J Supercomput
https://doi.org/10.1007/s11227-018-2277-x

Emergent models, frameworks, and hardware technologies for Big data analytics

Sven Groppe[1]

Abstract Today's state-of-the-art Big data analytics engines handle masses of data, but will reach to their limits, as the future Big data flood is predicted to still grow with an increasing speed. Hence we need to think about the next development phase and future features of Big data analytics engines. In this paper, we discuss possible future enhancements in the area of Big data analytics with focus on emergent models, frameworks, and hardware technologies. We point out a selection of new challenges and open research questions.

Keywords Big data · Computer architectures · FPGA · GPU · Cloud Computing · Fog Computing · Dew Computing · Semantic Web

1 Introduction

Big data experts characterize Big data with certain words starting with a big 'V'. The most common ones are coined to Laney [47], which uses Volume to express that Big data must have some certain data quantity, Velocity to refer to an enormous speed of new data, and Variety to remember that in usually heterogeneous environments there are a lot of data types and formats to be processed. The community actively took over the idea of big V's and, as a result, proposed an increasing number of V's (e.g., [90] for 7 V's and [19] for the top 10 list of V's). One of these most important newly proposed big V's is the Value, which refers to the data usefulness: We should be able to process the today's masses of heterogeneous data in order to conclude new information, to use

✉ Sven Groppe
groppe@ifis.uni-luebeck.de

[1] Institute of Information Systems, University of Lübeck, Ratzeburger Allee 160, 23562 Lübeck, Germany

Published online: 20 February 2018

the data for new applications and have some benefits from them. Otherwise, handling with Big data would not be meaningful for us.

Big data analytics engines are already able to handle masses of data and offer sophisticated technologies to address the volume, velocity, and variety properties. However, in order to handle the future's increasing demand of Big data processing, we need to think about new technologies and computing paradigms to be integrated into future generations of Big data analytics engines. Furthermore, we need to discuss ways to increase the value of Big data.

In this paper, we hence discuss possible next steps for future generations of Big data analytics engine. We explain and analyze the possible future integration of emergent hardware technologies into existing frameworks for Big data analytics. Furthermore, we look at new computing paradigms besides cloud computing, which may be applied in future Big data analytics engines as well. Moreover, we shortly discuss the Semantic Web offering a data model targeting at the value property of Big data.

The contributions of this paper include:

- an overview of the current state of big data analytics engine, emergent hardware technologies and new computing paradigms, and their relation to the Semantic Web,
- a comparison of Moore's law with the properties of top ranked parallel computers,
- an analysis of high-end hardware of different types of parallel architectures, and
- a selection of new challenges and open research questions in the intersection of the discussed areas for data management problems of Big data analytics engines with special focus on
  - Big data analytics engines for new computing paradigms like fog and dew computing,
  - the Semantic Internet of Things area and
  - index accesses of hardware-accelerated Big data analytics engines.

2 Moore and data growths

Gordon Moore observed that the number of transistors in a dense integrated circuit doubles approximately every 2 years [63,64]. His law was later adapted to other areas and describes in general the exponential growth of processing capabilities of a single computer. However, what we today recognize is that the growth of data to be processed is also somehow exponential, but growing faster than the processing capabilities.

Some decades before, it was common sense to expect that the next generation of computers can handle those problems not process-able by the older computers. Today this is on the one hand still true, but on the other hand, it seems to be that today's applications and infrastructure are designed to produce as much data as possible leading at some point to a processing demand not to be handled even by future computer generations. Furthermore, because of physical laws like those of quantum effects, the Moore era seems to come to a natural end [65,91], i.e., the growth of processing capabilities of a single computer may at least slow down or even the processing capabilities do not increase at all any more.

The end of the Moore era can be made visible by looking at top ranked parallel computers[1]: Whereas the achieved GFlops[2] per core seemed to be increasing following an exponential growth until 2012 (see Fig. 1), afterward the achieved GFlops per core even decreases. This effect can be easily overseen whenever only the achieved performance is considered (see Fig. 2): The exponential growth of the achieved performance is only gained by an exponential growth of used cores (see Fig. 3).

Hence there must be other ways to handle the Big data flood. Completely new computing paradigms like quantum computing promising exponential speedup but only for certain calculations and neuromorphic computing modeling processing elements on neurons in the brain are still not far away from laboratory [91]. The only currently feasible way seems to be using parallel computing, which requires parallel algorithms to work on partitioned data, scalable with the Big data flood whenever the data can be partitioned in disjoint fragments to be processed (mainly) independently from each other. Indeed, the number of cores grows exponentially for top ranked parallel computers, as we already discussed above (see Fig. 3) and which is a sign for the great success of using more and more parallelism in order to overcome problems of the end of Moore era.

3 Evolution of typical Big data analytics engines

Because the Big data landscape [89] covering the most important technologies, frameworks, involved companies, and products is too huge, we can deal only with a small excerpt of it. Hence we focus on Big data analytics engines in this paper.

The typical Big data analytics stack (see Fig. 4) includes a layer for the data and the access to it. Engines for Big data analytics usually provide access to data in databases, in distributed storages or streams. Engines especially deal with distributed resource management and distributed execution. They provide their capabilities to application developers and Big data experts in terms of an application programming interface (API) and often by supporting a high-level scripting language and even SQL-like query languages. Big data engines often have specialized APIs for processing streams, applying machine learning and graph processing applications.

Big data engine developers make huge efforts to integrate their product into the existing Big data landscape and reuse a lot of the existing technologies for import and export of data and the processing results. However, some core modules like the APIs for data, stream, and graph processing and machine learning as well as scripting and query languages are often reinvented, such that there is also a big migration effort necessary for changing an underlying Big data analytics engine in an existing application.

Let us look at the evolution of Big data analytics engines (see Table 1). While the original Hadoop version 1 system [85] introduces the Map-/Reduce programming paradigm with the focus on batch processing of Big data with applications, e.g., in targeted advertisement and log analysis, the next generation of engines already sup-

[1] As given in the November list at https://www.top500.org.

[2] Floating-point operations per second (Flops) is a measure of computer performance representing the number of floating-point calculations computed per second.

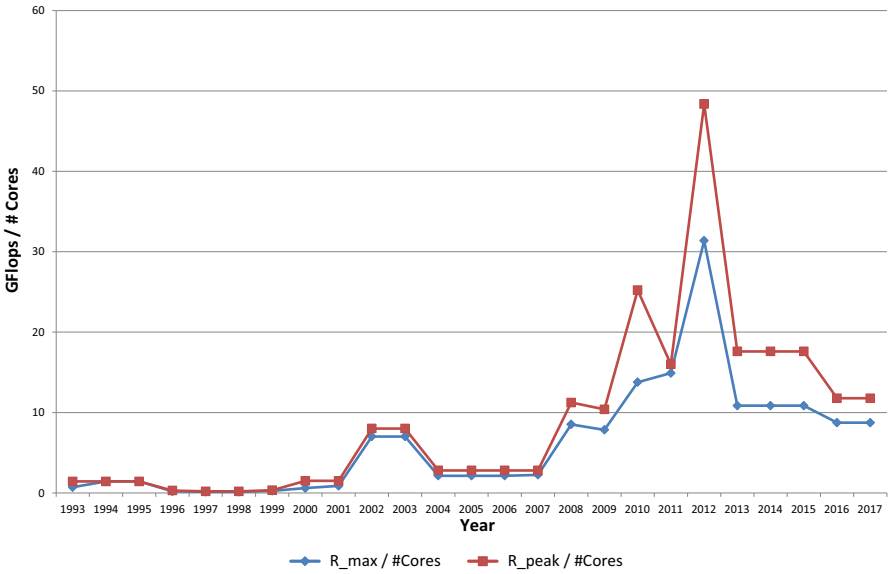

Fig. 1 Achieved GFlops per core of the top ranked parallel computer of the November list of the corresponding year. Whereas $R_{\max}$ represents the maximal LINPACK performance achieved, R_{peak} contains the numbers of the theoretical peak performance

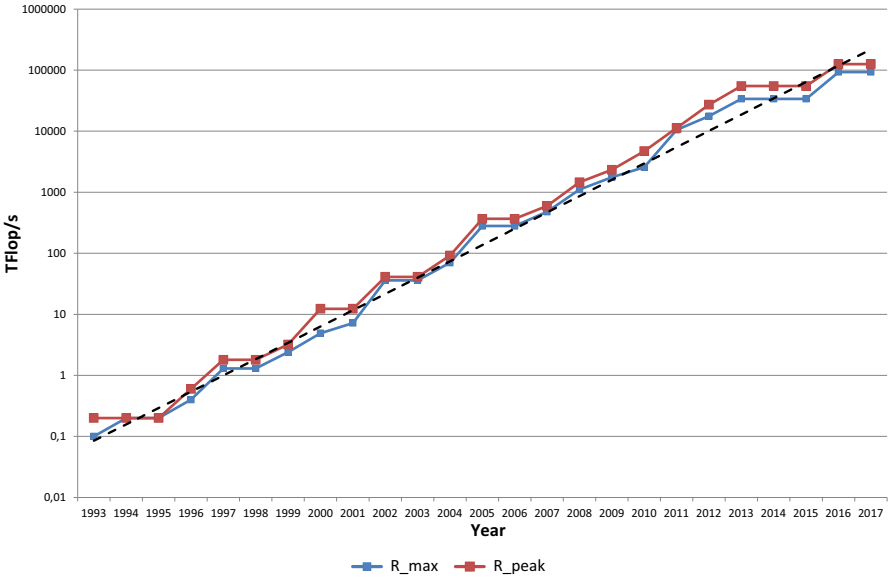

Fig. 2 TFlops of the top ranked parallel computer of the November list of the corresponding year. Note that the *y*-axis has a logarithmic scale. Whereas $R_{\max}$ represents the maximal LINPACK performance achieved, R_{peak} contains the numbers of the theoretical peak performance

