THE TECHNOLOGY STACKS OF HIGH PERFORMANCE COMPUTING AND BIG DATA COMPUTING:

What they can learn from each other

ETP 4 HPC
EUROPEAN TECHNOLOGY PLATFORM FOR HIGH PERFORMANCE COMPUTING

BDV BIG DATA VALUE ASSOCIATION

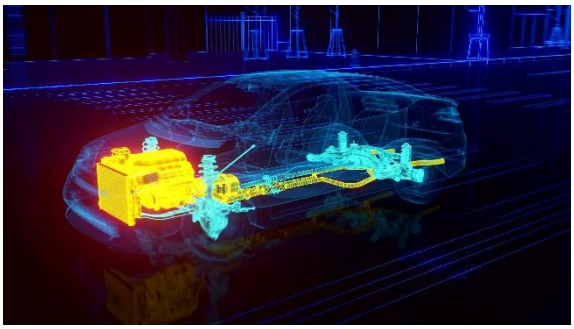
ABSTRACT

As some Big Data Computing (BDC) workloads are increasing in computational intensity (traditionally an HPC trait) and some High Performance Computing (HPC) workloads are stepping up data intensity (traditionally a BDC trait), there is a clear trend towards innovative approaches that will create significant value by adopting certain techniques of the other. This is akin to a form of "transfer learning" or "gene splicing" - where specific BDC techniques are infused into the HPC stack and vice-versa. This document is intended to inform the discussion regarding current strengths and differences between the (software and hardware) stacks of BDC and HPC, and how the current strengths of one stack may address a shortcoming/need in the other stack. This paper reflects an under-taking between two distinct ecosystems - the European associations for HPC (www.ETP4HPC.eu) and Big Data Value (www.BDVA.eu).

TABLE OF CONTENTS

1.0 INTRODUCTION

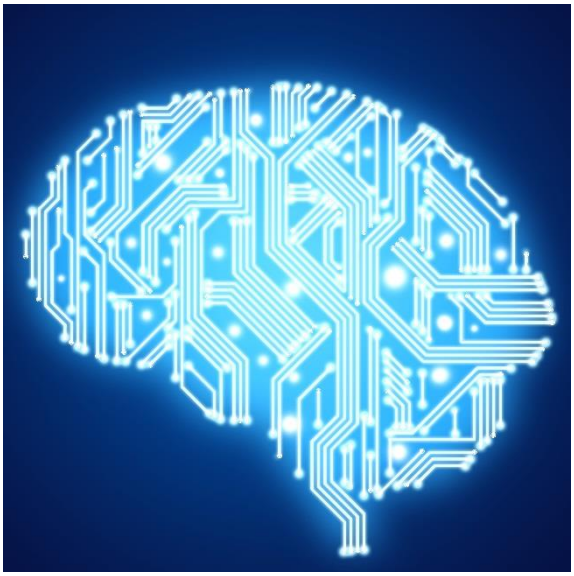
To aid consistent interpretability across High Performance Computing (HPC) and Big Data Computing (BDC) ecosystems, key concepts need to be explicitly explored upfront before addressing the potential of cross stack transfer learning opportunities. We begin by examining the top-level characteristics and objectives for both HPC and BDC stacks (and extending this to related workloads of Machine Learning (ML) and High Performance Data Analytics (HPDA)).



HPC (High Performance Computing) in this paper refers to modelling and simulation workloads from science, industry, public decision making and commerce perspectives that require extreme amounts of computation and a class of highly tuned stacks that provide scalable compute and local communication capabilities that far surpass hardware capabilities of a single server and can take on the afore mentioned workloads. Typically executed on ultra-fast, high-capacity "scale up" architecture. Classic HPC modelling and simulation seeks to represent the dynamics of a system based on a mathematical model for that system, which is usually derived from scientific or engineering laws. HPC focuses on interaction amongst parts of a system and the system as a whole. For example, weather modelling will take initial conditions (such as historical temperatures, estimates of energy input from the sun, and so on) as an input. Then, using a commonly accepted model about the causal interactions between these variables (captured in equations and code), it will produce forecasts of the weather in, e.g. five days. It is also possible to invert the process – here, the forecasts are compared with the actual weather, and conclusions are drawn about the original model assumptions. The relationship between modelling and simulation relates to level of abstraction and accuracy – abstraction decreases while accuracy increases moving from models to simulations[1]. Major HPC application areas include biochemistry (e.g. protein behaviours), chemistry (e.g. materials), physics (e.g. astrophysics), environment (e.g. weather), and industry (e.g. design and prototyping).

Big Data Computing (BDC) is the process of collecting, organising and analysing large sets of data (called Big Data). Big Data typically means datasets (both structured and unstructured) possessing data volume, velocity and variety characteristics, which are so large it is difficult to process using traditional database and software techniques. Big Data can be captured from any number of sources (e.g. web activity, business operations, customer contact centre, social media, mobile data, machine-to-machine data, and so on), then it is typically formatted, manipulated, stored and finally analysed to gain useful insights i.e. building a representation, query, or model that enables descriptive, interactive or predictive analysis for insights and value.

Machine Learning (ML) workloads (straddles the disciplines of Artificial Intelligence (AI) and Big Data Computing) are of particular interest regarding the intersection between HPC and BDC. ML uses computer-implementations of mathematical algorithms that can classify data, extract features or create insights. Traditional ML is most suitable where feature extraction[2] of data is tractable. A subfield within ML is the rapidly evolving Deep Learning (DL) (algorithms inspired by neural networks with multiple layers of neurons that learn successively complex representations) to implicitly recognise properties of sounds, images, and other data in a training phase, and then make predictions on new data sets (i.e. inferencing). DL is most useful in cases where feature extraction using classical machine learning is intractable, or where the underlying features are not a priori known. The rise in popularity of DL is due to the explosion in data generation/storage and vast improvements in available compute performance. Major application areas include statistical analysis, search, planning, classification and clustering, imaging and vision, natural language processing, and machine reasoning.



Another rapidly evolving field that increasingly draws from the domains of HPC and BDC is **High Performance Data Analytics (HPDA)** – which pursues extreme data analytics at scale and in a manner that is sensitive to the trade-offs between the timely consumption of computational resources and the practical value of the predictions obtained. Demands for HPDA originate from the need for extremely fast analysis results (e.g. real-time high-frequency analysis), extreme problem complexity requiring high capability analytics (e.g. those found in large-scale scientific research and industrial settings), and where patterns or insights are of an extremely valuable nature (e.g. economic, scientific or social).

