GPU Provisioning: The 80 − 20 Rule

✉Eleni Kanellou[1], Nikolaos Chrysos[1], Stelios Mavridis[1], Yannis Sfakianakis[1],
and Angelos Bilas[1,2]

[1] ICS-FORTH, N. Plastira 100, GR-70013, Heraklion, Greece
{kanellou,chrysos,mavridis,jsfakian,bilas}@ics.forth.gr
[2] Computer Science Department, University of Crete, Voutes Campus GR-70013
Heraklion, Greece

Abstract. The use of accelerators, such as GPUs and FPGAs, in datacenters has been increasing in an effort to improve response time for user-facing tasks. Although accelerators offer performance improvements for certain types of applications, they contribute to total cost of ownership and need to be deployed thoughtfully. In addition, the complexity of modern applications and different accelerator types, makes this a challenging task. In this paper, we derive a generalized model of workload core performance in datacenters. We find that the sweet spot for cost/benefit is when deploying a relatively low number of GPU accelerators compared to the number of servers. We also quantify this effect in the presence of data transfers and verify our observations using performance simulations and experiments in a realistic testbed with multiple GPUs. Overall, we detect aspects of accelerator deployment that should be taken into account to achieve trade-offs for their use in datacenters.

1 Introduction

With the end of Dennard scaling [1], the evolution of general-purpose processors cannot benefit from technology improvements alone. Thus, modern datacenters actively pursue heterogeneity by including special-purpose accelerators. For instance, Microsoft, Google, and Amazon reportedly use FPGA, ASIC, and GPU accelerators in order to speedup the execution of user-facing tasks, such as web search, speech recognition, etc. [2, 3]. The spreading integration of accelerators in datacenters raises a question that we address in this paper: *how many and what type of accelerators should we deploy in a functional datacenter?*

Deployment decisions may influence many datacenter cost and performance aspects, such as machine utilization, energy consumption, task latencies, cost of purchase and maintenance (as evidenced for instance in [4–7]). In addition, modifying a deployment paradigm can be a costly and time-consuming process. In this paper, we evaluate deployment compositions for heterogeneous datacenters. We use a model that approximates the performance of a datacenter when executing a given workload. We experiment with workloads that have a determined specific characteristics on some aspects (e.g. task duration and type) but we have also experimented with workloads that we designed so as to mimic the real-life traces available in literature that are representative for at least a class of datacenters. In summary, our main observation is the $80-20$ *rule*, i.e. that under certain circumstances, building a deployment with more than 20% GPUs does not significantly improve performance. Experiments in small-scale systems validate this finding.

Our results can help and guide datacenter designers when they plan a deployment. The data transfer overhead can limit the performance gain from accelerators, challenging our key observations, especially considering deployments with *remote* accelerators (or GPUs). Nevertheless, we were able to partly validate our results in such deployments using simulations and realistic testbeds. Our main contributions in this paper are:

1. A model to evaluate workload core performance in datacenters.
2. A powerful rule-of-thumb that can help direct GPU provisioning.
3. Evaluation some of our results on a realistic testbed of 6 GPUs and 6 CPUs.

The remainder of this paper is organized as follows. In Section 2, we outline the conceptual model of the datacenter, on which we base our analysis. Then, Section 3 describes the theory behind the 80-20 rule. In Section 4, we elaborate further on this rule, by providing simulations results taking data transfer into account and by presenting our results from small scale testbed with a mix of CPUs and GPUs. Related efforts in literature are summarized in Section 5. Section 6 provides a discussion of further aspects of the work and a summary.

2 Conceptual Datacenter Model

In this section, we construct a model that will allow us to define and evaluate various datacenter deployments. We actively neglect processor architecture details, such as out-of-order execution, core number, memory hierarchy, etc. The performance variability of the heterogeneous nodes is not (re)constructed by the model itself, but comes as user input[1]. We consider heterogeneous *processing units*, such as CPUs, GPUs, FPGAs, and ASICs. Each processing unit, or accelerator, has coordinates that identify its position (rack and shelve IDs) inside the datacenter. In the following, we may abuse terminology and refer to processing units in general as *accelerators*. The processing units execute tasks (or kernels), which are pieces of computation. Tasks may have input and output data, both of which are characterized by size and rack/shelve coordinates. Similarly with processing units, tasks are also characterized by a type. To simplify things, we assume that all tasks can be executed on all processing units, an assumption that we expect to hold true in the near future.

