DOI: 10.2507/32nd.daaam.proceedings.011

OPTIMIZING KUBERNETES PERFORMANCE, EFFICIENCY AND ENERGY FOOTPRINT IN HETEROGENOUS HPC ENVIRONMENTS

Vedran Dakic, Mario Kovac & Jasmin Redzepagic

This Publication has to be referred as: Dakic, V[edran]; Kovac, M[ario] & Redzepagic, J[asmin] (2021). Optimizing Kubernetes Performance, Efficiency and Energy Footprint In Heterogenous HPC Environments, Proceedings of the 32nd DAAAM International Symposium, pp.0075-0083, B. Katalinic (Ed.), Published by DAAAM International, ISBN 978-3-902734-33-4, ISSN 1726-9679, Vienna, Austria
DOI: 10.2507/32nd.daaam.proceedings.011

Abstract

This paper describes the most common complexities in heterogenous HPC (High Performance Computing) environments, the most common problems facing heterogenous HPC environments arising from these complexities, the most commonly used solutions to these problems and the potential solutions to these problems by using new methodologies. One of these methodologies could be using containers and container orchestration methodologies instead of virtual machines to lower compute overhead when dealing when large-scale, distributed computing concepts like HPC. This will require significant efforts in different phases of the HPC process - workload preparation, producing different types of images and layers for various HPC applications, potential changes in HPC schedulers etc. In the era of green computing, that will also require significant improvements in energy management process, that needs to be an integral part of the overall design.

Keywords: High Performance Computing; Kubernetes; containers; virtual machines; automation; orchestration.

1. Introduction

Ever since the invention of supercomputers in the 1960s, science world has been one of the key customers on that market. As technology, programming and computational models improved, another type of supercomputer was introduced, and its subsequent application was called HPC (High Performance Computing). High Performance Computing is the application of supercomputers to computational problems that are either too large for standard computers or would take too long.

In the past decade, HPC environments have been extensively using virtual machines/clouds because of the need for computational power, agility, and manageability. But there are other methodologies for running workloads, like containers. Containers offer a broad array of benefits, including a consistent lightweight runtime environment through OS-level virtualization, as well as low overhead to maintain and scale applications with high efficiency [1]. This makes them a good candidate to apply to HPC environments, especially as HPC environments become more and more heterogenous. IT industry in general became more and more aware of the fact that significant gains in computational power can be achieved by moving towards containers, years ago. Further operational advantages can be achieved by implementing automation and orchestration principles as they can help us do a lot more while working manually a lot less. The question that we need to ask is - can the same idea be used in the area of HPC?

What are the real obstacles on the way of using containers as compute resources for HPC? Are there different ways of overcoming those issues and tapping into the idea of green computing at the same time? We think that the correct way forward is to implement tighter hardware and software integration, and to create programmable, custom Kubernetes schedulers for HPC workloads. And this is precisely the subject of our paper.

2. Theoretical background

By definition, heterogenous computing means mixing different types of cores within the limits of a single node. We can use this approach in a variety of different ways, like:

- by using CPU and GPU in the same node (for example, x86 CPU and NVIDIA Tesla GPU).
- by using CPU and ASIC in the same node (for example, x86 CPU coupled with Intel eASIC or Xilinx UltraScale).
- by using different CPU architectures in the same node (for example, AMD SkyBridge).

The most widely defined goal of heterogenous computing is to maximize performance and to provide energy efficiency. We can add additional performance by utilizing other hardware devices in the same nodes, by using concepts like VFs (Virtual Functions) [2]. As an example, we can partition a PCI Express device to multiple sub-devices that can be independently configured (for performance, cost, Quality of Service or other reasons). We can use specialized PCI Express cards to partition them across different compute objects. Typical example of that methodology is SR-IOV (Single Root Input/Output Virtualization), that enables us to use Virtual and Physical Functions to manage access to physical, underlying specialized hardware. Virtual functions (VFs) are logical PCI Express instances of Physical Functions (PF) on a specialized PCI Express device that can be virtualized by using a Hypervisor. For example, if we have a SR-IOV compatible physical network card that is placed in a server's PCI Express slot that supports SR-IOV, we can present that physical network card as multiple virtual network cards. We can then connect these virtual network cards to compute objects like virtual machines and containers, in essence giving them much more direct approach to the underlying specialized hardware, which means less overhead, more bandwidth and less latency. Having in mind that HPC applications are very sensitive to latency and bandwidth, using specialized devices like these can have a positive impact on the performance of HPC environment.

If we were to use this kind of functionality inside a container, other criteria need to be met. For example, hardware vendor needs to have a driver that's able to expose these functionalities to a container as it's a completely different methodology to using a hypervisor. For example, Mellanox has a plugin called k8s-rdma-sriov, which can be used in such a scenario if we have a Mellanox InfiniBand adapter and want to assign it to multiple different containers. There are multiple different technologies that are prominent - on desktop, server and HPC markets. Some common examples include:

- GPGPU (General-Purpose Graphics Processing Unit) technologies like CUDA (Compute Unified Device Architecture) and OpenCL (Open Computing Language) on various hardware platforms like NVIDIA/AMD GPUs paired with x86 processors.
- Intel Xeon Phi and similar accelerators, that can take advantage of concepts like OpenCL, OpenMP, C/C++, Fortran etc.
- Xilinx and similar FPGA/ASICs, that can also take advantage of various concepts like OpenCL, HLS etc.