The remainder of this paper deals primarily with the domains of HPC and Big Data Computing (Machine Learning and High Performance Analytics representing states between both are highlighted when appropriate). The paper is structured as follows: overview of respective stacks, workloads characteristics of each stack, stack segregation and profiling, cross stack transfer learnings, and conclusions. (The needs of respective stacks will be broader than what this paper covers, as this paper confines itself to how the current capability strengths of one stack may address a shortcoming/need in the other stack).

2.0 STACK OVERVIEWS

Be it on a leadership class supercomputer, small institutional cluster, or in the Cloud – HPC and Big Data Computing (BDC) stacks traditionally have distinct features, characteristics and capabilities.

HPC stacks are designed for modelling and simulation workloads which are compute intense, and focus on the interaction amongst parts of a system and the system as a whole. Typical application areas include physics, biology, chemistry, and Computer Aided Engineering (CAE). HPC targets extremely large amounts of compute operations, which often operate on large sets of data and are executed in parallel on potentially high number of compute nodes. Data is usually partitioned between the nodes, and any data sharing is effected by message exchanges over a high-speed interconnect between the nodes. HPC applications are usually iterative and closely coupled – the underlying mathematical models do cause dependencies between the data blocks owned by different nodes, and this requires frequent communication of data updates (usually of lower-dimensional parts of blocks) between the nodes. As a consequence, the interconnect fabric has to be very fast in terms of bandwidth and latency, essentially turning the entire set of nodes into one single "supercomputer". This requires expensive high performance hardware, i.e. high floating-point processing power and very fast (in terms of both latency and bandwidth) network fabric between the nodes. A HPC application operates as a closely coupled and synchronised over many nodes (up to hundreds of thousands) and accesses the data on a storage entity that is attached via the fast network to all the nodes. In effect, such applications use large parts of the system in "exclusive mode".

> HPC APPLICATIONS ARE USUALLY ITERATIVE AND CLOSELY COUPLED

Big Data Computing stacks are designed for analytics workloads which are data intense, and focus on inferring new insights from big data sets. Typical application areas include search, data streaming, data preconditioning, and pattern recognition. In the case of a Hadoop-type architecture[3] (which is one of the more prevalent ecosystems in Big Data Computing), the data set can be divided into multiple independent sub-problems (i.e. "embarrassingly parallel problems"[4]) and those sub-problems can be solved by multiple "simple" nodes without need for significant communication between processes during the "map" step. Only the "reduce" step requires bringing the results of parallelised parts together. As the problem size increases, the number of the sub-problems increases accordingly. And as the number of sub-problems grows, the number of simple nodes to solve them also rises. For these architectures, the power lies in having a huge number of relatively simple nodes (rather than highly tuned nodes as for HPC), which do not have to be tightly coupled. Several applications (or, *instances* of the same application) run simultaneously on multiple nodes (i.e. opposite to HPC where a *single* application uses all the nodes in the cluster). Common practices today highlight that HPC uses Batch Queuing system while BDC uses interactive python interfaces – although this difference is diminishing with introduction of newer HPC workloads.

> THE DATA SET CAN BE DIVIDED INTO MULTIPLE INDEPENDENT SUB-PROBLEMS

So in case of HPC workloads we are deploying a single large supercomputer to solve the entire problem. Adding more nodes for running the same problem ("strong scaling") will initially reduce runtime, yet at a diminishing rate due to the relative increase in communication and coordination overheads, and will later

even increase runtime again. To achieve good strong scaling, the performance of the interconnect fabric also has to be improved. In the second case (i.e. BDC), just adding more simple modes will reduce the amount of work per node, and since much less communication is required between the nodes, also continue to reduce the runtime. Figure 1 illustrates these conceptual differences. High Performance Data Analytics (HPDA) and Machine Learning (ML) inference will look largely like conventional BDC "on steroids", that is running on large numbers of fast nodes. While current practice in ML training is to use multiple, interconnected accelerators in a single node. For the latter, strong scaling across multiple nodes is a R&D topic, which will result in HPC-like closely coupled training workloads, most likely each running on comparatively small numbers of nodes.

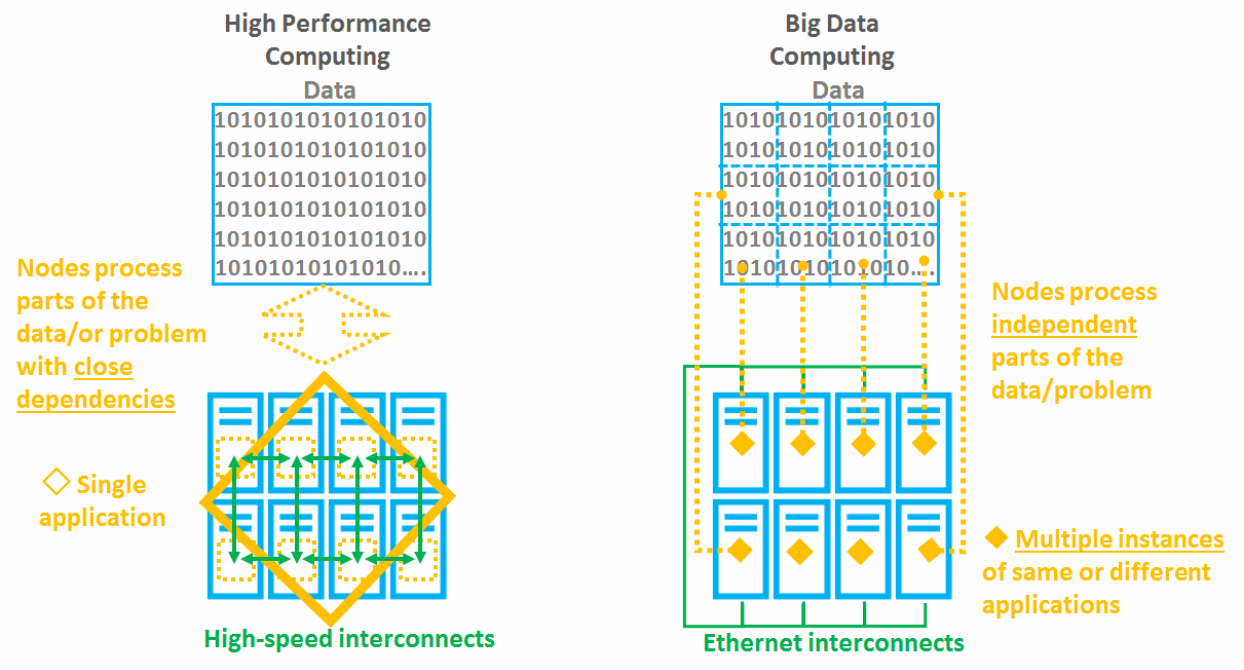


Figure 1: Overviews of HPC and Big Data Computing (Hadoop-type Cloud) architectures

