_Bcast` operation.
5. Finally, the all-reduce operation (`MPI_Allreduce`) is executed on all tensors ready to be reduced.

In order to implement this behavior, we have created in the VEF traces framework a new model called HOROVOD. To choose this model, we have configured the VEF trace file so that the different parameters of a Horovod-based application can be configured using a few number of records in the VEF trace header. To select the HOROVOD model in the VEF trace file, we need to add the `HOROVOD` value to the second record of the trace (i.e., `EXT HOROVOD`). After that, in the third line of the trace, a record is included to specify the number of training batches and the number of the DNN layers. After this common configuration, a fourth record must be included in the VEF trace to provide a default configuration for all the layers. Code listing 7 shows the format to establish the initial configuration of the DNN traffic.

Listing 7: Initial configuration for the HOROVOD model.

```
BATCHES b LAYERS l
DEFAULT COMPUTE_MS c_ms NEGOTIATION_LOOP_MS nl_ms
    GATV_BYTES gb GATV_TOTAL_BYTES gtb GATV_PROB gprob
    BCAST_BYTES bb ARED_KBYTES akb
```

After the word DEFAULT, all the fields in the default layer must be included, but the fields can be specified in any order. We assume that all the parameters must be integers greater than zero, except for those parameters that are specified otherwise. The meaning of each of these fields is the following:

- COMPUTE_MS c_ms is the average time, measured in milliseconds, that a worker is only performing computation at the beginning of a layer, before starting the negotiation phase. c_ms must be a float number greater than 0.
- NEGOTIATION_LOOP_MS nl_ms is the average time, measured in milliseconds, that a worker spends between each loop iteration of the negotiation phase. The parameter nl_ms must be a float number greater than 0.
- GATV_BYTES gb is the average size of the MPI_Gatherv message, measured in bytes, that the workers send to the master process for communicating the tensors indexes that must be reduced.
- GATV_TOTAL_BYTES is the size, measured in bytes, that the master must receive at least in several MPI_Gatherv messages to consider that a worker has sent all its tensor indexes to be reduced in the master process.
- GATV_PROB gprob is the probability that a worker sends tensor indexes to the master in any iteration. gprob must be a float number between 0 and 1.
- BCAST_BYTES bb is the size of the MPI_Bcast operation that the master sends to all the workers when it has received from them all the tensor indexes.
- ARED_KBYTES akb is the size of the MPI_Allreduce operation that the master performs in the last step to update all the tensor indexes in all workers.

Apart from the configuration of the default layer, each layer can be independently configured in the VEF trace. For this purpose, the record format is the same as for the default layer, changing the word DEFAULT by the layer identifier. This identifier must be an integer in the range $[0, l-1]$.

Listing 8: VEF trace example configuring HOROVOD model.

```
VEF3 128 0 0 0 0 0 1000
EXT HOROVOD
BATCHES 2 LAYERS 8
DEFAULT COMPUTE_MS 20 NEGOTIATION_LOOP_MS 1.5
    GATV_BYTES 60 GATV_TOTAL_BYTES 600 GATV_PROB 0.01
    BCAST_BYTES 1000 ARED_KBYTES 50000
3 BCAST_BYTES 2000 ARED_KBYTES 150000
```

Code listing 8 shows a VEF trace example of the HOROVOD model. In this example, we can observe that this model works in the following way:

1. All the workers spend 20 ms in average performing only computation.
2. After that, all the workers start the negotiation phase. First, the workers send the master an MPI_Gather of 4 bytes indicating the vector size of tensor indexes. All the workers have a probability of 0.01 of sending tensor indexes. If a worker

has tensor indexes to be sent to the master, it performs an `MPI_Gatherv` of average size of 60 bytes. In other case, it sends a empty message to the master via `MPI_Gatherv` of 25 bytes.

3 If the master has received at least 600 bytes of tensor indexes from each worker, the update step is performed:

   3.1 The tensor update is not carried out. The master performs two `MPI_Bcast` of 4 and 25 bytes to communicate that there are no tensors to update. The execution continues in step 2 after 1.5 milliseconds in average.

   3.2 The tensor update is carried out. The master performs two `MPI_Bcast` of 4 and 1000 bytes (or 2000 bytes in layer 3) to communicate the tensor indexes to update. The execution continues in step 4.

