Dynamic OpenCL

DISTRIBUTED COMPUTING ON CLOUD SCALE

FLORIAN ROESLER

OUTLINE

1. Motivation
2. Related Work
3. Basics
4. Contributions
5. Evaluation
6. Future Work
7. Conclusion

MOTIVATION

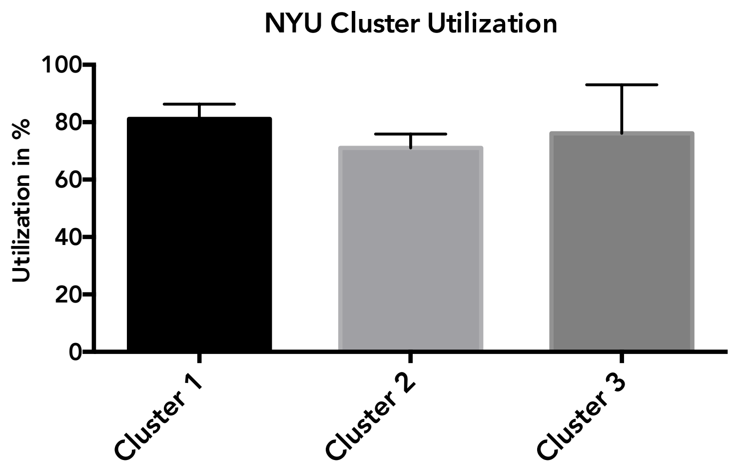
COMPUTATIONAL COMPLEXITY

- Certain computations can not be efficiently computed on a single machine
- Single-threaded code → Multi-threaded code → Distributed code
- Code complexity increases drastically

Related Technologies: MapReduce, OpenMP, MPI, CUDA, OpenCL

COST EFFICIENT CLUSTERS

- **Shared clusters face trade off scenario:**
  - Underutilization → high total costs of ownership
  - Overutilization → job queues and increased waiting time
- **Solution: dynamic resource adjustments**



RESEARCH GOALS

Build a framework that provides …

- Cluster execution of jobs on CPUs and GPUs of various vendors
- Dynamic scaling of cluster resources through cloud services
- Handling multiple simultaneous jobs efficiently by employing suitable scheduling algorithms
- Easy-to-use API in high-level language

RELATED WORK

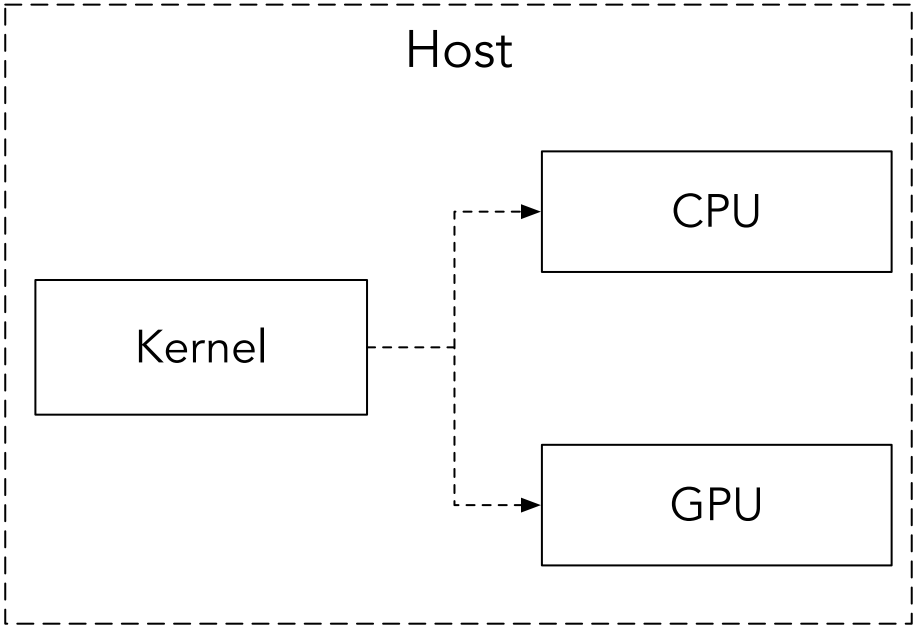
RELATED WORK

- rCUDA
- Virtualizing CUDA Enabled GPGPUs on ARM Clusters
- DistCL
- Hadoop+Aparapi

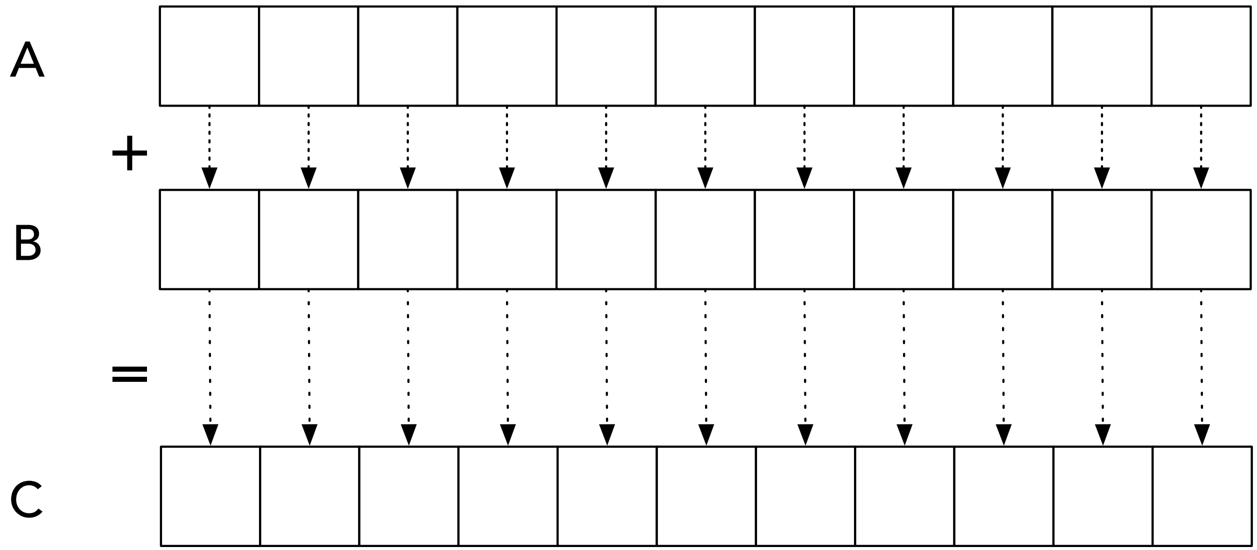
BASICS

OpenCL

- Execute parallel programs (Kernels) on heterogeneous hardware (CPU, GPU, FPGA and more)
- Kernels written in OpenCL C
- Kernels are started on host from C or C++ programs



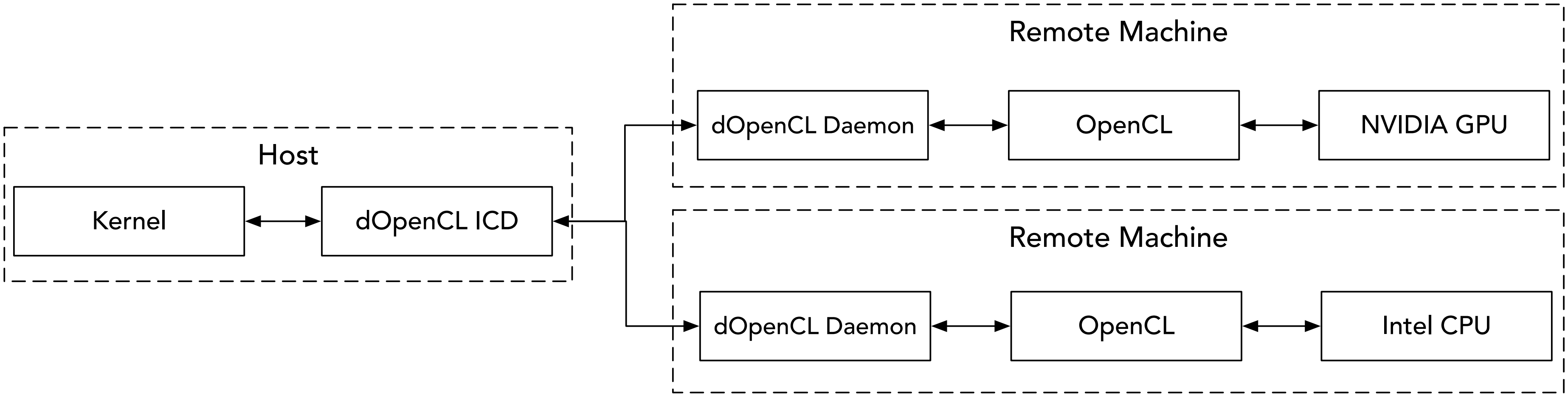
OpenCL Vector Addition Example




```
__kernel void run(__global double *a, __global double *b, __global double *c)
{
      int i = get_global_id(0);
      c[i] = a[i] + b[i];
}
```

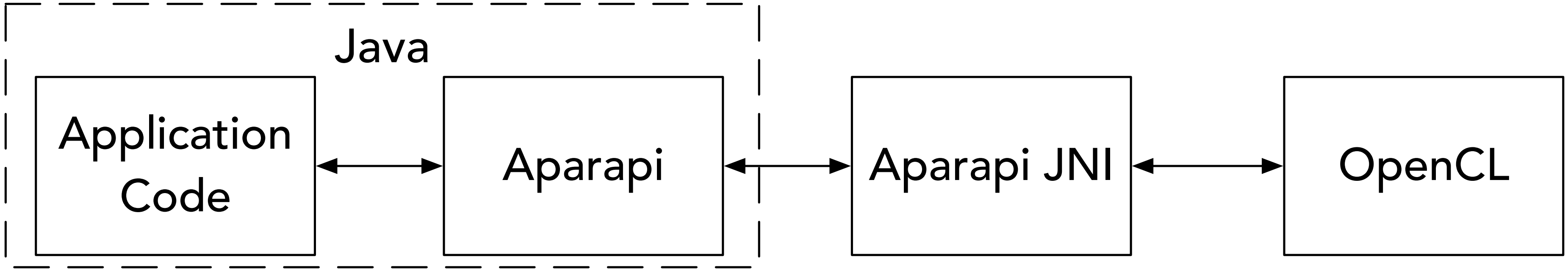
OpenCL API Forwarding (dOpenCL)

- Access OpenCL devices on remote host
- No code changes necessary
- Reduces distribution complexity



Aparapi

- Translates Java code to OpenCL Kernels
- Kernels are started from Java
- Reduces programming complexity
- Minimizes auxiliary code