3.0 STACK WORKLOADS

So far we have examined rather high-level differences between how HPC and Big Data Computing stacks are architected. A closer analysis of the respective application/workload requirements teases out further differences, including

- **Interconnect sensitivity:** Some applications require very fast interconnects[5] i.e. low latency and high throughput (e.g. Infiniband-class fabrics or High-Performance Ethernet). To minimise software overheads, these interconnects often leverage user-space[6] communication pathways that bypass the Operating System kernel and avoid context switches and buffer copies. This makes the use of a Cloud-type architecture very difficult because most virtualisation schemes do not support such a "kernel bypass". If high-performance interconnects are not available, the application may run slowly and get no performance gain when adding more nodes i.e. suffers from poor scalability. There are applications/workloads that are highly scalable or "embarrassingly parallel" (e.g. image rendering) and there are those that are difficult to scale, which we will call "interconnect sensitive" (e.g. computational mechanics). HPC workloads that have moved to Big Data Cloud-type stacks are typically embarrassingly parallel i.e. do not communicate frequently or with high data volumes between threads and cores, and therefore do not require a high-performance interconnect.
- **I/O sensitivity:** I/O-sensitive applications without a very fast I/O subsystem will run slowly because of storage bottlenecks, i.e. because computation will be waiting to read or write data. To avoid these bottlenecks, technologies such as optimised Networked Attached Storage - NAS, distributed file systems, parallel file systems, or fast storage media such as solid-state drives (SSDs), can drastically increase the I/O bandwidth of computing nodes. A significant difference between HPC and Hadoop-type Big Data Computing stacks (Map/Reduce, Spark, Flink or other) is that HPC codes tend to generate significant amounts of results data that have to be stored without impeding the computation. This holds in particular for applications simulating the time-dependent behaviour of a complex phenomenon (like the weather) or artefact (like a rocket engine). I/O subsystems for large HPC computers are usually designed to support such use cases. In contrast, I/O subsystems for Big Data Computing stacks are usually designed to ingest the highest volumes, variety and velocity of data, and depending on the exact application, store significantly smaller results relatively to HPC I/O subsystems i.e. HPC generates relatively larger volumes of output data (i.e. insights) relative to its data inputs, whilst BDC generates relatively smaller volumes of output data (i.e. insights) relatively to its data inputs.
- **Compute sensitivity:** Specialised hardware, i.e. performance accelerators, can give a significant boost to certain workloads, depending on the algorithms used. Optimised libraries are often required for the workloads to fully leverage hardware capabilities and performance. For instance, the dominance of dense linear algebra kernels plus the opportunity to use lower precision representation and arithmetic is the reason why accelerator hardware is extremely useful for Deep Learning training workloads. A typical example is the use of multiple general-purpose GPU (GPGPU) accelerators in a single server node to reduce training time to acceptable levels. Overall, the class of problems being addressed is that of very high-order optimisation problems – it is thus natural that Deep Learning workloads have entered the realm of high-performance computing. That is, the forefront of Deep Learning research is now looking at scaling Deep Learning training out across nodes – an approach that uses a core HPC discipline (i.e. scalable dense linear algebra).

Both HPC and Big Data Computing stack workloads can be affected by load imbalances caused by a variety of reasons. For example, differences in computational or I/O load across nodes, contention and congestion in the use of shared resources (like network links or storage targets), and different performance characteristics across nodes. Such imbalances can severely limit workload scalability – i.e. performance does no longer increase commensurate to the number of nodes used by an application.

We can observe that HPC and Big Data Computing workload types have affinity towards the capabilities offered by discrete stack architectures and hardware. Table 1 presents a summary, and is extended to include ML and HPDA workloads.

Table 1: Summary of Workload Sensitivities to their Underlying Resources

	Classical HPC Workload	Typical Big Data Workload	ML Workload	HPDA Workload
Interconnect sensitivity	High [+]	Depends [+/-]	Depends [+/-]	High [+]
IO sensitivity	Depends [+/-]	Depends [+/-]	Depends [+/-]	High [+]
Compute Sensitivity	High [+]	Low [-]	High [+]	High [+]

To appreciate this in more detail, we must dive deeper into the individual characteristics and composition of these respective stacks.

4.0 STACK PROFILES

Figure 2 offers a more fine grained exploration of the respective stacks – presenting a disaggregated profile view for the Supercomputing, Big Data Computing and Deep Learning stacks. We use the term Supercomputing here to denote a specific implementation of HPC, however, many aspects will also apply to institutional clusters. Deep Learning is a Big Data Computing workload that readily stands to benefit from HPC's experience due to its reliance on (dense) linear algebra, hence we examine it in more detail at this point. Deep Learning (aka artificial neural networks) algorithms typically scale well to produce increasingly better results where larger models can be fed with more data (i.e. Big Data), this in turn requires more computation to train (i.e. more compute is required) – typically using hand-crafted tuning of deep neural networks.

High Performance Data Analytics (HPDA) is not represented in Figure 2 – however architecture-wise, it is close to the BDC column with the Server and Network layers from the HPC column and potentially selected, more highly performant communication and I/O services. This "breed" of stack can bring performance and efficiency improvements for large-scale BDC problems, and it could open the door to tackling data analytics problems with inherent dependencies between nodes.

