The HCl Scheduler: Going all-in on Heterogeneity

Michael Kaufmann
IBM Research Zurich
Institute of Telematics, KIT

Kornilios Kourtis
IBM Research Zurich

Abstract

Heterogeneity is a growing concern for scheduling on the cloud. Hardware is increasingly heterogeneous (e.g., GPUs, FPGAs, diverse I/O devices), emphasizing the need to build schedulers that identify the internal structure of applications and utilize available hardware resources to their full potential. In this paper we present our initial efforts to build a scheduler that tackles heterogeneity (in hardware and in software) as a primary concern. Our scheduler, HCl (**H**eterogeneous **Cl**uster), models applications as annotated directed acyclic graphs (DAGs), where each node represents a task. It maps tasks onto hardware nodes, also organized in DAGs. Initial results using application models based on TPC-DS queries running on Apache Spark show that HCl can improve significantly upon approaches that do not consider heterogeneity and generate schedules that approach the critical path in length.

1 Introduction

Cloud computing enables execution of large-scale distributed applications in datacenters comprising tens or hundreds of thousands of machines [9]. One problem that has drawn significant research effort over the last years is scheduling: how to allocate hardware resources to these applications [11,15,17,19,20,22,24,38,46–48].

Many cloud schedulers have been built under the assumption of hardware homogeneity, but there is an increasing need to support heterogeneity in terms of both hardware and software. Hardware is increasingly diverse. Modern datacenters include different types of hardware (cores, memory), accelerators such as GPUs [1,3] and FPGAs [8], or specialized hardware extensions (e.g., Intel's SGX [27]). Software running in the cloud is, also, heterogeneous. The cloud is used by applications with varying size, execution time, and latency requirements [20,22,46–48]. At the same time, many distributed applications consist of parts with different resource requirements [19] and different behavior.

Most of the existing schedulers either assume a homogeneous environment, or deal with heterogeneity in a limited matter (we expand on this in §5). As heterogeneity increases, both hardware and software heterogeneity need to be considered when making scheduling decisions. Doing this requires removing assumptions about the hardware and applications from the scheduler, externalize them, and provide them as input to the scheduling algorithm.

The challenge, as with most scheduling algorithms, is to reach an attractive tradeoff between the overhead of taking a decision, against the cost of making a suboptimal decision. In the presence of hardware and software heterogeneity the latter increases, leading to a significant amount of lost potential. At the same time, incorporating heterogeneity into the scheduling also increases the scheduling overhead, since more factors need to be considered.

In this paper, we take a first step and build a new scheduler, called HCl, that takes an extreme approach fully modeling heterogeneity. To that end, HCl operates on two annotated DAGs: one representing the application, and one representing the available hardware resources. Our goal is to investigate the potential benefits of considering heterogeneity in scheduling, and, if they prove significant, use it as a basis for building a scheduler that can be used in practice. We provide some background and extend on our motivation in §2. We describe HCl in §3 and perform an initial evaluation in §4. We discuss related work in §5 and conclude in §6.

2 Background and Motivation

The cloud aims to realize the concept of utility computing, i.e., provide an infrastructure that allows cloud users to use computing resources on-demand, without owning them. A primary goal of cloud computing is simplicity. As such, a homogeneous cloud is preferable. There are, however, several reasons that make supporting heterogeneity in the cloud important.

2.1 Hardware Heterogeneity

Processing Units As architects struggle to maintain exponential performance improvements with each new hardware generation, systems move towards heterogeneity [14], using specialized hardware such as GPUs and FPGAs. Embracing heterogeneity is the only way to maintain exponential performance improvements, resulting in them being increasingly adopted in datacenters. Cloud providers such as Amazon and Softlayer already offer GPU-accelerated (virtual) machines [1, 3]. Moreover, FPGAs are used for acceleration of network and application functions in Microsoft datacenters [8]. Taking this trend further, Google has built and deployed custom ASICs for speeding machine learning applications [23].

Networks and Storage Networks are getting faster. 10 Gbit/s speeds are currently considered commodity, while 40 Gbit/s and even 100 Gbit/s with and without hardware offloading are increasingly used [2]. Hence, it is not unreasonable to expect datacenters that include multiple generations of networking equipment. Storage is also increasingly diverse. Flash-based SSDs, and HDDs offer different trade-offs between performance and cost. Both are extensively used in data centers [7, 30, 36] and offered in public clouds. Furthermore, non-volatile memories [33] will increase heterogeneity. Moreover, fully exploiting the capabilities of new IO devices depends on accelerators that enable direct transfer from one device to another. One example is combining NVIDIA's GPUDirect technology with RDMA network interface cards (NIC) [39] to allow direct access from a local GPU or CPU to a remote GPU. Another one is NICs that provide offloads for storage protocols such as NVMe over Fabrics [29].

