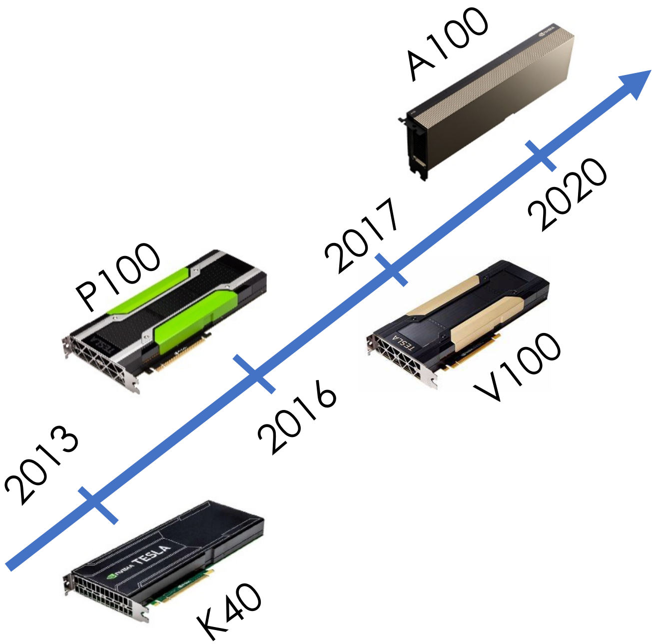
ShadowVM: Accelerating Data Plane for Data Analytics with Bare Metal CPUs and GPUs

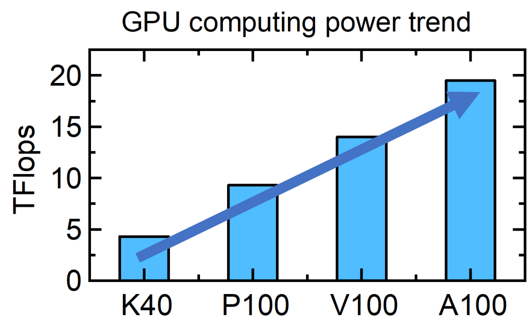
Zhifang Li, Mingcong Han, Shangwei Wu, and Chuliang Weng