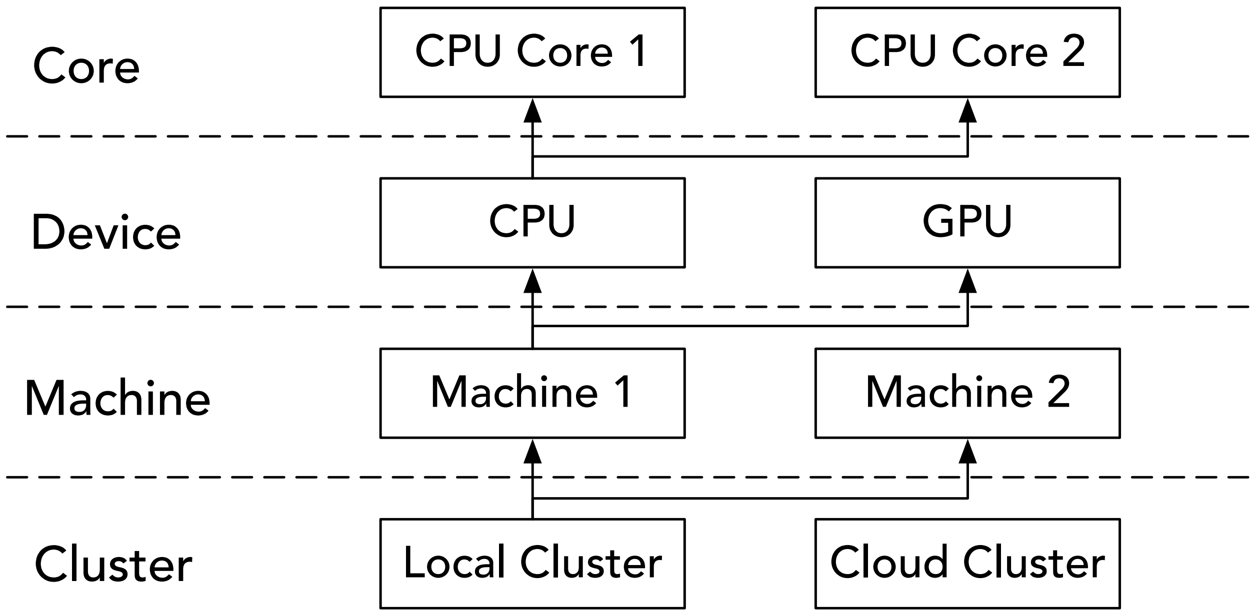
Aparapi Example

```
final double[] a = new double[]{0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
final double[] b = new double[]{0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
final double[] c = new double[10];

Kernel kernel = new Kernel() {
   @Override
   public void run() {
       int i = getGlobalId();
       c[i] = a[i] + b[i];
   }
};
kernel.execute(10);
```

CONTRIBUTIONS

LEVELS OF PARALLELIZATION



CORE & DEVICE LEVEL

Choosing OpenCL:

- Code utilizes all cores of CPUs and GPUs
- Programs are portable and follow a fixed programming model

Alternatives:

- Low-level technologies like OpenMP and MPI provide complex APIs without a fixed programming model
- Targeting GPUs requires additional solutions like OpenACC

MACHINE & CLUSTER LEVEL

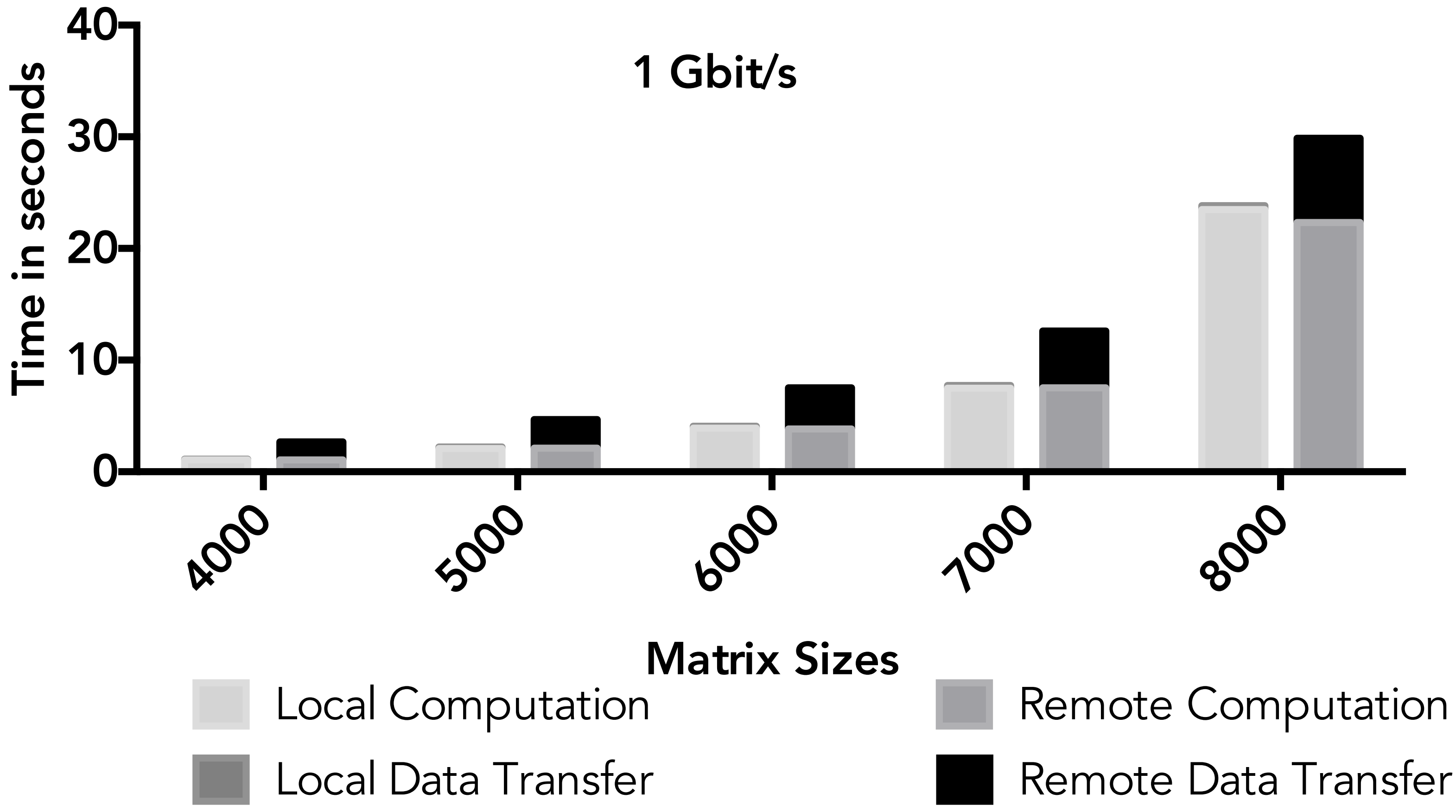
Choosing dOpenCL:

- API forwarding requires no code changes
- Minimal overhead and cluster management
- Similar libraries like SnuCL and VirtualCL could not be operated without errors

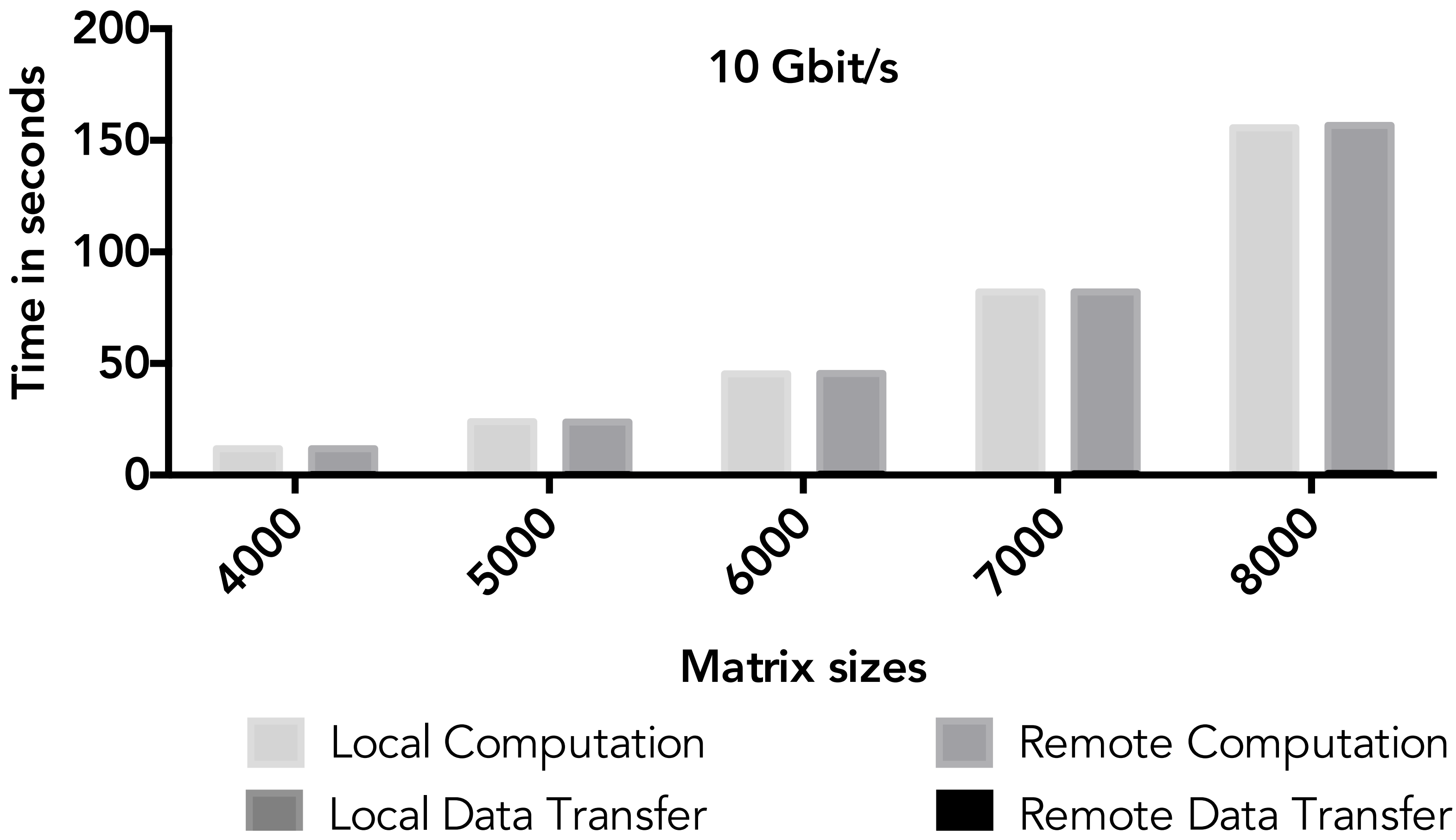
Alternatives:

- MPI increases code complexity
- MapReduce requires cluster management and adds startup/memory overhead due to JVM