Implications for Scheduling The shift towards hardware heterogeneity creates additional challenges for scheduling. Ignoring the diversity of hardware resources is not an option, because their capabilities cannot be fully exploited. In a heterogeneous setting, it is not enough for the scheduler to answer how many resources of a given type (e.g., cores for computation) will be allocated, it also needs to determine the type and location of these resources. This results in a significantly larger search space and, therefore, a more complex problem. Moreover, application behavior changes depending on the type and quantity of the resources it uses, which should be factored in. Complicating things further, different applications have different capabilities to exploit hardware [12, 13, 26, 50]. For example, some distributed applications can benefit from fast networks while others cannot [28, 34, 44].

2.2 Software Heterogeneity

Applications as Dataflow Graphs The dominant programming model in cloud systems is expressing applications as dataflow graphs. Distributed programming frameworks such as MapReduce [10], Dryad [21], Spark [49], Naiad [32], and others [4, 25, 40] aim to provide a convenient way of writing distributed applications where the programmer does not have worry about partitioning the work, communication, fault-tolerance, and other details of distributed execution. One pattern that seems to be emerging from these systems is that expressing programs as dataflow graphs, where nodes represent computations and edges data dependencies between them, is suitable for addressing these challenges. Fortunately, as systems like Dandelion [35], TensorFlow [4], and others [31] demonstrate, the dataflow programming model provides a good basis for building distributed applications that utilize heterogeneous hardware.

Scheduling Dataflow Graphs The dataflow graph abstraction is not only useful for writing applications, but also for executing them. Musketeer [16], for example, uses a DAG of dataflow operators as a common intermediate representation to decouple the front-end and the back-end of distributed frameworks. Similar to other works [18, 19, 42], HCl accepts the application DAG as input to the scheduling algorithm.

Such a DAG representation exposes the heterogeneity of applications, i.e., that different application parts have different characteristics. A recent study [19] performs an analysis on production DAGs, showing that they are indeed heterogeneous in terms of their structure, resource demands, etc.

The heterogeneity of applications offers an opportunity to improve scheduling efficiency by allocating resources on a finer granularity than the whole application, granting better control. This is particularly important for heterogeneous hardware, so that the, potentially different variants of, specialized hardware are better utilized.

3 The HCl Scheduler

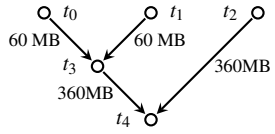
In this section we discuss how HCl schedules heterogeneous applications on heterogeneous clusters. We introduce the core concepts and the base algorithm, we present an illustrating example, and conclude with a discussion on heuristics to reduce scheduling time.

3.1 Core Concepts

HCl is based on four core concepts that allow it to address various aspects of the heterogeneous scheduling problem.

Execution Cost Awareness Like other schedulers that address resource heterogeneity [12, 13, 18, 45], HCl uses task runtime estimations (such as in figure 1b) for each

resource to find the optimal mapping for a task, and estimate future availability of resources.



(a)

Task	N_0cpu	N_1cpu	N_1gpu
t_0	4.0s	5.0s	2.0s
t_1	4.0s	5.0s	2.0s
t_2	8.0s	10.0s	5.0s
t_3	1.0s	1.5s	-
t_4	1.0s	1.5s	-

(b)

Figure 1: Example of a annotated DAG of a heterogeneous application. The edge labels in (a) represent the I/O volume between two tasks and the table (b) shows the runtimes of each task on the available resources.

I/O Cost Awareness I/O adds non-negligible overheads. Hence, similarly to other schedulers (e.g. Spark [49]), HCl tries to minimize them. Therefore, the HCl application model (Figure 1a) includes annotations about the estimated I/O volume between related tasks, and the resource graph (Figure 2) is annotated with the performance characteristics of the (network or storage) I/O system. Using this information, HCl weights the I/O and execution costs against each other, preferring a remote resource if the benefit from executing a child task on it exceeds the I/O cost of transferring its input data there, and a local resource otherwise.

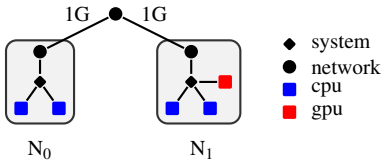


Figure 2: Example of a annotated graph for a heterogeneous cluster with two nodes, each with 2 CPUs. N_1 additionally has one GPU while the N_0 has faster CPUs.