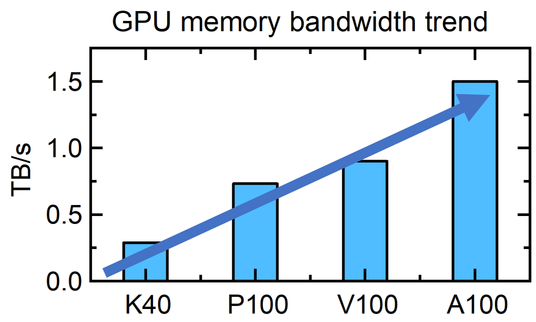
East China Normal University

Background

https://www.nvidia.com/



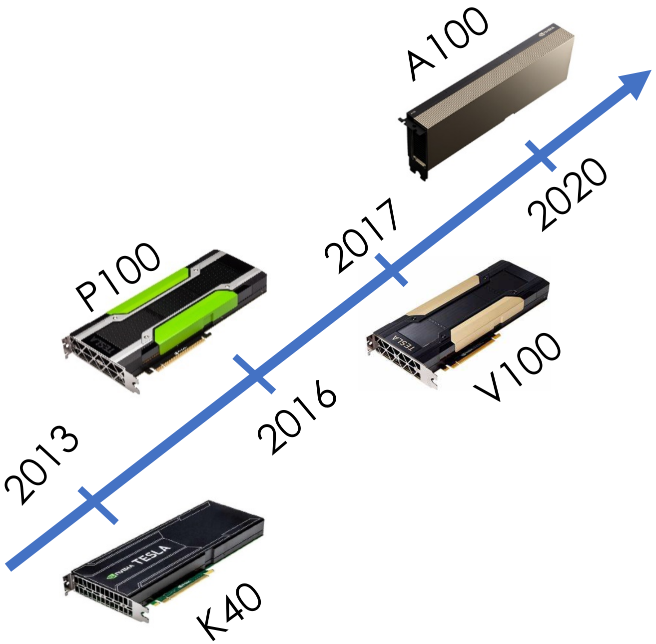


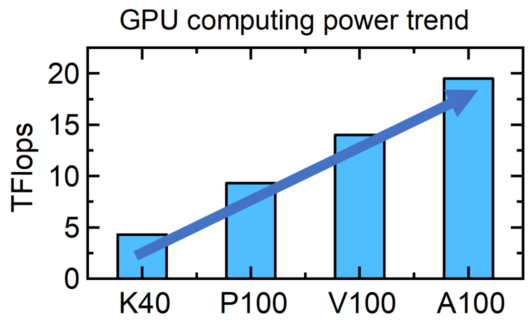


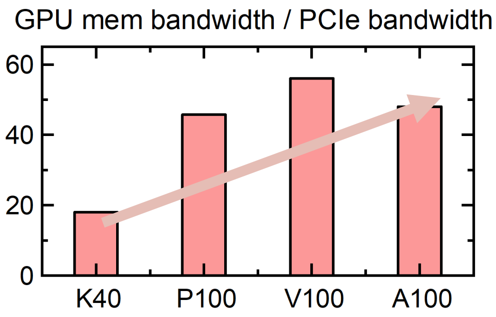
GPU itself continues to improve in the pass decade,
luckily for Graphics, HPC, Deep Learning, etc. ☺

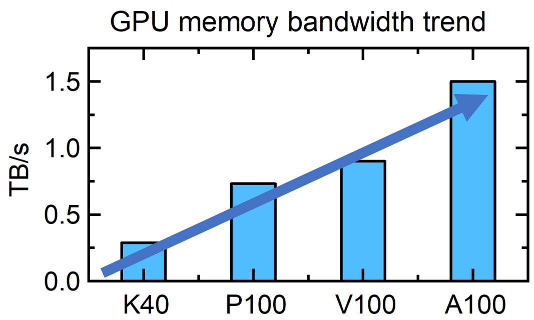
Background

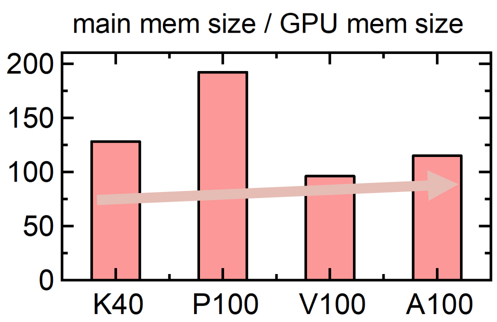
https://www.nvidia.com/







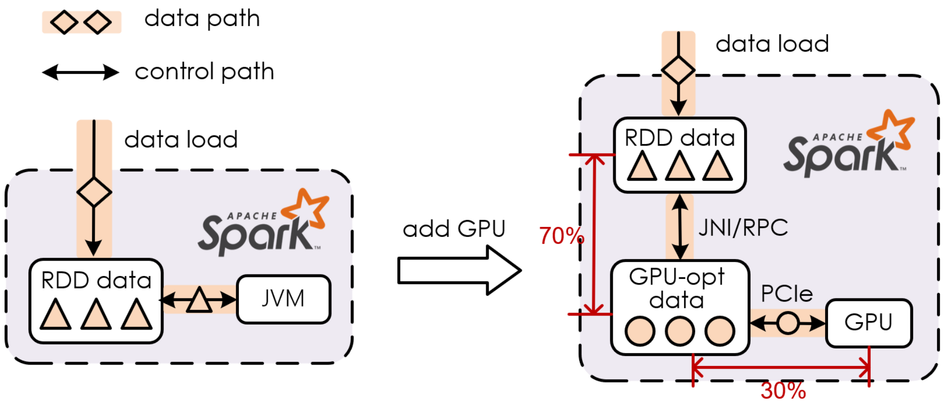




However, owing to the nature of CPU-GPU architecture,
how to exploit GPU in data analytics is still an open issue !

Motivation

- Challenge 1: a long data path moving data to GPU
  - Standard Spark abstracts partitioned data as RDD and computes RDD with JVM
  - When adding GPU to compute RDD, it involves extra JNI/RPC calls before data reaches GPU through PCIe



Spark + JVM (standard)

Spark + GPU (naive)

Motivation

- Challenge 2: current works are based on the case-by-case approach
  - Diverse analytics systems and devices will lead to tedious engineering
  - Needs to modify current analytics systems or even build new systems
  - Implementations and optimizations are hard to reuse among different combinations

N analytics systems

Spark, Flink, Hive, Apache Storm

M devices

N x M combinations !