		Supercomputing (SC)	Deep Learning (DL)	Big Data Computing (BDC)
Apps...	Boundary Interaction Services	**In-house, commercial & OSS applications** [e.g. Paraview], **Remote desktop** [e.g. Virtual Network Computing (VNC)], **Secure Sockets Layer** [e.g. SSL certificates]	**Framework-dependent applications** [e.g. NLP, voice, image], **Web mechanisms** [e.g. Google & Amazon Web Services], **Secure Sockets Layer** [e.g. SSL certificates]	**Framework-dependent applications** [e.g. 2/3/4-D], **Secure Sockets Layer** [e.g. SSL certificates]
Middleware & MGMT	Processing Services	**Domain specific frameworks** [e.g. PETSc], **Batch processing of large tightly coordinated parallel jobs** [100s – 10000s of processes communicating frequently with each other]	**DNN training & inference frameworks** [e.g. Caffe, Tensorflow, Theano, Neon, Torch], **DNN numerical libraries** [e.g. dense LA]	**Machine Learning** (traditional) [e.g. Mahout, Scikit-learn, BigDL], **Analytics / Statistics** [e.g. Python, ROOT, R, Matlab, SAS, SPSS, Sci-Py], **Iterative** [e.g. Apache Hama], **Interactive** [e.g. Dremel, Drill, Tez, Impala, Shark, Presto, BlinkDB, Spark], **Batch / Map Reduce** [e.g. MapReduce, YARN, Sqoop, Spark], **Real-time / streaming** [e.g. Flink, YARN, Druid, Pinot, Storm, Samza, Spark]
	Model / Information Management Services	**Data Storage**: Parallel File Systems [e.g. Lustre, GPFS, BeeGFS, PanFS, PVFS], **I/O libraries** [e.g. HDF5, PnetCDF, ADIOS]	**Data Storage** [e.g. HDFS, Hbase, Amazon S3, GlusterFS, Cassandra, MongoBD, Hana, Vora]	**Serialization** [e.g. Avro], **Meta Data** [e.g. HCatalog], **Data Ingestion & Integration** [e.g. Flume, Sqoop, Apache Nifi, Elastic Logstash, Kafka, Talend, Pentaho], **Data Storage** [e.g. HDFS, Hbase, Amazon S3, GlusterFS, Cassandra, MongoBD, Hana, Vora], **Cluster Mgmt** [e.g. YARN, MESO]
	Communication Services	**Messaging & Coordination** [e.g. MPI/PGAS, direct fabric access], **Threading** [e.g. OpenMP, task-based models]	**Messaging & Coordination** [e.g. Machine Learning Scaling Library (MLSL)]	**Messaging** [e.g. Apache Kafka (streaming)]
	Workflow / Task Services	**Conventional compiled languages** [e.g. C/C++/Fortran], **Scripting languages** [e.g. Python, Julia,]	**Scripting languages** [e.g. Python]	**Workflow & Scheduling** [e.g. Oozie], **Scripting languages** [e.g. Keras, Mocha, Pig, Hive, JAQL, Python, Java, Scala]
System SW	System Management & Security Services	**Domain numerical libraries** [e.g. PETSc, ScaLAPACK, BLAS, FFTW, ...], **Performance & debugging** [e.g. DDT, Vtune, Vampir], **Accelerator APIs** [e.g. CUDA, OpenCL, OpenACC] **Data Protection** [e.g. System AAA, OS/PFS file access control] **Batch scheduling** [e.g. SLURM], **Cluster management** [e.g. OpenHPC], **Container Virtualization** [e.g. Docker], **Operating System** [e.g. Linux OS Variant]	**Batching for training** [built into DL frameworks], **Reduced precision** [e.g. Inference engines], **Load distribution layer** [e.g. Round robin/load balancing for inference], **Accelerator APIs** [e.g. CUDA, OpenCL], **Hardware Optimization Libraries** [e.g. cuDNN, MKL-DNN, etc.] **LA numerical libraries** [e.g. BLAS, LAPACK, etc] **Virtualisation** [e.g. Dockers, Kubernetes, VMware, Xen, KVM, HyperX], **Operating System** [e.g. Linux (RedHat, Ubuntu, etc.), Windows]	**Distributed Coordination** [e.g. ZooKeeper, Chubby, Paxos], **Provisioning, Managing & Monitoring** [e.g. Ambari, Whirr, BigTop, Chukwa], **SVM systems** [e.g. Google Sofia, libSVM, svm-py, ...], **Hardware Optimization Libraries** [e.g. DAAL, DPDK, MKL, etc.] **Virtualisation** [e.g. Dockers, Kubernetes, VMware, Xen, KVM, HyperX], **Operating System** [e.g. Linux (RedHat, Ubuntu, etc.), Windows]
Hardware	Infrastructure	**Local storage** [e.g. Storage & I/O nodes, NAS] **Servers** [e.g. CPU & Memory [Gen Purpose CPU nodes, GPUs, FPGAs] **Network** [e.g. Infiniband & OPA fabrics]	**Local storage** [e.g. Local storage or NAS/SAN] **Servers** [e.g. CPU & Memory [Gen Purpose CPU + GPU/FPGA, TPU] **Network** [e.g. Ethernet]	**Local storage** [e.g. Direct attached Storage] **Servers** [e.g. CPU & Memory, [Gen Purpose CPU hyper- convergent nodes] **Network** [e.g. Ethernet fabrics]

Figure 2: Disaggregated Stack Profiles (Note: some applications serve multiple functions, for illustration simplicity they were assigned to their dominant function)

Using Figure 2 as a backdrop, some notable observations which can be illuminated across the three stacks include:

4.1 Supercomputing

MINIMUM TIME-TO-SOLUTION

Traditionally designed for *minimum time-to-solution* across a wide range of computationally intensive simulation tasks in various fields e.g. climate modelling, physics simulation, molecular modelling, structural modelling and so forth. Design principles for infrastructure and software are typically optimised for performance (i.e. FLOP/s[7]) first – (i.e. devotes most of its execution time to computation), then for minimal cost (e.g. energy management – FLOP/s per watt).