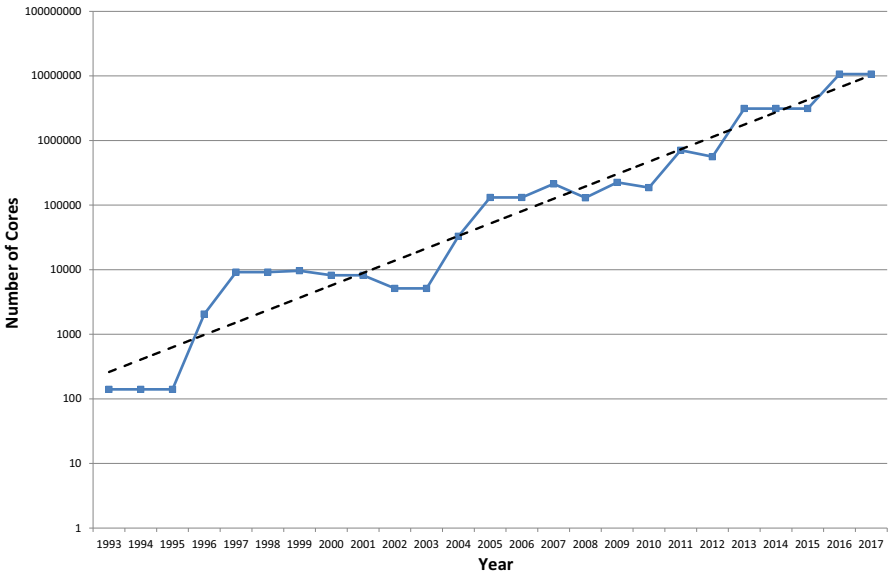

Fig. 3 Number of cores of the top ranked parallel computer of the November list of the corresponding year. Note that the *y*-axis has a logarithmic scale

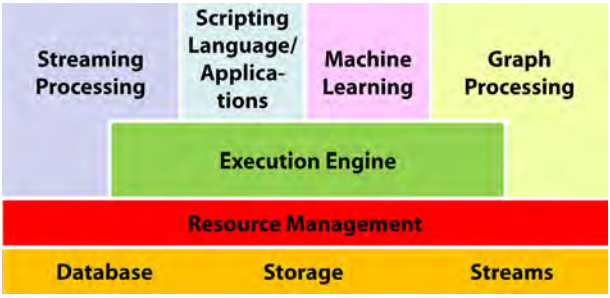

Fig. 4 Typical Big data analytics stack (e.g., Spark, Flink, Storm)

Table 1 Evolution of Big data analytics engines

	Generation			
	1	2	3	4
Features	Batch	+ Interactive	+ Near-Real-Time + Iterative Processing	+ Real-Time Streaming + Native Iterative Processing
Processing model	MapReduce	DAG Dataflows	Resilient Distributed Datasets (RDD)	Cyclic Dataflows
Engine	Hadoop	TEZ	Spark	Flink

ported enough performance for user interactivity. TEZ [87] hence also extended the processing model to allow directed acyclic graphs for the data flow between different processing units. The third generation to which SPARK [88] belongs, already supports near-real-time and iterative processing by handling mini-batches and introducing Resilient Distributed Datasets enabling in-memory computations on large clusters in a fault-tolerant manner. Apache Flink [86] is one engine of the current fourth generation, which supports real-time processing of streams by native iterative processing instead of internal mini-batch processing as in SPARK. The processing model of Flink also supports cyclic dataflows, where iterative computations are optimized by Flink.

The interesting question is now: What will be the key features of the next generation? We will try to speculate in the following sections about this issue...

4 Possible directions for the fifth generation of Big data analytics engines

Possible directions for the fifth generation of Big data analytics engines might be the support of co-processors.

There are already some few research efforts to use GPUs and FPGAs in Spark [25, 80], but these systems seem to be preliminary. Another direction could be to integrate the use of smart SSDs (e.g., [92]) into the engines, such that data is already filtered at the SSD using the processing capabilities of smart SSDs in order to drastically reduce transfer costs between the SSD and the memory of a cluster node.

There might be also tendencies to go away from cloud computing [23] as the transfer to nodes are bottlenecks especially for Internet of Things [59] scenarios such that processing capabilities closer to the sources of data generation are used instead or complimentary. Approaches like fog and dew computing [18,82] may be adapted for future generations of Big data analytics engines.

We try to shed lights on these emergent hardware technologies and computing paradigms for the usage in Big data analytics engines in the following subsections.

4.1 Co-processors

We will compare multi-core CPUs, many-core CPUs, GPUs, and FPGAs with each other.

In a simplified schematic view (see Fig. 5) of the parallel architectures, we can divide the components of the architecture in computational units, execution controllers, interconnection network, and on-chip memory.

4.1.1 Multi-core CPU

The multi-core CPU has shared memory for all cores, but each core owns own memory (typically fast memory in form of caches), its own execution controller and a computational unit.

Multi-cores are suitable for applications [74] that

– benefit from high single-core performance,

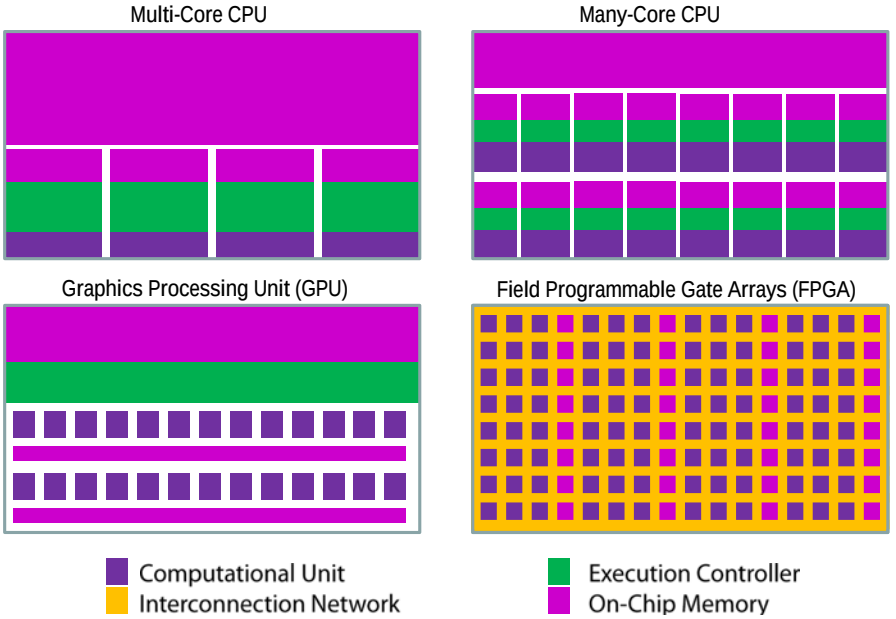

Fig. 5 Schematic view of parallel architectures (adapted from [74])

- scale only moderately,
- run threads with different functionality (according to the multiple-instruction multiple-data (MIMD) paradigm),
- are control flow dominated,
- rely on shared memory, and
- access memory only in irregular patterns (with benefits from caches).

The use of multi-core CPUs is currently standard in Big data analytics engines.

4.1.2 Many-core CPU

Many-core CPUs manage now much more cores, but the architecture remains similar to the one of the multi-core CPU. However, many-core CPUs are optimized for a higher degree of explicit parallelism, and often for higher throughput at the expense of latency and lower single thread performance.

Many-cores are suitable for applications [74] that

- scale well,
- have threads with different control flows, and
- compute mostly on private data (sometimes with access to some shared data),

Because of the similarity to multi-core CPUs, the extension of the support of Big data analytics engines to many-core CPUs seems to be straightforward on a first look, but there are many challenges remaining for future research like for optimizing massively parallel execution plans and coordinating competing accesses to data.

4.1.3 GPU

A special type of many-core CPUs are graphics processing units (GPUs) suitable for vector processing evaluating the same instructions on multiple data in different cores. Groups of cores typically share some faster memories most often used as caches of the global memory.

General-purpose graphics processing units (GPGPUs) turn the massive computational power of a modern graphics accelerator's shader pipeline into general-purpose computing power. Although graphics processing units (GPUs) are not necessarily meant for general-purpose computing, GPGPUs and GPUs are often used synonymously. GPUs basically work according to the single-instruction multiple-data paradigm, where the same instruction is executed on different data on different cores at the same time, i.e., not all parallel algorithms are suitable for GPUs. Modern GPUs offer several thousand computing cores, such that some applications are greatly speeded up in comparison with multi-core CPUs. There exists a lot of proprietary and also vector-independent programming standards for programming GPUs. The most important one is OpenCL [83].

GPUs are suitable for applications [74] that

- utilize data parallelism,
- run threads with identical control flow,
- access memory in regular patterns,
- work on small, static working sets without dynamic memory allocation, and
- use floating-point arithmetic.

It seems to be that one of the most important challenges for a tight integration of GPU acceleration into future Big data analytics engines is the further design of algorithms for typical processing tasks tailored to the GPU hardware architecture utilizing the massive parallelism on the one hand, but masking the overhead for communication between host computer and GPU on the other hand.

In this context, it is worth to consider AMD Accelerated Processing Units (APU) [7], which are designed to act as a CPU and GPU on a single die. While the processing capabilities are typically more restricted in comparison with (high-end) single GPUs, they have the great feature to access the main memory coherently to and as fast as the CPU, i.e., communication costs for the transfer to the GPU are avoided in contrast to PCI-attached GPUs. In this way computations may benefit already for smaller problem sizes from GPU acceleration, because APUs do not have to first neutralize communication overhead (as PCI-attached GPUs have to do).

4.1.4 FPGA

Field-programmable gate arrays (FPGAs) consist of an array of programmable logic blocks as well as reconfigurable interconnects for connecting these blocks with each other. Logic blocks can be configured to perform simple logic gates (like AND and XOR) up to complex combinational functions.