Spark+GPU, Flink+GPU,

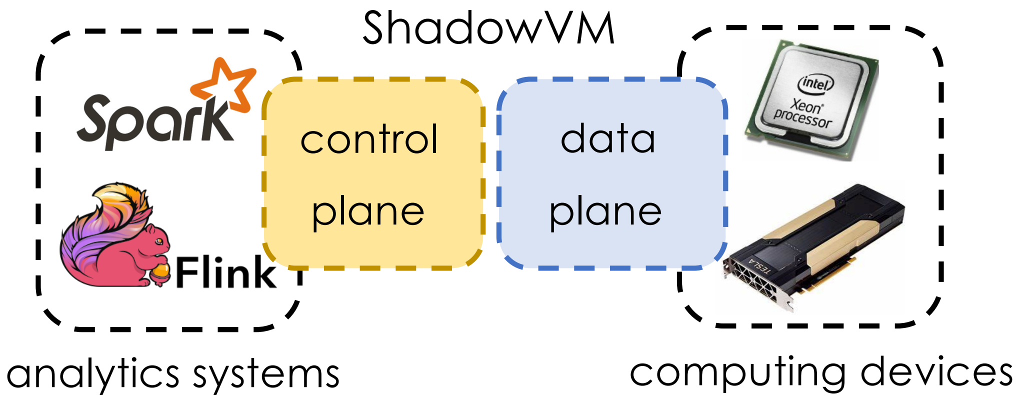
Spark+FPGA, blabla….

Motivation

- Challenge 3: the underlying optimization space of GPU is restricted by the upper analytics systems
  - RDD data is stateless, while GPU context is stateful (initialization is costly)
  - RDD processing model is row-oriented (optimized for CPU), while GPU prefers a column-oriented approach
  - RDD scheduling only consider homogeneous resources (CPU or GPU), while a heterogeneous machine contains both CPU and GPU

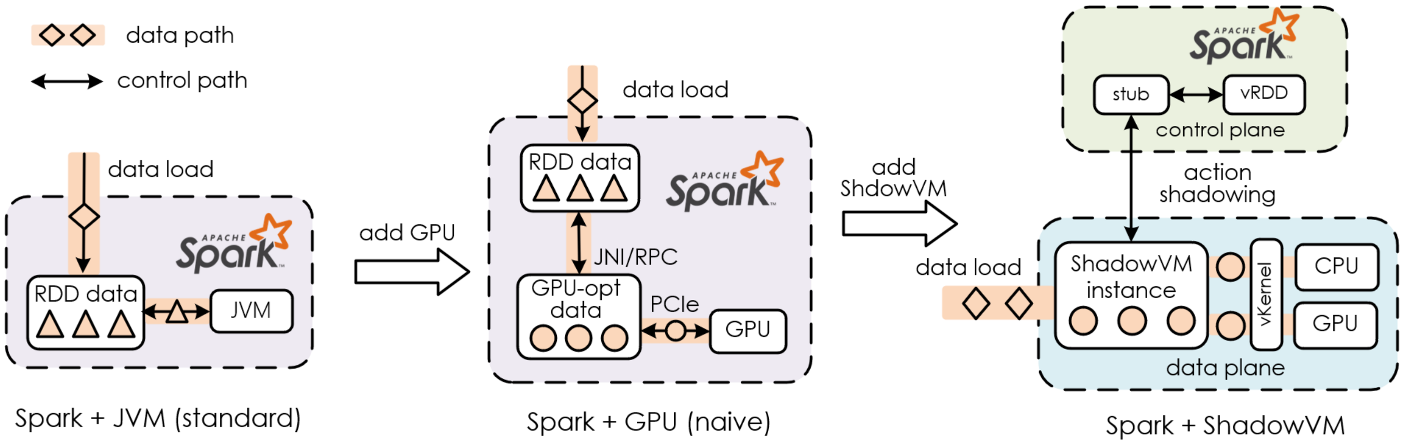
Motivation

- Our key idea to solve all challenges
  - Decouple control plane & data plane in the end-to-end data path
- ShadowVM
  - A lightweight control plane to adapt analytics systems
  - An efficient data plane to utilize CPUs and GPUs without the restrictions from the control plane