4 An `MPI_Allreduce` of 50 MB is performed to update the tensors. Note that, the `MPI_Allreduce` performed in layer 3 has a size of 150 MB.

5 The number of layers completed in the current training batch is updated, and also the number of batches if all the layers are completed. When all the batches are completed, the simulation finishes. In other case, the simulation of the current layer of the current batch continues in step 1.

4 Extended traces use cases

The aim of this section is to show how extended traces work by using series of use cases (or tests). These tests are obtained with a network simulator that uses *VEF-TraceLib* and extended VEF traces. To perform the simulations, we have used an open-source interconnection network simulator called *Hiperion* (HIgh PERformance InterconnectiOn Network) [24], which integrates*VEF-TraceLib*. Since the goal of this study is not to evaluate different configurations of the interconnection network, we have chosen a simple switch architecture to evaluate how the different network loads generated affect the network performance.

For this study, we have considered a single Input Queued (IQ) switch [25] with Virtual Cut-Through (VCT) switching [26], a credit-based flow control at link level, and a two-stage allocation algorithm composed of a round-robin virtual channel allocator and a round-robin switch allocator. The input-port buffer size is 1024 flits, and the flit size is 16 bytes, i.e., the buffer size is 16 KB. Each buffer has one virtual channel, and the packet size is 8 flits. Each switch port is connected to a multi-core node by means of a network interface (NIC). The number of switch ports and the number of node cores depend on the extended trace considered and will be specified in the following sections.

Moreover, to properly calculate the average results of the performance metrics, each simulation test has been run 30 times. We have considered the following performance metrics:

- *Execution time* (cycles) shows the time instant when all the traffic generated by the extended VEF trace has been received in all the computing nodes.

- *Average flit end-to-end latency* and *average flit network latency* (cycles) show the average time spent in the network by the all the flits. The end-to-end latency also includes the queuing delay in the source node.
- *Average throughput* (flits/cycles/NIC). This is the amount of information that the network in total delivers to all the connected nodes. The maximum throughput reachable is 1, i.e., each NIC receives one flit each cycle.

DCNW model use cases

For testing the behavior of the DCNW model, we have used the trace shown in Listing 3 as a reference. Since this trace has 128 tasks, the switch has 8 ports and the nodes have 16 cores, being a total amount of 128 cores. We have compared different traces to show how the changes in trace configuration affect the achieved performance. Specifically, we have tested four VEF traces:

- *ref*: Trace shown in Listing 3 setting the load to 1 Gbps.
- *task256*: *ref* trace with 256 tasks, a 16-port switch and 16-core nodes.
- *gauss*: Same trace as *ref* but using the *Gauss* distribution.
- *load*: Same trace as *ref* but setting the load to 0.5 Gbps.

Figure 1 shows the results obtained. First, when the number of tasks of *ref* configuration is doubled (*task256* configuration), we can observe that the execution time is reduced. Since the number of tasks is higher, the required amount of time to generate the same number of messages is lower. However, the higher the amount of messages simultaneously traveling throughout the network, the higher the degree of contention. Hence, this slightly worsens the latencies and throughput. When the Uniform destination distribution is changed to a Gaussian distribution (see the *gauss* bar), the execution time and latencies increase, while the throughput decreases. Since the destination nodes where all the application tasks send traffic are concentrated around the task 64, the network congestion increases and the network performance is reduced. Finally, when the message generation load rate is halved (see the *load* bar), a large increase in execution time is observed because the system takes twice as long as for the *ref* configuration to generate the same number of messages. Moreover, since the network load is reduced, there is a significant reduction in end-to-end and network latency. Therefore, according to the experimental results, the observed network performance is the expected one for the DCNW model.

FLOWS use cases

In this subsection, we test the behavior of the FLOWS model using as a reference the trace shown in Listing 6 . The switch has 8 ports and the nodes have 16 cores. Specifically, we have tested four VEF-trace configurations:

- *ref*: Trace shown in Listing 3, but both classes use the Uniform distribution.
- *task256*: *ref* trace with 256 tasks, a 16-port switch and 16-core nodes.
- *gauss*: *ref* trace but both classes use the Gauss distribution.
- *perc*: *ref* trace but each class generates the 50% of flows.