All these solutions can be mixed and matched and used side-by-side with other solutions, like Open MPI (Open Message Passing Interface), Intel MPI, MPICH (Message-Passing Interface Chameleon) for distributed parallel computing, which fits well with the concept of HPC. Idea behind these technologies is quite straightforward – to run code on multiple processors and use one of these solutions via library calls. Other aspects of modern computing – Big Data, AI, HPC - are just further proof to the following point – the present and future of computing are based on heterogenous model. There are different reasons for that, such as:

- x86-based ecosystem has been well-developed over the past 40+ years, which means that a lot of workloads can be handled by it.
- ARM-based ecosystem has been focused more on non HPC markets in the past However, the newly developed 64-bit architecture along with their respective SoCs will have a large impact on the CPU performance and memory bandwidth of these ARM processors in HPC domain [3].
- RISC-V based ecosystem has been focused on open-sourcing CPU technology via open instruction set and could be the “Linux of CPU design” if everything goes well.
- GPUs are very fast for specific types of workloads.
- ASICs are even faster for specific types of workloads.

Combining these different technologies in one or multiple nodes could yield the best possible design and architecture for a modern HPC data centre. Of course, there are various geopolitical and strategical reasons why current state of the industry in which x86-based environments are prominent might not be feasible for the long-term feature. EU has been investing a lot of resources into EPI (European Processor Initiative), showing its intent to break free from non-EU owned technology.

3. Hardware and software integration

Let’s now discuss the software aspect. For the most part, workloads that we’re running today can use either virtual machines (VMs) or containers (Docker, Podman). Integrating these two types of solutions offers the best of both worlds, and it’s also a valid approach for HPC workloads. But that also means we need to pay close attention to how we’re going to integrate these two concepts. Simple brute-force integration just won’t do in the 21st century - randomly placing virtual machines or containers on various nodes isn’t efficient nor is it a good policy. Nodes differ in capabilities, compute power, GPU/ASIC availability, storage subsystem performance, performance per Watt figures etc. Connecting performance per Watt figures with node capabilities, automatization and orchestration technologies like Kubernetes can be a good approach for intelligent workload placement, given a specialized, fully programmatic scheduler. There are multiple questions that need answers if we were to use the Kubernetes approach as a valid alternative to virtual machines. Questions like:

- Are containers a better solution than virtual machines?
- Are containers a solution that can be used for any workload?
- How can using containers improve execution of HPC workloads?
- If containers offer a possibility for improving HPC workload execution, can we somehow connect that to the idea of green computing?

Let's start by discussing differences between virtual machines and containers and how these differences can be used to improve HPC workload efficiency. Key architectural difference between a virtual machine and a container is the size of those two concepts. Virtual machines are much bigger objects than containers, as they require more resources to work. First and foremost, they require a virtual disk with installed operating system, for every virtual machine installed on a node. If we multiply that by X virtual machines, that overhead can easily mean terabytes of disk space being used to store virtual machine elements that are needed for it to boot, let alone do some kind of usable work. There are cloud and VDI (Virtual Desktop Infrastructure) concepts that we could use to alleviate this problem, but they also bring additional performance and management overhead. For example, we could use linked clones - a concept that uses virtual machine virtual disk (usually referred to as "Golden image") assigned to multiple virtual machines at the same time.

As it's impossible to assign the same virtual disk to multiple virtual machines in read-write mode at the same time, a hypervisor would need to create *delta* virtual disk per virtual machine so that we can store data to all virtual machines linked to the same image. But, at the same time, multiple virtual machines using the same image to boot operating system and work can lead significant performance degradation. Booting multiple virtual machines using the same image at the same time leads to a phenomenon called *boot storms*. Result of this phenomenon is a significant delay in booting virtual machines that can lead to hours of boot processes if it happens in a big environment. Furthermore, real-life usage of linked-clone virtual machines is also slower than using regular virtual machines. As there are a lot of HPC applications that are very sensitive to storage I/O and latency, this might have a significant influence on HPC application performance. Containers use a completely different approach, as they use container images that are made up out of different layers unique to specific container and application that's containerized. Generally speaking, for everything else, containers use resources from the underlying operating system, therefor not wasting disk space on content that's used X times for X containers.