Design

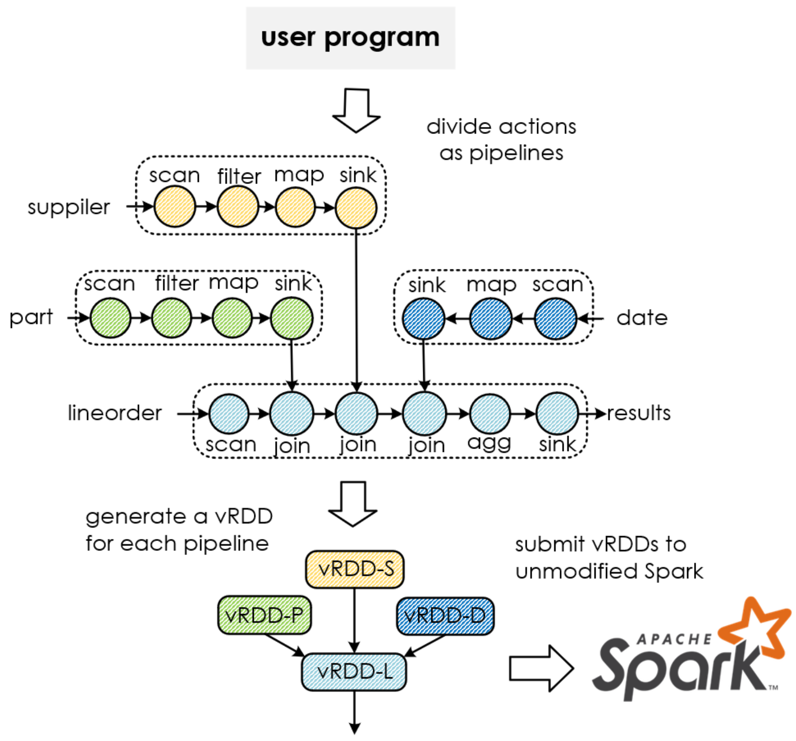
- Core abstracts in ShadowVM
  - vRDD (control plane): a "virtual" RDD that holds control information rather than real data
  - vKernel (data plane): an execution unit that runs on a pod of bare metal CPUs and GPUs
  - Action shadowing: offloading computing from vRDD to vKernel



syslab@ECNU

Design

- How to generate and execute vRDDs
  - Write a user program with ShadowVM control plane API
  - Divide computing actions as pipelines (actions in a pipeline can be executed without blocking)
  - Generate a vRDD for each pipeline (reduce pthread/kernel launching in data plane)
  - Submit vRDDs to unmodified Spark (vRDD is still a “normal” RDD)



Design

- Compare the implementations of RDD and vRDD
  - Take a simple map (one to one) as an example

```
rdd.mapPartitions(
    .......
    @Override
    public Iterator<Integer> call(Iterator<Integer> input){
        List<Integer> output = new ArrayList<>();
        /* compute each input value */
        while(input.hasNext()){
            Integer value = input.next();
            output.add(value +1);
        }
        return output.iterator();
    }
});
```