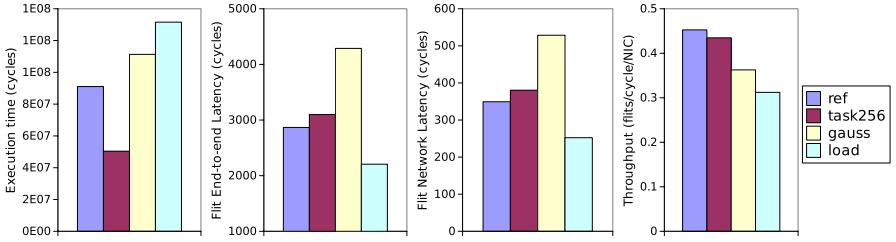

Fig. 1 Evaluation results for DCNW model

Figure 2 shows the results obtained using the FLOWS model. The conclusions are very similar than obtained in Sect. 4 . For configuration *task256*, increasing the number of tasks reduces the execution time and slightly worsens the performance, while for the configuration *gauss*, concentrating the traffic in fewer tasks increases the contention and reduces the network performance. For the configuration *perc*, a large increase in execution time is observed, due to 30% of flows change from class 0 (message size $\in [1-10]$ KB) to class 1 (message size $\in [1-10]$ MB). Moreover, note that, the execution time for the different configurations is different than that observed in Fig. 1, since the amount of generated messages in total differs from that of the DCNW model. Interestingly, despite the increased load, there is hardly any impact on latencies and throughput. It is worth noting that we used a simple switch architecture model (e.g., virtual channels are not used to attempt to reduce the contention). Therefore, the switch is already working on saturation for the configuration *ref*, and then, increasing the message size simply increases the execution time but does not affect to the latency and throughput.

HOROVOD use cases

Finally, we test the behavior of the HOROVOD model. In first place, we obtained a VEF trace running a real DNN training application using Horovod and the training model AlexNet [27], running the application with 98 tasks. The trace has been collected using the GALGO High Performance Computing Cluster [28]. After that, we have tried to replicate its behavior using the HOROVOD model. This task is especially complicated since the optimizations introduced by Horovod software, such as the Tensor Fusion technology [19], make it difficult to find a repeating pattern.

Despite these drawbacks, we have managed to create a trace with a similar behavior than the original AlexNet trace, which is shown in Sect. 4 of the Supplementary

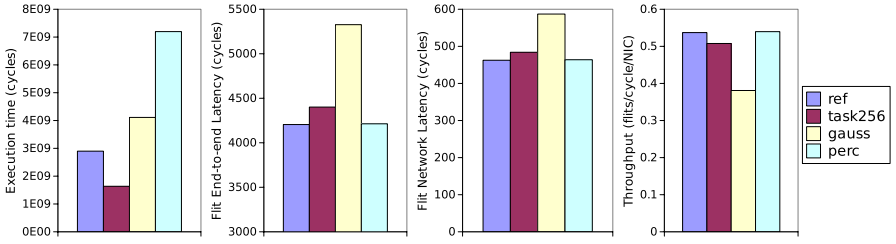

Fig. 2 Evaluation results for FLOWS model

Information. The amount of traffic generated has a little reduction of 5% (from 249 to 239 GB), but the runtime is very similar (the difference is less than 1%). Since our goal is not to exactly model the same trace, this trace is accurately enough to test the HOROVOD model. The switch configuration has been modified accordingly to have 98 cores. Now, the switch has 7 ports and the nodes have 14 cores. We have compared these five traces:

- *alexnet*: VEF trace obtained from a execution of *AlexNet* model in GALGO.
- *ref*: Horovod trace shown in Sect. 4 of Supplementary Information.
- *allreduce*: *ref* trace, but the size of AllReduce in all the layers is duplicated.
- *compute*: *ref* trace, but dividing by 10 all the compute times.
- *loop*: *ref* trace, but dividing the negotiation loop time by 2, being the final time equal to 0.7 ms.