Lookahead Scheduling HCl performs lookahead scheduling [5,6], and operates on application DAGs (see Figure 1a) that includes information such as parent/child and sibling relationships of tasks. This enables HCl to better understand the implications of its current decisions onto future task schedules. In general, a locally optimal schedule (i.e. for a single task) is not necessary globally optimal (i.e. for a parent task and its descendents). Furthermore, lookahead scheduling allows HCl to further optimize I/O costs by placing parent tasks closer to their children's optimal resource instead of only placing child tasks closer to their parents (as done, e.g., in Spark).

Path Balancing Path balancing is enabled by lookahead scheduling and means that HCl tries to balance the execution times of converging paths in a DAG such that the waiting time for tasks at convergence points is minimized. In other words, HCl tries to reduce the critical path of the executed graph. As consequence, HCl might not use a task's preferred resource even if it is available, if it determines that there is no benefit to the critical path in using said resource. Therefore the preferred resource (e.g. a fast CPU or GPU) is available for other applications (or parts of the same application) which can potentially use it more beneficially.

3.2 Scheduling Algorithm

As discussed previously, HCl works on application and resource graphs as well as runtime estimations of each task on each resource. Upon submission of an application (given as annotated DAG, such as in figure 1a), tasks are grouped into levels according to their uprank (i.e. the longest distance to an input task)[1]. Tasks on the same level can can be executed concurrently and do not have any data dependencies to each other. Once the application tasks have been sorted by their uprank (level), HCl performs an exhaustive search, i.e. it recursively evaluates all possible combinations of task to node mappings in a depth-first fashion in order to find shortest path through the search space considering task runtimes on the selected resources as well as all I/O costs between tasks in different levels. The shortest path represents the optimal schedule for the application. A simplified pseudo code of the basic algorithm is shown in Figure 3.

```
map(level, prev):
  if (level > maxLevel): return 0
  costs  = execution_costs(this)
  costs += io_costs(prev, this)
  forall (tasks[level+1] -> resources):
    next_costs = min(next_costs,
                     map(level+1, this))
  return costs + next_costs
```

Figure 3: Pseudocode of the basic HCl algorithm

Using the application DAG and the task runtimes in 1 and the cluster in figure 2 as example, HCl would first sort tasks t_0, t_1 and t_2 into level 0, t_3 into level 1 and t_4 into level 2 and continue from there to find all possible mappings of $\{t_0,t_1,t_2\} \rightarrow \{N_{0cpu},N_{1cpu},N_{1gpu}\}$ and for each of those mappings recursively all mappings for $\{t_3\} \rightarrow \{N_{0cpu},N_{1cpu}\}$ and $\{t_4\} \rightarrow \{N_{0cpu},N_{1cpu}\}$. For example, the schedule $\{t_0 \rightarrow N_{1gpu}, t_1 \rightarrow N_{0cpu}, t_2 \rightarrow N_{0cpu}\}$, $\{t_3 \rightarrow N_{0cpu}\}$, $\{t_4 \rightarrow N_{0cpu}\}$ is runtime optimal

[1]This corresponds to a as soon as possible (ASAP) strategy. A as late as possible (ALAP) strategy would also be possible and we intend to investigate this in future work.

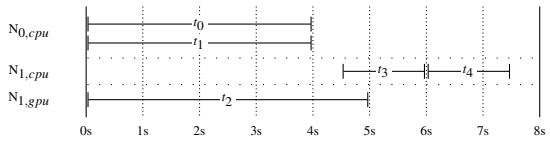


Figure 4: Globally optimal schedule for the application (see 1a) on the cluster (see 2) as computed by HCl w.r.t. to task runtimes (1b), I/O costs and task relations.

for each task (given that there is only a single GPU). Due to its lookahead and path balancing feature, HCl notices that assigning the GPU to t_0 (or t_1) does not allow t_3 to start any earlier than 4s because one of its parents can never finish earlier than that. Instead HCl is uses the GPU for t_2 which reduces t_2's runtime to 5s.

Furthermore, HCl determines that the I/O volume between t_3 and its parents ($2\times$60MB) is much lower ($\approx$ 0.5s of network I/O) than between t_3 and its child t_4 (360MB) as well as the between t_2 and t_4 (360 MB), each potentially inflicting about 3s of network I/O. Since the benefits of t_2 running on the GPU are very high and moving t_4 to the faster node N_0 would also inflict 3s of network I/O, HCl schedules t_4 on N_1 and with a similar reasoning t_3 as well, despite their runtime being 0.5s longer on N_1 than on N_0. The final is shown in figure 4.