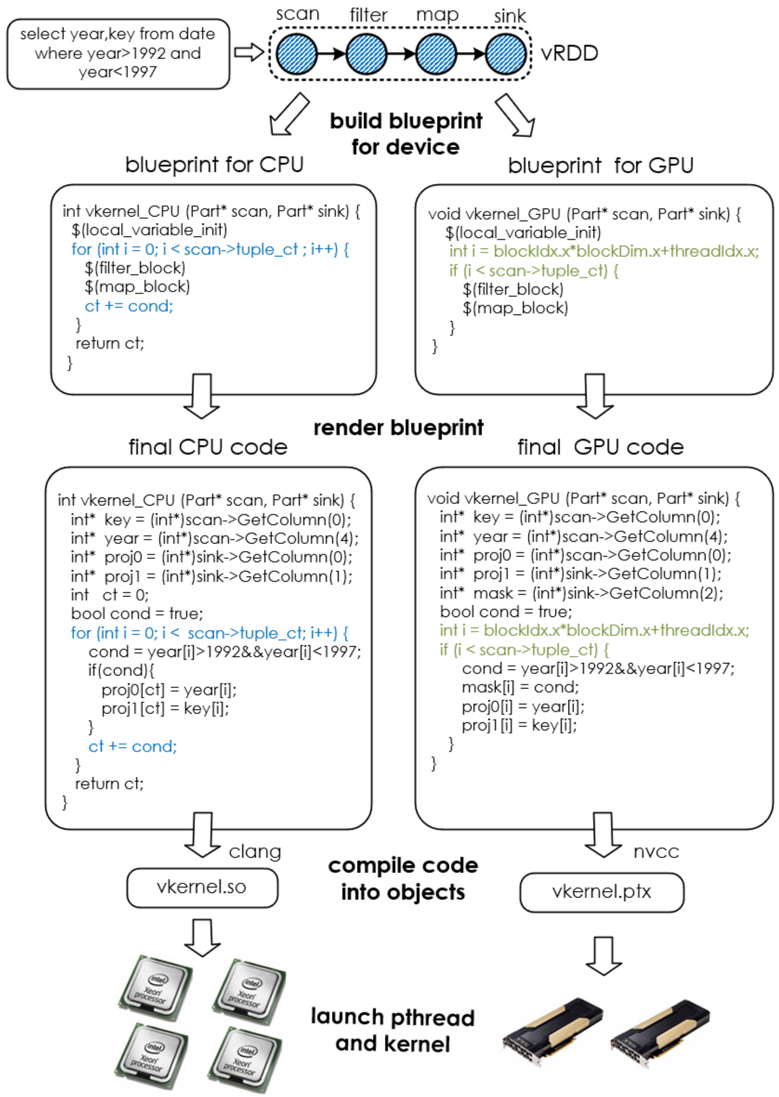
RDD: directly compute and deliver data

```
vrdd.mapPartitions(
    .......
    @Override
    public Iterator<Barrier> call(Iterator<Barrier> input){
        List<Barrier> output = new ArrayList<>();
        Barrier b = input.next();
        Pipeline pip = new Pipeline(b)
                .AddAction(b, SVM.Map, "$(0)+1" );
        /* offload computing to the data plane */
        stub.ActionShadowing(pip) ;
        /* generate barrier for the successor vRDD */
        output.add(new Barrier (pip, "Map is OK" ));
        return output.iterator();
    }
});
```

vRDD: offload computing and deliver barrier

Design

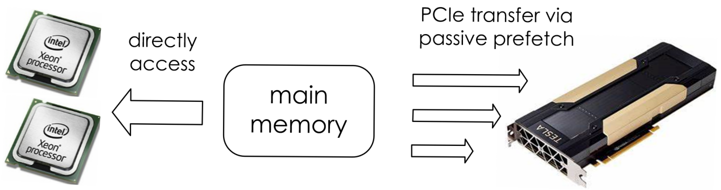
- How to execute vkernel in the data plane
  - Generate CPU and GPU code code with two steps
    1. build blueprint for device (CPU/GPU)
    2. render blueprint with implementations
  - Compile code into target objects (.so/.ptx)
  - Launch pthread for CPU and CUDA kernel for GPU
    - partitioned parallelism
    - CPU/GPU ratio is based on runtime statistics
    - speculative execution to narrow scheduling errors



syslab@ECNU

Design

- Traditional active copy: put data to GPU memory as far as possible
  - Restrict the workload size (below GPU memory)
  - Restrict the opportunity to reduce PCIe transfer
- Passive prefetch: put data in host memory and delay real transfer to vkernel runtime
  - vkernel is executed in a pipelined manner (locality is low)
  - Skip unused data when there are multiple selective actions in a pipeline (e.g., filters/joins)
  - Integrate in vkernel codegen by using UVM pointers and skip conditions for GPU thread
  - Decoupled data plane allows to reuse GPU memory context (UVM initialization is costly)