A datacenter is used in order to execute *workloads*. A workload is a collection of tasks. Tasks arrive in different time slots, as can be defined by a time-series or trace. The tasks in a workload may be stand-alone or they may belong to a *taskset* or *job*. Tasks belonging to a user-facing job are latency-critical tasks. Typically, they all have to complete fast in order for the job to terminate on

---

[1] In practice, we derive it by profiling applications or by using the raw capacity numbers advertized by device vendors.

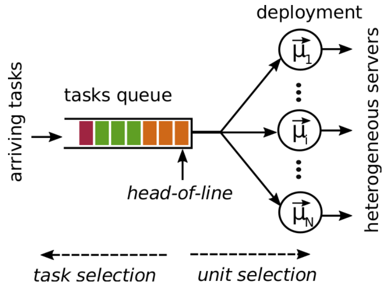


Fig. 1. High level model of the task selection and routing performed by the deployment scheduler. The service rate of the accelerators is not a scalar, but a vector with a scalar value for each task type.

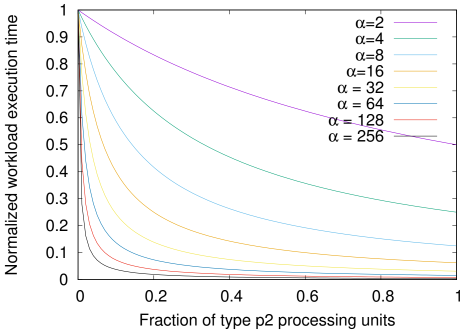


Fig. 2. Total execution time normalized to that of a CPU-only deployment as a function of $f = \frac{n}{N}$, i.e. the fraction of GPUs in the deployment, for varying affinity ratios between CPUs and GPUs.

time [8]. In our model, we vary the workload by using different mixes of task types and sizes. We consider that tasks arrive at the datacenter at a configurable arrival rate λ. At one extreme, tasks arrive slowly, and the datacenter is largely idle; at the other, many tasks arrive concurrently, in a burst, in which case they may have to spend some time in a task queue of infinite capacity.

Our model takes as input a 2D *affinity matrix* that defines the throughput of the processing units for each task type, measured in tasks per second. The affinity matrix expresses the heterogeneity of computation and hides the node-level intricacies of nodes, while nevertheless allowing the model to accurately model node-level performance. Any particular instance of a datacenter, i.e. a datacenter for which the specific numbers, types, and affinities of processing units is referred to as a datacenter *deployment*.

We may view the datacenter as an $M/M/c$ queuing system (multiple servers), where, however, not only the servers are heterogeneous in terms of service rate, but where the arriving "customers", i.e. tasks, also belong to several different classes. Effectively, as shown in Figure 1, the mean service times of the N computing units/accelerators are vectors instead of scalar values. Arriving tasks enter the system task queue waiting for service. The task queue is controlled by a *task scheduler*, which assigns tasks to time processing units, scheduling their execution. Each task will be executed by a processing unit. We identify the following two decisions in task scheduling: (i) Task selection, which determines the order in which the scheduler visits the tasks in the task queue, and (ii) Unit selection (or task routing), which selects the processing unit that will execute the currently examined task. The task selection order can be *FIFO or non-FIFO*, but in either case, multiple tasks can be selected at the same time when multiple computing units are available. The tasks can be issued in parallel, as would happen in a Map-Reduce workload, and can complete out of order. The unit selection can be *work conserving*, when it always picks one of the available accelerators for the next task, or *non-work conserving*, when the scheduler may prefer to wait for a better match to become available. The unit selection may be aware of resources heterogeneity, or oblivious of affinity relationships of tasks to accelerators.

We could set out to find the performance of an examined deployment using optimal scheduling. For instance, the minimization of workload completion time can be formulated as a maximum-weight flow problem in directed graphs. However, in principle, the optimal task scheduling may depend on the workload and probably also on the deployment. For this reason, in this paper we chose to compare deployments for the following set of schedulers for heterogeneous datacenters: (i) the *oblivious scheduler* visits tasks in FIFO order, and assigns them to an available processing unit; and (ii) the *fastest server first* visits tasks in FIFO order and the next (head of line) task is assigned to the available processing unit with the highest affinity. In either case, the unit selection is work-conserving.