EVALUATING DOPENCL



EVALUATING DOPENCL

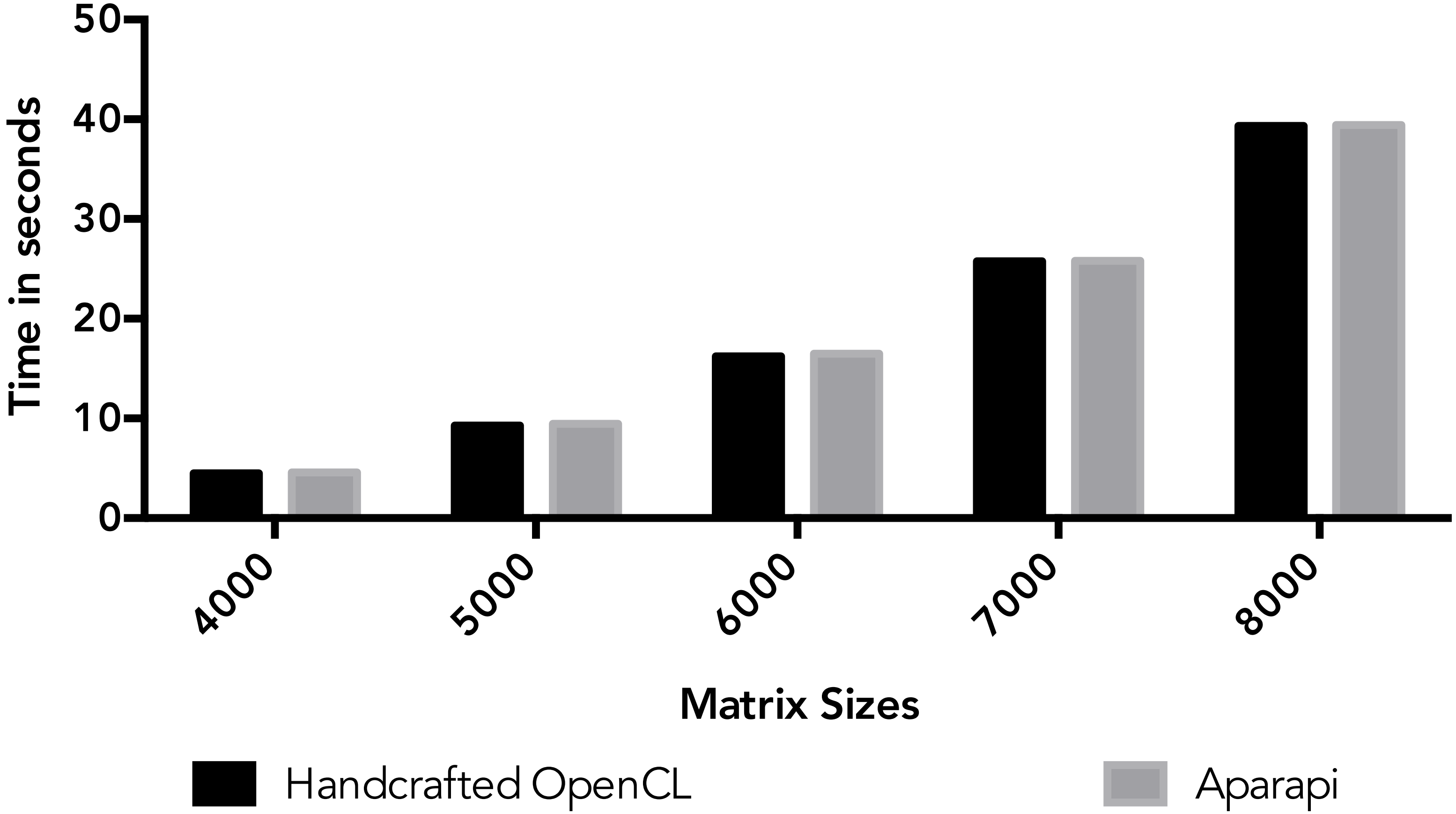


HIGH-LEVEL ABSTRACTION

- **OpenCL requires much auxiliary code for data initialization and device selection**
- **Aparapi allows to write OpenCL in Java**
  - Flatten learning curve
  - Reduce auxiliary code

But is it fast enough?

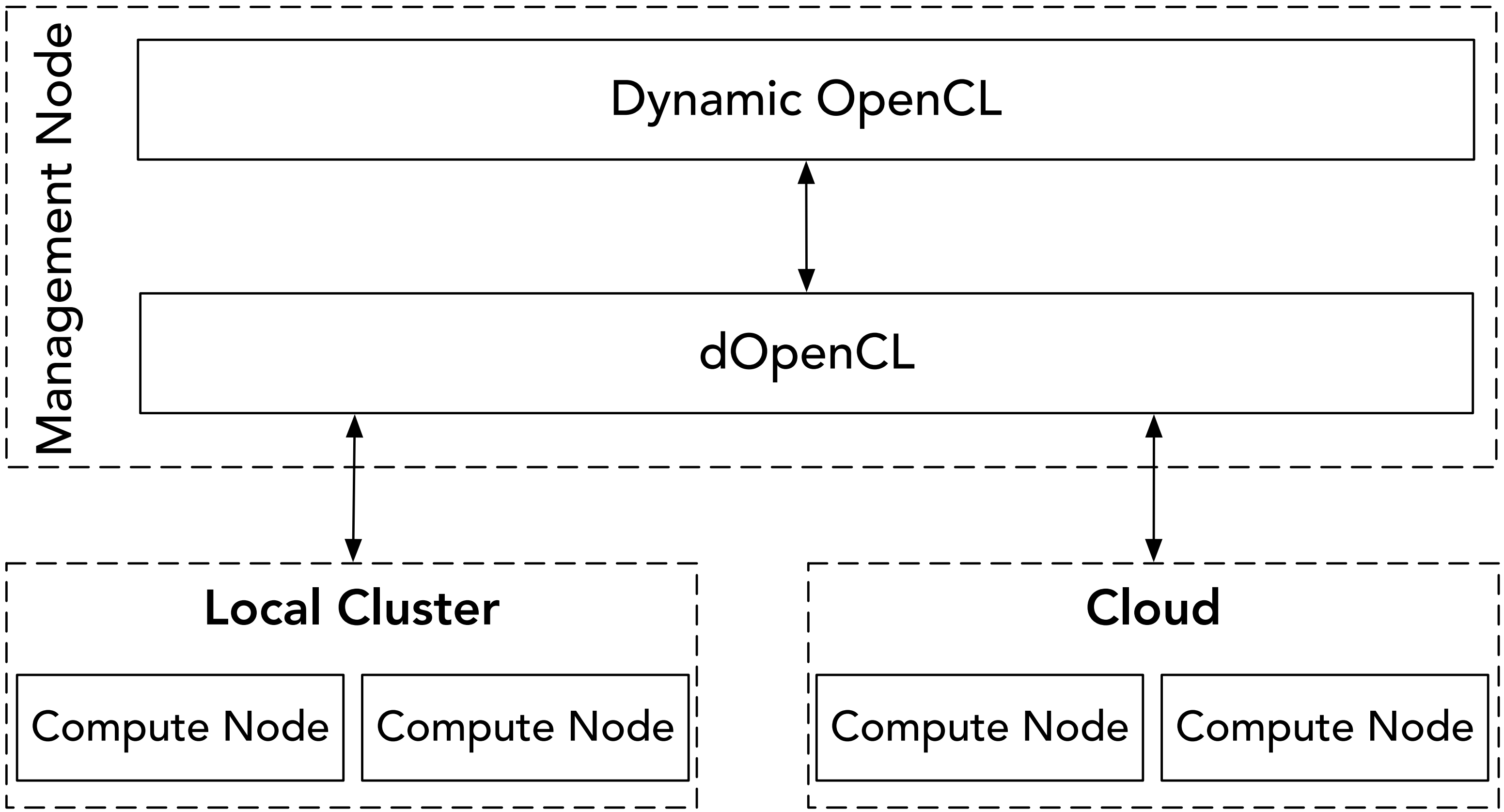
EVALUATING APARAPI



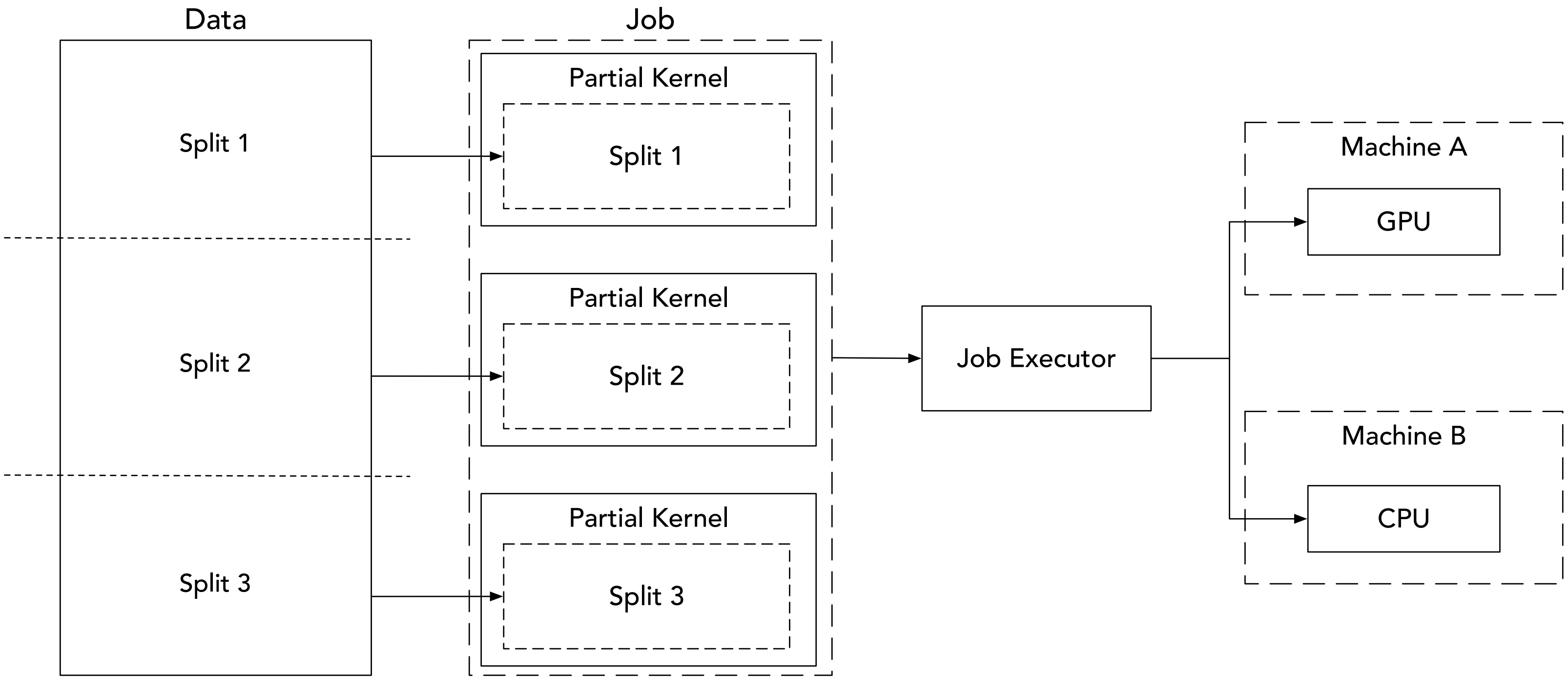
Connecting Aparapi and dOpenCL

- **Both include incompatible design decisions**
- **Forked both libraries**
- **Fixed several bugs and design decisions**
  - Dynamic resource adjustments
  - Device selection