Furthermore, as far as compute size goes (compute size equals CPU resources plus memory resources), virtual machines are a much bigger concept. If we want virtual machines to run properly, we need to reserve resources for them, which increases their context size. And that's on top of the fact that virtual machines need a lot of compute power just to run the operating system. And again, we can multiply that by X number of virtual machines. In contrast to that, containers are regular processes running directly on the underlying operating system. We can manipulate container resources by using well-known ideas like *cgroups*, kernel limits (by using *pam_limits* module), *ulimit*, *cpulimit,* etc. We can also easily manipulate container execution priorities, before we start them or after we start them - by using concepts like *nice* and *renice*. We can't implement these concepts on virtual machines easily, and even when we do, similar concepts in virtualization work by manipulating behavior of virtual machine scheduler, not virtual machines *before* they undergo their scheduling process.

These two key architectural differences can be summed up from our perspective to say that virtual machines need a lot of resources and time (measured in minutes) to be spooled up so that they can do the work. In contrast, containers use less resources and take seconds to spool up. All of this leads us to the following conclusion - given the right set of circumstances, containers are easily the better solution to run almost any workload. "Almost any workload" is an important part of that conclusion, as it helps us to answer the next two questions. It's impossible to state that containers are the better solution for any kind of workload. There are various capabilities of virtual machines that containers don't have at all or have partial virtual machine capabilities. For example, working with advanced PCI Express devices like ASICs, GPUs and FPGAs is sometimes possible *only* if we're using virtual machines. This will depend solely on the ASIC/GPU/FPGA vendor, if they're willing to offer software support that enables us to use all their products' capabilities inside containers. We mentioned *k8s-rdma-sriov* project, which is a good example of what can be done with containers if a manufacturer develops deep integration between a hardware device and its corresponding software stack. So, the answer to our next two questions is simple - containers can not necessarily be used for *any* type of workload, but given the right set of circumstances, they *can* be used.

The answer to the last question seems logical - if we have a large infrastructure and compatible workloads, using containers can offer huge strides towards the idea of green computing. Containers offer the capability to use our infrastructure in a much more efficient manner, without added overheads of running virtual machines. That means that we can use our nodes to run more containers, which leads to an environment where compute power is used to do more computing, less general, unnecessary work. So, generally speaking, because of improved efficiency, we're either given a chance to shrink our infrastructure to suit our computational requirements which means less power usage, or we can do more work given the same infrastructure. Both of these scenarios are completely in line with the idea of green computing. We can further this point by saying that introducing various, more efficient CPU architectures will also have a huge impact on performance-per-Watt calculation for any infrastructure.

There are other ways in which using containers can lead to efficiency improvements. We mentioned container images and layers. Container images are objects that consist of sub-objects - layers. These layers can be used to create new images, that can be customized to create new container images. For example, let's consider the following scenario:

- we have a layer that contains all the necessary binaries and libraries for application 1.
- we have a layer that contains all the necessary binaries and libraries for application 2.
- we have a layer that contains all the necessary binaries and libraries for application 3.
- we have a layer that contains all the necessary binaries and libraries for application 4.
- …..
- we have a layer that contains all the necessary binaries and libraries for application n

At any given moment, we can use layer with application 1 and layer with application 2 to create a new image that will have both of these applications inside. Or, we can use layer with application 3 and layer with application n to create a new image that will have those two applications. We just need to create a file that describes which layers need to be combined and create a new image. The resulting image will be small in size, and constantly re-usable in our environment, which is a very good approach from both operational and CapEx standpoint.

If we were to try to do the same thing on a virtual machine, we'd soon realize that it's almost impossible to do this in environments that don't have additional capabilities not native to HPC and regular virtualized environments. There are concepts in VDI that can make this approach happen, but this is a completely different use case with different architectural requirements and prohibitive licensing cost. If we exclude VDI principles as non-viable for our HPC use cases, the only thing left to do for us in HPC environments would be to create X number of virtual machines with different application combinations. That would just help us confirm what we already discussed - that from overhead standpoint, virtual machines are a demanding concept that's not necessarily the best for every use case we might need them for. Having the capability to do compute operations with smaller objects is the way to go, if possible.

This is the reason why, as a concept, Kubernetes is going to have a major role to play in our work. Using containers in manual fashion would mean a lot of additional administrative work, which we want to avoid at all costs. We also don't want to use built-in methods for container placement (concepts like Docker Swarm), as they're too limiting in their capabilities and not aimed at use cases that involve large infrastructures with hundreds or thousands of nodes.

Kubernetes is there to do all that work for us, as its primary objective is to *manage* and *schedule* containers to run assigned workloads on Kubernetes nodes. We want these workloads to be placed on Kubernetes nodes in an intelligent way, that doesn't just use node CPU and memory availability as the only way to determine container placement. We are going to use Kubernetes to provide us with storage orchestration capabilities, as we need permanent storage mounted to our containers, but we also want that done in a secure way. Then, we are going to use Kubernetes for its clustering and self-healing capabilities, as it can re-spool containers that fail (for example, when a Kubernetes node fails).