3.2.1 Heuristics

The basic HCl algorithm described above is prohibitively costly. We employ heuristics in order to reduce the complexity, and enable using HCl in practice.

DAG Partitions We partition application DAGs into sub-graphs with a limited depth ending in a convergence point. The rationale behind this is that our predictions are afflicted with a growing uncertainty further into the future. Partitions are scheduled independently from each other using the basic algorithim described above.

Node Equivalence Classes To reduce complexity we handle identical nodes with pair-wise identical communication costs as a single resource. Within a node equivalence class, we employ a simple greedy task execution and I/O cost aware homogeneous scheduler.

Task Equivalence Classes distributed application frameworks (DAFs) achieve a high level of concurrency by partitioning data and spawning multiple tasks to perform the same operation independently. We sort all tasks into buckets depending on the size of each partition and treat all tasks within one bucket as equivalent.[2]

Limited Path Exploration Instead of exploring all possible paths through the search space, we randomly select a limited number of paths at every level in the recursion. This effectively limits the runtime of the scheduling algorithm at the cost of introducing significant jitter (as can seen at the error bars in Figure 5), hence we will investigate more deterministic methods in future work.

[2]Implementation of task equvalence classes is work in progress.

4 Evaluation

In this section, we evaluate the benefits of the HCl model when dealing with heterogeneity using our prototype implementation. Our goal is twofold. First, to quantify the potential benefits of HCl, and, second, to analyse the effect of the lookahead in the behaviour of HCl. We evaluate HCl with a subset of the TPC-DS [43] queries that were executed unmodified using Apache Spark on a single nodewhile limiting the number of executors such that inter-task interference can be excluded. We then extracted task dependencies and run-times as well as I/O volumes for inter-task communication from Spark's event traces to build the application DAG for each query.

For our execution platform, we considered an 8-node cluster with 6 slow and 2 fast nodes ($\times$1.5 better performance), a simple model for a cluster where accelerators (e.g. GPUs, FPGAs, etc.) are, due to their cost and applicability, only available in a limited number of nodes. We assume that all nodes are connected by a 1G Ethernet network with a single switch.

We use two reference points. First, we compute the total time for each task in the critical path of the DAG ignoring any I/O data transfer times. This serves as a low bound. Second, we use a hardware oblivious (homogeneous) greedy scheduler that attempts to distribute tasks across all available resources. As a simple optimization, it tries to schedule child tasks on the first available node with the largest amount of local input data.

We execute each query multiple times to accomodate for randomization (e.g., the order of tasks and nodes for the H/W oblivious reference and the randomized limited path exploration for HCl). For each task, we execute a H/W oblvious schedule 10 times, and the HCl algorithm 4 times. For the latter, we use lookahead values from 0 to 4. We summarize the results in Figure 5 that depicts the execution time of the each query as the percentage of the critical path execution (i.e., our low bound).

Overall, the HCl scheduler improves the execution time of most of the evaluated queries. On average, the schedules computed by HCl are only 15% longer than the critical path of the DAG and in some cases approach it to less than 1%. In contrast to that, the H/W oblivious scheduler computed schedules that are up to 61% (48% on average) longer than the critical path. Furthermore, we observe diminishing benefits of a lookahead larger than 1 in most cases. Whether this is a property that holds for other applications is not clear and requires further evaluation. There are also cases where a larger lookahead value leads to a worse schedule. We attribute

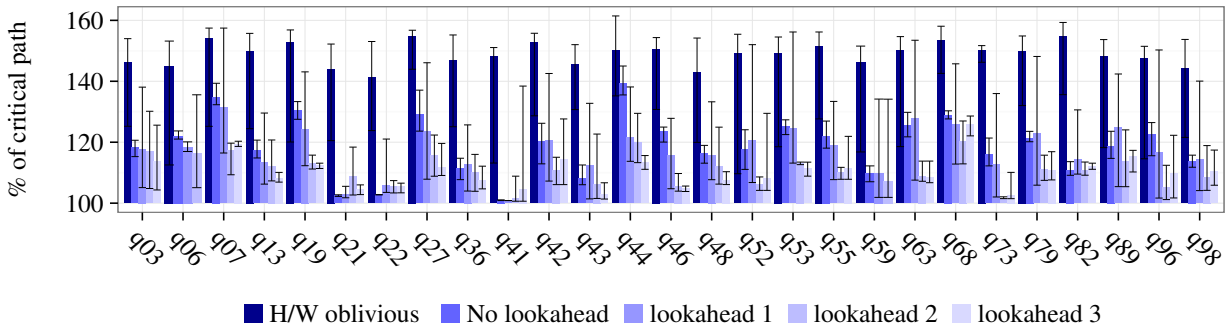


Figure 5: Execution time of different scheduling algorithms relative to the execution time of the critical path. All values are mean values relative to the critical path length with min/max error bars.

this to the randomized limited path exploration but require further investigation to determine the cause for this behavior with certainty.