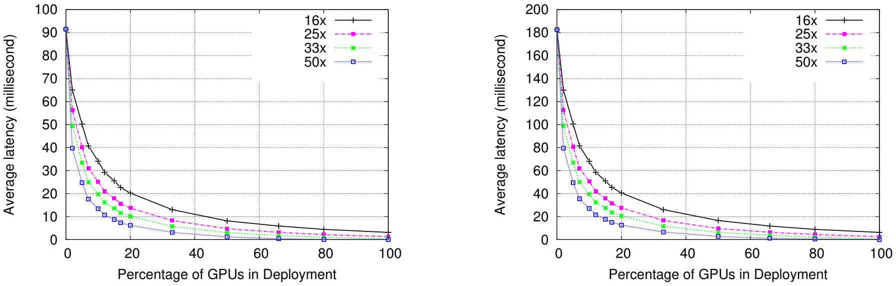
3 Deciding How Many Accelerators to Deploy

Suppose that we want to populate a rack with N machine slots using a mix of type p_1 and p_2 computing units. To improve the correspondence with a realistic (and popular nowdays) setup, assume that p_1 units are CPUs and p_2 units are GPUs. The workload of interest, W, consists of M number of tasks. We will first consider that all tasks are of the same type, t, and that, possibly belonging to a single job, they all arrive together, in a burst. Such workloads with a *predominant task-type* may be seen in dedicated clusters of production datacenters [9] working e.g. on web-search or deep learning. Furthermore, assume that it's a peak hour for the service provided by the cluster. In this work, we find that for a highly loaded system, i.e. $M >> N$, if the cluster is working on a single task type, any work-conserving scheduler performs optimally. Below we assume such a scheduler.

Assume that the affinity of CPUs on type t tasks is equal to 1 and that of GPUs is $\alpha > 0$. (Note that in general α is a function of task type t.) A starting deployment contains only CPUs (p_1 units), without any GPU (p_2 unit). The aggregate throughput (or service rate) in tasks per second of the deployment is $A^W_{CPU} = N$ tasks per time unit. Now consider the deployment that results if we substitute n out of N CPUs with GPUs. The aggregate service rate of the new deployment for this workload is $A^W_n = (N - n) + \alpha \cdot n$.

The system can be modeled as a queue of tasks drained by the heterogeneous processing units. Let T^W be the total time necessary to execute workload W. According to Little's law, the total drain time of the task queue is inversely proportional to the aggregate service rate of the deployment. Therefore, we can express the *speedup* obtained by a deployment containing n GPUs and $N - n$ CPUs relative to the CPU-only deployment as: $S_n = \frac{T^W_{CPU}}{T^W_n} = \frac{A^W_n}{N} = \frac{N+(\alpha-1)\cdot n}{N}$, and if we set $f = n/N$, we obtain:

$$S^W_n = 1 + (\alpha - 1) \cdot f. \tag{1}$$



(a) Average latency.

(b) 99.9 percentile.

Fig. 3. Latency under high load of workload consisting of GPU-F task-type only, in deployments where α, the relative affinity of GPUs and CPUs, changes.

If $\alpha = 1$, changing the proportions of CPUs and GPUs does not affect the execution speed. If $\alpha > 1$, then adding GPUs speeds up the execution; reversely, if $0 < \alpha < 1$, adding GPUs slows it down.

Indicative curves of the execution time of the n-GPUs deployment relative to the CPUs-only one, i.e. $\frac{T_n^W}{T_{CPU}^W} = \frac{1}{S^W}$, are presented in Figure 2 for various values of $\alpha > 1$. Because $\alpha > 1$, the latency is reduced as we add more GPUs. For each curve, the latency is minimized when all processing units are of GPUs, at the right end of graph. The figure shows clearly that, for large α, *the deployments with more than than around* 20% *GPUs, depending on* α*, provide diminishing benefits.* We refer to this observation as *the 80 − 20 rule*. At first, this may seem somewhat surprising because according to Equation 1 (and normal intuition), the speedup is proportional to f. The catch is that the speedup is also proportional to α, which can outweigh the effect of f: a few but well customized accelerators may cut down sharply the workload execution time, masking out the benefits of bringing in more accelerators.

One can further show that, for $\alpha >> 1$, we can have at least 75% of the latency reduction achieved by the N-GPUs system $(T_0^W - T_N^W)$ with a deployment containing $(f =)\ \frac{4+a}{3}$ times fewer GPUs. For example, if a GPU kernel achieves a speedup $\alpha \geq 12$ relative to CPUs, we need to replace just 20% of the processors to reach this performance point. To get the remaining 25% of latency reduction, we need to buy $5x$ more GPUs. Many accelerators with good affinity are reported in the literature [10]. For instance, GPUs contain thousands of cores and can greatly speedup traditional high-performance computing or deep-learning tasks.