The FPGA configuration is typically specified by using a hardware description language (HDL). In comparison with software development, developing HDL programs is several factors slower and more error prone [1].

Recently, high-level synthesis (e.g., OpenCL) is more mature (e.g., [102]), such that the development efforts of FPGA programs becomes comparable to software development (but still performance-critical parts should not be implemented in OpenCL, as they have a high overhead). Furthermore, although OpenCL programs developed for GPUs are also running on FPGAs, they need to be optimized for FPGAs to avoid worse performance and achieve better results with FPGAs.

A main disadvantage of FPGAs is that it takes a very long time from an HDL or OpenCL program to running the configuration on the FPGA. Often developers need to wait hours. There is no problem if the FPGA should run only one configuration. However, if the FPGA should be reconfigured during runtime, there are some tricks for an efficient procedure necessary. Hence there are still many open challenges for the integration of FPGAs in the processing pipeline of Big data analytics engines in order to allow a certain flexibility in specifying the processing tasks at runtime, but having a short time for reconfiguration.

Summarizing, FPGAs are special because the interconnection network between its cores can be configured making them ideally suitable for data-flow-driven algorithms as well as fast on-chip memory is massively distributed and hence can be accessed in parallel by its numerous cores.

Hence FPGAs are suitable for applications [74] that

- are designed in a data flow style by implementing processing pipelines, and
- process streams of data.

Also for FPGAs is one main task for future research the investigation of new algorithms for Big data analytics tasks especially designed for the FPGA. Note that besides PCI-attached FPGA acceleration cards (running on the most widespread platforms), recently shared-memory systems like HARP2[3] (e.g., [94]) are developed, which do not have the disadvantages of communication overhead (as for PCI-attached FPGA cards), because CPU and FPGA can (coherently) operate on the same main memory (in parallel). Analogous to APUs, calculations may benefit for much smaller problem sizes from accelerations by shared-memory systems in comparison with PCI-attached FPGAs.

4.1.5 Comparison

Table 2 contains the high-end hardware (in terms of achieved GFlops (single precision)) for the recent 10 years, which we have considered for our comparisons of the different types of parallel hardware architectures. For a reasonable selection and simplicity of presentation, we consider only leading manufacturers of different types of hardware[4]: Intel CPUs for servers, many-core CPUs of Intel and GPUs from NVIDIA

[3] https://software.intel.com/en-us/hardware-accelerator-research-program.

[4] Inspired by [77], the numbers for the considered hardware are collected from https://www.intel.com/content/www/us/en/support/articles/000006779/processors.html for Intel Xeon processors,

Table 2 Considered high-end hardware

Year	Intel server CPU	Intel many-core	NVIDIA GPU	AMD GPU
2007	Xeon X5482		GeForce 8800 GTS	Radeon HD 3870
2008	Xeon X5492		GeForce GTX 280	Radeon HD 4870
2009	Xeon W5590		GeForce GTX 285	Radeon HD 5870
2010	Xeon X5680		GeForce GTX 580	Radeon HD 6970
2011	Xeon X5690		GeForce GTX 580	Radeon HD 6970
2012	Xeon E5-2690	Xeon Phi SE10	GeForce GTX 680	Radeon HD 7970 GHz Ed.
2013	Xeon E5-2697 v2	Xeon Phi 7120	GeForce GTX Titan	Radeon HD 8970
2014	Xeon E5-2699 v3	Xeon Phi 7120	Tesla K40	FirePro W9100
2015	Xeon E5-2699 v3	Xeon Phi 7120	GeForce GTX Titan X	FirePro S9170
2016	Xeon E5-2699 v4	Xeon Phi 7290	NVIDIA Titan X	FirePro S9170
2017	Intel Xeon Platinum 8180	Xeon Phi 7290	Tesla V100	Radeon Vega Frontier Edition

and AMD. We can expect that the main trends remain for high-end hardware of other manufacturers.

For this high-end hardware of the recent years, the end of the Moore era can be shown by looking at the frequency of computational units (i.e., CPU and GPUs) (see Fig. 6), which does not increase any more in big jumps and sometimes even decreases (e.g., Intel Xeon) in order to allow more computational units on a die for more parallel computations.

The number of computational units[5] grows more extensive in the considered recent high-end hardware (see Fig. 7), which is a sign for the trend to using more and more parallelism for satisfying the increasing demand for processing capabilities.

Figure 8 and Table 3[6] contain the performance metrics GFlops (single precision) of the considered high-end hardware (see Table 2) of the different parallel architectures: Many-core CPUs close the gap between CPUs and GPUs in terms of GFlops performance. FPGAs are also somewhere between CPU and GPU GFlops performance.

Footnote 4 continued
http://www.intel.com/content/dam/www/public/us/en/documents/product-briefs/high-performance-xeon-phi-coprocessor-brief.pdf and https://www.intel.com/content/www/us/en/processors/xeon/scalable/xeon-scalable-platform-brief.html for Xeon Phi, http://en.wikipedia.org/wiki/List_of_Nvidia_graphics_processing_units for NVIDIA GPUs and http://en.wikipedia.org/wiki/Comparison_of_AMD_graphics_processing_units for AMD GPUs.

[5] We define the number of computational units of CPUs as the number of cores, and compare these number with the number of streaming multi-processors (NVIDIA GPU) and compute units (AMD GPU). We do not present here the number of compute units of GPUs, groups of which are meant for handling single-instruction multiple-data (SIMD) in GPUs and hence are not independent of each other as the cores of CPUs.

[6] Note that there is a big discussion of about how to determine the GFlops of an FPGA (see, e.g., https://www.altera.com/en_US/pdfs/literature/wp/wp-01222-understanding-peak-floating-point-performance-claims.pdf). We considered the numbers presented in previously mentioned web document. Further note that due to the different architectures it is much more difficult to define the number of computational units of an FPGA, which are comparable to those of CPUs and GPUs. Hence we do not provide the numbers of computational units for FPGAs here.

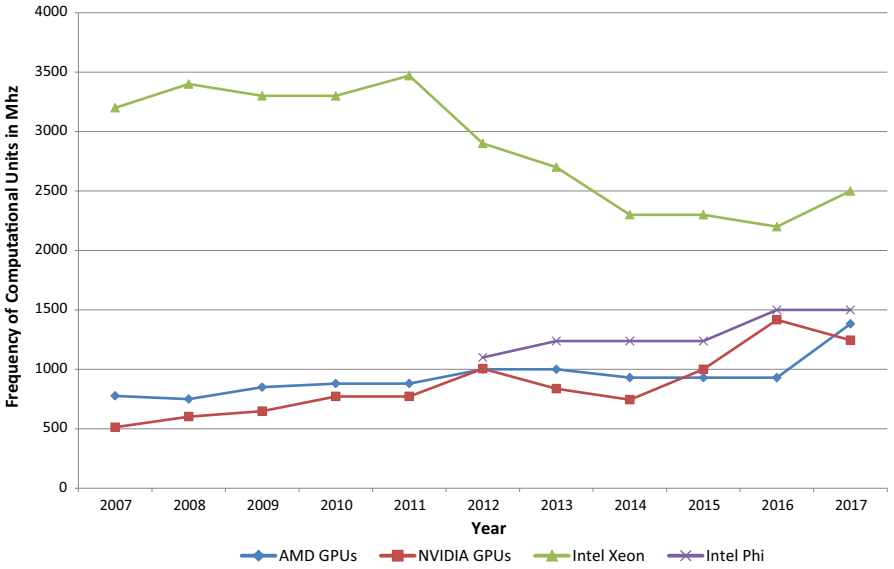

Fig. 6 Frequency of computational units of high-end hardware (Intel Server CPUs, Intel many-core chips, NVIDIA and AMD GPUs)

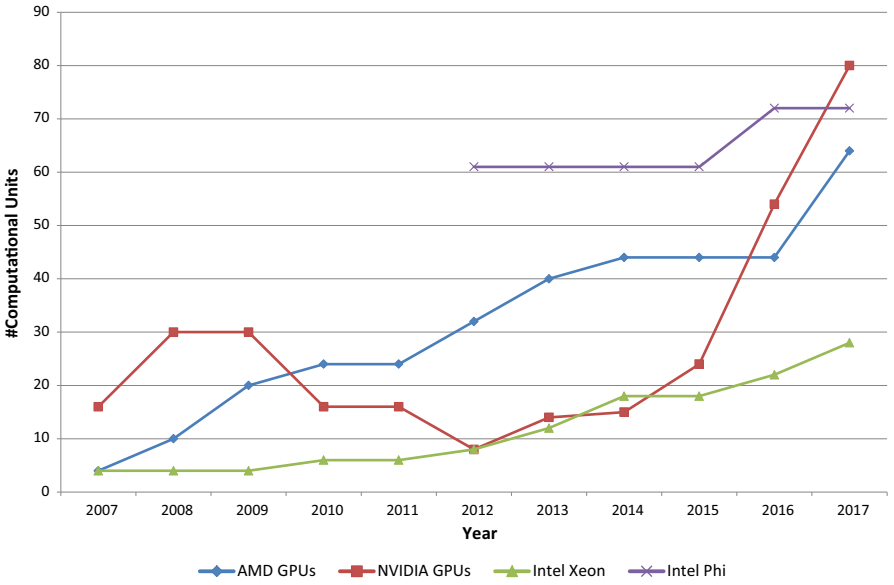

Fig. 7 Number of computational units of high-end hardware (Intel Server CPUs, Intel many-core chips, NVIDIA and AMD GPUs)