5 Related work

A signifcant amount of research work tackling the problem of scheduling distributed applications on clusters.

Mesos [20] and YARN [46] are popular resource managers with limited support for heterogeneity, only allowing for annotating nodes with text labels (e.g., "GPU"), leaving it to the application to implement the desired allocation. Furthermore, they lack understanding of the application structure and task dependencies.

Quincy [22] schedules tasks by solving a min-cut/max-flow problem in a flow network that represents the scheduler state (available tasks and resources). Firmament [17] follows the same approach as Quincy, improving scalability and flexibility (e.g., by allowing arbitrary policies). Contrarily to HCl these works do not exploit the application DAG. Firmament includes a policy framework that could be used to support heterogeneity [37], but we are not aware of any works that do so.

TetriSched [45] attempts to address scheduling heterogeneous tasks and resources by forming a Mixed Integer Linear Programming (MILP) problem, and using a solver to produce an optimal schedule. TetriSched, however, is also DAG-oblivious, i.e., not aware of the task relations in the application. It does plan ahead, however, to estimate the future availability of resources.

Carbyne [18] is an altruistic scheduler, i.e. it relinquishes resources if they don't reduce the application runtime, thus allowing them to be used by other applications. Like HCl, it uses an application DAG as well as resource quantity and duration requirement estimations but it is resource and network heterogeneity oblivious. The decicion space of Carbyne is therefore one dimensional (time) and excludes the 2nd dimension (space), i.e. the possibility to choose between different non-equal resources alternatives.

Graphene [19] is DAG-aware and considers task dependencies. However, contrarily to HCl, it assumes that H/W resources are homogeneous. Graphene schedules application tasks offline in a virtual space, scheduling troublesome tasks first. An online component performs global scheduling in a greedy manner.

Quasar [13] and Paragon [12] implement methods to determine task runtimes on multiple resource types for reocurring tasks (via short benchmarks), as well as unknown tasks (via similarities with known tasks). HCl also uses runtime estimations and the methods presented in [12, 13] may be used to automatically gather this information. In contrast to HCl, these works use a greedy task scheduling and are not aware of task relations.

HEFT [41], LHEFT [6] an PEFT [5] are DAG based list schedulers that aim at reducing the makespan of DAGs on heterogeneous hardware. LHEFT and PEFT extend HEFT by employing lookahead scheduling strategies similar to HCl but while they select resources for tasks one task at a time, HCl does so on a per partition basis, which enables path balancing. Moreover, HCl implements a variety of heuristical methods to cope with the inherent complexity of DAG scheduling.

6 Final words

Motivated by the emerging trends in the cloud, we presented HCl, a scheduler that attempts to tackle heterogeneity in hardware and software as a primary concern. While our initial evaluation showed potential by produce good schedules, there are limitations that render HCl impractical. There are two main challenges to make HCl usable in practice. First, HCl needs to provide good schedules in a reasonable time (sub-second) when dealing with the amount of resources commonly found in cloud datacenters. We are tackling this problem investigating proper heuristics to reduce the search time. Second, for HCl to operate as a global scheduler it needs to support global scheduling policies (e.g., fairness) among different applications.

7 Discussion

The approach that we have chosen in HCl contains several risk factors. First and foremost, the current base algorithm is very time consuming and despite the fact that we have implemented several heuristics and have yet to implement others, there is a risk that the approach won't scale to relevant real-world sized clusters and workloads without loosing the anticipated benefits that we could show in this paper, hence the concept depends on the solution of these related problems, which we expect would also be one of the most controversial points about our work.

To date, we have also only evaluated the potential benefits for one class of applications (SQL) and it is not obvious what benefits our approach may have on other application classes, such as distributed machine learning algorithms. Furthermore, multi-application scheduling and global policies are two important aspects that we have not addressed yet.

Finally, we would be interested in learning about relevant heterogeneous application classes that might be able to benefit from the concepts used in HCl and further ideas on how to reduce the complexity, e.g. by using more directed partial path exploration techniques.

References

[1] Softlayer GPU Accelerated Computing. http://www.softlayer.com/GPU.

[2] IBM Expands High Performance Computing Capabilities in the Cloud. http://www.softlayer.com/press/ibm-expands-high-performance-computing-capabilities-cloud, July 2014.