Adding GPUs to a fixed deployment: Adding GPUs (accelerators) to servers *without removing CPUs* does not modify the aforementioned results. One can easily show that the speedup obtained by a deployment with N CPUs *and* m GPUs is: $S_m^W = \frac{T_{CPU}^W}{T_m^W} = \frac{A_m^W}{N} = \frac{N+(\alpha+1)\cdot m}{N}$ therefore: $S_m^W = 1+(\alpha+1)\cdot f$.

Bad performing accelerators: The hardware of accelerators is frequently built to match the requirements of certain tasks. Effectively, some tasks with a lot of branching activity may run faster on CPUs instead of GPUs. In order to account for such tasks in our model, allow for values of $\alpha < 1$. The results are analogous to those considered so far: adding GPUs in place of CPUs increases the workload execution time, along curves that are symmetrical to the ones shown in Figure 2.

4 Exploring the $80-20$ Rule

In this section, we validate $80-20$ rule with experiments for the above analysis, both in a custom-made C++ simulator, as well as in a setup with real machines. **System simulations.** Our simulator essentially follows the abstract datacenter model described in Section 2 and outlined in Figure 1. In our first experiment, we model a *single rack* consisting of $N = 40$ machines, each of which can be either CPU or GPU. We feed the simulator with a workload consisting of same-type tasks. A new job arrives in the system every 10 μs and consists of 30 tasks. We use an abstract task type *GPU-friendly* (GPU-F) that can be accelerated on GPUs. These tasks take 500 μs to execute on the simulated CPU. On a simulated GPU a task takes $\frac{500}{a}$ μs, i.e. a single GPU is α times faster than a CPU. Thus, α defines the relative affinity of GPUs and CPUs with respect to GPU-F tasks. Notice that we have based decision of considering a proportional relation of relative affinity between GPUs and CPUs, on real-life experiments that we have performed, where we measured individual execution time of tasks on GPUs and CPUs. In each of those experiments, our workload consisted of a single task of a different type for each experiment, focusing on making sure the execution times that we measured excluded any data transfer or queuing delays.

With respect to our simulated experiments, the small job inter-arrival that we use (10 μs) creates a high load for any deployment, thus resembling peak hours in a datacenter. In this experiment, we use the fastest-server first scheduler,described in Section 2. However, we have observed that for single-task workloads operating under high load, any work-conserving scheduling performs the same. In this experiment, we assume that no data transfers are needed in order to execute a task.

Figure 3 shows the results of a GPU-friendly workload, executed in configurations that contain a varying percentage of GPUs. It depicts multiple plots, representing deployments with a varying number of GPUs and a fixed affinity relationship (i.e. α) between CPUs and GPUs for tasks of type GPU-F. The x-axis shows a varying percentage of GPU contents. In Figure 3a, the y-axis shows the average task latency in ms; in Figure 3b, we depict the 99.9 percentile of task latency. Our simulation results match our theoretical expectations (cf. Section 3): the higher the affinity ratio between GPUs and CPUs, the greater the overall performance improvement, *both in average and tail latency.* Currently, a usual practice is to over-provision datacenters, aiming at always ensuring resource availability. However, as we see in the plot, after 20% of GPUs, “throwing more hardware at the problem” does not significantly improve latency.

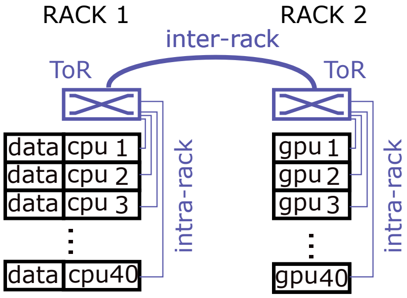


Fig. 4. A deployment spanning two racks. In rack 1, each of the 40 servers has a CPU plus a data storage device. The data needed by each task is hosted in one of these servers. Rack 2 has 40 processing units which can be either a CPU or a GPU. The hosts inside each rack are connected to a top-of-rack (ToR) switch using 10 Gb/s links. The two racks are connected with a link running at 200 Gb/s.

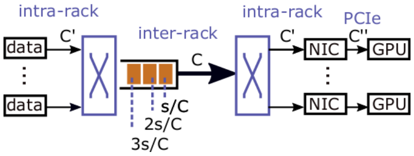


Fig. 5. Transfers between the host that holds the task data and a remote accelerator, which can be a CPU or a GPU. The GPUs are attached to the PCI interface in another rack. As multiple transfers of size s may be concurrently active between the two racks, they are necessarily serialized on the inter-rack link of capacity $C = 200$ Gb/s. The maximum rate of a *single transfer* is, however, always limited by the capacity of an intra-rack link which runs at 10 Gb/s.