Figure 3 shows the results obtained using the HOROVOD model. When comparing *ref* and *alexnet* traces, we can see that execution time and the throughput are practically the same in both traces, but the end-to-end and the network latency are slightly different. Since the amount of traffic generated is lowered, the situations where contention appears is also reduced, decreasing the network latencies. In the other three cases, the results are the expected ones. Note that, by doubling the size of `All_Reduce` communications (see the *allreduce* configuration), the execution time and latencies increase, and also the throughput. When the compute time is reduced (*compute* configuration), the execution time is slightly lowered, although it does not have the same impact as reducing the negotiation loop time. Since now the compute time is closer to the negotiation loop time, in some NICs, there may be communications in the DNN in which the computation of the previous layer overlaps with the first communication of the next layer, thus slightly increasing the network and end-to-end latency. Furthermore, decreasing the time of the negotiation loop (*loop* configuration) has a great impact on the execution time, and therefore, on the network throughput. The execution time reduction is almost 50%, doubling the throughput. Note that, the *loop* and *allreduce* throughput is practically the same. For the *loop* configuration, the execution time is halved, while for the *allreduce* configuration, the amount of traffic in the network is doubled. Also, as the message generation is the same as in *ref* (the loop time only affects the time between negotiation loops, not communications within a negotiation loop), there is no impact on

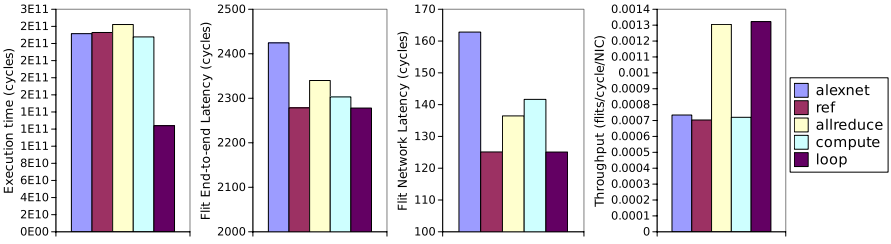

Fig. 3 Evaluation results for HOROVOD model

latencies. As we can see, our HOROVOD model allows us to replicate the behavior of DNN models, and it works as expected.

5 Conclusions

In previous works, we have presented the open-source *VEF traces framework* to model the workloads generated by HPC applications in high-performance interconnection network simulators. This framework provides two main tools: *VEF-Prospector* and *VEF-TraceLib*. The first one is used to capture the communication operations generated in the network by MPI applications and generate trace files storing these operations. The second is used to feed an interconnection network simulator with the workloads stored in the traces. The main problem with the *VEF traces framework* is that it was only focused on MPI-based applications, although there are other interesting workloads to feed the network simulation tools, such as those generated by data-center applications (e.g., deep-learning training or online data-intensive).

In this work, we have extended the VEF traces framework to model Data-Center Network (DCN) workloads and reproduce them in DCN simulators. These workloads have been modeled according to the information publicly available, such as that of workloads generated in Facebook and Google data centers. Moreover, we have also extended the VEF traces framework with deep-learning training applications workloads. The *VEF-TraceLib* has been also updated to run the *Extended VEF traces*, although no modifications on the library interface are required. This preserves the backward compatibility with the previous version of the library and allows the *VEF-TraceLib* users to use the new workloads simply by updating the library from the repository. In addition, users can still use all the *VEF-TraceLib* features with the new traces. Finally, in order to validate the proposed models, we have simulated several configurations for the new traffic models. The obtained results show that the modeled traffic patters reproduce realistic DCN workloads in the simulation tools using the new traffic models included in the *VEF traces framework*.

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1007/s11227-022-04692-0.

Acknowledgements This work is part of the R &D Project Grant PID2019-109001RA-I00, funded by MCIN/AEI/10.13039/501100011033.

Funding Open Access funding provided thanks to the CRUE-CSIC agreement with Springer Nature.