[3] Amazon EC2 Pricing. https://aws.amazon.com/ec2/pricing/, Mar. 2016.

[4] ABADI, M., BARHAM, P., CHEN, J., CHEN, Z., DAVIS, A., DEAN, J., DEVIN, M., GHEMAWAT, S., IRVING, G., ISARD, M., KUDLUR, M., LEVENBERG, J., MONGA, R., MOORE, S., MURRAY, D. G., STEINER, B., TUCKER, P., VASUDEVAN, V., WARDEN, P., WICKE, M., YU, Y., AND ZHENG, X. Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)* (GA, 2016), USENIX Association, pp. 265–283.

[5] ARABNEJAD, H., AND BARBOSA, J. G. List scheduling algorithm for heterogeneous systems by an optimistic cost table. *IEEE Transactions on Parallel and Distributed Systems 25*, 3 (2014), 682–694.

[6] BITTENCOURT, L. F., SAKELLARIOU, R., AND MADEIRA, E. R. M. Dag scheduling using a lookahead variant of the heterogeneous earliest finish time algorithm. In *2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing* (Feb 2010), pp. 27–34.

[7] BREWER, E., YING, L., GREENFIELD, L., CYPHER, R., AND T'SO, T. Disks for data centers. Tech. rep., Google, 2016.

[8] CAULFIELD, A. M., CHUNG, E. S., PUTNAM, A., ANGEPAT, H., FOWERS, J., HASELMAN, M., HEIL, S., HUMPHREY, M., KAUR, P., KIM, J.-Y., ET AL. A cloud-scale acceleration architecture. In *Microarchitecture (MICRO), 2016 49th Annual IEEE/ACM International Symposium on* (2016), pp. 1–13.

[9] DEAN, J. The rise of cloud computing systems. https://youtu.be/4_BeSgiNoQ0, Oct. 2015. SOSP History Day.

[10] DEAN, J., AND GHEMAWAT, S. MapReduce: Simplified data processing on large clusters. *Commun. ACM 51*, 1 (Jan. 2008), 107–113.

[11] DELGADO, P., DINU, F., KERMARREC, A.-M., AND ZWAENEPOEL, W. Hawk: Hybrid datacenter scheduling. In *2015 USENIX Annual Technical Conference (USENIX ATC 15)* (Santa Clara, CA, July 2015), USENIX Association, pp. 499–510.

[12] DELIMITROU, C., AND KOZYRAKIS, C. Paragon: Qos-aware scheduling for heterogeneous datacenters. *SIGPLAN Not. 48*, 4 (Mar. 2013), 77–88.

[13] DELIMITROU, C., AND KOZYRAKIS, C. Quasar: Resource-efficient and qos-aware cluster management. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems* (New York, NY, USA, 2014), ASPLOS '14, ACM, pp. 127–144.

[14] ESMAEILZADEH, H., BLEM, E., ST. AMANT, R., SANKARALINGAM, K., AND BURGER, D. Dark silicon and the end of multicore scaling. *IEEE Micro 32*, 3 (May 2012), 122–134.

[15] GODER, A., SPIRIDONOV, A., AND WANG, Y. Bistro: Scheduling data-parallel jobs against live production systems. In *Proceedings of the 2015 USENIX Conference on Usenix Annual Technical Conference* (Berkeley, CA, USA, 2015), USENIX ATC '15, USENIX Association, pp. 459–471.

[16] GOG, I., SCHWARZKOPF, M., CROOKS, N., GROSVENOR, M. P., CLEMENT, A., AND HAND, S. Musketeer: All for one, one for all in data processing systems. In *Proceedings of the Tenth European Conference on Computer Systems* (New York, NY, USA, 2015), EuroSys '15, ACM, pp. 2:1–2:16.

[17] GOG, I., SCHWARZKOPF, M., GLEAVE, A., WATSON, R. N. M., AND HAND, S. Firmament: Fast, centralized cluster scheduling at scale. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)* (GA, Nov. 2016), USENIX Association, pp. 99–115.

[18] GRANDL, R., CHOWDHURY, M., AKELLA, A., AND ANANTHANARAYANAN, G. Altruistic scheduling in multi-resource clusters. In *Proceedings of OSDI16: 12th USENIX Symposium on Operating Systems Design and Implementation* (2016), p. 65.

[19] GRANDL, R., KANDULA, S., RAO, S., AKELLA, A., AND KULKARNI, J. Graphene: Packing and dependency-aware scheduling for data-parallel clusters. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)* (GA, Nov. 2016), USENIX Association, pp. 81–97.