The case of remote accelerators. In this experiment, we examine whether the effect of data transfers influences the observed trends. For this purpose, we simulate two racks, racks 1 and 2. We assume that all GPUs in the deployment are located in rack 2, as shown in Figure 4. This models a deployment with remote (network-attached) accelerators, which highly stresses the network. All tasks must take their data from a server in rack 1. Therefore, whenever a processing unit in rack 2 is used to accelerate the execution, a data transfer over the inter-rack link is triggered.

As noted in Figure 5, the intra-rack link runs at 200 Gb/s. In our model, the rate of a single transfer is limited by the minimum of the current fair share on inter-rack link and the 10 Gb/s rate of a single intra-rack link–assuming that the modern PCI can run faster than that. In this experiment, we assume a relative affinity between GPUs and CPUs equal to $\alpha = 16$. Interestingly, the $80 - 20$ rule still holds. Figure 6 contains multiple plots corresponding to different task data size. In our configuration, moderate data sizes correspond to up to 320K bytes. For larger transfers, the data transfers dominate and the use of accelerators does not improve the performance even at 100% GPUs in the deployment.

This effect is depicted in Figure 7. The vertical blocks depict the delay incurred by task execution and the required data transfers over inter-rack links, assuming the data that the task needs has to be transferred to the CPU or the GPU from a remote rack. The higher the block, the longer the corresponding delay. Notice that in the case of GPUs, we assume that, once the data reaches the rack and node where the GPU is located, then the eventual PCI transfer delay is negligible, compared to the rack-to-rack transfer delay. The observed effect is

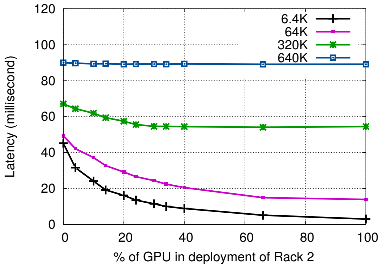


Fig. 6. Average latency, under high load, of workload consisting of same-size, GPU-F task-type only, with different data transfer sizes. X-axis indicates % of GPUs in the deployment of rack *B*.

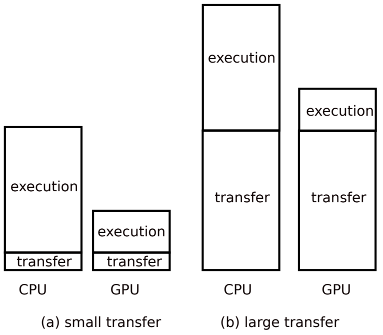
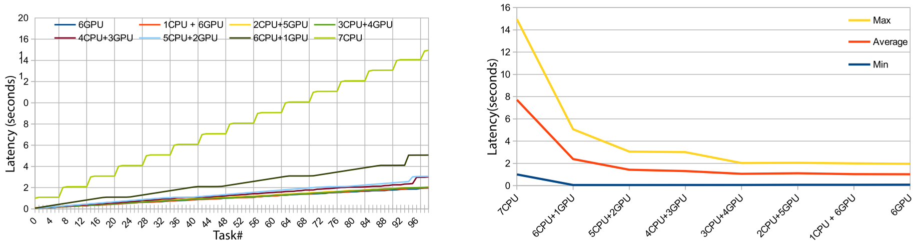


Fig. 7. The effect of data transfers on accelerator speedup.



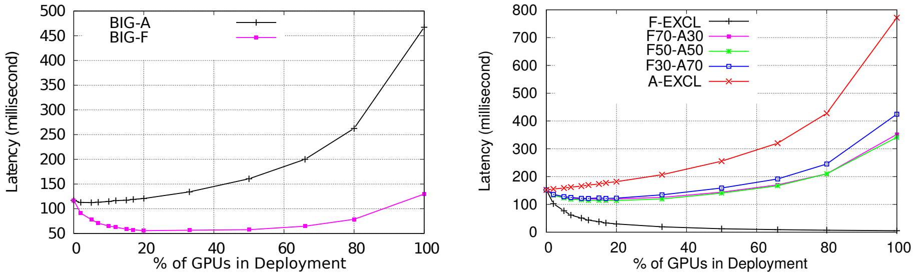
(a) Individual task latency with varying GPU/CPU ratio deployments.

(b) Average task latency with varying GPU/CPUs ratio deployments.