References

1. Congdon P (2018) IEEE 802 Nendica report: the lossless network for data centers. IEEE-SA industry connections white paper. https://mentor.ieee.org/802.1/dcn/18/1-18-0042-00-ICne.pdf
2. Varga A OMNeT++ User Manual. OpenSim Ltd. Accessed 15 Feb 2022. http://omnetpp.org/doc/omnetpp/manual/usman.html
3. Riley GF, Henderson TR (2010) The ns-3 network simulator. In: Wehrle K, Güneş M, Gross J (eds) Modeling and tools for network simulation. Springer, Berlin, pp 15–34
4. The Structural Simulation Toolkit, Sandia National Laboratories, USA, 2015. Accessed 15 Feb 2022. http://sst-simulator.org
5. Mubarak M, Carothers CD, Ross RB, Carns P (2017) Enabling parallel simulation of large-scale HPC network systems. IEEE Trans Parallel Distrib Syst 28(1):87–100
6. Andújar FJ, et al. (2015) VEF traces: a framework for modelling MPI traffic in interconnection network simulators. In: The 1st IEEE international workshop on high-performance interconnection networks in the exascale and big-data era, Chicago, IL, USA, pp 841–848
7. Andujar FJ et al (2016) An open-source family of tools to reproduce MPI-based workloads in interconnection network simulators. J Supercomput 72(12):4601–4628
8. Congdon P, Guo L (2021) IEEE 802 nendica report: intelligent lossless data center networks. IEEE-SA industry connections white paper. https://ieeexplore.ieee.org/abstract/document/9457238
9. Gonzalez-Naharro L, et al (2019) Modeling traffic workloads in data-center network simulation tools. In: 2019 International Conference on High Performance Computing Simulation (HPCS), pp 1036–1042
10. Montazeri B, et al (2018) Homa: a receiver-driven low-latency transport protocol using network priorities. SIGCOMM '18, pp 221–235. Association for Computing Machinery, New York, NY, USA
11. Sze V, Chen Y, Yang T, Emer JS (2017) Efficient processing of deep neural networks: a tutorial and survey. Proc IEEE 105(12):2295–2329
12. Knüpfer A, Brunst H, Doleschal J, Jurenz M, Lieber M, Mickler H, Müller MS, Nagel WE (2008) The vampir performance analysis tool-set. In: Resch M, Keller R, Himmler V, Krammer B, Schulz A (eds) Tools for high performance computing. Springer, Berlin, pp 139–155
13. Knupfer A, Rossel C, an Mey D, Biersdorff S, Diethelm K, Eschweiler D, Geimer M, Gerndt M, Lorenz D, Malony A, Nagel WE (2012) Score-P: a joint performance measurement run-time infrastructure for periscope, Scalasca, TAU, and Vampir
14. Geimer M, Wolf F, Wylie BJN, Ábrahám E, Becker D, Mohr B (2010) The scalasca performance toolset architecture. Concurr Comput Pract Exp 22(6):702–719
15. Structural Simulation Toolkit (SST) DUMPI trace library. Accessed 15 Feb 2022. https://github.com/sstsimulator/sst-dumpi
16. Extrae documentation-Extrae 3.8.3 documentation. Accessed 15 Feb 2022. https://tools.bsc.es/doc/html/extrae/index.html
17. MVAPICH Benchmarks. Accessed 15 Feb 2022. http://mvapich.cse.ohio-state.edu/benchmarks/
18. Introducing Intel R MPI Benchmarks. Accessed 15 Feb 2022. https://www.intel.com/content/www/us/en/developer/articles/technical/intel-mpi-benchmarks.html
19. Sergeev A, Balso MD (2018) Horovod: fast and easy distributed deep learning in TensorFlow. arXiv preprint arXiv:1802.05799
20. Sergeev A (2017) Horovod source code. Accessed 15 Feb 2022. https://github.com/horovod/horovod
21. Alizadeh M, Greenberg A, Maltz DA, Padhye J, Patel P, Prabhakar B, Sengupta S, Sridharan M (2010) Data Center TCP (DCTCP). SIGCOMM Comput Commun Rev 40(4):63–74
22. Atikoglu B et al (2012) Workload analysis of a large-scale key-value store. SIGMETRICS Perform Eval Rev 40(1):53–64
23. Roy A, Zeng H, Bagga J, Porter G, Snoeren AC (2015) Inside the social network's (datacenter) network. SIGCOMM Comput Commun Rev 45(4):123–137
24. Hiperion repository homepage. Accessed 15 Feb 2022. https://gitraap.i3a.info/fandujar/hiperion
25. Karol M, Hluchyj M (1988) Queueing in high-performance packet switching. IEEE J Select Areas Commun 6(9):1587–1597
26. Anderson T, Owicki S, Saxe J, Thacker C (1993) High-speed switch scheduling for local-area networks. ACM Trans Comput Syst 11(4):319–352

27. Krizhevsky A, Sutskever I, Hinton GE (2017) Imagenet classification with deep convolutional neural networks, vol 60, pp 84–90. Association for Computing Machinery, New York
28. GALGO - Supercomputer Center homepage. Accessed 15 Feb 2022. https://www.i3a.uclm.es/i3a_t/galgo-supercomputing/