[20] HINDMAN, B., KONWINSKI, A., ZAHARIA, M., GHODSI, A., JOSEPH, A. D., KATZ, R. H., SHENKER, S., AND STOICA, I. Mesos: A platform for fine-grained resource sharing in the data center. In *NSDI* (2011), vol. 11, pp. 22–22.

[21] ISARD, M., BUDIU, M., YU, Y., BIRRELL, A., AND FETTERLY, D. Dryad: Distributed data-parallel programs from sequential building blocks. In *Proceedings of the 2Nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007* (New York, NY, USA, 2007), EuroSys '07, ACM, pp. 59–72.

[22] ISARD, M., PRABHAKARAN, V., CURREY, J., WIEDER, U., TALWAR, K., AND GOLDBERG, A. Quincy: Fair scheduling for distributed computing clusters. In *Proceedings of the ACM SIGOPS 22Nd Symposium on Operating Systems Principles* (New York, NY, USA, 2009), SOSP '09, ACM, pp. 261–276.

[23] JOUPPI, N. Google supercharges machine learning tasks with TPU custom chip. https://cloudplatform.googleblog.com/2016/05/Google-supercharges-machine-learning-tasks-with-custom-chip.html.

[24] KARANASOS, K., RAO, S., CURINO, C., DOUGLAS, C., CHALIPARAMBIL, K., FUMAROLA, G. M., HEDDAYA, S., RAMAKRISHNAN, R., AND SAKALANAGA, S. Mercury: Hybrid centralized and distributed scheduling in large shared clusters. In *2015 USENIX Annual Technical Conference (USENIX ATC 15)* (Santa Clara, CA, July 2015), USENIX Association, pp. 485–497.

[25] LOW, Y., GONZALEZ, J. E., KYROLA, A., BICKSON, D., GUESTRIN, C. E., AND HELLERSTEIN, J. Graphlab: A new framework for parallel machine learning. *arXiv preprint arXiv:1408.2041* (2014).

[26] MARS, J., AND TANG, L. Whare-map: Heterogeneity in "homogeneous" warehouse-scale computers. In *Proceedings of the 40th Annual International Symposium on Computer Architecture* (New York, NY, USA, 2013), ISCA '13, ACM, pp. 619–630.

[27] MCKEEN, F., ALEXANDROVICH, I., BERENZON, A., ROZAS, C. V., SHAFI, H., SHANBHOGUE, V., AND SAVAGAONKAR, U. R. Innovative instructions and software model for isolated execution. In *Proceedings of the 2Nd International Workshop on Hardware and Architectural Support for Security and Privacy* (2013), HASP '13.

[28] MCSHERRY, F., AND SCHWARZKOP, M. The impact of fast networks on graph analytics. part 1: https://www.cl.cam.ac.uk/research/srg/netos/camsas/blog/2015-07-08-timely-pagerank-part1.html, part 2: https://www.cl.cam.ac.uk/research/srg/netos/camsas/blog/2015-07-31-timely-pagerank-part2.html, 2015.

[29] MELLANOX TECHNOLOGIES. ConnectX-5. http://www.mellanox.com/related-docs/prod_adapter_cards/PB_ConnectX-5_EN_Card.pdf.

[30] MEZA, J., WU, Q., KUMAR, S., AND MUTLU, O. A large-scale study of flash memory failures in the field. In *ACM SIGMETRICS Performance Evaluation Review* (2015), vol. 43, pp. 177–190.

[31] MITTAL, S., AND VETTER, J. S. A survey of CPU-GPU heterogeneous computing techniques. *ACM Comput. Surv. 47*, 4 (July 2015), 69:1–69:35.

[32] MURRAY, D. G., MCSHERRY, F., ISAACS, R., ISARD, M., BARHAM, P., AND ABADI, M. Naiad: A timely dataflow system. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles* (New York, NY, USA, 2013), SOSP '13, ACM, pp. 439–455.

[33] NANAVATI, M., SCHWARZKOPF, M., WIRES, J., AND WARFIELD, A. Non-volatile storage. *Commun. ACM 59*, 1 (Dec. 2015), 56–63.

[34] OUSTERHOUT, K., RASTI, R., RATNASAMY, S., SHENKER, S., AND CHUN, B.-G. Making sense of performance in data analytics frameworks. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)* (Oakland, CA, 2015), USENIX Association, pp. 293–307.

[35] ROSSBACH, C. J., YU, Y., CURREY, J., MARTIN, J.-P., AND FETTERLY, D. Dandelion: A compiler and runtime for heterogeneous systems. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles* (New York, NY, USA, 2013), SOSP '13, ACM, pp. 49–68.