Fig. 8. Task latency behaviour for deployments with 6 (CPU/GPU) accelerator.

reminiscent of Amdahl's law: the data transfer is the serial part of a computation that limits the speedup that can be obtained by any good accelerator. When the latency of the communication required to move the data to a remote accelerator approaches the computation time on the CPU, the maximum speedup that can be obtained is upper bounded by 2×. In this case, we might be better off if we avoid the data transfer and schedule the task to a processing unit that can access the data locally [11]. **Validation with NVidia GPUs.** We have tested our theoretical and simulation findings on a real machine setup. Specifically, we populated six (6) Gen PCIe (8x) slots of an Intel Xeon CPU $E5-2630$ processor with (6) P1000 GPUs from NVidia. We employ a custom middleware [12], which allows for the sharing of several accelerators, such as GPUs, among the processors residing on a single node. We use *DarkGrey* kernels as workload. DarkGrey is acustom-created kernel that converts an RGB image to grayscale and then darkens the result.

In our experiments, we manipulate a 62 MB image files. On a CPU, the kernel takes $895ms$ to execute, whereas it takes only $7.26ms$ on P1000. However,



(a) BIG-A: Big jobs contain GPU-A tasks. The inverse holds for the BIG-F trace.

(b) A trace AX-FY contains X% GPU-A jobs and Y% GPU-F jobs. -EXCL stands for "exclusive".

Fig. 9. Realistic synthesized job size distribution. Average task latency during simulated peak hours.

in order to execute the task on the GPU, we need to *transfer the data over the PCI*, an operation that takes 38 *ms*. The workload tested in our experiments is set of 100 DarkGrey tasks that enter a task queue. Tasks are scheduled across the available processing units, regardless of task affinity, using the oblivious work-conserving FIFO scheduler policy. In Figure 8, we keep the total number of processing units constant to six (6), and we examine how the task latency is affected when vary the mix of GPUs and CPUs in the deployment.

Figure 8a depicts the individual latency of each one of the 100 tasks for each deployment. The latency of each task includes the queuing delay, i.e. the total execution time of the tasks that precede it in the queue. Tasks that can be executed concurrently from start to finish, incur the same queuing delay. This is most clearly shown in the plot depicting the task latency in the deployment with 6 CPUs. This deployment produces a stair-step-type graph, where task latency increases in every batch of 6 consecutive tasks, since 6 tasks can be executed in parallel on the 6 processing units. Figure 8b shows the average task latency for each of the deployments. As can be seen, we obtain a great reduction of average task latency after trading the first two CPUs for two GPUs. However, this trend does not continue so prominently after the inclusion of more GPUs. This is in accordance with our theoretical expectations outlined previously in Figure 2. Note that this effect is still observable, despite the fact that executing on the GPU incurs an additional latency of 38 *ms*.

Mixed Workloads. We now turn our attention to workloads that may contain both GPU-F tasks, as well tasks that are not suitable for execution on GPU, meaning that the GPU cannot sufficiently accelerate them, possibly even harming their execution time with respect to the execution on CPU. We refer to this type of tasks as *GPU-averse* (GPU-A)[2]. The GPU-A type represents, for example, tasks with branching or memory access patterns ill-suited for GPUs.

We start by examining average task latency in Figure 9. We have used the fastest-server-first scheduling policy and showcase various workloads in situations of high load. Given that information regarding the task length, exact type, distribution, etc., of a datacenter trace, is not readily available, we attempt to synthesize workloads that are as realistic as possible, basing our choice of values on datacenter trace analyses such as [13–17] and others. We use simple algorithms such as the ones used in [17] in order to create synthetic datacenter traces containing combinations of GPU-A and GPU-F tasks.

Figure 9a shows task traces with realistic taskset size distributions, where the task type depends on the job size; and Figure 9b shows task traces with realistic taskset size distributions and different task type mixes. We observe that the highest gain in task latency occurs once more with the inclusion of already a few GPUs. However, this only occurs on those traces that are predominantly or exclusively composed of GPU-F tasks. On the contrary, when traces are exclusively of type GPU-A or when GPU-A tasks dominate, the inclusion of more GPUs than CPUs ceases to offer benefit and is instead even detrimental. But what is more, when GPU-F and GPU-A tasks are in equivalent numbers or when even GPU-A tasks are in the trace even though GPU-F tasks dominate, then again only the inclusion of the first few GPUs offers benefit.