- **Communication:** Applications are usually iterative (i.e. repeating computation and communication patterns) and require significant communication and coordination between nodes. For performance reasons, this necessitates the use of scalable and high performance (latency and bandwidth) interconnect fabrics. Also, user-space access to the interconnect fabric is important for performance.
- **Compute:** Use of dedicated compute engines (processing mostly floating point calculations) often with accelerators attached.
- **Storage:** HPC storage is often designed for a specific application set and user base. This requires use of dedicated parallel file system middleware (Lustre, GPFS, etc) conforming to POSIX semantics and dedicated file servers. I/O libraries are used to encapsulate data formats and their organisation. HPC relies on System Authentication Authorisation Accounting (AAA) and protections implemented by the parallel file systems (PFS) and operating system (OS) layers, e.g. file access controls. HPC applications partition the data with the amount of "shared data" being relatively small, and storage systems are optimised to quickly provide access to each node's block of data.
- **OS:** Linux is the most common OS in use, with container virtualisation gaining presence. To minimise latency and maximise bandwidth, HPC applications often need to bypass the OS kernel and communicate directly with remote user processes. Optimised HPC libraries and applications work closely with a given hardware architecture to achieve maximum performance, thus requiring specific OS drivers and hardware support. (This is in contrast to Big Data Computing applications which tend to rely on a middleware layer i.e. tend to be agnostic of the details of the underlying system, so long as performance is - relatively speaking - satisfactory). For the most part, VM/Hypervisor style virtualisation is mostly not considered – due to concerns about performance loss by the extra levels of indirection and software involved.
- **Scheduling & Workflows:** Most HPC systems use a batch scheduler to administer use of shared, limited resources and ensure high system throughput. User jobs must wait until the required set of resources becomes available (resources are not directly managed by the user). This requires specific cluster management techniques that handle creation, provisioning and management of large system partitions (from tens to hundreds-of-thousands of nodes) used to run individual HPC applications. (This is a significant difference to "scheduling" in Big Data Computing and Deep Learning – makes it more difficult to complete interactive work. HPC has limited hardware redundancy; if nodes supporting a HPC application fail – the application eventually fails. Therefore, applications need to periodically save their application state (e.g. check-pointing) to guard against losing a lot of work.) This is very different to

"redundancy" for Big Data Computing applications (see '*Scheduling & Workflows*' in following BDC section).

- **Processing:** HPC systems run tightly coupled applications on hundreds to tens-of-thousands of nodes in exclusive mode, using compiled languages, numerical libraries, accelerator APIs[8], threading and high-performance messaging (MPI, GASPI) layers - scripting languages like Python are gaining presence. There is limited use of higher level frameworks and IDEs[9] – a canonical example is PETSc for PDEs and ODEs. Applications are typically compiled into binaries and distributed and executed in a classical way, i.e. started and controlled by the OS on a constant set of resources. Accessing HPC applications via Cloud interfaces is emerging.
- **Analytics:** analysis of input and of results data makes use of statistical and domain-specific, descriptive data analysis, plus performance and fault diagnostics.
- **Interaction Services:** HPC workloads use advanced 3D, interactive visualisation for results presentation and computational steering[10]. Outside of this, the defaults are methods like remote desktop and SSL. Uptake of Web/Cloud access methods remains limited.

4.2 Big Data Computing

ECONOMICALLY INGEST, STORE AND ANALYSE

Big Data Computing stacks are traditionally designed to *economically* ingest, store and analyse massive volumes and varieties of data at speed (i.e. velocity). Design principles for infrastructure and software are typically optimised for cost effectiveness (i.e. storage IOP/s[11]/Euro) first - i.e. devotes most of its processing time to I/O and manipulation of data, then for maximum performance (e.g. data-ingestion, -processing, -persistence, -analytics). For the most part, Big Data Computing stacks are generally run on a Cloud type environment, therefore this paper examines it from that perspective.

- **Communication:** Applications typically are non-iterative, requiring minimal communication or coordination between nodes (during the Map phase of a Map/Reduce application). Ethernet use is prevalent, being generally viewed as cost efficient and providing "satisfactory" performance for most workloads.
- **Compute:** Nodes (processing mostly integer[12] or byte/string operations[13]) are "hyper-convergent" i.e. one kind of node fulfils all functions (compute, data storage, access) and accelerators can include field-programmable gate array (FPGA) processors (to accelerate a simple, repetitive task, like pattern matching).
- **Storage:** BDC uses a multitude of storage systems, including distributed file systems/parallel file systems, key/value stores, object stores, on hyper-convergent or dedicated infrastructure. Two application patterns are prevalent, namely Map/Reduce and Data Streaming, with frameworks for each.
- **OS:** Virtualisation (hypervisor and container virtualisation) and Virtual Machine Monitors (VMMs) are prevalent. In this environment, more resources are created by adding (if the underlying hardware resources are already present) more virtual machines (OS instances). Users can design personalised software environments and scale their "instances" to suit their needs. Instances can be easily moved to and from *similar* Cloud setups. The user, for the most part, does not care (or control) on which exact

hardware their Big Data Computing applications run on. (This is a contrast to HPC workloads, which is typically *tuned* to run on specific hardware architectures.)

- **Scheduling & Workflows:** Big Data Computing running in the Cloud offers almost instantaneous access to data or compute resources (whereas in an HPC system, the queuing system might impose a significant wait). Big Data Computing applications are usually complex workflows, therefore distributed coordination is important. Orchestration and resource management/scheduling is fine-grained (at a level of nodes, i.e. not in large partitions as in the case of HPC). Failing nodes in this (Map/Reduce) context can simply re-run the failed tasks on replacement nodes. (This is a significant difference to how HPC redundancy is handled. C++ plays a role, but scripting or interpreted languages are widely used.)
- **Processing:** BDC typically runs large workflows (often expressed as task graphs[14]), with each step running on (part of) a node. Users are guaranteed a certain *minimal* level of performance according to a Service Level Agreement (SLA).
- **Analytics:** In Big Data Computing, the data analytics leverages the full spectrum of statistical (averages, trends, correlations, causalities, etc.) and Machine Learning (clustering, modelling, decision trees and rule systems, graph analytics[15], spatio-temporal[16]) analysis methods. Machine Learning (outside of Deep Learning, see next section) is still the predominant usage model.
- **Interaction services:** Users of Big Data Computing applications mainly rely on service-based techniques to interact with Cloud infrastructures. Algorithms typically produce numerical output, such as a classification or score, and can utilise interactive 2D and 3D visualisation techniques for an exploration of results.

4.3 Deep Learning

EXPEDITE TRAINING AND INFERENCE

Traditionally designed to *expedite training and inference* of models that will minimise training errors when applied to test/validation datasets. Design principles for infrastructure and software are typically optimised for reduced-precision calculations (floating or fixed point for training, floating point or integer for inference), and storage IOP/s for performing irregular accesses in storage when working with unstructured data. More recently, distributed scaling of Deep Learning training performance is aiming for network performance to match the Flop/s available on the nodes.

- **Communications:** Fabric is predominantly Ethernet (with current R&D looking at scale-out over high-performance fabrics for model training[17]).
- **Compute:** Use of dedicated compute nodes with many (reduced-precision) accelerators attached, integrated into a flexible data centre architecture - similar to Big Data Computing today yet there is significant ongoing research into using multiple nodes and fast inter-node communication. Parallelism and scalability (as well as floating-point performance) are key to finding the model that accurately represents the training data quickly with a low error. For scale-out, Deep Learning is investigating HPC-style interconnects.
- **Storage:** Data stores are typically the same as for Big Data Computing discussed previously.
- **OS:** Usually same as for Big Data Computing, however efficient user-space access to accelerators is key.