DYNAMIC OPENCL



JOB DESIGN

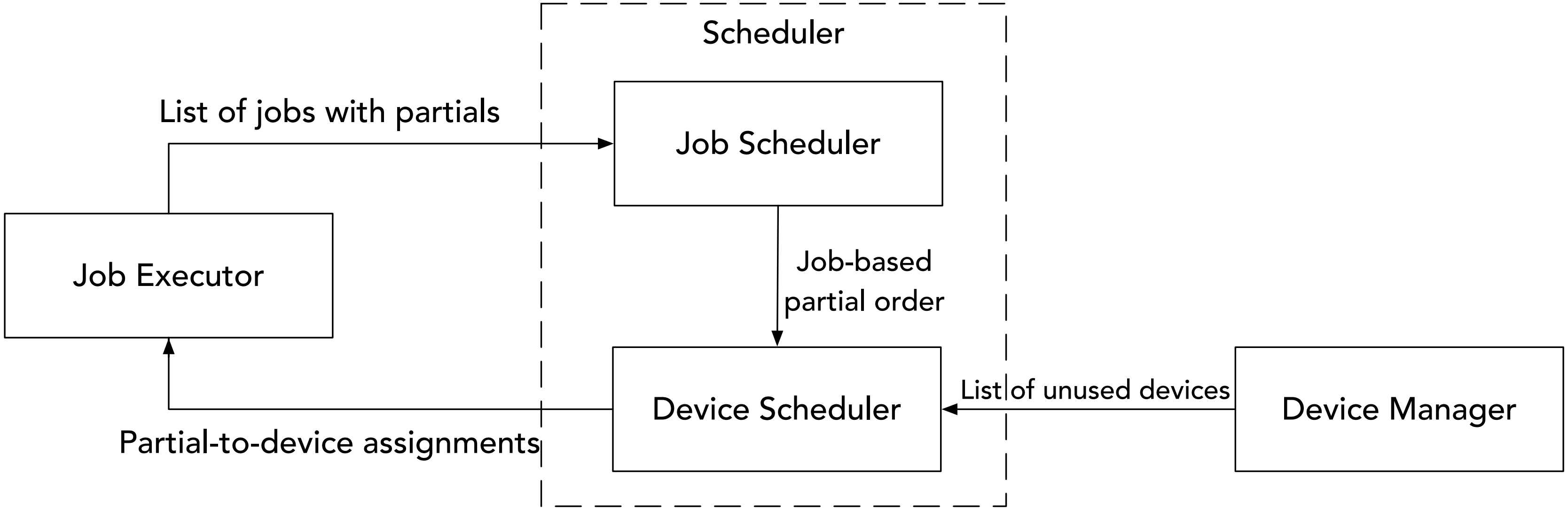

Data
Split 1
Split 2
Split 3
Job
Partial Kernel
Split 1
Partial Kernel
Split 2
Partial Kernel
Split 3
Job Executor
Machine A
GPU
Machine B
CPU

HYBRID CLUSTER

- **Created abstract class "*Machine Manager*"**
- **Handles dOpenCL cluster management**
- **One implementation per cloud service**
  - Provides cloud service communication
  - Implementations required to fill 2 methods
  - Exemplary implementation for Amazon EC2

SCHEDULING

- **Fairness vs. Efficiency**
- **Heterogeneous hardware offers optimization potential**



SCHEDULING ALGORITHMS

JOB SCHEDULER

Round-Robin

First-In-First-Out

DEVICE SCHEDULER

Device Preference

Performance Based

Network Based

USE CASES

- Job-based library
- Local cluster
- Hybrid cluster
- Cloud Cluster

EVALUATION

BENCHMARK SETUP

- **Local FSOC hardware**
- **EC2 CPUs and GPUs**
- **Local, hybrid and cloud cluster**

- **Various Computations**
  - Matrix Multiplication (data-heavy)
  - Mandelbrot Set (computation-heavy)
  - Multiple jobs in parallel: Matrix Multiplication, Mandelbrot, K-means and N-body

HARDWARE

Local Machine Type A:

144 logical cores and 1 Gbit/s Ethernet

Local Machine Type B:

8 logical cores and 10 Gbit/s Ethernet

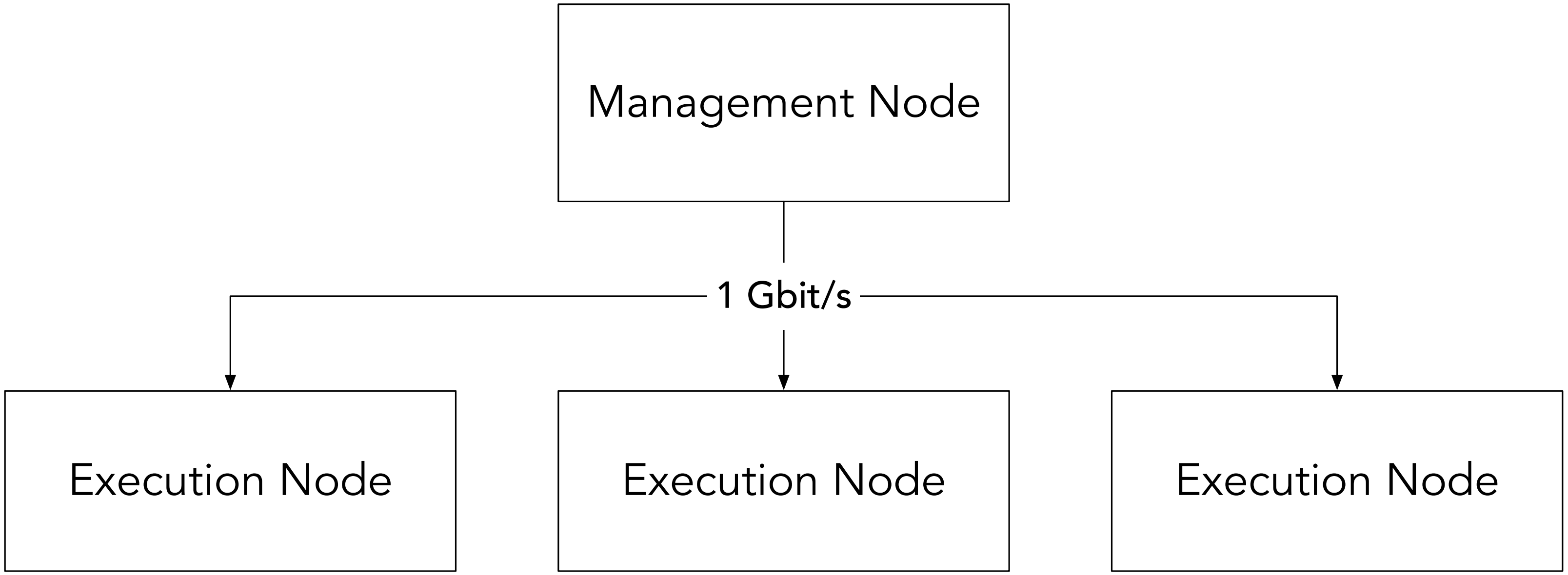
EC2 c4.8xlarge:

36 logical cores and 10 Gbit/s Ethernet

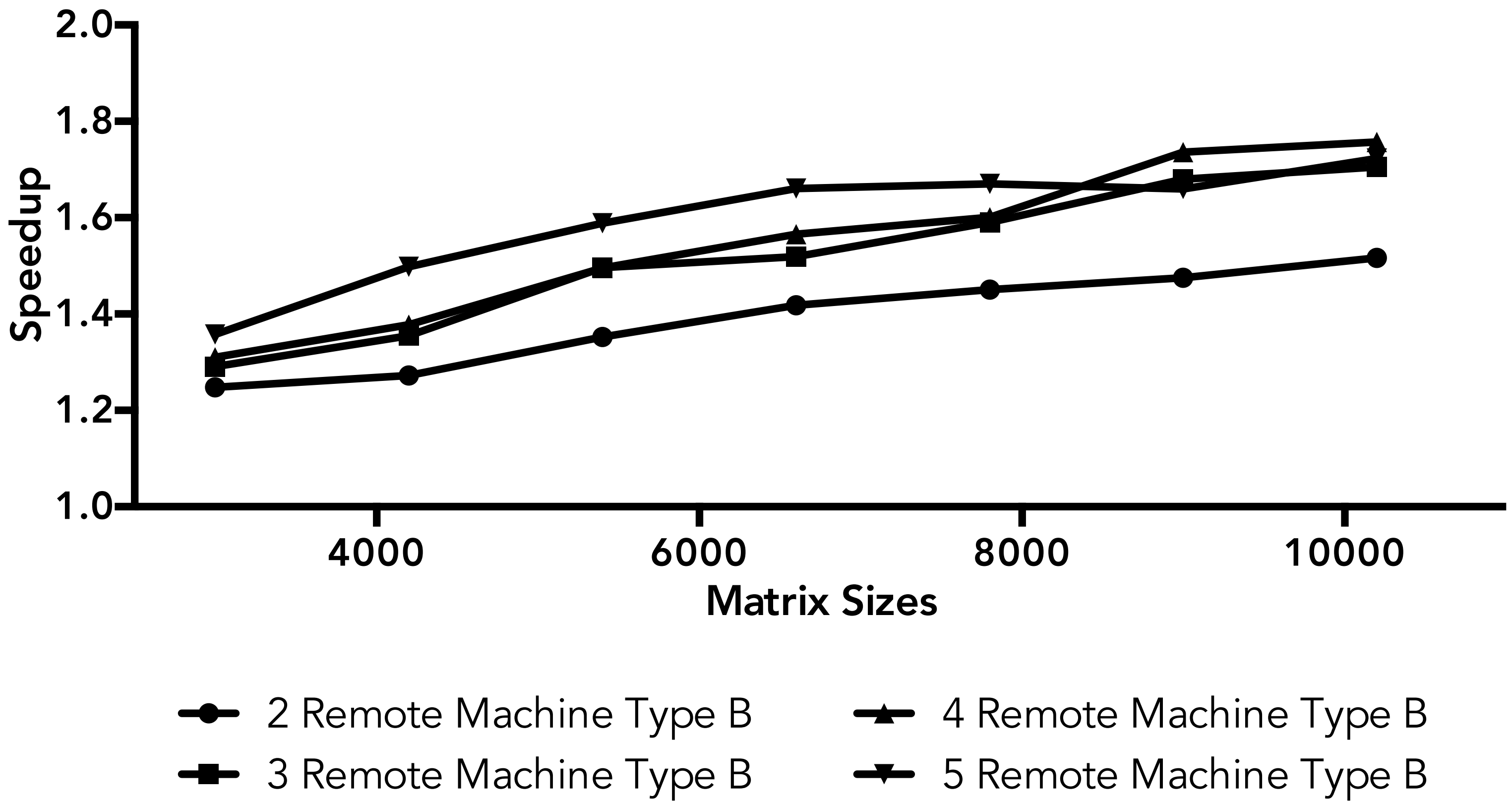
EC2 g2.2xlarge:

NVIDIA GRID K520 and 1 Gbit/s Ethernet

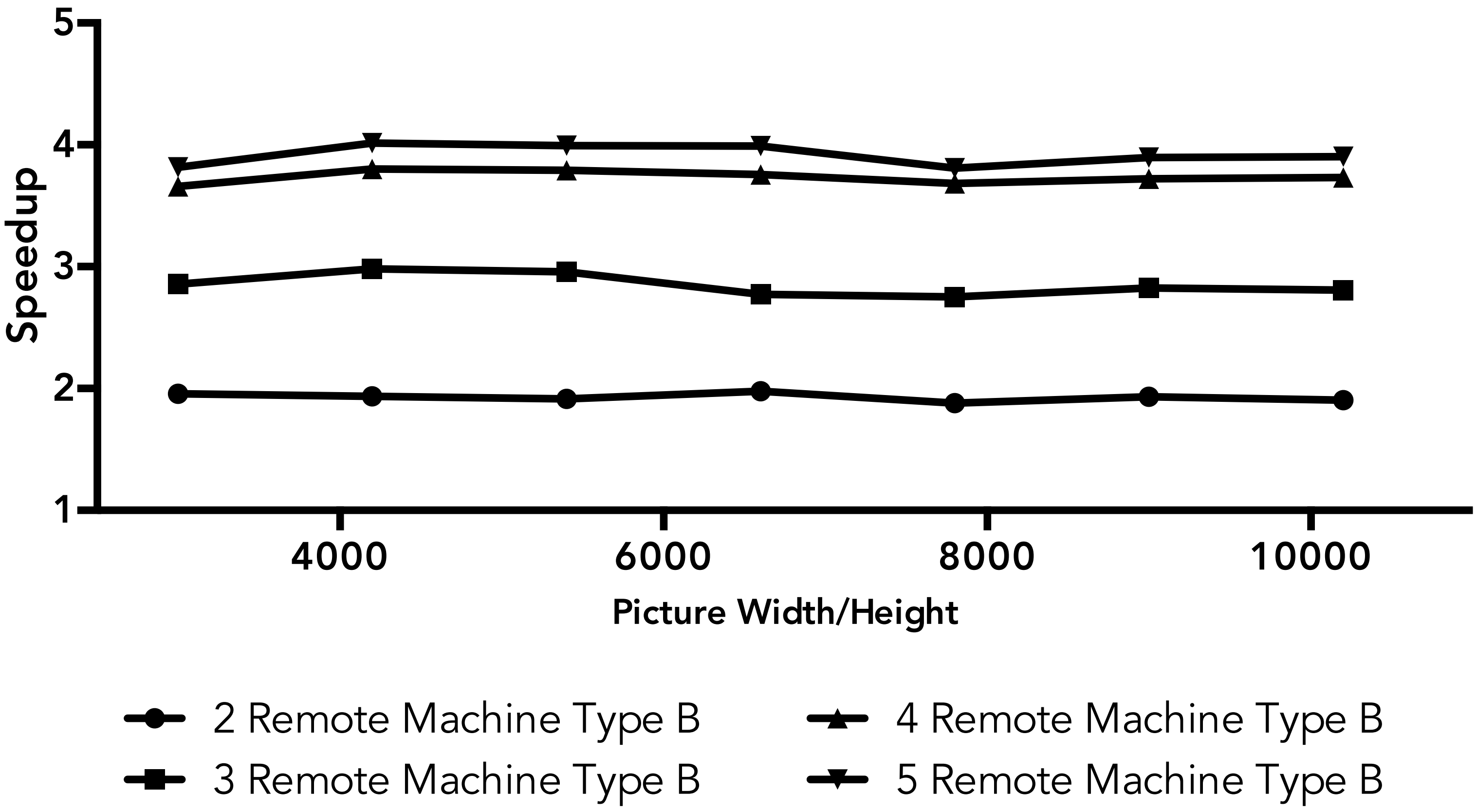
LOCAL FULLY ASSISTED SETUP



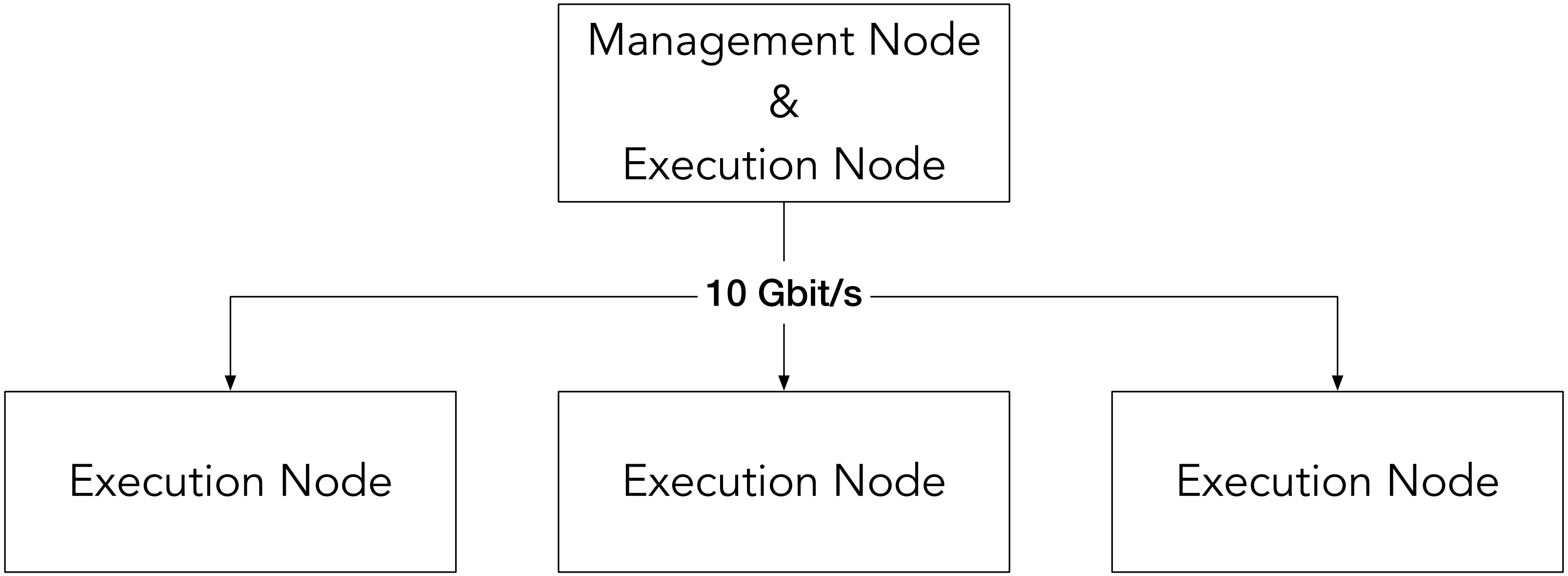
FULLY ASSISTED MATRIX MULT.