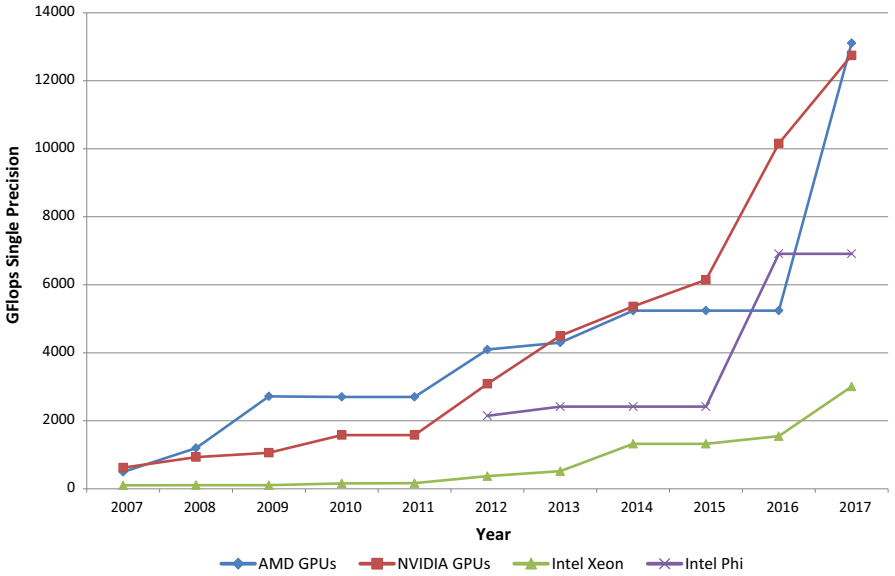

Fig. 8 Single-precision GFlops

Table 3 GFlops of FPGAs

Vendor	FPGA	GFlops
Intel/Altera	Midrange Arria 10 devices	160 to 1500
	High-end Stratix 10 devices	10,000
Xilinx	Zynq-7000	778
	7 Series	2000
	UltraScale	4903

However, if we compare the performance (in terms of GFlops) per computational units of the different types of high-end hardware (see Fig. 9), we recognize that these performances are much more close to each other than one would expect just when looking at the overall performance (see Fig. 8).

Based on these and further observations, we now compare the different architectures concerning performance in terms of integer and floating-point operations per second, ease of coding, flexibility of usage, suitability for hard real time, and the operations per Watt by rough estimations. An overview is provided in Fig. 10.

The multi-core CPU is known to be very flexible and easy to program, but is not very efficient as only few cores are available for parallel programs and consumes much electric power.

GPUs are highly parallel and hence have a better performance by a slightly reduced flexibility, but consuming less electric power.

FPGAs are still fully flexible, having a better integer performance by reduced consumption of electric power by a factor of about 10 in comparison with a CPU/GPU

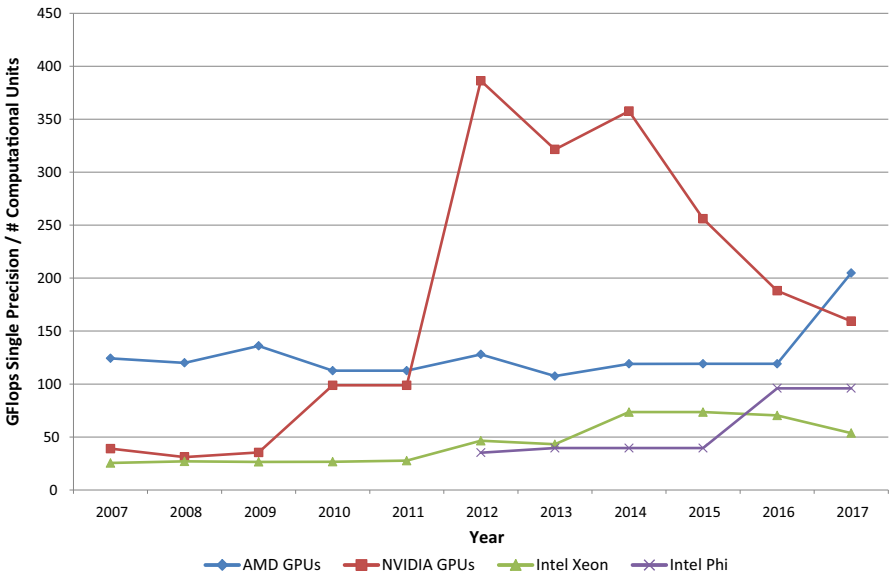

Fig. 9 Single-precision GFlops divided with the number of computational units

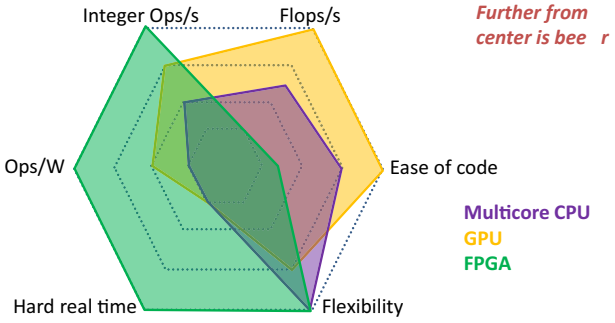

Fig. 10 Comparison of parallel architectures

system according to some recent studies [70], but are very difficult to program. The situation about their ease of coding is currently changing with the support of OpenCL, but still FPGA providers state that at least performance-critical parts should not be implemented with high-level synthesis like OpenCL.

4.2 Cloud and new computing paradigms

While acceleration by co-processors like GPU and FPGA are suitable as extensions to today's computing infrastructure, which have a central gateway like it is the case for cloud computing [23], we want to discuss about other emergent computing paradigms here. For this purpose, we first point out the problems of cloud computing in context of new application scenarios and contexts like the one of Internet of Things [59].

4.2.1 Cloud Computing

In the Internet of Things, many small devices produce a lot of sensor data and other data. Just let us consider the smart home scenario, where your things at home can be accessed via internet and maybe high-level applications need to process all their generated sensor data in real time. Today's solutions typically just use a cloud to collect and store all the sensor data and other data of your things.

However, the connection from your things to the cloud is a bottleneck, as these data must be continuously transferred over wide-area networks. It is also not a scalable solution, as the problem of the bottleneck becomes bigger for more things at your home. Furthermore, the transferred data may contain sensible data like exact times of your presence at home. Hence there is a security risk for you as the data may be misused by the company, by criminals within the company or criminals hacking the company's computer infrastructure.

4.2.2 Fog Computing

The fog computing paradigm [18,82] is mainly coined to Cisco systems. There are also other names for the same paradigm like edge computing.

Fog computing utilizes near-things edge devices with higher capabilities to carry out a substantial amount of storage (rather than stored primarily in cloud data centers), communication (rather than routed over the internet backbone), and application logic.

The main advantage is also that data resides at owners infrastructure, which reduces the discussed security risks of cloud computing.

However, fog computing is not really scalable in the number of connected things, as the near-things edge devices do not increase in number and capabilities in the same way.

4.2.3 Dew Computing

Hence there are ideas collected under the dew computing [31,82,93] paradigm, which also utilize the things themselves for storing, processing and management tasks according to the observation that their capabilities also steadily increase.

In dew computing, the near-things edge devices are still used, but additionally now the things themselves fulfill tasks of the high-level application and store the raw data and its processed results. Additional transfer of data to, storing and processing in the cloud are optional. Hence the advantage remains that security risks are reduced compared to cloud computing as the data resides at the owner's infrastructure. Furthermore, dew computing is really scalable with the number of things: As more things are there and produce data, as more things also offer their capabilities for storing and processing. Disadvantages are besides problems of heterogeneous environments in general, that the poor availability of some things needs to be compensated, which leads to further redundancies. And small things typically have limited storing and computing capabilities, such that the overall system must take care about the properties of each thing, too.

Table 4 Comparison of computing paradigms in the context of the Internet of Things

Computing paradigm	Advantages	Disadvantages
Cloud Computing	Scalable in processing, (illusion of) unlimited storing and processing capabilities	communication bottleneck between things and Cloud
Fog Computing	Data resides at owners infrastructure (reduces security risks)	Not really scalable
Dew Computing	Data resides at owners infrastructure (reduces security risks), scalable with number of things	Poor availability of some things need to be compensated, limited storage and computing capabilities of things

Our Semantic Web framework LUPOSDATE [35–37] already supports P2P networks as data source [61], which is a reasonable communication infrastructure for Internet of Things environments [62]. However, extended research is necessary to address a lot of open questions concerning fog and dew computing

4.2.4 Comparison and discussion

Table 4 summarizes the results of our comparison between the different computing paradigms.

Big data analytics engines are typically designed to run in the cloud. Hence, for the support of fog and dew computing, Big data analytics engines need to be evolved such that a heterogeneous hard and software environment is supported, near-things edge devices and things are detected, deployed, and managed, and computing and storage tasks are distributed between them by optimizing according to their capabilities and availability. Especially because of the low availability of things, further (efficient) strategies about redundancies in the computing and storage infrastructure need to be developed in order to avoid lost data and lost computing results.

4.2.5 Open challenges for Big data analytics engines for new computing paradigms

We point out a selection of open challenges and new research questions for data management of Big data analytics engines for new computing paradigms in this section.

Current Big data analytics engines are developed to be deployed in cloud environments. However, other execution environments like fog and dew computing impose new challenges for Big data analytics engines:

1. The hardware in fog and dew computing environments are much more heterogeneous in comparison with clouds. Hence new technologies are needed to distribute resources fairly in environments with components of diverse capabilities.
2. The network components in fog and dew computing may also differ much more than in clouds. Hence we need new research on routing for data management purposes in such networks with various different bandwidths between its components.