Evaluation

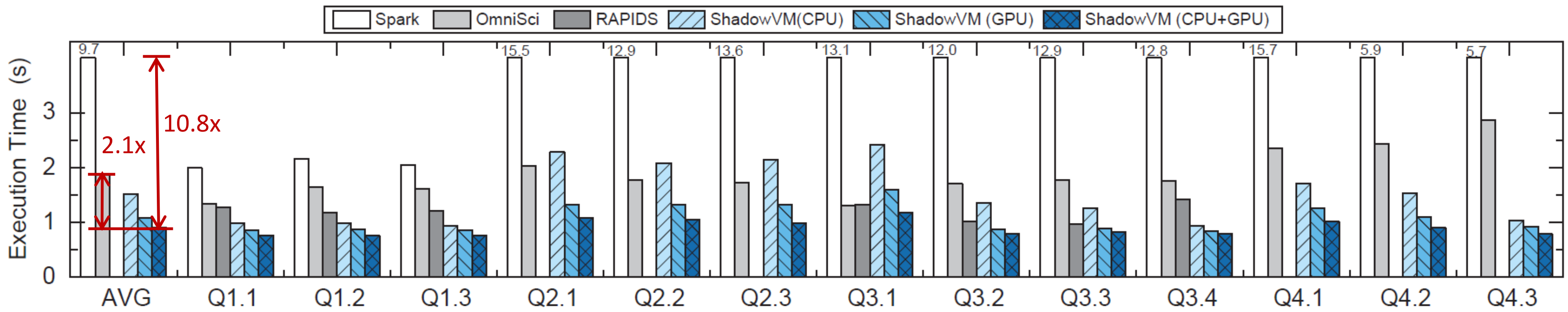
- CPU node: 16 CPU cores (Intel Xeon Silver 4110x2)
- Het node: 16 CPU cores + 1 GPU card (NVIDIA Titan RTX)
- Main memory: 128 GB
- GPU memory: 24 GB
- Ubuntu Server 18.04 + CUDA 10.0 + Clang 6.0 + OpenJDK 8.0

- 3 x CPU nodes for JVM-based **Spark**
- 1 x Het node for GPU-aware systems, including **ShadowVM**, **OmniSci**, **RAPIDS**

- Star Schema Benchmark (SSB) with all 13 analytics SQLs

Evaluation

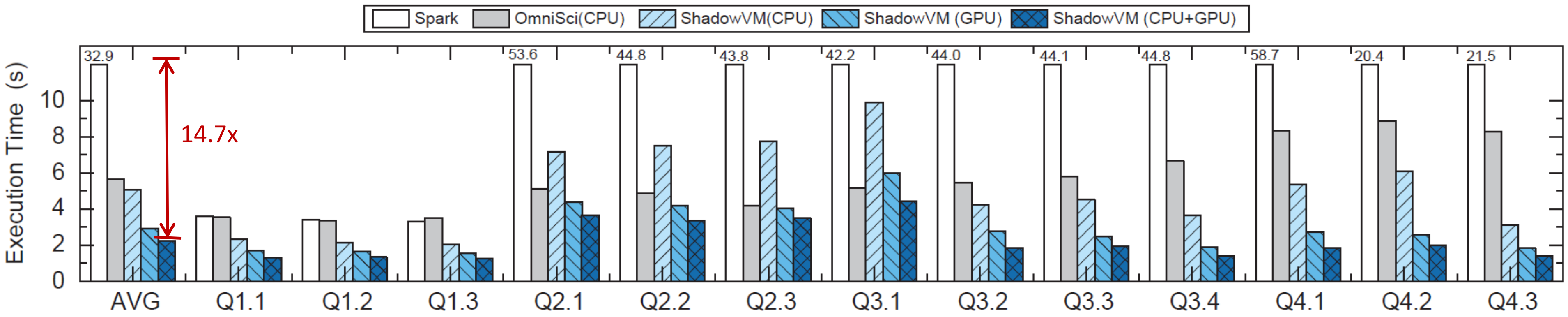
- ShadowVM outperforms Spark by 6.4×, 8.9×, and 10.8× on three modes
- ShadowVM (mixed CPU-GPU) outperforms GPU-centric OmniSci by up to 2.1×
- RAPIDS is slightly faster than OmniSci by using overlapping but fails in some SQLs



medium-scale workload (SSB, scale factor = 20GB)

Evaluation

- ShadowVM outperforms Spark by up to 14.7×
- OmniSci is switched to the CPU mode due to out of GPU memory
- RAPIDS fails due to out of GPU memory



large-scale workload (SSB, scale factor = 100GB)

syslab@ECNU

Conclusion

- ShadowVM introduces a new approach to harness data analytics systems with modern computing devices
  - A lightweight control plane to adapt upper analytics systems
  - An efficient data plane to directly utilize bare metal CPUs and GPUs without the restrictions from the control plane
  - Offloading computing from control plane to data plane
- ShadowVM code (Java control plane + CPP/CUDA data plane) is available at https://github.com/syslab-ecnu/ShadowVM

ShadowVM: Accelerating Data Plane for Data Analytics with Bare Metal CPUs and GPUs

Zhifang Li, Mingcong Han, Shangwei Wu, and Chuliang Weng

East China Normal University

Thank you
Q&A