We are going to focus on problems that are facing our use cases – heterogenous HPC environments based on all technologies - x86, ARM, RISC-V, ASIC, and GPU or any combination thereof. We will make sure to have virtual machine-based data so that we have basis to compare to, which we will later use to work on improving efficiency of our Kubernetes scheduler. Our goal will be to try to answer to the following question - how to combine performance, workload priority, efficiency, and energy footprint to get the best possible “green” result on an infrastructure/data centre level, by using all available technologies. From the green computing perspective, it is crucial to be able to estimate and predict the power and consequently the energy of a data centre [4].

4. Related research

Katenbrink et al. presented the idea of Dynamic Scheduling for Seamless computing for Kubernetes or DYSCO [5], a methodology that uses Kubernetes to schedule applications between different nodes at runtime [5]. Their application of this methodology aims to solve the problem of movement of IoT devices, as Kubernetes itself doesn’t really consider the mobility of nodes [5] in multi-layer environments that have Edge, Fog and Cloud infrastructure layers. The reality is that not all the HPC environments have capability to do a fallback to cloud solution. Furthermore, there might be privacy problems involved, especially if we’re working in fields like molecular biology, stem cell research, AI-assisted facial or speech recognition, motion estimation for video surveillance etc. Load balancing workloads across multiple nodes is a very important aspect of container-based workloads. We can look at this problem from two different levels – how to load balance workloads across nodes, and how to load-balance traffic that’s coming to those nodes.

As we mentioned, Kubernetes' default workload load-balancing schedulers are based on Filtering and Scoring. These mechanisms are decent when we have more generic environments running similar types of workloads – for example, many containers running Apache web server, or PHP solution. If we're talking about traffic load balancing when using Kubernetes in larger environments, we will have to investigate additional solutions, ranging from HAproxy and nginx to hardware load balancers. Qingyang et al. explain the reason in their paper about how to design a multi-metric load balancer for Kubernetes, by stating that the built-in load balancing component has a few functions, only supports static load balancing strategies, and cannot adapt to complex needs [6]. They propose a solution that checks additional I/O metrics for network IO, CPU usage, memory usage and disk IO in real time with ability to check application running state, which is the basic idea behind load balancing on OSI Layer 4 or 7, depending on scenario and application used.

Scalability of Kubernetes-based solutions is very important for any kind of enterprise-level deployment. Previously, large-scale deployments went through a phase where workloads were installed on physical servers (in HPC space, often on supercomputers), and virtualized servers, by using an underlying hypervisor. Dewi at al. concluded that virtualizing using containers enables much flexibility for capacity management in a server [7]. Furthermore, their research indicates that implementation of scalability to the containers (on multiple nodes) reduces the CPU usage in a pod due to the distribution of loads [7], by running them across multiple nodes.

Flexibility and effectiveness of Kubernetes-based environment plays a key role in designing a Kubernetes-based data centre. Tesliuk et al. discussed the possibility of using MPI, HPC and data parallelism for HPC applications. Deployment of virtual MPI cluster allows us to run MPI software without any modification of the code [8]. They also note that detailed analysis of the performance of this approach is a subject of future research [8]. By using kube-openmpi [9], we have a standardized way of integrating Kubernetes with Open MPI, which is a good starting point for our research.

When designing HPC data centre and thinking about using container-based methodology, performance of Docker Swarm and Kubernetes-based solutions for HPC workloads is the one of the key aspects to evaluate. Cloud-based methodologies at scale (when using massive number of nodes) are problematic at best – we need to have enough resources for these nodes, which means multiple regions or cloud datacentres. That, in turn, means increase in latency, along with additional infrastructure complications. Let's use Microsoft Azure as an example. If we wanted to deploy one thousand nodes in Azure across five or six different regions, we would first need to connect those regions by using VPN gateways, software services that aren't necessarily consistent in bandwidth and latency. That's a big administrative overhead, as well, and if we need to move those workloads to other regions, it's only going to get worse, even without discussing the variable cost and other problems across different regions. Which is why local, high-speed networking in local datacentres (local infrastructure) is the best way to get low-latency, high-bandwidth network backend. Beltre at al. note that Kubernetes and Docker Swarm can achieve near bare metal performance over RDMA communication when high performance transports are enabled [10]. They also note that using Kubernetes presents overheads for several HPC applications over TCP/IP protocol [10]. But, when Infiniband is used as network backend, performance in within 1% of the bare metal performance both for Docker Swarm and Kubernetes in low complexity applications like HPL, while HPCG overhead drops from 13.15% to 10.31% [10].