3. Components especially in dew computing environments fail more often than in clouds (because of limited battery capacities, lower lifetimes, mobile components, the networks got partitioned, ...). Hence we need sophisticated technologies to overcome related problems. Furthermore, in fog and dew computing, there is often a need of stream processing applying long-time queries in form of continuous queries. Hence processing tasks for long-time queries must be distributed among the heterogeneous components in fog and dew computing environments, such that as less work as possible must be repeated for fails and crashes. For this purpose, currently a backup of the states in stream processing is often proposed, which could be seen as a form of data replication (e.g., [20,69]). Another approach would be the replication of tasks, i.e., the replicated processing of tasks within the dew computing network for avoiding data loss. Also a predictive migration of tasks before components fail may be a promising future research direction.
4. There is a need for advanced privacy and security protection approaches, which work well in heterogeneous fog and dew computing environments, where often new components are deployed and the hardware of the components are more accessible by other possibly unauthorized persons (compared to close-able rooms of server farms for cloud computing).
5. Fog and dew computing environments are often organized as decentralized approach. Hence data in these environments must be also managed in a decentralized way. However, databases rely on a centralized (for centralized and parallel databases) or only slightly distributed architecture (for distributed databases). Recent approaches [32,60] try to use blockchain technologies (as known from the Bitcoin currency [68] and its transactions) for massive decentralized databases. However, these approach still do not adapt the blockchain technologies in a really consequent way to be suitable for fog and dew computing environments. Furthermore, there are still problems with the performance of blockchain technologies, the improvement of which must be further researched, which is a key for their success.

5 Semantic Web

Having in mind that the growth of data is today exponential, we must take care that the value of the data and the value of the result of data processing also scales with this exponential growth. Hence there are some efforts to support scalable value already in the data model. One of these efforts is the Semantic Web [15].

The Semantic Web uses ontologies as additional abstraction layer. In this additional abstraction layer, background knowledge can be specified (based on which new facts can be derived), which avoids redundant information, supports reuse of data and data integration, but increases computational complexity.

An example for initiatives for Big data collection resides in the Semantic Web community and includes the Linking Open Data (LOD) project [52], which maintains a Big data collection of freely available and accessible large-scale data sets. These data sets currently contain approximately 150 billion triples in over 2800 data sets [54,55] as well as links (of equivalent entities) between these data sets. In particular,

the links of equivalent entities address the volume and value properties of Big data, as only with these links the information of different data sets can be combined.

There are already many contributions running Semantic Web tasks on Big data analytics engines (e.g., [2,34,38,58,79]). For the future development of Big data analytics engines, investigations are necessary in order to point out drawbacks in the design of Big data analytics engines when running Semantic Web tasks, such that Big data analytics engines can be further evolved for faster processing of not only Semantic Web tasks. There seems to be much space left for future enhancements in the optimizer of Big data analytics engines when handling simpler data models (like the triples of Semantic Web) and more complex operations (like OWL inference resulting in complex cyclic data flows).

5.1 Open challenges for Semantic IoT

Research activities in the Semantic Web community are still very active. However, we want to point out a selection of open challenges especially for the *Semantic Internet of Things (SIoT)*, which applies Semantic Web technologies to the Internet of Things area, as we expect a big attention of the scientific community to this area in the near future.

5.1.1 Open research questions in heterogeneous environments

It is expected that the number of IoT devices per user will quickly and steadily increase with the consequence that the user will soon have a zoo of different IoT devices. Today, manufacturers typically offer for each IoT device (or for a subset of their IoT devices) another application (as app for smartphones, web application and/or PC software). Hence, for the control of its IoT devices, the user has to run a lot of different applications, especially if the user owns IoT devices of different manufacturers. From the users' point of view, there is a need for interoperable IoT devices, such that all its IoT devices can be managed by one application. Furthermore, interoperable IoT devices can be utilized for applications and services across different IoT devices of different manufacturers (e.g., combining IoT devices for the detection of persons at home with the data about favorite music songs of persons to play only music which is liked by all present persons).

In the following paragraphs, we discuss different approaches to support interoperable IoT devices.

1. For the support of interoperable IoT devices, we need standardized ontologies for data exchange, protocols for communication with and between the IoT devices, and maybe an additional code for specific applications utilizing all the features of the IoT device. The typical way is that manufacturers offer some form of data structure (e.g., described by ontologies), protocols and applications for one or a group of IoT devices, where often versions are released to deal with new devices, new requirements or just to fix errors (see a) in Fig. 11). The applications are based on the offered data structures and protocols, which can be manufactured-specific or standardized. If an application is developed for a newer version than

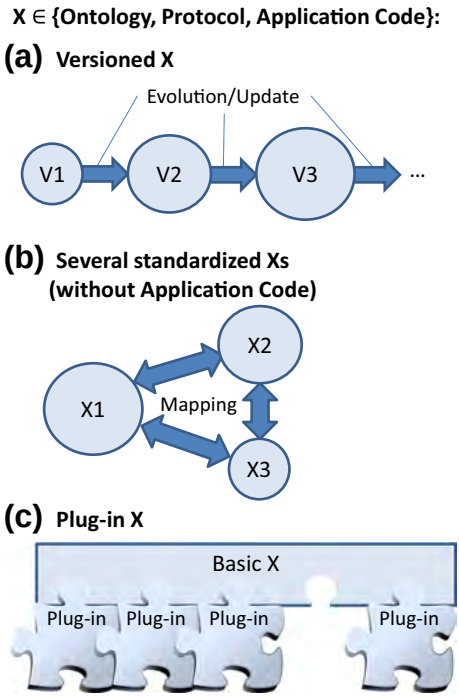

Fig. 11 Possibilities for ontologies, protocols, and application code in heterogeneous environments

supported by its addressed IoT devices, it must be taken care of releasing only downwardly compatible versions, or ontology and protocol evolution approaches must be applied. New IoT devices or their full potential may not be usable, for the time their special features are not considered in data structures, protocols, and applications. In particular, standardizations need time, which often lead to manufacturer-specific proprietary solutions in the meanwhile. New research is necessary for the development of adequate evolutionary approaches for jointly evolving ontologies and protocols in SIoT.

2. Standardizations typically can never support all the heterogeneous IoT devices and their features. This may lead to the development of several standardized or manufacturer-specific ontologies and protocols for different domains and subsets of IoT devices. To enable applications to use different ontologies and protocols, mapping approaches need to be applied (see b) in Fig. 11). In particular, for direct communication between IoT devices, there needs to be research on mapping approaches, which can overcome the limits of the sparse (storage, computing and energy) resources of IoT devices.
3. The two approaches above are not flexible enough for the fast-growing heterogeneous universe of IoT devices for the purpose of their quick and easy deployment. We hence envision a plug-in support of new IoT devices (see c) in Fig. 11): an IoT device should bring all necessary data with them such that it can be deployed and used in any environment immediately. Based on a basic ontology, protocol, and application, the IoT device could have a plug-in ontology, which extends the

basic ontology, a plug-in extension of the basic protocol as well as code to be plugged into the application for the purpose of fully utilizing the IoT device in the application. The development of such a plug-in environment and infrastructure is one of the biggest challenges for research and standardization.

5.1.2 Open research questions for continuous queries for Semantic IoT

Nowadays, Internet of Things applications generates large amounts of data at an ever increasing rate. These flows of continuously generated data have led to the emergence of a new kind of data known as data stream. Data streams can be defined as an infinite sequence of data arriving continuously in real time. (Infinite) data streams are generated from, e.g., sensor networks, which obtain data from their environment in high speed rate. In order to determine useful knowledge(like a probably upcoming earthquake) from data streams, we need to consider the infinite nature and support the computation of intermediate results based on a window, which contains the recent data of the infinite data stream. In many scenarios, the intermediate results must be calculated in a timely fashion, e.g., a probably upcoming earthquake must be detected as early as possible allowing no delays for the computations.

Analyzing data streams requires special techniques and method for querying. We distinguish three types of queries: (a) *One-time queries* are comparable to traditional queries in databases and consider all data in the data stream. Their results are computed and returned right after they are set up. (b) *Trigger queries* are waiting until their results are non-empty and return their non-empty results. (c) The results of *continuous queries* are periodically computed and returned.

Table 5 provides an overview over areas with new research possibilities for the different query types (one-time, trigger, and continuous queries) as well as considering the support of ontology inference with respect to the different execution environments local computer, cloud, and P2P networks. Following the observation that trigger queries are a special form of continuous queries and trigger queries are seldom looked at in research, we only deal with continuous queries in the following paragraphs, although special research for trigger queries may lead to new results.

Without support of ontology inference, all query types are already considered in every of the execution environments, but sometimes only few contributions exist:

There exist many approaches to query RDF data stored in a cloud [46]. Features of continuous queries like windowing is supported by Cloud computing environments like Apache Spark[7] and Apache Storm.[8] There are some contributions to support SPARQL processing in the Spark environment (e.g., [24]), but we are not aware of any approach to evaluate continuous SPARQL queries in Spark or Storm.

Continuous queries in P2P networks are supported by, e.g., P2P-DIET [42,43], CoQUOS [76] and CQC/CSBV [51]. There are only few contributions in this area, such that much space is left for new research and ground-breaking approaches.

[7] http://spark.apache.org/.

[8] http://storm.apache.org/.

Table 5 Discussed versus missing approaches for different query types in different execution environments ✓ = many approaches exist, ✓ = only few approaches exist, ↯ = not yet dealt with in scientific literature (to the best of our knowledge)

Ontology inference	Query type	Local	Cloud	P2P
With	Continuous	✓	✓	↯
	trigger	✓	✓	↯
	one-time	✓	✓	✓
Without	Continuous	✓	✓	✓
	trigger	✓	✓	✓
	one-time	✓	✓	✓

Only few approaches exist for the support of ontologies during query processing in the cloud [46]. Most of them support only RDFS and only few a very restricted fragment of OWL called OWL Horst fragment consisting of 24 rules [84]. SPOWL [53] maps axioms in the T-Box of a given ontology (in a slightly larger fragment than OWL2RL) to Spark programs for iterative execution in order to materialize a closure of reasoning results. Other approaches utilize a cluster to distribute reasoning on messages of IoT devices [44,57] avoiding to reason on large data.