- **Scheduling & Workflows:** Using the same techniques as for Big Data Computing, with Deep Learning training and inference embedded in workflows. At a lower level, Deep Learning inference uses load balancers to distribute queries across nodes in a uniform way, and Deep Learning training tightly controls the progress of the training across the assigned resources. Deep Learning inference[18] is a step in a large Cloud or Big Data workflow, and training can be run stand-alone or also as part of a workflow.
- **Processing:** Deep Learning training/inference steps are typically based on a multitude of higher-level frameworks (with a rather high frequency of change) e.g. Caffe, Tensorflow, Theano as some of the more prevalent ones (at the time of writing). Use of numerical libraries (such as DGEMM/SGEMM/IGEMM) is key to achieve high compute performance, accelerating gradient descent and "tensor" data manipulation. Numerical algorithms like Fast Fourier Transforms (FFTs)[19] will likely become more important in the future. Training runs computationally intensive codes on "fat" nodes (i.e. lots of cores, disk, and memory), and is looking to scale this o a moderate number (hundreds) of nodes. Domain-specific messaging layers are emerging (Intel Machine Learning Scaling Library, etc.). Inferencing uses much less compute and communication, and is regularly performed at reduced precision (FP16 or INT16, 8-bit, sometimes even less).
- **Analytics:** Algorithms are self-directed (for the most part) on the data analysis once put in to production, using artificial neural networks that pass data through many processing layers to interpret data features and relationships i.e. the many hidden or computational layers between the input neurons, where data is presented for training or inference, and the output neuron layer where the numerical inference results can be read.
- **Interaction services:** Deep learning outputs include identifying patterns, recognising objects, understanding concepts, natural-language processing, etc., and uses mostly 2D visualisation in the frameworks and Web/Cloud access mechanisms (internally most frameworks use object orientated languages - like python).

5.0 CROSS STACK TRANSFER LEARNINGS

The prior sections till now provide an appreciation of the main differences and similarities of the respective stacks, this provides necessary context as we now consider how the capabilities present in either HPC or Big Data Computing stacks may benefit the other. It is worth restating at this point that the needs of each stack will be broader than what this paper covers, as this paper confines itself to how the current capability strengths of one stack may address a shortcoming/need in the other stack.

5.1 What Big Data Computing can Learn from HPC

Big Data Computing applications are expected to move towards more compute-intensive algorithms to reap deeper insights across descriptive (explaining what is happening), diagnostics (explaining why it happened), prognostics (predicting what can happen) and prescriptive (proactive handling) analysis. HPC capabilities are expected to be of assistance to faster decision making across more complex data sets.

As already mentioned, Deep Learning and automated deep neural network design creation are workloads that readily stand to benefit from HPC's experience in optimising and parallelising difficult optimisation algorithms problems[20]. Major requirements include highly scalable performance, high memory bandwidth, low power consumption, and excellent reduced precision arithmetic performance.

Some specific challenges include:

- **Fast interconnects:** more bandwidth within the node (PCI Express evolution or alternatives), faster communication between nodes for both local and global communication schemes, in particular for the distributed gradient descent at the heart of Deep Learning training.
- **More efficient algorithms for linear algebra:** model compression and moving from dense matrix to sparse data structures leveraging sparse linear algebra (matrix-matrix and matrix-vector arithmetic).
- **Low precision data representation and arithmetic:** Deep Learning algorithms usually work well at low precision (training for 16-bit floating point, inference for even lower precision up to 8-bit and below), depending on the exact network. Today's requisite computational resources are clusters whose nodes are populated with a sufficient number of accelerators. These provide the needed performance while keeping power consumption low, and will also reduce the amount of data to be communicated between nodes. A challenge is to know a priori how small the key data structures can be for the specific network to continue working.
- **Introduction of new hardware capabilities:** For example, FPGA or dedicated Deep Learning processors will require optimisation of software layers. For non-van-Neumann computing architectures (like Quantum or Neuromorphic), further research is required to evaluate applicability.

- **Efficient distributed data access (using parallel data stores):** modifying Deep Learning algorithms to reduce the total amount of communication between nodes in a scale out cluster.
- **High energy efficiency:** it highly likely that research results achieved in the HPC ecosystem will be beneficial e.g. the use of minimal precision and compression for the recursive gradient descent (or of less communication-intense communication) methods in Deep Learning.
- **Multi-level, tiered architectures:** integration between accelerators, CPUs, high-performance fabrics and I/O that combines best-of-breed technical capabilities.
- **Hybrid architectures:** creating the best-matching hybrid architecture to optimise - at scale - across training and inference to serve the most extreme/challenging data analytics scenarios, and be appropriate for large-scale system deployments to synergistically utilise centre (HPC) and edge (IoT/Cyber-physical Systems) stacks.

5.2 What HPC can Learn from Big Data Computing

HPC is now generating models of unprecedented realism and accuracy. At the most extreme, these models represent the real world using trillions of discrete degrees of freedom, which require huge memory and compute performance on extremely large (typically scale-out) systems. These simulations generate enormous amounts of output data. Researchers need to apply advanced and highly complex analytics and processing (including visualisation) to this data to generate insights, which means that off-loading to remote platforms is simply not an option. Thus, data analytics needs to take place in-situ, and perhaps in close coordination with tightly coupled synergistic computing platforms (e.g. visualisation engines). These investigations will have important ecosystem benefits for multi-scale, multi-physics coupled applications, where instances are running on tens to hundreds-of-thousands of nodes.

Therefore, analytics is expected to become a fully-fledged software component of the HPC ecosystem to process the massive results of large numerical simulations or to feed numerical models with complex data produced by scientific and industrial instruments/systems (e.g. telescope, satellite, sequencers, particle accelerators, etc) or by large scale systems of systems (e.g. edge devices, smart sensors, IoT devices, cyber-physical systems, etc).

In addition, HPC simulations could profit significantly from iterative refinements of their underlying models effected by advanced data analytics tools and machine learning techniques, e.g. by accelerated convergence. HPC can also benefit from Big Data management approaches, especially in the case of dynamic scenarios (Big Data Computing is much more flexible with the notions of data at rest, data on move, data in change).