Fairness of workload distribution is also an important characteristic of Kubernetes scheduler. By design, default Kubernetes schedulers don't have any available options here, if we discount node/pod affinity and anti-affinity capabilities. These enable us to do a static selection which nodes are going to be selected by Kubernetes scheduler. As this approach decreases efficiency and agility, using it might mean that we can't schedule a workload to be run on Kubernetes pod. According to Hamzeh et al., fairness is a missing point in Kubernetes [11]. They also note that some applications may require intensive resources such as CPU and memory that should be maximized to satisfy them [11], which can be done by using different fair allocation policies based on algorithms – Hamzeh et al. MLF-DRS [12], Hamzeh et al. FFMRA [13], and Ghodsi et al. DRF [14]. All very important points when designing a Kubernetes scheduler. Efficiency and power usage for a large HPC datacenter are very important from the design standpoint, as we live in the era of green computing. From datacenter standpoint, green computing principles can be used on multiple levels:

- Approach to building a data centre from architectural standpoint – for example, materials used, energy certifications etc.
- Approach to energy recovery, waste management and utilization – for example, are we only using grid power or have solar or wind-based power generation.
- Efficient design from the hardware standpoint – for example, trying to get either as much performance from a given power envelope or customizing our hardware design to fit a specific set of workloads to be as efficient as possible for those specific workloads.
- Data centre location – is it inland, underwater, or in higher altitude, as this will have significant impact on data centre design. For example, Microsoft conducted a series of tests using Northern Isles data centre underwater (Project Natick), which might define Microsoft's future data centre strategy.

Our approach will be to be as efficient as possible in terms of workload placement to be done in an environmentally responsible way. That means that we will not be searching just for outright performance – we will try to place workloads to be done with as less energy usage as possible. Guzek et al. proposed a holistic model for hypervisors in this respect, by dividing a system into three layers: physical node (server), container (VM), and task (application) [4]. Their research yielded two approaches – multi-stage linear regression approach and neural network approach.

By using the first approach, they were able to accurately estimate the power consumption in a virtualized system with acceptable temporary and cumulative errors [4]. Second approach introduced a lot more errors. This stems from the usage of neural networks, as they require a lot of time for training. As they state, research should be continued by adding more elements to their model, and to be used on a larger set of applications and machines [4].

The other aspect of achieving high efficiency comes from the platforms used. Maqbool et al. noted that ARM-based CPUs performed four times better in tests of the performance to power ration on a single core and 2.6 times better for tests on multiple cores [15]. In terms or memory performance, as expected, Intel CPUs performed significantly better in massively parallel applications, while ARM CPUs performed better in other I/O bound applications [15]. But one of the most important conclusions is related to other parts of the HPC system used for testing revealed the impact of network bandwidth, workload type, and messaging overhead on scalability, floating point performance, the performance of large-scale applications and energy-efficiency of the cluster [15]. This was also emphasized by Nam et al. They state that improving the QoS of latency-sensitive workloads is crucial to the users at a data centre [16].

5. Application of programmable, custom Kubernetes scheduler on HPC workloads

From a container workload perspective, we need to consider multiple different scenarios. In the following cases we will assume that GPU can in general case be replaces by other type of HW accelerator. Let's go through them:

- If the workload that we need to execute cannot be GPU-accelerated, we need to have capability to place those workloads on node(s) without GPUs. We don't want to waste CPU on a GPU-enabled node on workloads that cannot use the GPU.
- If the workload we need to execute can be GPU-accelerated, it needs to be placed on a GPU-enabled node to be done as fast and as efficiently as possible.
- If we have nodes with multiple different GPUs, we need to find the best way of using them (in terms of power, speed, and overall efficiency).
- If the workload that needs to be done is bound to a certain CPU architecture (for example, HPC application that's only available for ARM), it needs to be placed on an adequate host that meets the demands of that specific workload.
- If the workload that needs to be done isn't bound to a specific CPU architecture (for example, HPC application that's available both on x86 and ARM CPU architectures), it needs to place the workload on the node that's going to go through that workload in an efficient manner from an overall environment and efficiency perspective.
- If there's a chance for further optimization on the host-level post container deployment, migrating a container to another node could offer better efficiency.
- If there's a node failure, workload needs to be migrated to another node and either continued or restarted (depending on the HPC application that we're using).
- It needs to take power per kWh pricing as a variable when making decisions on workload placement.
- For ease of use, it would be best to use multi-architecture container images, a Docker image feature that allows the specification of the same container image for multiple CPU architectures [5].
- Therefore, a fair, efficiency-focused custom scheduler for Kubernetes needs to take all these scenarios into consideration and have the following characteristics:
- It needs to take the actual workload into account (whether it can be accelerated by using a GPU or other type of accelerator, if these devices support Docker or Podman containerization). Therefore, it needs to be able to distribute workloads based not only on capacity or CPU speed, but on node capability.
- It needs to have insert capability, so that we can insert workloads into the existing workload queue at will for some priority workloads.
- It needs to be mindful of the energy footprint as one of the key priorities.
- It needs to be able to learn about workload placement either directly or by using external service, and, if possible, interact with the underlying OS platform to redistribute workloads on node level.

This would give us „the best of both worlds" type of result – we will have the capability to insert any high-priority workload to the fastest nodes, but at the same time, be mindful of every other non-priority workload that can be placed on the best „performance-per-Watt" node selected by our scheduler. Kubernetes scheduler component is responsible for deploying pods and services in the suitable nodes [11]. Fastest nodes aren't just nodes with fastest CPU's – it's a terminology that we use to describe nodes where the workload should be placed to be fastest in terms of execution time. We mentioned the idea of placing workloads based on capabilities – let's call that capability-based placement. For example, if we have a HPC workload that can take extensive usage of NVIDIA GPU, then it makes sense to put that workload on a node that has a NVIDIA GPU and assign it to that specific workload.