5 Related Work

As an alternative to pure simulation, analytical modeling and analysis can be used. In [18], the performance of a given workload is expressed as a function of the CPU and GPU throughput and the data transfer time between them. Mathematical solvers are used to predict the optimal relative content of CPUs and GPUs. However, this method requires some partial profiling and is targeted at specific applications. While it remains to be seen whether such analysis could be applied to datacenters, in our paper we wished to eschew analytical solving and explore performance over a more generic variety of application types and deployment compositions. In [19], a queuing-theoretical approach is favored, in order to determine which deployment decisions lead to robustness, i.e. maintaining the minimum worst-case energy consumption in the face of a varying workload. In contrast, in this work, we are concerned with reducing task queuing time and execution latency. An exploratory approach is also followed in [20]. There, however, analytic workloads are modeled and used as workload inputs to a machine-learning system. After training on a variety of hardware configurations, the system predicts expected performance of a given workload on a given datacenter setup. We tackle the issue differently, striving to predict the hardware configuration that is sufficient, given expected workload characteristics.

---

[2] We opt for these two categories because our model relies on expressing the suitability of an accelerator for a task instead of the nature of its computation.

6 Conclusion

In this paper, we studied what constitutes a good datacenter deployment in the face of heterogeneity. We focused on the use of GPUs in datacenters and developed a model in order to explore various deployment options, but our insights may prove useful in cases of ad hoc heterogeneity, as well. Our main finding shows that even including few GPUs significantly improves performance, while often, a few but powerful GPUs can outperform many, but less powerful ones. We proceed to discuss a number of assumptions made in obtaining these results, as well as some general trends worth noting.

All tasks can be executed on all processing units: We made the assumption that any task may be executed on any accelerator. While this was done for the sake of abstraction, it may gradually become a reality as people port their codes into accelerators. For instance, specialized, highly parallel applications, traditionally executed on dedicated grids, are now moved to datacenters for the sake of lowering energy consumption [21] and applications such as analytics are being actively rewritten to take advantage of new accelerator architectures [22].

Independent tasks: Our model assumes that tasks in a taskset are not interdependent, and therefore, that they can be executed in parallel as long as accelerators are available. This assumption is valid for jobs belonging to Online Data-Intensive (OLDI) services and Map-Reduce-style computations.

Homogeneous workloads: Datacenter workloads do not necessarily consist of a single task type. However, many high-performance datacenter clusters work on a predominant task type, e.g. web search, and many datacenters are deployed to run tasks that benefit from the presence of certain accelerators. For instance, one may deploy a set of physical machines, or of virtual machines in the cloud, in order to run machine learning algorithms. Both the learning and inference phases of neural networks can benefit from GPUs or ASIC accelerators [22]. Our work can help the user estimate the number of accelerators needed in order to obtain the targeted performance.

While our exploration is not exhaustive, we regard our findings as a step towards understanding the benefits offered by accelerators, considering as a plus that they were derived without being dependent on a specific datacenter configuration or a particular application. Besides datacenter planners, our results can be also used by users that deploy virtual machines and GPUs (or generally accelerators) in the cloud. Given that scarce FPGA resources are a constraint, our results may also prove useful when populating cluster of FPGAs [23, 24]. In future work, we plan to deal with diversifying some of the aforementioned assumptions. For instance, our current efforts focus on workloads with interdependent tasksets, and we consider that heterogeneous workloads and the validation of the $80-20$ rule under different scheduling techniques are interesting avenues of future research.

Acknowledgments. This work has received funding from the European Union's Horizon 2020 Research and Innovation program "VINEYARD: Versatile Integrated Accelerator-based Heterogeneous Data Centers", under grant agreement No 687628.