Some specific challenges include:

- **Data stream processing:** Streaming capabilities are becoming increasingly important for scientific and industrial HPC applications (e.g. CERN's Large Hadron Collider (LHC), Square Kilometre Array (SKA) project, astrophysics, physical simulations, digital twins, etc) supporting important needs such as the ability to act on incoming data and computational steering. For example, supporting the processing of high data rate streams, e.g. predictions and outlier detection algorithms on it, while running larger models in batch mode on the entire dataset, is a challenging task. Coupling data streams produced by such experiments to computational HPC capabilities is an important challenge, and Big Data Computing's

near real-time processing architectures and stream processing capabilities hold promise i.e. to rapidly analyse high-bandwidth, high-throughput streaming data. However, they require enhancements across infrastructure (e.g. delivery guarantees, low latencies, varying data rates, storage for flexible streaming and batch processing), abstractions (e.g. decouple application concerns from streaming infrastructures) and applications (e.g. adaptation of batch algorithms for inclusion of data steaming into their current state) to accommodate the massive bandwidth requirements of these scientific and industrial HPC applications.

- **In situ/in transit processing:** Traditionally, in the HPC area, datasets resulting from scientific simulations are typically shipped to some auxiliary post-processing platforms for offline visualisation, processing and analysis, which becomes more and more costly in terms of storage requirements as data volumes grow. In situ processing is a more efficient alternative, allowing data visualisation and analysis to happen online, as data is generated by the simulations, thus reducing the volume of refined data to be stored and in consequence saving energy. Big Data management approaches include in situ processing capabilities that are of particular interest for addressing this challenge, i.e. by bringing the computation to where data is located. Furthermore, exploring how stream-processing architectures and tools could combine with in situ/in transit processing architectures is clearly an opportunity. It can prove relevant for scenarios where multiple data processing steps are carried out by combining edge compute processing with centralised Cloud/HPC processing.
- **Blending of traditional HPC simulation with Deep Learning techniques:** To improve simulation results whilst requiring less computational effort. Tentative results from early investigations show a potential to enhance and augment existing simulations, steer simulations between successive iterations, and combine numerical simulation models with Machine Learning based equivalent approximations for instance for short-range predictions.
- **Maximising operational efficiency and throughput for large multi-node systems:** This is a Big Data problem in itself, which requires near-real-time analytics of huge data streams (like monitoring data from all nodes and fabrics), prediction and prognostics of future system behaviour and state, and finally prescriptive system management. All of this has to fit in the compute centres production policies, and needs to reach complex resource management and orchestration decisions.
- **Demand by HPC users for interactive analytical capabilities added to HPC workloads:** Challenge is to integrate these with the current systems and systems software architectures and ensure that HPC resources are effectively utilised.

5.3 Mutual Software Engineering Learnings

The specification of infrastructure requirements for a BDC or HPC workload entails a complex task involving multiple variables and diverse criteria, such as data types, scalability and type of processing, communication between tasks or processes, among many others. Software engineering has a critical role to play in ensuring those workloads can make best use of the underlying hardware resources.

For example, BDC workloads typically run a Cloud Computing setup designed to leverage dynamic and adaptive cluster resource management, dimensioning and configuration according to economic cost, required quality of service, and availability. Software engineering of BDC applications is typically orientated around agile frameworks – designed to lower barriers between Development and Operations teams, accelerate workflows (i.e. high deployment rates for faster feedback, better code quality leading to

less errors and lower costs, etc), and increase the reliability, stability, and resilience of the production environment.

HPC workloads typically leverage well-established models and processes for optimising performance and efficiency of very large, tightly coupled workloads and execution systems. Software engineering of HPC applications is typically combining the modeling of compute, communication and I/O operations required with proven ways to identify and correct efficiency and performance bottlenecks in their execution on large-scale parallel computer systems.

Software engineering frameworks associated with respective ecosystems of BDC and HPC may still require further research and advances; even so, there are opportunities for each ecosystem to benefit from results already created by the other ecosystem as respective workloads increase in compute intensity and/or in data intensity.

6.0 CONCLUSIONS

Whilst a proportion of Big Data Computing workloads are increasing in compute intensity and a proportion of HPC workloads are increasing in data intensity, the generally unique (technical and economic) requirements of HPC and Big Data Computing workloads call for distinct compute stacks tailored to the needs of their respective workloads. HPC and BDC ecosystems should continue to identify and research idiosyncratic challenges particular to their respective domains, but not in isolation from each other. Fostering increased collaboration between HPC and Big Data ecosystems will lead to the identification and exchange of complementary capabilities (e.g. applications, setups, workflows, etc) that will accelerate the pace of innovation for their respective stacks i.e. by transplanting specific HPC-inspired-capabilities for addressing the compute-intensity of certain Big Data Computing workloads, and similarly transplanting specific Big Data-inspired-capabilities for addressing the data-intensity of certain HPC simulation and modelling workloads.

> Increased collaboration to identify and exchange complementary capabilities

ACKNOWLEDGEMENTS

This paper is possible by member contributions and reviews across BDVA and ETP4HPC associations.

Contributors

Big Data Value Association (BDVA)
Ana García Robles, BDVA Office
Roberta Turra, Cineca
Andrea Manieri, Engineering Ing. Inf. SpA
Davide Dalle Carbonare, Engineering Ing. Inf. SpA
Gabriel Antoniu, Inria
Jim Kenneally, Intel Corporation
Alexandru Costan, Irisa
Paul Czech, KNOW-CENTER
Nenad Stojanovic, Nissatech
Thierry Nagelle, Orange
Arne Berre, SINTEF
Eneko Osaba, TECNALIA
Sergio Campos, TECNALIA
Maria Perez, Universidad Politécnica De Madrid
Jukka K Nurminen, VTT

European Technology Platform for HPC (ETP4HPC)
Pascale Bernier-Bruna, Atos
Michael Malms, ETP4HPC Office
Dirk Pleiter, Forschungszentrum Jülich
Thomas Eickermann, Forschungszentrum Jülich
Jens Krüger, Fraunhofer ITWM
Stephane Requena, GENCI
Hans-Christian Hoppe, Intel Corporation
Erwin Laure, KTH Royal Institute of Technology
Guy Lonsdale, Scapos AG
Dirk Pleiter, Forschungszentrum Jülich
Maike Gilliot, Teratec
Mark Asch, Total S.A.
Jean-Francois Lavignon, TS-JFL

To comment on this document, contact editors jim.kenneally@intel.com (on behalf of BDVA) or hans-christian.hoppe@intel.com (on behalf of ETP4HPC).