[36] SCHROEDER, B., LAGISETTY, R., AND MERCHANT, A. Flash reliability in production: The expected and the unexpected. In *14th USENIX Conference on File and Storage Technologies (FAST 16)* (Santa Clara, CA, 2016), USENIX Association, pp. 67–80.

[37] SCHWARZKOPF, M. *Operating system support for warehouse-scale computing*. PhD thesis, University of Cambridge Computer Laboratory, Oct. 2015.

[38] SCHWARZKOPF, M., KONWINSKI, A., ABD-EL-MALEK, M., AND WILKES, J. Omega: Flexible, scalable schedulers for large compute clusters. In *Proceedings of the 8th ACM European Conference on Computer Systems* (New York, NY, USA, 2013), EuroSys '13, ACM, pp. 351–364.

[39] SHAINER, G., AYOUB, A., LUI, P., LIU, T., KAGAN, M., TROTT, C. R., SCANTLEN, G., AND CROZIER, P. S. The development of Mellanox/NVIDIA GPUDirect over InfiniBand—a new model for GPU to GPU communications. *Computer Science - Research and Development 26*, 3 (2011), 267–273.

[40] THUSOO, A., SARMA, J. S., JAIN, N., SHAO, Z., CHAKKA, P., ANTHONY, S., LIU, H., WYCKOFF, P., AND MURTHY, R. Hive: A warehousing solution over a map-reduce framework. *Proc. VLDB Endow. 2*, 2 (Aug. 2009), 1626–1629.

[41] TOPCUOGLU, H., HARIRI, S., AND WU, M.-Y. Task scheduling algorithms for heterogeneous processors. In *Heterogeneous Computing Workshop, 1999. (HCW '99) Proceedings. Eighth* (1999), pp. 3–14.

[42] TOPCUOGLU, H., HARIRI, S., AND WU, M.-Y. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Transactions on Parallel and Distributed Systems 13*, 3 (Mar 2002), 260–274.

[43] TPC. TPC BenchmarkDS (TPC-DS). http://www.tpc.org/tpcds/, Mar. 2017.

[44] TRIVEDI, A., STUEDI, P., PFEFFERLE, J., STOICA, R., METZLER, B., KOLTSIDAS, I., AND IOANNOU, N. On the [ir]relevance of network performance for data processing. In *8th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 16)* (Denver, CO, 2016), USENIX Association.

[45] TUMANOV, A., ZHU, T., PARK, J. W., KOZUCH, M. A., HARCHOL-BALTER, M., AND GANGER, G. R. Tetrisched: Global rescheduling with adaptive plan-ahead in dynamic heterogeneous clusters. In *Proceedings of the Eleventh European Conference on Computer Systems* (New York, NY, USA, 2016), EuroSys '16, ACM, pp. 35:1–35:16.

[46] VAVILAPALLI, V. K., MURTHY, A. C., DOUGLAS, C., AGARWAL, S., KONAR, M., EVANS, R., GRAVES, T., LOWE, J., SHAH, H., SETH, S., SAHA, B., CURINO, C., O'MALLEY, O., RADIA, S., REED, B., AND BALDESCHWIELER, E. Apache Hadoop YARN: Yet another resource negotiator. In *Proceedings of the 4th Annual Symposium on Cloud Computing* (New York, NY, USA, 2013), SOCC '13, ACM, pp. 5:1–5:16.

[47] VERMA, A., PEDROSA, L., KORUPOLU, M., OPPENHEIMER, D., TUNE, E., AND WILKES, J. Large-scale cluster management at Google with Borg. In *Proceedings of the Tenth European Conference on Computer Systems* (New York, NY, USA, 2015), EuroSys '15, ACM, pp. 18:1–18:17.

[48] ZAHARIA, M., BORTHAKUR, D., SEN SARMA, J., ELMELEEGY, K., SHENKER, S., AND STOICA, I. Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling. In *Proceedings of the 5th European Conference on Computer Systems* (New York, NY, USA, 2010), EuroSys '10, ACM, pp. 265–278.

[49] ZAHARIA, M., CHOWDHURY, M., FRANKLIN, M. J., SHENKER, S., AND STOICA, I. Spark: Cluster computing with working sets. In *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing* (Berkeley, CA, USA, 2010), HotCloud'10, USENIX Association, pp. 10–10.

[50] ZHURAVLEV, S., SAEZ, J. C., BLAGODUROV, S., FEDOROVA, A., AND PRIETO, M. Survey of scheduling techniques for addressing shared resources in multicore processors. *ACM Comput. Surv. 45*, 1 (Dec. 2012), 4:1–4:28.