References

1. Esmaeilzadeh, H., Blem, E., St. Amant, R., Sankaralingam, K., Burger, D.: Dark silicon and the end of multicore scaling. In: Proceedings of the 38th Annual International Symposium on Computer Architecture. (ISCA), NY, USA, ACM (2011) 365–376
2. Jouppi, N.P., Young, C., Patil, N., Patterson, D., Agrawal, G., Bajwa, R., Bates, S., Bhatia, S., Boden, N., Borchers, A., et al.: In-datacenter performance analysis of a tensor processing unit. arXiv preprint arXiv:1704.04760 (2017)
3. Putnam, A., Caulfield, A.M., Chung, E.S., Chiou, D., Constantinides, K., Demme, J., Esmaeilzadeh, H., Fowers, J., Gopal, G.P., Gray, J., et al.: A reconfigurable fabric for accelerating large-scale datacenter services. In: ACM/IEEE 41st International Symposium on Computer Architecture (ISCA), IEEE (2014) 13–24
4. Kindratenko, V.V., Enos, J.J., Shi, G., Showerman, M.T., Arnold, G.W., Stone, J.E., Phillips, J.C., m. Hwu, W.: Gpu clusters for high-performance computing. In: IEEE International Conference on Cluster Computing and Workshops. (2009)
5. Wu, C., Buyya, R.: Cloud Data Centers and Cost Modeling: A Complete Guide To Planning, Designing and Building a Cloud Data Center. 1st edn. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2015)
6. Koomey, J., Brill, K., Turner, P., Stanley, J., Taylor, B.: A simple model for determining true total cost of ownership for data centers. Technical report, Uptime Institute (2008)
7. Popa, L., Ratnasamy, S., Iannaccone, G., Krishnamurthy, A., Stoica, I.: A Cost Comparison of Datacenter Network Architectures. In: Proceedings of the 6th International Conference on Emerging Networking Experiments and Technology, New York, NY, USA, ACM (2010) 16:1–16:12
8. Dean, J., Barroso, L.A.: The tail at scale. Communications of the ACM **56**(2) (February 2013) 74–80
9. Barroso, L.A., Dean, J., Hölzle, U.: Web search for a planet: The google cluster architecture. IEEE Micro **23** (2003) 22–28
10. Kachris, C., Soudris, D.: A survey on reconfigurable accelerators for cloud computing. In: Field Programmable Logic and Applications (FPL), 2016 26th International Conference on, IEEE (2016) 1–10
11. Zaharia, M., Borthakur, D., Sen Sarma, J., Elmeleegy, K., Shenker, S., Stoica, I.: Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling. In: Proceedings of the 5th European conference on Computer systems, ACM (2010) 265–278
12. Mavridis, S., Pavlidakis, M., Stamoulias, I., Kozanitis, C., Chrysos, N., Kachris, C., Soudris, D., Bilas, A.: Vinetalk: Simplifying software access and sharing of fpgas in datacenters. In: Field Programmable Logic and Applications (FPL), 2017 27th International Conference on, IEEE (2017) 1–4
13. Chen, Y., Alspaugh, S., Katz, R.: Interactive Analytical Processing in Big Data Systems: A Cross-industry Study of MapReduce Workloads. Proc. VLDB Endow. **5**(12) (August 2012) 1802–1813
14. Reiss, C., Tumanov, A., Ganger, G.R., Katz, R.H., Kozuch, M.A.: Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In: Proceedings of the

Third ACM Symposium on Cloud Computing. SoCC '12, New York, NY, USA, ACM (2012) 7:1–7:13

15. Awasthi, M., Suri, T., Guz, Z., Shayesteh, A., Ghosh, M., Balakrishnan, V.: System-level characterization of datacenter applications. In: Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering, New York, NY, USA, ACM (2015) 27–38
16. Moreno, I.S., Garraghan, P., Townend, P., Xu, J.: Analysis, modeling and simulation of workload patterns in a large-scale utility cloud. IEEE Transactions on Cloud Computing **2**(2) (April 2014) 208–221
17. Wang, G., Butt, A.R., Monti, H., Gupta, K.: Towards synthesizing realistic workload traces for studying the hadoop ecosystem. In: 2011 IEEE 19th Annual International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems. (July 2011) 400–408
18. Shen, J., Varbanescu, A.L., Sips, H.: Look before you leap: Using the right hardware resources to accelerate applications. In: Proceedings of 16th IEEE International Conference on High Performance Computing and Communications (HPCC 2014), IEEE (August 2014) 383–391
19. Garg, S., Sundaram, S., Patel, H.D.: Robust heterogeneous data center design: A principled approach. SIGMETRICS Perform. Eval. Rev. **39**(3) (2011) 28–30
20. Venkataraman, S., Yang, Z., Franklin, M., Recht, B., Stoica, I.: Ernest: Efficient performance prediction for large-scale advanced analytics. In: Proceedings of the 13th Usenix Conference on Networked Systems Design and Implementation. NSDI'16, Berkeley, CA, USA, USENIX Association (2016) 363–378
21. Li, K.: Power and performance management for parallel computations in clouds and data centers. Journal of Computer and System Sciences **82**(2) (2016) 174–190
22. Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., et al.: Tensorflow: Large-scale machine learning on heterogeneous distributed systems. arXiv preprint arXiv:1603.04467 (2016)
23. Katevenis, M., Chrysos, N., Marazakis, M., Mavroidis, I., Chaix, F., Kallimanis, N., Navaridas, J., Goodacre, J., Vicini, P., Biagioni, A., et al.: The exanest project: Interconnects, storage, and packaging for exascale systems. In: Digital System Design (DSD), 2016 Euromicro Conference on, IEEE (2016) 60–67
24. Abel, F., Weerasinghe, J., Hagleitner, C., Weiss, B., Paredes, S.: An FPGA Platform for Hyperscalers. In: IEEE 25th Annual Symposium on High-Performance Interconnects (HOTI), IEEE (2017) 29–32