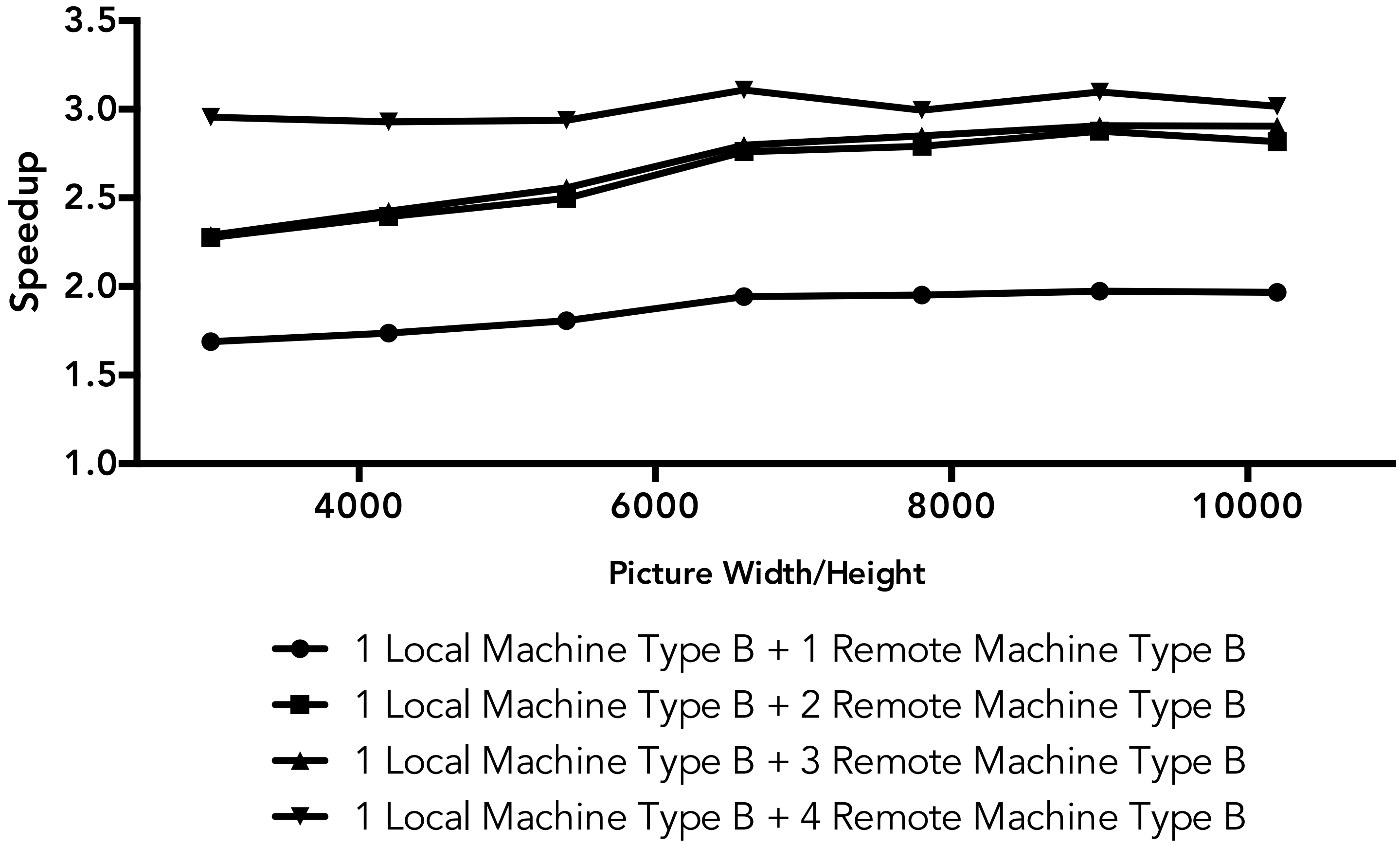
FULLY ASSISTED MANDELBROT



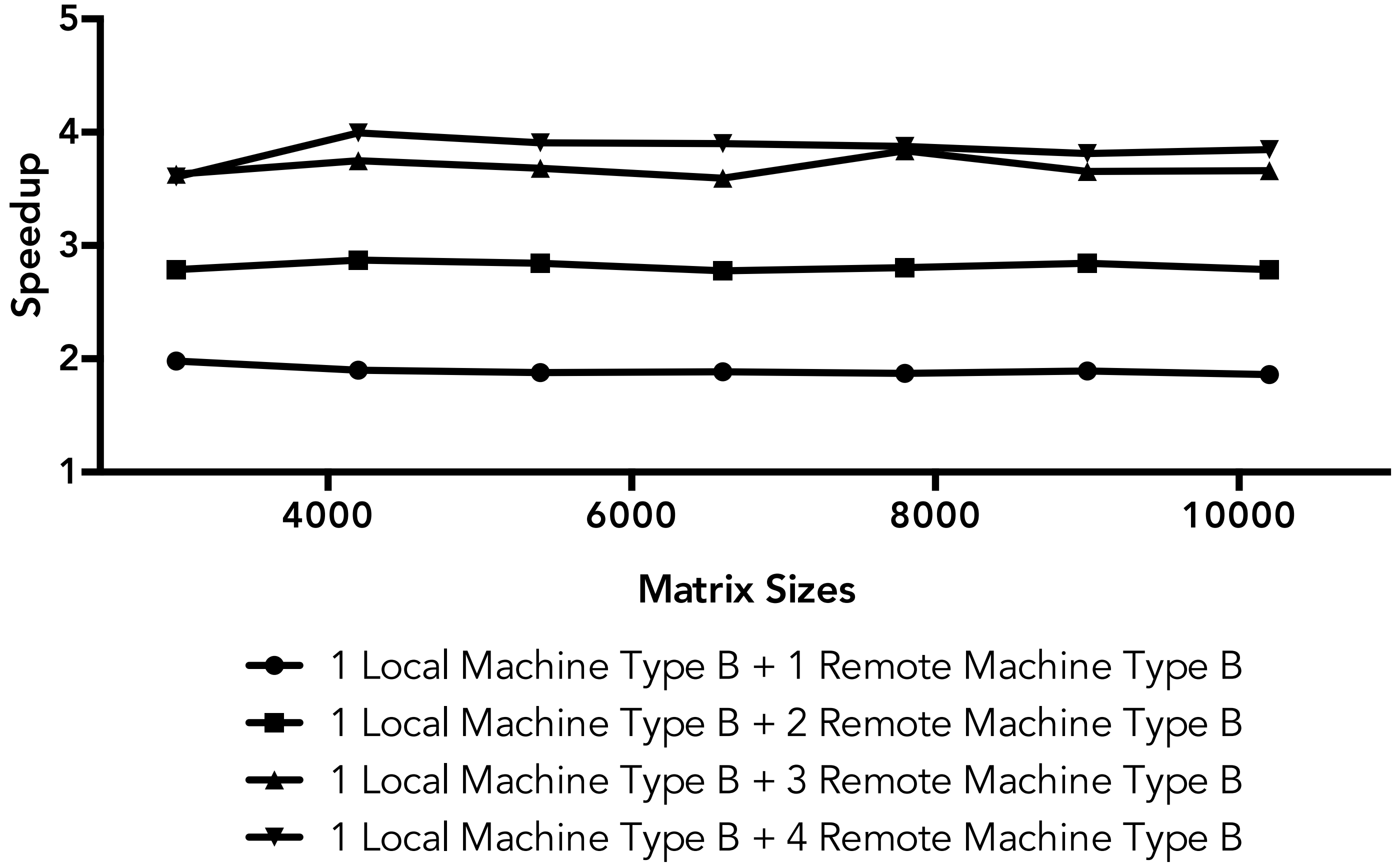
LOCAL PARTLY ASSISTED SETUP



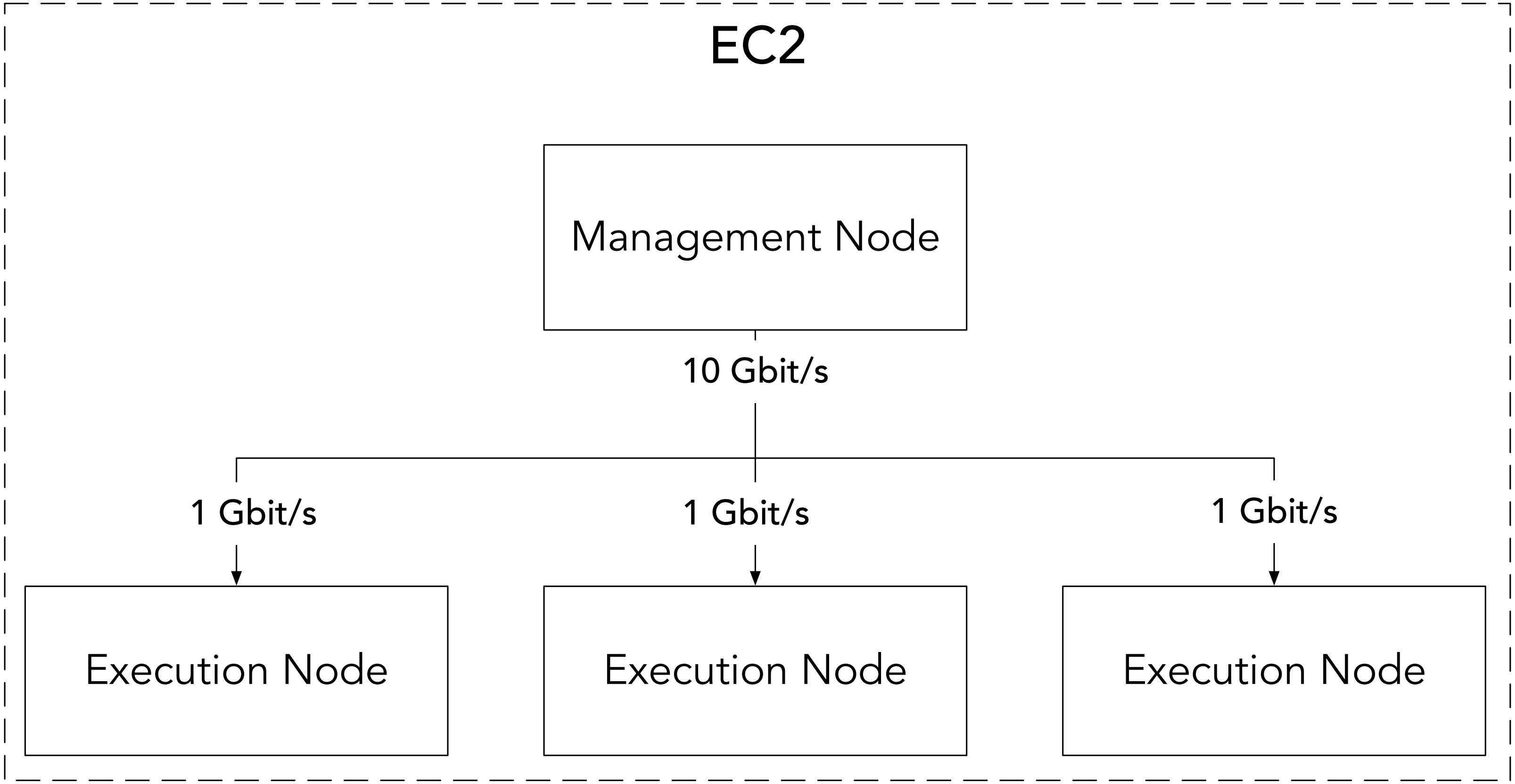
PARTLY ASSISTED MATRIX MULT.



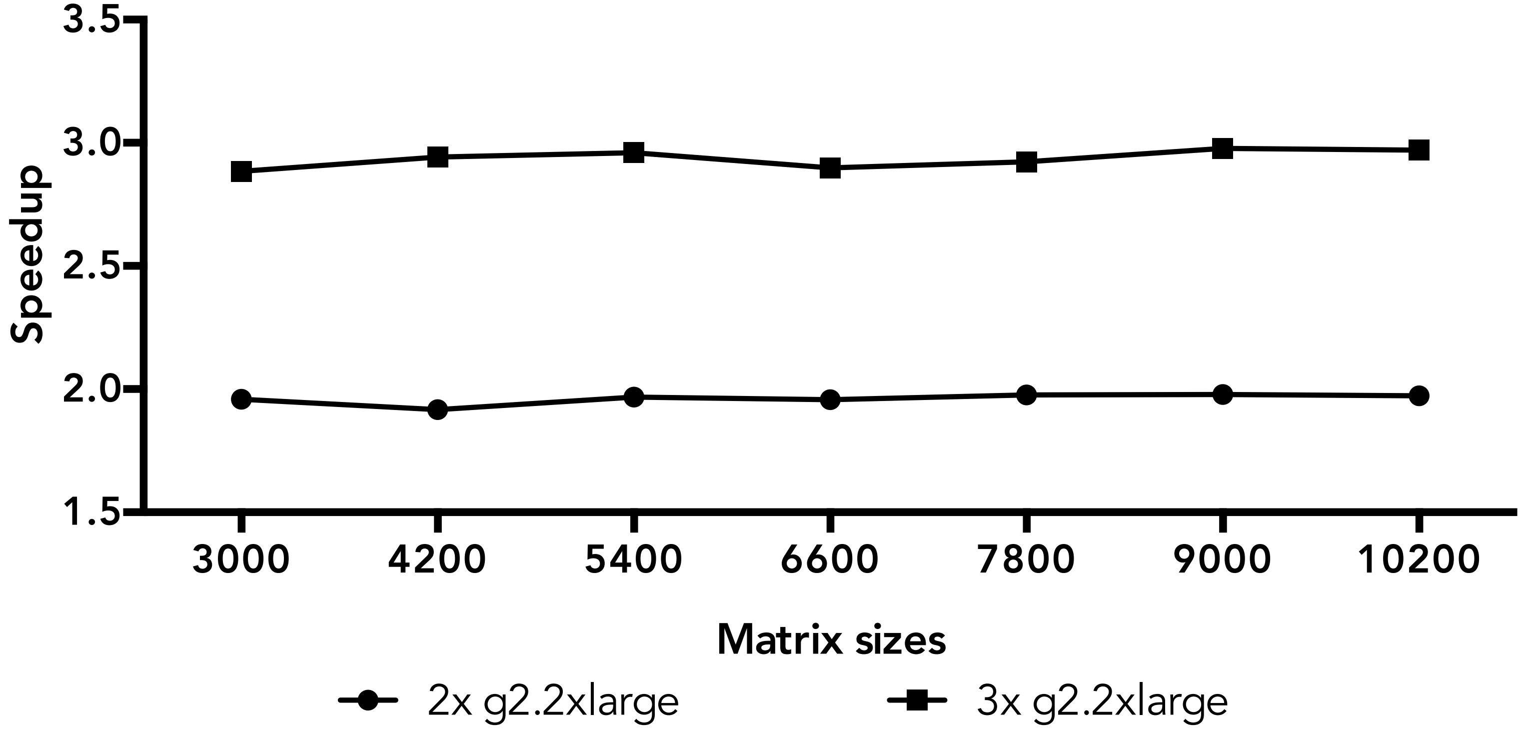
PARTLY ASSISTED MANDELBROT



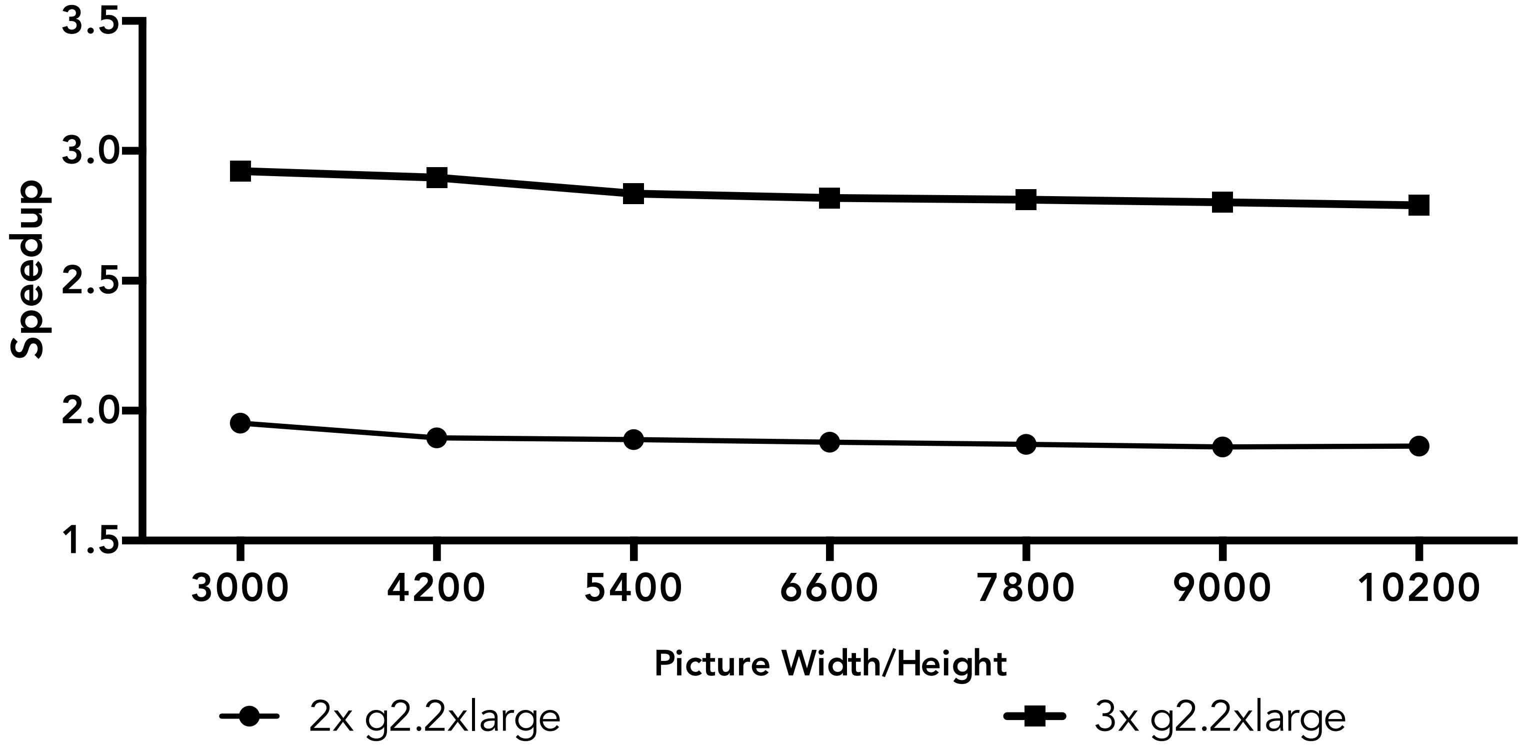
CLOUD


EC2
Management Node
10 Gbit/s
1 Gbit/s
1 Gbit/s
1 Gbit/s
Execution Node
Execution Node
Execution Node