6. Initial performance measurement and baselining

We must establish performance baselines so that we can make sure that our programmatic approach to Kubernetes scheduler gives real-life benefits. First step in that process is going to be introducing general and HPC-workload simulation benchmarks to measure baseline performance.

After that, next step will be cross-referencing these results with power usage data points and develop methodology that will take both data sets as inputs to decision-making process for Kubernetes scheduler. Generic Kubernetes schedulers use a two-step process to select a node for running a Kubernetes pod:

- Filtering – does a node have enough compute resources to handle the workload.
- Scoring – after filtering, scoring system is applied to remaining nodes to find the best fit for any given workload.

There are some attributes and methods that we could configure in generic scheduler by using Extension Points, Scheduling Profiles and Policies, but these are nowhere near fine enough to manage big environments like HPC-based data centres. By extension, that also means that we can't get finely grained workload-to-node assignments which can lead to much greater energy consumption, especially on a HPC data centre scale. Number of cores in HPC data centres are often in the millions, and numbers of servers running those cores are often in the hundreds of thousands. This represents a significant chance for optimization. These synthetic and HPC tests will include various cross-platform benchmarks like:

- sysbench tests for CPU (single and multi-core), and memory performance – a good general measurement of all-around CPU and memory I/O, by using prime number calculation (64-bit integers). Memory tests are sequential read/write tests.
- stress-ng tests for CPU (single and multi-core), and memory performance.
- HPCC benchmarks – a collection of benchmarking applications that consists of multiple tests Specifically, the suite is composed of several well-known computational kernels (STREAM, HPL, matrix multiply – DGEMM, parallel matrix transpose – PTRANS, FFT, RandomAccess, and bandwidth/latency tests – b_eff) that attempt to span high and low spatial and temporal locality space [1]. This is a standard HPC benchmark suite that has been used for years.
- HPCG (High Performance Conjugate Gradient) benchmarks – a second standard HPC benchmark suite that consists of multiple tests that are much more memory-bound than HPL/HPCC stack of benchmarks. HPCG is composed of computations and data-access patterns commonly found in scientific applications [2].

By running these benchmarks on our Kubernetes nodes, we can get a really good idea about the general level of performance we can expect from our nodes. We could also run all these tests in Docker/Podman containers, to get information how the physical performance any given Kubernetes node translates to container-based performance. That will also give us a good insight into overall compute overhead of the underlying OS. We can also use information from servers' onboard management hardware to get a rough estimate of how much power it uses, which we will turn into a more reliable power measurement method with time and additional data. We're going to use a shell script to run through all these tests on physical Kubernetes nodes. Output from that script is going to be CSV file which we can then use to do our analysis. Running these benchmarks in a container requires external storage for results, which we will achieve by mounting a NFS storage from network attached storage device. We're going to use the same basic shell script principle to get the same type of output to make our results directly comparable.

7. Preliminary test results

We created a small test environment with different x86 and ARM-based nodes. Purposefully, we chose different x86 nodes in terms of capabilities so that we can gather as much data as possible, as shown in the following table:

Hardware	**Kubernetes node specifications**			
	Node 1	***Node 2***	***Node 3***	***Node 4***
CPU type	2xXeon E5-2630L	2xXeon E5-2450L	2xXeon E5-2680	1xCortex A72 (ARM v8)
Core number /SMT core number	12/24	16/32	16/32	4/4
Base frequency	2 GHz	1.8 GHz	2.7 GHz	1.8 GHz
L1 cache (per core)	64 KB	64 KB	64 KB	80 KB
L2 cache	256 KB per core	256 KB per core	256 KB per core	1 MB shared (with GPU)
L3 cache (shared)	15 MB	20 MB	20 MB	-
Memory	96 GB	96 GB	96 GB	4 GB
Memory frequency	1333 MHz	1333 MHz	1333 MHz	3200 MHz
Average power consumption	220 W	230 W	250 W	13W

Table 1. Nodes used for preliminary testing

Let's now contrast that to testing results, using previously mentioned benchmarking tools:

Hardware	Kubernetes node performance			
	Node 1	***Node 2***	***Node 3***	***Node 4***
Sysbench single core (events per second)	643.89	593.16	815.52	1787.74
Sysbench multicore (events per second)	1299.85	1196.87	1630.47	3574.08
Sysbench memory (total time)	9.55366	10.0014	7.57572	10.0002
Stress-ng single core (bogo ops)	12258.4	11274.4	14836.4	1253
Stress-ng multi core (bogo ops)	24286.8	22398.0	29626.8	2134
HPL (Gflops)	2.01437	1.76892	2.47621	10.9701
RandomAccess (GUP/s)	0.0622563	0.0471568	0.0416779	0.0075283
FFT (Gflops)	1.99787	1.74458	2.06731	0.186364
STREAM (GB/s)	6.19918	5.35783	6.1504	0.537999
PTRANS (GB/s)	1.59295	1.35349	1.70072	0.130684
DGEMM (Gflops)	1.52678	1.32662	1.58446	4.19122
HPL time (s)	331.03	376.96	269.29	60.7851
HPCG (Gflops)	1.48279	1.37273	1.85292	0.100408