Continuous queries with OWL ontology support (in the OWL2RL fragment) have been dealt with, e.g., in [10,11]. Again, much space is left for further research trying to increase the performance of continuous queries with ontology support. Furthermore, no investigations have been done so far in scenarios where the ontologies are dynamic and change over time, i.e., where the ontological statements come along the streams and are themselves considered for processing in windows.

[13,45] describe approaches to consider inference for RDFS ontologies during the processing of one-time queries. The support of OWL in P2P networks is still missing.

Continuous queries with the support of ontologies (considering their inferences in the query answers) have not been investigated so far in P2P networks and only few approaches exist for cloud clusters [44,57], which are favored networks types for fog and dew computing environments.

5.1.3 Semantic blockchain databases

There are already first approaches for Semantic blockchains (e.g., [78]), which are designed for Semantic resource and service discovery layer built upon a basic blockchain infrastructure in order to gain a consensus validation. However, while the studies show proof-of-concepts, the performance is still a problem especially on devices with low storage and processing capabilities facing a higher processing demand because of inferring tasks of the Semantic reasoner.

The techniques presented in [78] need to be refined to be applicable for a Semantic blockchain database.

6 Hardware-accelerated Big data analytics engines

We deal with hardware-accelerated Big data analytics engines in this section

– by first summarizing our own contributions to FPGA-accelerated processing of Semantic Big data, and
– by reviewing scientific literature of hardware acceleration of database indices for the purpose of identifying open research questions in this area.

6.1 Own contributions to hardware-accelerated processing of Semantic Big data

We developed LUPOSDATE [35–37], which is an open-source Semantic Web database supporting different types of indices for large-scale data sets (disk based) and for medium-scale data sets (memory based) as well as processing of (possibly infinite) data streams. LUPOSDATE supports the RDF query language SPARQL 1.1, the rule language RIF BLD, the ontology languages RDFS and OWL2RL, parts of geosparql and stsparql, visual editing of SPARQL queries, RIF rules and RDF data, and visual representation of query execution plans (operator graphs), optimization and evaluation steps, and abstract syntax trees of parsed queries and rules. The advantages of LUPOSDATE are the easy way of extending LUPOSDATE, its flexibility in configuration (e.g., for index structures, integration of different data sources, using or not using a dictionary, ...), and it is open-source [37]. These advantages make LUPOSDATE best suited for any extensions for scientific research. Hence LUPOSDATE is already extended for Internet of Things scenarios by supporting dew computing utilizing a P2P network spanning over the things for data storage and processing [61]. Also traditional cloud computing environments can be utilized by LUPOSDATE [38].

Recently there are some efforts to use FPGAs as hardware accelerators for query processing in LUPOSDATE [40,41,96–99]. Werner [95] describes a fully integrated hardware-accelerated query engine for large-scale data sets in the context of Semantic Web databases, where the proposed architecture automatically adapts and executes a user-defined query on a runtime reconfigurable FPGA. By using dynamic partial reconfiguration, the query can be exchanged within few milliseconds resulting in speedups of up to 32 for queries on data sets of over 1 billion triples.

It seems to be promising to use our experience of processing Semantic Web tasks in order to research on future generations of Big data analytics engines.

6.2 Open research questions for hardware-accelerated Big data analytics engines

Whereas there are already numerous implementations for hardware-accelerated query processing for FPGAs (e.g., [95–99]) and GPUs (e.g., [39,49,73]), it is maybe time to step back and research on hardware acceleration of even more basic tasks like data access based on indexes having in mind that index accesses are a dominant factor in query processing times.

Surprisingly, there are only few contributions to hardware accelerations of well-known indexes. Whereas many of the contributions presented in the following sections contain experimental evaluations based on PCI-attached GPUs or FPGAs, the utilization of GPUs and FPGAs may become even more beneficial if shared-memory systems

(APUs [7] or, e.g., HARP [94]) are applied. Shared-memory systems avoid communication costs, such that speedups are already achieved for smaller problem sizes.

6.2.1 B-trees

B-trees [26] and its variants like the B^+-tree are optimized for block-oriented storages (like hard disks, but also SSDs) for Big data. B-trees are very fast for read accesses, but typically suffer a bad write performance.

[41] utilizes an FPGA to search in parallel in nodes of the upper levels of a given B^+-tree. The extended contribution in [40] contains a discussion of how much update operations can benefit from the hardware acceleration of their necessary searches. [81] describes a (main memory) B^+-tree implementation for GPUs, and [27] for APUs.

6.2.2 Cache-oblivious lookahead array

Cache-Oblivious Lookahead Array (COLA) [14] consists of levels 0 to n, where the k-th level contains 2^k sorted elements. A level is either completely full up or empty. Insertions are first done in the level 0. If the level 0 is full up, then the inserted element is merged with the level 0. The result is stored in the next level, if the next level is empty and otherwise merged with the elements of the next level (repeatedly proceeded until an empty level is found). All full levels must be investigated for a search by utilizing binary search or (in a variant) *fractional cascading* [22], which uses additional pointers between the elements of full levels for speeding up the search.

We are not aware of hardware acceleration approaches for the COLA data structure, although it seems to be promising to investigate how parallel search capabilities of FPGAs and GPUs can speed up index accesses as well as how the merging step can be accelerated. There is great potential for FPGA acceleration of the merging step by applying merge networks [3,9,12,29,67], which can merge in logarithmic time instead of a linear one as in software.

6.2.3 Log structured merge: trees

Log Structured Merge (LSM)—trees [71] are optimized for update operations. LSM trees are slightly slower for reading accesses compared to B^+-trees. The LSM-tree consists of several levels, which all must be considered for search queries. Insertions (and deletions implemented by inserting deletion markers) are processed in the first level, which are traditionally located in the fast main memory. If one level is full, then this level (or big parts of this level) are rolled out into the next level, which can be optimized for presorted elements by merging algorithms.

LSM trees may have benefits from hardware acceleration whenever the management of the first level is taken over by GPU or FPGA. Also searching in higher levels or merging higher levels are good candidates for hardware acceleration. Note that runs in higher levels are often stored in a variant of B^+-trees [71] or in SSTables (e.g., BigTable [21], LevelDB [33], RocksDB [30] and Cassandra [28]). Hardware acceleration of B^+-trees has already been investigated [40,41,81] and adapted variants for SSTables seem to be feasible.

There exists first attempts to GPU acceleration of LSM trees [8], but we are not aware of approaches utilizing FPGAs for speeding up the index accesses of LSM trees.

Zhang et al. [101] introduces a variant of the LSM-tree for main memory databases: Here, the first level contains a data structure for fast updates (called dynamic index), which are rolled out in the second level, which contains a compressed data structure optimized for reading and low memory footprint (called static index).

Here, hardware acceleration for this index structure for main memory databases promises performance boosts. Basic steps to be accelerated are the insertion in the dynamic index, the search in the dynamic and static indexes, merging the dynamic and static indexes as well as the generation of the static index from the dynamic index.

6.2.4 Hash tables

Hash tables are often used in main memory databases for data indexing. According to [6], *cuckoo hashing* [72] is the most efficient index structure for point queries, but B^+-trees are faster for range queries. Cuckoo hashing [72] uses k hash tables and k different hash functions h_1 to h_k. Whereas in all k hash tables is searched for a key, the insertion of x is first tried only in the first hash table. In the case that there is already a value y at the position $h_1(x)$ in the first hash table, this value will be replaced with x and y will be probed in the second hash table maybe pushing another value to the third hash table, and so on. A value of the kth hash table will be inserted in the first hash table, where an endless cycle is detected when the same value is to be inserted into the same hash table (such that a reorganization must be triggered).

Alcantara et al. [5] demonstrate that large-size hash tables (using cuckoo hashing) can be built and accessed on the GPU at interactive rates. Liang et al. [50] describes that also FPGAs can greatly speed up cuckoo hashing [72] by pipelining the single steps of cuckoo hashing.

6.2.5 Adaptive radix trees

An alternative to hash tables in main memory databases are *adaptive* variants of *radix trees* (e.g., [48]). Because radix tree are order-preserving, range queries are faster than in hash tables, but not so fast than in B^+-trees [6].

We are not aware of any adaptive radix tree implementations for FPGAs, but for GPUs [4]. However, there are first implementations of radix trees on FPGAs [16,17].

6.2.6 Other data structures

There are some few contributions to hardware acceleration of other data structures optimized for GPUs like skip lists [66] and R-trees [56,75,100].

7 Summary and conclusions

We observe an increasing speed in the growth of the Big data flood, which may show the limits of state-of-the-art Big data analytics engines in the near future.

Hence we start a discussion about possible future development phases of Big data analytics engines. For this purpose, we look at the integration of emergent hardware technologies as co-processors for Big data analytics engines. Moreover, we discuss new computing paradigms, which may be also utilized for future generations of Big data analytics engines. We also take a short look at the Semantic Web, the data model of which supports mechanisms to increase the value of Big data.

We discuss open challenges and new research questions with special focus on Big data analytics engines for new computing paradigms like fog and dew computing, the Semantic Internet of Things area and index accesses of hardware-accelerated Big data analytics engines.