Reference details to cite this paper:

> "Kenneally, Jim, and Hoppe, Hans-Christian, editors. *The technology stacks of High Performance Computing and Big Data Computing: What they can learn from each other*. 2018. A joint publication between the European associations of www.ETP4HPC.eu and www.BDVA.eu."

ABOUT BDVA

The Big Data Value Association (BDVA) is an industry-driven international non–for-profit organisation with 200 members (as of November 2018) all over Europe and a well-balanced composition of large, small, and medium-sized industries as well as research and user organizations. BDVA is the private counterpart to the EU Commission to implement the Big Data Value PPP program. BDVA and the Big Data Value PPP pursue a common shared vision of positioning Europe as the world-leader in the creation of Big Data Value. The mission of the BDVA is to develop the Innovation Ecosystem that will enable the data-driven digital transformation in Europe delivering maximum economic and societal benefit, and, achieving and sustaining Europe's leadership on Big Data Value creation and Artificial Intelligence. BDVA enables existing regional multi-partner cooperation, to collaborate at European level through the provision of tools and knowhow to support the co-creation, development and experimentation of pan-European data-driven applications and services, and knowhow exchange. For further information: www.bdva.eu / info@core.bdva.eu / @BDVA_PPP

ABOUT ETP4HPC

ETP4HPC is the European Technology Platform (ETP) in the area of High-Performance Computing (HPC). It is an industry-led think-tank comprising of European HPC technology stakeholders: technology suppliers, research centres as well as Independent Software Vendors and HPC industrial and academic end-users (as of November 2018, ETP4HPC has 93 members, out of which 45 are companies from the private sector /34 are SMEs/). The main task of ETP4HPC is to define research priorities and action plans in the area of HPC technology provision. Since 2013, we have been issuing and updating our Strategic Research Agenda as a multi-annual European HPC technology roadmap and a mechanism to help the European Commission define the contents of the HPC calls for projects in the successive Horizon 2020 Work Programmes. At the end of 2013, ETP4HPC signed the HPC contractual Public Private Partnership with the European Commission. ETP4HPC is the private side partner of this cPPP, contributing to H2020 HPC programme development and steering, in particular synchronising the efforts in the areas of technologies and applications (the Centres of Excellence in Computing Applications joined the cPPP Partnership Board in 2015). More at www.etp4hpc.eu.

ENDNOTES

[1] A model of a system is "a set of instructions, rules, equations, or constraints for generating [input/output] behaviour". A simulation executes to the model, so that it computes the modelled system behaviour.
[2] A feature is an individual measurable property or characteristic of a phenomenon being observed. Choosing informative, discriminating and independent features is a crucial step for effective algorithms in pattern recognition, classification and regression.
[3] Hadoop-type architecture refers to software frameworks for storage and large-scale processing of data-sets on clusters of commodity hardware.
[4] An embarrassingly parallel problem (EPP) is one for which little or no effort is required to separate the problem into a number of parallel tasks. This is often the case when no dependency exists between those parallel tasks, i.e. they neither have to communicate data nor do the tasks have to be synchronised. A common EPP problem is one in which a very large data set is chopped into pieces which are dispatched to various computers for processing; or, several copies of a smaller data set are distributed across computers to perform different computations on it (e.g. running the application with different parameters). After the processing is finished, the resulting data is re-assembled or the results from all computers summarised.
[5] HPC generally relies on the MPI (Message Passing Interface) industry standard which explicitly handles communication between computing nodes within the program code. Alternative models, like for instance PGAS (partitioned global address space) programming models only account for a very small part of HPC use.
[6] User-space is the CPU privilege level area where application software and some drivers execute; it is also used to refer to the memory areas accessible to such software codes.
[7] Floating-point operations per second (FLOP/s) – represent a unit of counting floating-point operations carried out by an algorithm or computer hardware. A floating point operation is a mathematical operation (addition, multiplication, division, and so on) on a number with a decimal point and exponent (for example, 1.2345 x 103). Considered important as HPC workloads carry out a huge amount of floating-point calculations.
[8] An application program interface (API) is a set of routines, protocols, and tools for building software applications. Basically, an API specifies how software components interact.
[9] An Integrated Development Environment (IDE) is a software application that provides comprehensive facilities to computer programmers for software development. An IDE normally consists of a source code editor, build automation tools and a debugger.
[10] Computational steering is the practice of manually intervening with an otherwise autonomous computational process, to change its outcome.
[11] Input/output operations per second (IOP/s) – measures computer storage in terms of the number of read and write operations it can perform in a second. IOP/s are a primary concern for Big Data environments where content needs to be efficiently accessed, added to or modified constantly.
[12] Integer is a whole number; a number that is not a fraction.
[13] Byte/string operations represent encoding and decoding of data. Everything must be encoded before it can be processed or written to computer storage/memory, and it must be decoded before it can be read by a human.
[14] Task Graphs break down a computation into a series of independent tasks with clearly defined dependencies which form a directed acyclic graph. The acyclic nature of the graph is important as it removes the possibility of deadlocks between tasks, provided the tasks are truly independent. The overall task graph is considered complete when all the tasks have completed.
[15] Graph Analytics includes graph modelling, visualisation, and evaluation for understanding large, complex networks.
[16] Spatial-temporal models arise when data is collected across time as well as space, includes specific processing and visualisation methods, in general GIS-related.
[17] Training (within the DL context) is very computationally expensive and can run 24/7 for very long periods of time. It is reasonable to think of training as the process of adjusting the model weights of an artificial neural network (ANN) i.e. model fitting to the training data set.
[18] Inferencing refers to the application of that trained model to make predictions or classify data on newly received data. In this case, the parameters are kept fixed based on a fully trained model. Inferencing can be done quickly and even in real-time to solve valuable problems. Inferencing consumes very little power, therefore inferencing can be incorporated into edge devices, smart sensors and IoT (Internet of Things) devices. Inferencing works because once the model is trained, the ANN can interpolate to correctly make predictions for data points it has never seen before.
[19] Fourier transform (FT) decomposes a function of time (a signal) into the frequencies that make it up, similar to how a musical chord can be expressed as the frequencies (or pitches) of its constituent notes.
[20] As the number of parameters and computational needs of neural networks grow, efficiently parallelising neural network training to run on many nodes and potentially many accelerators (beyond the number that can be attached to one node) becomes more and more important, because long waiting times for large networks to train slows down experimentation and limits further development.