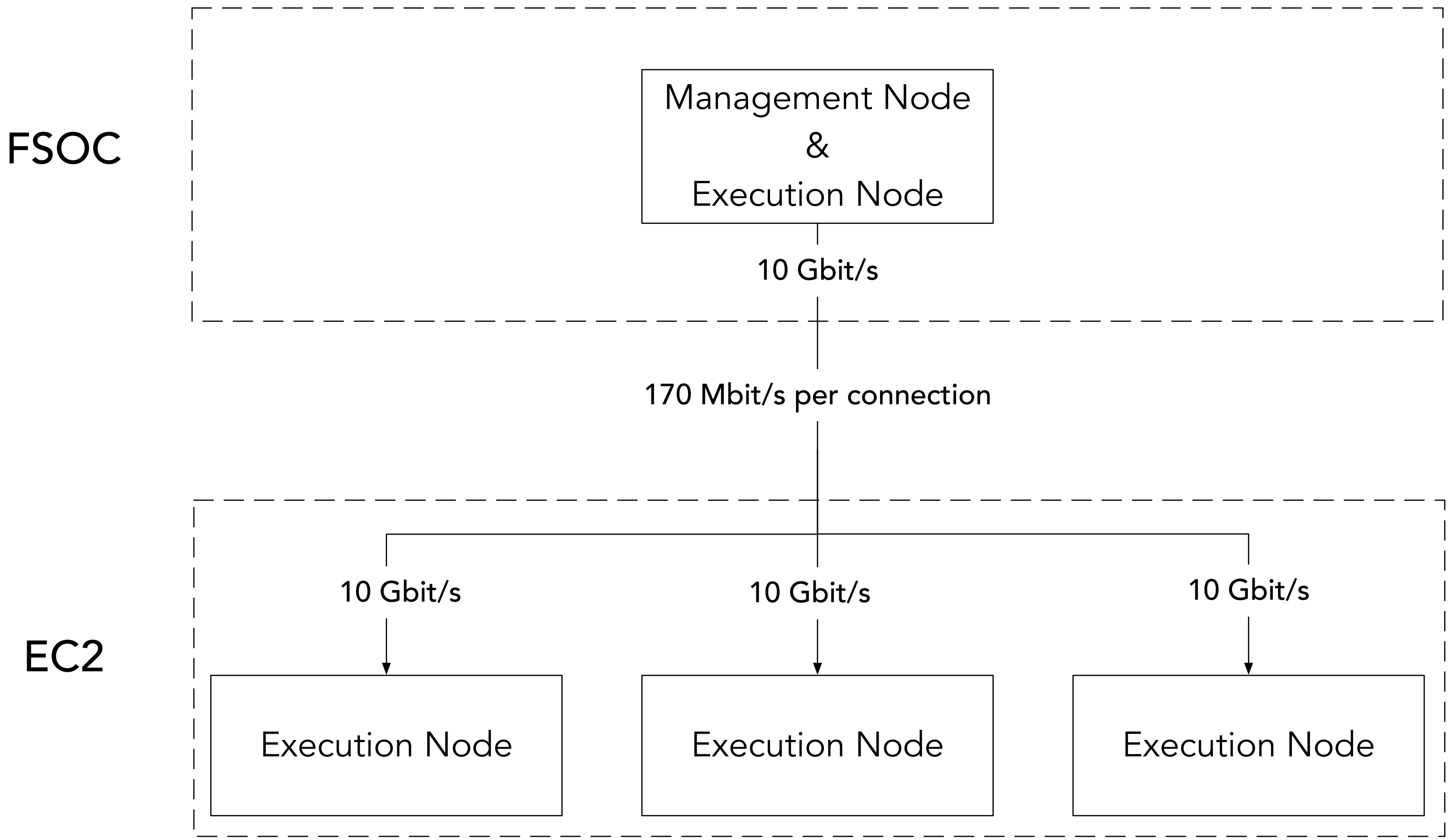
CLOUD MATRIX MULT.



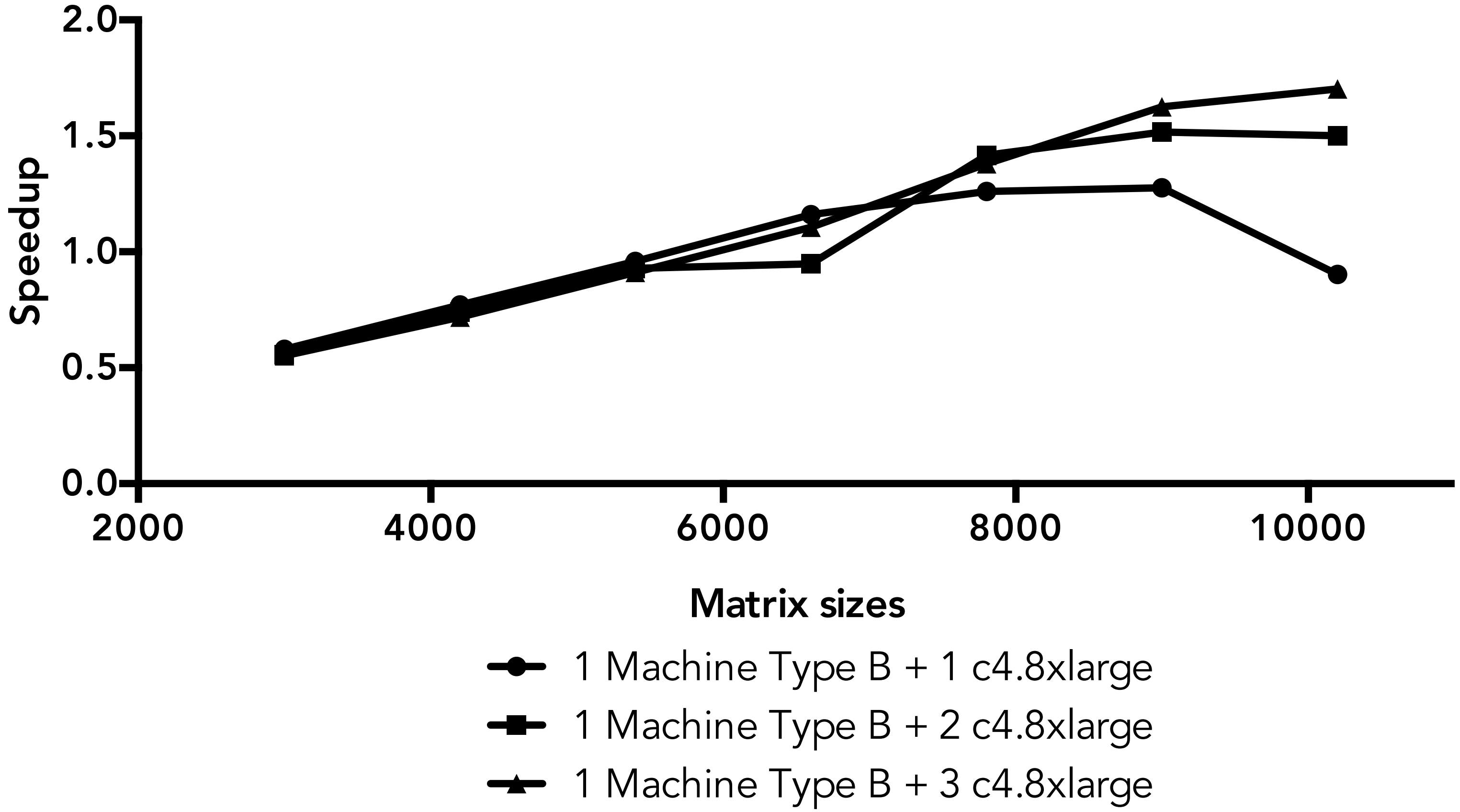
CLOUD MANDELBROT



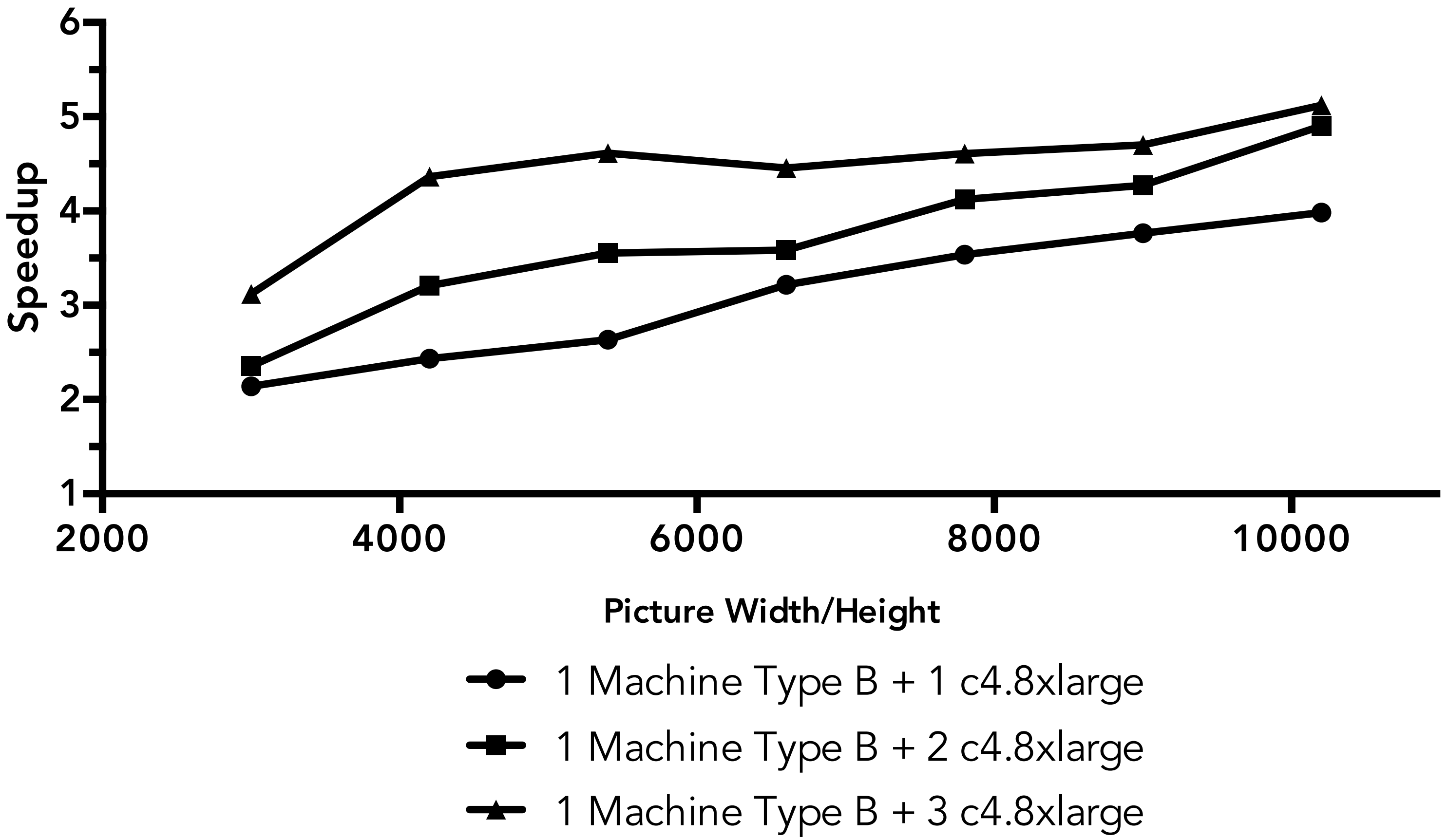
HYBRID


FSOC
Management Node
&
Execution Node
10 Gbit/s
170 Mbit/s per connection
EC2
10 Gbit/s
10 Gbit/s
10 Gbit/s
Execution Node
Execution Node
Execution Node