Table 2. Node performance

We used the same settings for all tests and limited the amount of available CPU cores for all tests on all servers to 4, so that we can have basis for per-CPU-core analysis. We also turned off all of the non-used CPUs (we only used cores from 0-3 on x86 platforms), and HyperThreading so that we can get a good estimate on power usage and performance. There are other factors that can influence efficiency and speed of HPC environments - we already mentioned low-latency and high-bandwidth networking as a must. HPC is increasingly faced with a QoS-sensitive computing demand, coming from that class of applications whose correctness depends on both performance and timing requirements, and the failure to meet either of them is critical [17]. Assigning network adapters in a direct fashion (PCI Express forwarding) or via SR-IOV could be the step in the right direction, which is the next step in Linux kernel development as not all of these features are supported if we're using containers as compute objects.

8. Conclusion and future work

If we take a deeper look at the preliminary test results, we can easily notice some trends. Although x86 CPU architecture has been the de-facto standard for decades now, its performance per Watt leaves a lot to be desired for. And although ARM's primary target isn't necessarily outright performance, even an inexpensive platform based on ARM A72 can give it a run for its money in certain use cases. Scaling that idea up to many nodes, coupled with other CPU architectures like RISC-V seems like a viable option for a high-performance, yet energy efficient HPC solution. Of course, we cannot discount x86 as its maturity and the scope of available hardware and software solutions still make it a platform to be used. Combining our preliminary test results with all the related scientific work that was previously done with relation to Kubernetes and HPC makes us confident that there are a lot of unexplored areas yet to be analysed. Therefore, focus of our future work should be on using all the measured performance parameters and expand the amount of performance testing to have as many data sets as possible for creating our much more efficient Kubernetes scheduler. We will also add more platforms and nodes to our environment – more ARM-based hardware and some RISC-V-based hardware, as well. This will help us gain even more understanding about performance scaling as we add more nodes to our environments. We will use all these tests to make scheduler as efficient as possible from the energy usage standpoint while keeping an eye on workload priority and distribution fairness. We will also have to introduce a methodology to do continuous power usage measurement. We can use the following data mechanisms for that:

- Remote management mechanisms like iLO (Integrated Lights Out), IPMI (Integrated Platform Management Interface) and others to get continuous stream of data from Out-of-band management technologies at our disposal.
- Software-based management utilities via hypervisor or OS-based tools (if available).
- PDU (Power Distribution Unit) remote management, which can give us much more insight and information about power usage over time.

This overall concept will require a fully orchestrated, programmatic approach to delivering containers. This is specifically what our Kubernetes scheduler needs to do for us. That also means that we must prepare a container image library for every HPC application and workload type, which requires time. Some of the latest research done by Zhou et al. clearly states that there are ways of integrating HPC schedulers and containers, which is why they developed a tool named Torque-Operator that bridges Kubernetes and TORQUE, one of the most commonly used HPC schedulers [18].

We can even go to greater lengths there and combine multiple container layers to create unified containers with multiple applications – such is the power of containers. This will help overcome major challenges implied by the HPC: performance, the total property cost (the cost of electric energy), programming, scalability and reliability [19]. Time involved in creating these customized containers will make it a lot easier for us to run all our HPC workloads in the future. So, it should be time well spent.

9. References

[1] https://www.researchgate.net/publication/238654737 (2005). University of Tennessee Knoxville, Oak Ridge National Laboratory, MITRE, High Performance Computing Center (HLRS), Information Sciences Institute, University of South California, MIT Lincoln Lab, IBM Austin, Lawrence Berkeley National Laboratory, University of Tsukuba, Introduction to the HPC Challenge Benchmark Suite, Accessed on: 2020-10-10

[2] Dogarra, J.; Heroux A. M. & Luszczek, P. (2016). High-performance conjugate-gradient benchmark: A new metric for ranking high-performance computing systems, The International Journal of High Performance Computing Applications Vol.30(I)3-10, DOI: 10.1177/1094342015593158

[3] Smith, J. & Hamilton, A. (2015). Massive affordable computing using ARM processors in high energy physics, Journal of Physics: Conference Series vol.608, IOP Publishing, DOI:10.1088/1742-6596/608/1/012001

[4] Guzek, M.; Varrette, S.; Plugaru, V.; Pecero, E.J. & Bouvry, P. (2013). A Holistic Model of the Performance and the Energy-Efficiency of Hypervisors in an HPC Environment, COST ICO804 European Conference on Energy Efficiency in Large Scale Distributed Systems, Volume 8046, Pages 133-152, DOI: 10.1007/978-3-642-40517-4_13