References

1. Abdelfattah MS, Hagiescu A, Singh D (2014) Gzip on a Chip: High performance lossless data compression on FPGAs using OpenCL. In: Proceedings of the International Workshop on OpenCL 2014, IWOCL '14. ACM, New York, NY, USA, pp 4:1–4:9
2. Ahn J, Im D, Kim H (2015) Sigmr: Mapreduce-based SPARQL query processing by signature encoding and multi-way join. J Supercomput 71(10):3695–3725
3. Ajtai M, Komlós J, Szemerédi E (1983) An 0(n log n) sorting network. In: Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing, STOC '83. ACM, New York, NY, USA, pp 1–9
4. Alam M, Yoginath SB, Perumalla KS (2016) Performance of point and range queries for in-memory databases using radix trees on GPUS. In: 2016 IEEE 18th International Conference on High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS), pp 1493–1500
5. Alcantara DA, Sharf A, Abbasinejad F, Sengupta S, Mitzenmacher M, Owens JD, Amenta N (2009) Real-time parallel hashing on the gpu. ACM Trans Graph 28(5):154
6. Alvarez V, Richter S, Chen X, Dittrich J (2015) A comparison of adaptive radix trees and hash tables. In: ICDE
7. AMD (2014) Compute Cores, White Paper. http://www.amd.com/Documents/Compute_Cores_Whitepaper.pdf. Accessed 19 Feb 2018
8. Ashkiani S, Li S, Farach-Colton M, Amenta N, Owens JD (2017) GPU LSM: a dynamic dictionary data structure for the GPU. CoRR. arXiv:1707.05354. Accessed 19 Feb 2018
9. Baddar SWA-H, Batcher KE (2011) Designing sorting networks: a new paradigm. Springer, Berlin
10. Barbieri DF, Braga D, Ceri S, Della Valle E, Grossniklaus M (2010) Incremental reasoning on streams and rich background knowledge. Springer, Berlin, pp 1–15
11. Barbieri DF, Braga D, Ceri S, Valle ED, Huang Y, Tresp V, Rettinger A, Wermser H (2010) Deductive and inductive stream reasoning for semantic social media analytics. IEEE Intell Syst 25(6):32–41
12. Batcher KE (1968) Sorting networks and their applications. In: AFIPS
13. Battré D, Heine F, Höing A, Kao O (2007) On triple dissemination, forward-chaining, and load balancing in DHT based RDF stores. In: Proceedings of the 2006 International Conference on Databases, Information Systems, and Peer-to-Peer Computing. Springer, pp 343–354
14. Bender MA et al (2007) Cache-oblivious streaming B-trees. In: SPAA
15. Berners-Lee T, Hendler J, Lassila O (2001) The Semantic Web. Scientific American Magazine 284:34–43
16. Blochwitz C, Joseph JM, Pionteck T, Backasch R, Werner S, Heinrich D, Groppe S (2015) An optimized radix-tree for hardware-accelerated index generation for Semantic Web Databases. In: International Conference on ReConFigurable Computing and FPGAs (ReConFig), Cancun, Mexico, December 7–9
17. Blochwitz C, Wolff J, Joseph JM, Werner S, Heinrich D, Groppe S, Pionteck T (2017) Hardware-accelerated radix-tree based string sorting for Big data applications. In: Architecture of Computing Systems (ARCS 2017) - 30th International Conference (LNCS), vol 10172, Vienna, Austria, pp 47–58, 3–6 April 2017

18. Bonomi F, Milito R, Zhu J, Addepalli S (2012) Fog computing and its role in the internet of things. In: Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing, MCC '12. ACM, New York, pp 13–16
19. Borne K (2014) Top 10 Big Data Challenges A Serious Look at 10 Big Data Vs. Gartner. https://www.mapr.com/blog/top-10-big-data-challenges-serious-look-10-big-data-vs. Accessed 19 Feb 2018
20. Carbone P, Ewen S, Fóra G, Haridi S, Richter S, Tzoumas K (2017) State management in apache flink: consistent stateful distributed stream processing. Proc. VLDB Endow. 10(12):1718–1729
21. Chang F et al (2008) Bigtable: a distributed storage system for structured data. ACM Trans Comput Syst 26(2):4:1–4:26
22. Chazelle B, Guibas LJ (1986) Fractional cascading: I. A data structuring technique. Algorithmica 1(1):133–162
23. Chellappa R (1997) Intermediaries in cloud-computing: a new computing paradigm. In: INFORMS
24. Chen X, Chen H, Zhang N, Zhang S (2014) Sparkrdf: elastic discreted rdf graph processing engine with distributed memory. In: Proceedings of the 2014 International Conference on Posters & Demonstrations Track, ISWC-PD'14, vol 1272, pp 261–264, Aachen, Germany. CEUR-WS.org
25. Chen Y-T, Cong J, Fang Z, Lei J, Wei P (2016) When spark meets FPGAs: a case study for next-generation DNA sequencing acceleration. In: 8th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 16), Denver, CO, 2016. USENIX Association
26. Comer D (1979) Ubiquitous B-tree. ACM Comput Surv 11(2):121–137
27. Daga M, Nutter M (2012) Exploiting coarse-grained parallelism in B+ tree searches on an APU. In: High performance computing, networking, storage and analysis (SCC), 2012 SC companion. IEEE, pp 240–247
28. DataStax, Inc (2016) How is data written?. http://docs.datastax.com/en/cassandra/3.0/cassandra/dml/dmlHowDataWritten.html. Accessed 19 Feb 2018
29. Dowd M et al (1989) The periodic balanced sorting network. J ACM 36(4):738–757
30. Facebook (2015) Indexing SST files for better lookup performance. https://github.com/facebook/rocksdb/wiki/Indexing-SST-Files-for-Better-Lookup-Performance. Accessed 19 Feb 2018
31. Fisher DE, Yang S (2016) Doing more with the dew: a new approach to cloud-dew architecture. Open J Cloud Comput 3(1):8–19
32. Gaetani E, Aniello L, Baldoni R, Lombardi F, Margheri A, Sassone V (2017) Blockchain-based database to ensure data integrity in cloud computing environments. In: ITASEC, pp 146–155
33. Google (2015) Leveldb file layout and compactions. https://rawgit.com/google/leveldb/master/doc/impl.html. Accessed 19 Feb 2018
34. Graux D, Jachiet L, Genevès P, Layaïda N (2016) SPARQLGX: efficient distributed evaluation of SPARQL with apache spark. In: The Semantic Web - ISWC 2016—15th International Semantic Web Conference, Kobe, Japan, October 17–21, 2016, Proceedings, Part II, pp 80–87
35. Groppe J, Groppe S, Schleifer A, Linnemann V (Nov. 2009) LuposDate: a semantic web database system. In: Proceedings of the 18th ACM Conference on Information and Knowledge Management (ACM CIKM 2009). ACM, Hong Kong, China, pp 2083–2084
36. Groppe S (2011) Data management and query processing in semantic web databases. Springer, Berlin
37. Groppe S (2017) LUPOSDATE Semantic Web Database Management System. https://github.com/luposdate/luposdate. Accessed 3 Feb 2017
38. Groppe S, Kiencke T, Werner S, Heinrich D, Stelzner M, Gruenwald L (2014) P-luposdate: using precomputed bloom filters to speed up sparql processing in the cloud. Open J Semant Web 1(2):25–55
39. Heimel M, Saecker M, Pirk H, Manegold S, Markl V (2013) Hardware-oblivious parallelism for in-memory column-stores. Proc VLDB Endow 6(9):709–720
40. Heinrich D, Werner S, Blochwitz C, Pionteck T, Groppe S (2017) Search & update optimization of a B+ tree in a hardware aided semantic web database system. In: Proceedings of the 7th International Conference on Emerging Databases (EDB)(Lecture Notes in Electrical Engineering (LNEE)). Springer, vol 461 , pp 172–182
41. Heinrich D, Werner S, Stelzner M, Blochwitz C, Pionteck T, Groppe S (2015) Hybrid FPGA approach for a B+ tree in a semantic web database system. In: Proceedings of the 10th International Symposium on Reconfigurable Communication-Centric Systems-on-Chip (ReCoSoC 2015), Bremen, Germany, June 29–July 1 2015. IEEE
42. Idreos S, Koubarakis M (2004) Methods and applications of artificial intelligence. In: Third Hellenic Conference on AI, SETN 2004, Samos, Greece, May 5–8, 2004. Proceedings, Chapter P2P-DIET:

Ad-hoc and Continuous Queries in Peer-to-Peer Networks Using Mobile Agents. Springer, Berlin, pp 23–32
43. Idreos S, Koubarakis M, Tryfonopoulos C (2004) Advances in database technology—EDBT 2004. In: 9th International Conference on Extending Database Technology, Heraklion, Crete, Greece, March 14–18, 2004, chapter P2P-DIET: One-Time and Continuous Queries in Super-Peer Networks. Springer, Berlin, pp 851–853
44. Jung HS, Yoon CS, Lee YW, Park JW, Yun CH (2017) Processing IoT data with cloud computing for smart cities. Int J Web Appl (IJWA) 9(3):88–95
45. Kaoudi Z, Koubarakis M, Kyzirakos K, Miliaraki I, Magiridou M, Papadakis-Pesaresi A (2010) Atlas: storing, updating and querying RDF(S) data on top of DHTs. Web Semant Sci Serv Agents World Wide Web 8(4):271–277
46. Kaoudi Z, Manolescu I (2015) Rdf in the clouds: a survey. VLDB J 24(1):67–91
47. Laney D (2001) 3D Data Management: controlling data volume, velocity and variety. Gartner, http://blogs.gartner.com/doug-laney/files/2012/01/ad949-3D-Data-Management-Controlling-Data-Volume-Velocity-and-Variety.pdf. Accessed 19 Feb 2018
48. Leis V, Kemper A, Neumann T (2013) The adaptive radix tree: artful indexing for main-memory databases. In: ICDE
49. Li J, Tseng H-W, Lin C, Papakonstantinou Y, Swanson S (2016) Hippogriffdb: balancing i/o and gpu bandwidth in big data analytics. Proc. VLDB Endow. 9(14):1647–1658
50. Liang W, Yin W, Kang P, Wang L (2016) Memory efficient and high performance key-value store on FPGA using cuckoo hashing. In: FPL
51. Liarou E, Idreos S, Koubarakis M (2007) The Semantic Web. In: 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, ISWC 2007 + ASWC 2007, Busan, Korea, November 11–15, 2007. Proceedings, chapter Continuous RDF Query Processing over DHTs. Springer, Berlin, pp 324–339
52. Linked Data (2016) Linked data—connect distributed data across the Web. Accessed 4 Nov 2016
53. Liu Y, McBrien P (2017) Spowl: spark-based owl 2 reasoning materialisation. In: Proceedings of the 4th ACM SIGMOD Workshop on Algorithms and Systems for MapReduce and Beyond (BeyondMR'17)
54. LOD2 (2016) LODStats. Accessed 4 Nov 2016
55. LOD2 (2016) Welcome—LOD2—Creating knowledge out of interlinked data. Accessed 4 Nov 2016
56. Luo L, Wong MDF, Leong L (2012) Parallel implementation of r-trees on the GPU. In: 17th Asia and South Pacific Design Automation Conference, pp 353–358
57. Maarala AI, Su X, Riekki J (2017) Semantic reasoning for context-aware internet of things applications. IEEE Internet Things J 4(2):461–473
58. Mammo M, Bansal SK (2015) Distributed SPARQL over big RDF data: a comparative analysis using presto and mapreduce. In: 2015 IEEE International Congress on Big Data, New York City, NY, USA, June 27–July 2, pp 33–40
59. Mattern F, Floerkemeier C (2010) From the internet of computers to the internet of things. In: From Active Data Management to Event-based Systems and More. Springer, pp 242–259
60. McConaghy T, Marques R, Müller A, De Jonghe D, McConaghy T, McMullen G, Henderson R, Bellemare S, Granzotto A (2016) Bigchaindb: a scalable blockchain database. White paper
61. Mietz R, Groppe S, Oliver Kleine DB, Fischer S, Römer K, Pfisterer D (2013) A P2P semantic query framework for the internet of things. PIK - Praxis der Informationsverarbeitung und Kommunikation 36(2):73–79
62. Mietz R, Groppe S, Römer K, Pfisterer D (2013) Semantic models for scalable search in the internet of things. J Sens Actuator Netw 2(2):172–195
63. Moore GE (1965) Cramming more components onto integrated circuits. Electronics 38(8):114–117
64. Moore GE (1975) Progress in digital integrated electronics. In: Electron Devices Meeting, 1975 International. IEEE, vol 21, pp 11–13
65. Moore GE (2015) The man whose name means progress, the visionary engineer reflects on 50 years of Moore's Law. IEEE Spectrum: special report: 50 years of Moore's Law (Interview). Interview with Rachel Courtland. http://spectrum.ieee.org/computing/hardware/gordon-moore-the-man-whose-name-means-progress. Accessed 19 Feb 2018
66. Moscovici N, Cohen N, Petrank E (2017) A GPU-friendly skiplist algorithm. In: 26th International Conference on Parallel Architectures and Compilation Techniques (PACT). IEEE, pp 246–259
67. Mueller R, Teubner J, Alonso G (2012) Sorting networks on fpgas. VLDB J 21(1):1–23

68. Nakamoto S (2008) Bitcoin: a peer-to-peer electronic cash system. https://bitcoin.org/bitcoin.pdf. Accessed 19 Feb 2018
69. Noghabi SA, Paramasivam K, Pan Y, Ramesh N, Bringhurst J, Gupta I, Campbell RH (2017) Samza: stateful scalable stream processing at linkedin. Proc VLDB Endow 10(12):1634–1645
70. Nurvitadhi E, Sim J, Sheffield D, Mishra A, Krishnan S, Marr D (2016) Accelerating recurrent neural networks in analytics servers: comparison of FPGA, CPU, GPU, and ASIC. In: 26th International Conference on Field Programmable Logic and Applications (FPL)
71. ONeil P et al (1996) The log-structured merge-tree (LSM-tree). Acta Inform 33(4):351–385
72. Pagh R, Rodler F (2004) Cuckoo hashing. J Algorithms 51(2):122–144
73. Pirk H, Moll O, Zaharia M, Madden S (2016) Voodoo—a vector algebra for portable database performance on modern hardware. Proc VLDB Endow 9(14):1707–1718
74. Plessl C (2012) Accelerating scientific computing with massively parallel computer architectures. IMPRS Winter School, Wroclaw. http://www.imprs-dynamics.mpg.de/pdfs/Plessl_talk.pdf. Accessed 19 Feb 2018
75. Prasad SK, McDermott M, He X, Puri S (2015) Gpu-based parallel r-tree construction and querying. In: Parallel and Distributed Processing Symposium Workshop (IPDPSW), 2015 IEEE International. IEEE, pp 618–627
76. Ramaswamy L, Chen J (2011) The coquos approach to continuous queries in unstructured overlays. IEEE Trans Knowl Data Eng 23(3):463–478
77. Rupp K (2016) CPU, GPU and MIC hardware characteristics over time. Posted in blog GPGPU/MIC computing. https://www.karlrupp.net/2013/06/cpu-gpu-and-mic-hardware-characteristics-over-time/, 2013, last update
78. Ruta M, Scioscia F, Ieva S, Capurso G, Sciascio ED (2017) Semantic blockchain to improve scalability in the internet of things. Open J Internet Things, **3**(1):46–61. Special Issue: Proceedings of the International Workshop on Very Large Internet of Things (VLIoT 2017) in conjunction with the VLDB 2017 Conference in Munich, Germany
79. Schätzle A, Przyjaciel-Zablocki M, Skilevic S, Lausen G (2016) S2RDF: RDF querying with SPARQL on spark. PVLDB 9(10):804–815
80. Segal O, Colangelo P, Nasiri N, Qian Z, Margala M (2015) SparkCL: A unified programming framework for accelerators on heterogeneous clusters. CoRR. arXiv:1505.01120. Accessed 19 Feb 2018
81. Shahvarani A, Jacobsen H-A (2016) A hybrid b+-tree as solution for in-memory indexing on CPU-GPU heterogeneous computing platforms. In: Proceedings of the 2016 International Conference on Management of Data (SIGMOD), pp 1523–1538
82. Skala K, Davidovic D, Afgan E, Sovic I, Sojat Z (2015) Scalable distributed computing hierarchy: cloud, fog and dew computing. Open J Cloud Comput 2(1):16–24
83. Stone JE, Gohara D, Shi G (2010) Opencl: a parallel programming standard for heterogeneous computing systems. IEEE Des. Test 12(3):66–73
84. ter Horst HJ (2005) Completeness, decidability and complexity of entailment for RDF schema and a semantic extension involving the OWL vocabulary. Web Semant 3(2–3):79–115
85. The Apache Software Foundation (2014) Welcome to Apache Hadoop!. http://hadoop.apache.org/. Accessed 19 Feb 2018
86. The Apache Software Foundation (2016) Apache Flink: scalable stream and batch data processing. https://flink.apache.org/. Accessed 19 Feb 2018
87. The Apache Software Foundation (2016) Apache Tez–Welcome to Apache Tez. https://tez.apache.org/. Accessed 19 Feb 2018
88. The Apache Software Foundation (2017) Apache Spark—Lightning-fast cluster computing. http://spark.apache.org/. Accessed 19 Feb 2018
89. Turck M (2016) Is Big data still a thing? (The 2016 Big data landscape). Blog of Matt Turck. http://mattturck.com/2016/02/01/big-data-landscape. Accessed 19 Feb 2018
90. Khan MA, Uddin MF, Gupta N (2014) Seven v's of Big data understanding Big data to extract value. In: Proceedings of the 2014 Zone 1 Conference of the American Society for Engineering Education, pp 1–5
91. Waldrop MM (2016) The chips are down for Moores law. Nature 530(7589):144–147
92. Wang J, Park D, Papakonstantinou Y, Swanson S (2017) SSD in-storage computing for search engines. IEEE Trans Comput (**to appear**)
93. Wang Y (2016) Definition and categorization of dew computing. Open J Cloud Comput 3(1):1–7

94. Weisz G, Melber J, Wang Y, Fleming K, Nurvitadhi E, Hoe JC (2016) A study of pointer-chasing performance on shared-memory processor-fpga systems. In: Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, FPGA '16, New York, NY, USA, 2016. ACM, pp 264–273
95. Werner S (2017) Hybrid Architecture for Hardware-accelerated query processing in semantic web databases based on Runtime Reconfigurable FPGAs. PhD thesis, University of Lübeck
96. Werner S, Groppe S, Linnemann V, Pionteck T (2013) Hardware-accelerated join processing in large semantic web databases with FPGAs. In: Proceedings of the 2013 International Conference on High Performance Computing & Simulation (HPCS 2013), Helsinki, Finland, July 1–5 2013. IEEE, pp 131–138
97. Werner S, Heinrich D, Piper J, Groppe S, Backasch R, Blochwitz C, Pionteck T (2015) Automated composition and execution of hardware-accelerated operator graphs. In: Proceedings of the 10th International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC 2015), Bremen, Germany, June 29–July 1 2015. IEEE
98. Werner S, Heinrich D, Stelzner M, Groppe S, Backasch R, Pionteck T (2014) Parallel and pipelined filter operator for hardware-accelerated operator graphs in semantic web databases. In: Proceedings of the 14th IEEE International Conference on Computer and Information Technology (CIT 2014), Xian, China, September 11–13 2014. IEEE
99. Werner S, Heinrich D, Stelzner M, Linnemann V, Pionteck T, Groppe S (2016) Accelerated join evaluation in semantic web databases by using FPGAs. Concurr Comput Pract Exp 28(7):2031–2051
100. You S, Zhang J, Gruenwald L (2013) Parallel spatial query processing on gpus using r-trees. In: Proceedings of the 2Nd ACM SIGSPATIAL International Workshop on Analytics for Big Geospatial Data (BigSpatial), pp 23–31
101. Zhang H, Andersen DG, Pavlo A, Kaminsky M, Ma L, Shen R (2016) Reducing the storage overhead of main-memory oltp databases with hybrid indexes. In: Proceedings of the 2016 International Conference on Management of Data (SIGMOD), pp 1567–1581
102. Zohouri HR, Maruyama N, Smith A, Matsuda M, Matsuoka S (2016) Evaluating and optimizing opencl kernels for high performance computing with FPGAs. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC '16, Piscataway, NJ, USA. IEEE Press, pp 35:1–35:12