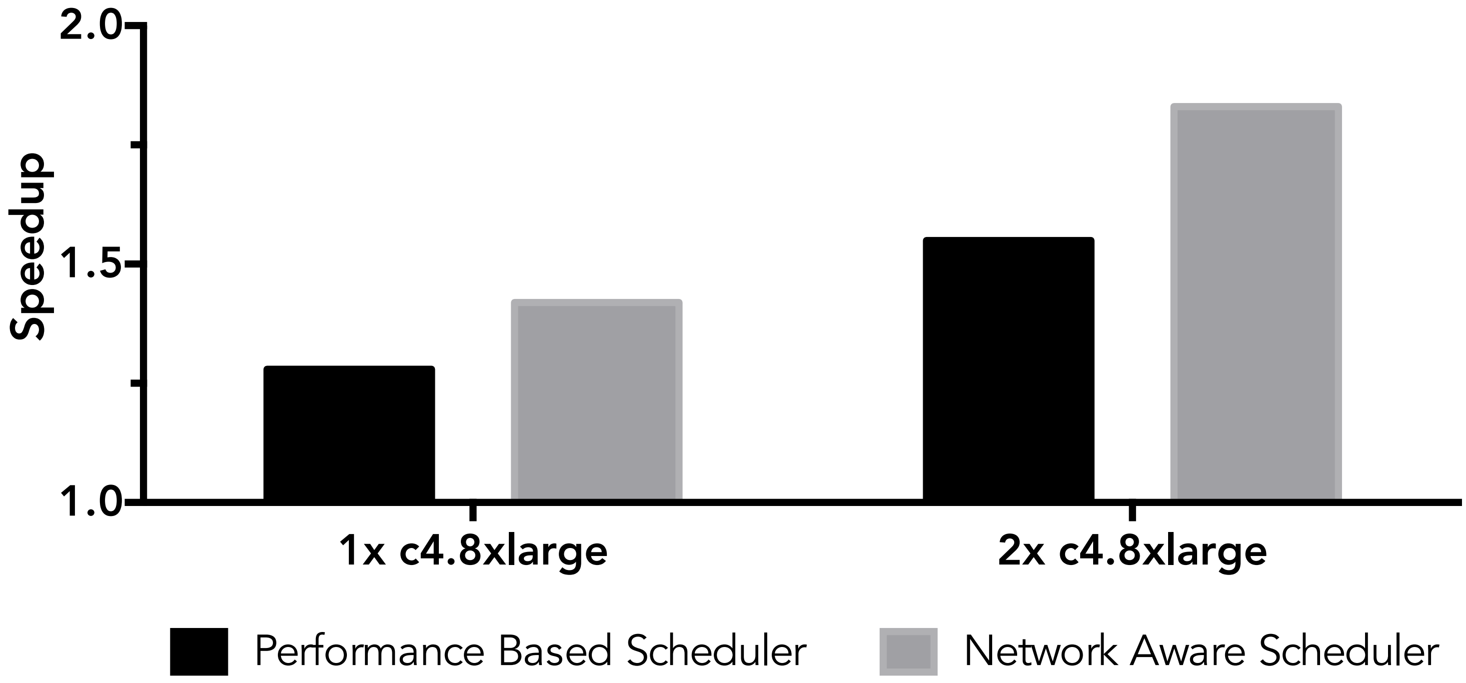
HYBRID MATRIX MULT.



HYBRID MANDELBROT

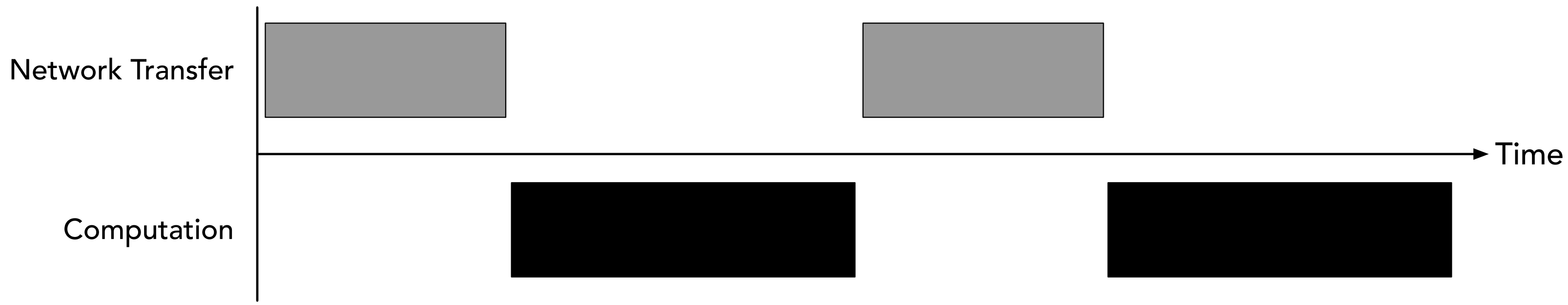


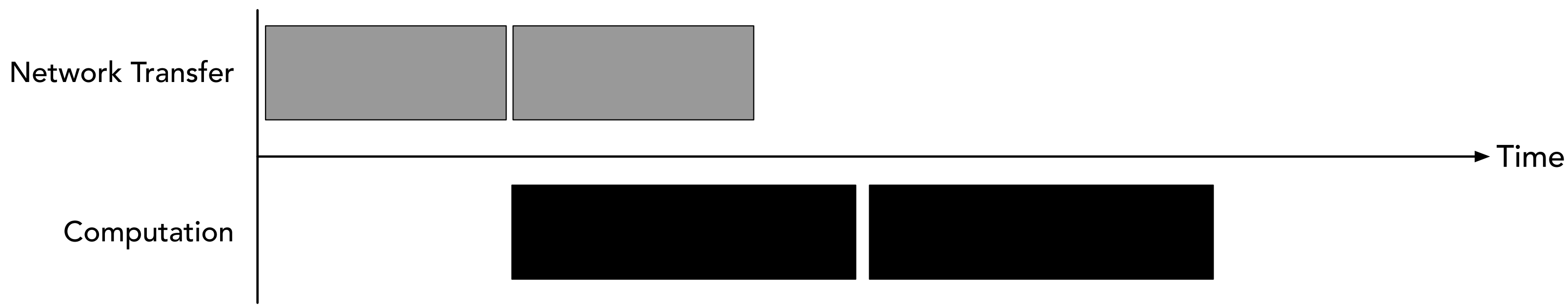
HYBRID JOB SUITE SCHEDULING



FUTURE WORK

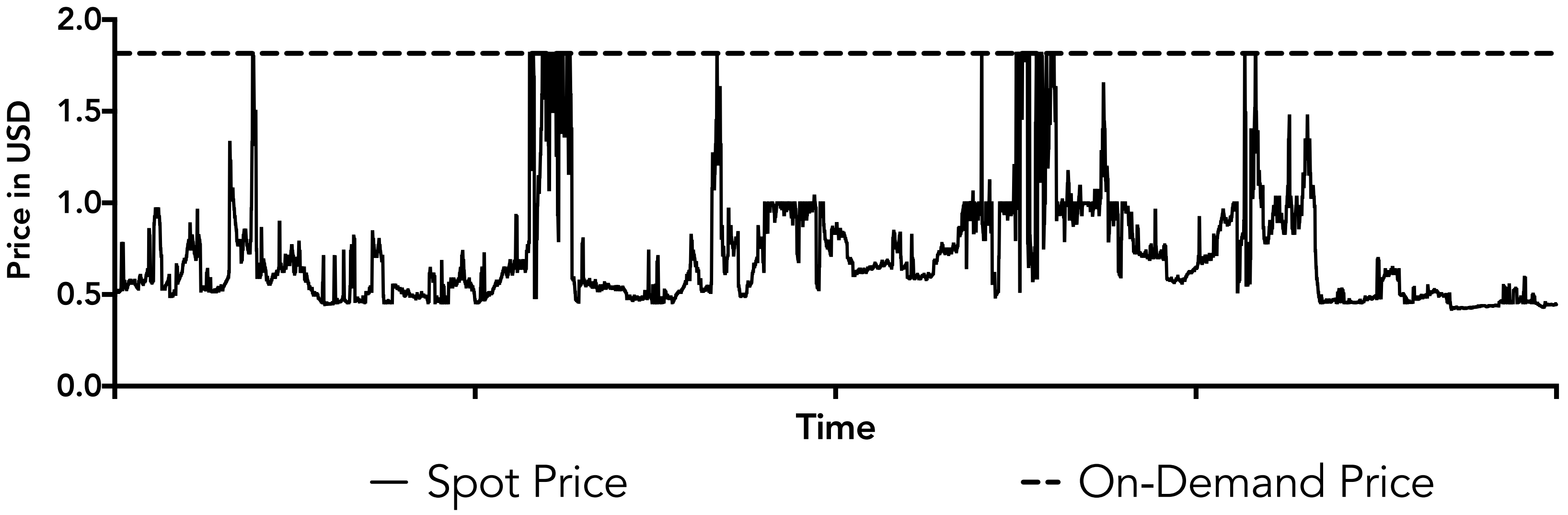
TASK QUEUE


Network Transfer
Time
Computation


Network Transfer
Time
Computation

EC2 SPOT INSTANCES

- Optimize cloud resource costs
- Reserve cheap instances over time
- Automated process with upfront user input



CONCLUSION

LIMITATIONS

- **Network connection major bottleneck**
- **Limitations of Aparapi**
  - Code translation
  - Device support
- **Memory may become bottleneck when many jobs are executed in parallel**

ACHIEVEMENTS

- Distributed computations on heterogeneous clusters
- Flat learning curve and little code necessary
- Cluster size can be dynamically increased by cloud resources
- Scheduling architecture adaptable to various use cases
- Small code base (less than 1500 Java LOC)

SOURCE CODES

https://github.com/florianroesler/dopencl

https://github.com/florianroesler/aparapi

https://github.com/florianroesler/dynamoopencl

https://github.com/florianroesler/dynamo-server