[5] Tatenbrink, A.; Seitz, A.; Mittermeimer, L; Mueller, H. & Bruegge, B. (2018). Dynamic Scheduling for Seamless Computing, IEEE 8th International Symposium on Cloud and Service Computing (SC2), IEEE, ISBN: 978-1-7281-0236-8, DOI: 10.1109/SC2.2018.00013

[6] Liu, Q.; Haihong, E. & Song, M. (2020). The Design of Multi-Metric Load Balancer for Kubernetes, Proceedings of the Fifth International Conference on Inventive Computation Technologies (ICICT-2020), IEEE, ISBN: 978-1-7281-4685-0 , DOI: 10.1109/ICICT48043.2020.9112373

[7] Dewi, L.P.; Noertjahyana, A.; Palit, H.N. & Yedutun, K. (2019). Server Scalability Using Kubernetes, The 2019 Technology Innovation Management and Engineering Science International Conference (TIMEs-iCON2019), ISBN:978-1-7281-3755-1, DOI: 10.1109/TIMES-iCON47539.2019.9024501

[8] Tesliuk, A.; Bobkov, S.; Ilyin, A.; Novikov, A.; Poyda, A. & Velikhov, V. (2019). Kubernetes container orchestration as a framework for flexible and effective scientific data analysis, 2019 Ivannikov Ispras Open Conference (ISPRAS), 67-71, ISBN:978-1-7281-6055-9, DOI: 10.1109/ISPRAS47671.2019.00016

[9] https://github.com/everpeace/kube-openmpi (2017). Omura, S., kube-openmpi [Accessed: 04-Jan-2021].

[10] Beltre, A.; Saha, P.; Govindaraju, M.; Younge, A.J. & Grant, R.E. (2019). Enabling HPC workloads on Cloud Infrastructure using Kubernetes Container Orchestration Mechanisms, 2019 IEEE/ACM Workshop on Containers and New Orchestration Paradigms for Isolated Environments in HPC (CANOPIE-HPC), 11-20, ISBN:978-1-7281-6028-3, DOI: 10.1109/CANOPIE-HPC49598.2019.00007

[11] Hamzeh, H.; Meacham, S. & Khan, K. (2019). A New Approach to Calculate Resource Limits with Fairness in Kubernetes, 2019 First International Conference on Digital Data Processing (DDP), 51-58., ISBN:978-1-7281-5363-6, DOI: 10.1109/DDP.2019.00020

[12] Hamzeh, H.; Meacham, S.; Virginas B. & Khan, K. (2019). MLF-DRS: A Multi-level Fair Resource Allocation Algorithm in Heterogeneous Cloud Computing Systems, 2019. IEEE 4th International Conference on Computers, Communications and Systems (ICCCS), ISBN:978-1-7281-1322-7, DOI: 10.1109/CCOMS.2019.8821774

[13] Hamzeh, H.; Meacham, S.; Khan, K.; Phalp, P. & Stefanidis, A. (2019). FFMRA: A Fully Fair Multi-Resource Allocation Algorithm in Cloud Environments, 2019 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation, 279-286, ISBN:978-1-7281-4034-6, DOI: 10.1109/SmartWorld-UIC-ATC-SCALCOM-IOP-SCI.2019.00091

[14] Ghodsi, A.; Zaharia, M.; Hindman, B.; Konwinski, A.; Shenker, S. & Stoica, I. (2011). Dominant Resource Fairness: Fair Allocation of Multiple Resource Types, Proceedings of the 8th USENIX conference on Networked systems design and implementation, 323-336.

[15] Maqbool, J.; Oh, S. & Fox, G.C. (2015). Evaluating ARM HPC Clusters for Scientific Workloads, Concurrency and Computation: Practice & Experience, Wiley, ISSN: 1532-0634.

[16] Nam, Y.; Kang, M.; Sung, H.; Kim, J. & Eom, H. (2016). Workload-aware resource management for software-defined compute , Cluster Computing, Vol. 19, Issue 3, 1555-1570., DOI: 10.1007/s10586-016-0613-6

[17] Fitch, J.; Agosta, G.; Ampletzer, P.; Alonso, D.A.; Cilardo, A. ; Fornaciari, W. ; Kovač, M.; Roudet, F. & Zoni, D. (2015). The MANGO FET-HPC Project: An Overview, 18th IEEE International Conference on Computational Science and Engineering, ISBN:978-1-4673-8297-7, DOI: 10.1109/CSE.2015.57

[18] Zhou, N.; Georgiou, Y.; Pospieszny, M.; Zhong, L.; Zhou, H.; Niethammer, C.; Pejak, B.; Marko, O.; Hoppe, D. (2021). Container orchestration on HPC systems through Kubernetes, Journal of Cloud Computing: Advances, Systems and Applications, DOI: 10.1186/s13677-021-00231-z

[19] Novacesku, F. (2011). Trends in the field of High Performance Computing and influence on HPC field, Annals of DAAAM for 2011 & Proceedings of the 22nd International DAAAM Symposium, Volume 22, No.1, ISSN 1726-9679.