RAY

An open source framework that provides a simple, universal API for building distributed applications

github.com/ray-project/ray

Melih Elibol

riselab
UC Berkeley

What is Ray?

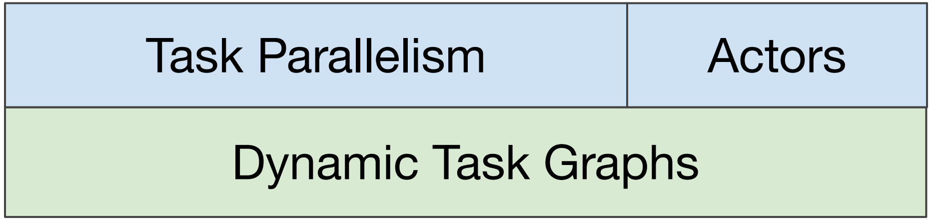
- Ray provides a **Task parallel** API and **actor** API built on **dynamic task graphs**

Dynamic Task Graphs

Ray execution model

What is Ray?

- Ray provides a **Task parallel** API and **actor** API built on **dynamic task graphs**

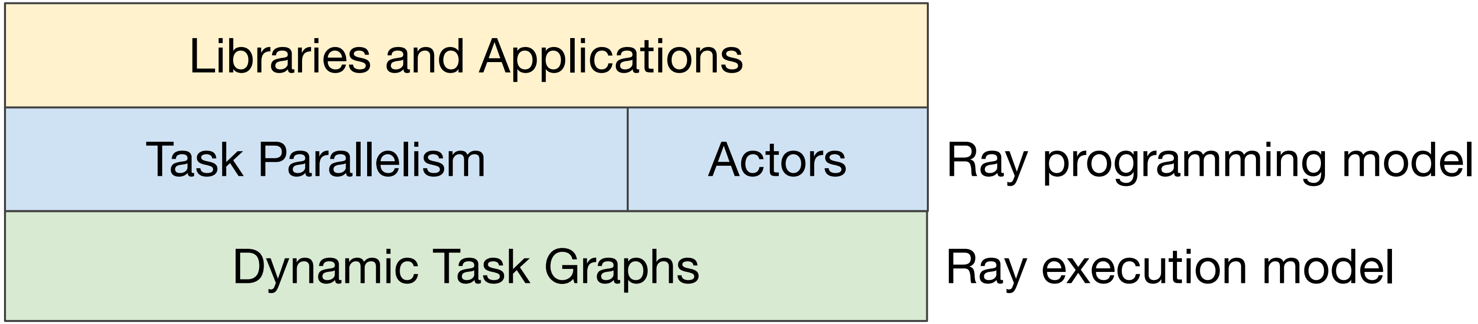


Ray programming model

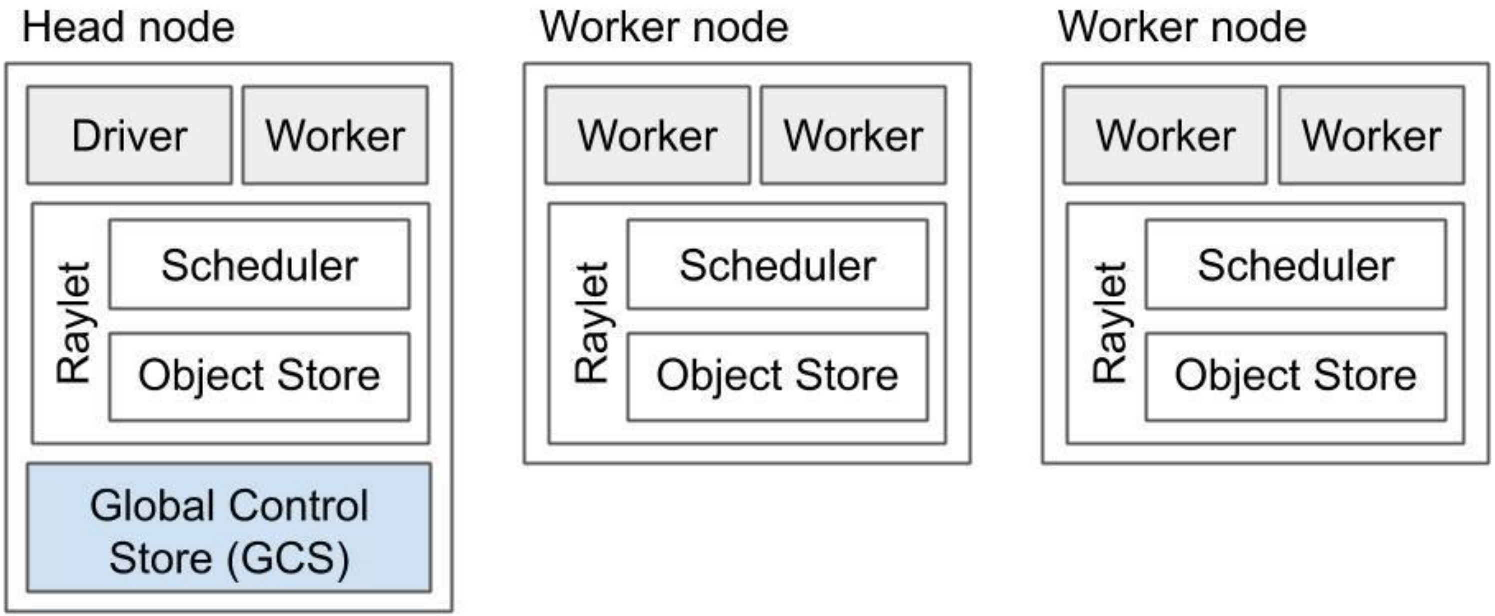
Ray execution model

What is Ray?

- Ray provides a **Task parallel** API and **actor** API built on **dynamic task graphs**



Ray Architecture



Ray Core API

- **put**(object) -> ObjectRef
- **get**(List[ObjectRef]) -> List[object]
- **remote**(Function) -> RemoteFunction
- **RemoteFunction**(*args, **kwargs) -> List[ObjectRef]

The Ray API

```
def zeros(shape):
    return np.zeros(shape)


def dot(a, b):
  return np.dot(a, b)
```

The Ray API

Tasks

```
@ray.remote
def zeros(shape):
    return np.zeros(shape)

@ray.remote
def dot(a, b):
  return np.dot(a, b)
```

The Ray API

Tasks

```
@ray.remote
def zeros(shape):
    return np.zeros(shape)

@ray.remote
def dot(a, b):
  return np.dot(a, b)

ref1 = zeros.remote([5, 5])
ref2 = zeros.remote([5, 5])
ref3 = dot.remote(ref1, ref2)
ray.get(ref3)
```

The Ray API

Tasks

```python
@ray.remote
def zeros(shape):
    return np.zeros(shape)

@ray.remote
def dot(a, b):
  return np.dot(a, b)

ref1 = zeros.remote([5, 5])
ref2 = zeros.remote([5, 5])
ref3 = dot.remote(ref1, ref2)
ray.get(ref3)
```

```python
class Counter(object):
    def __init__(self):
        self.value = 0
    def inc(self):
        self.value += 1
        return self.value
```

The Ray API

Tasks

```
@ray.remote
def zeros(shape):
    return np.zeros(shape)

@ray.remote
def dot(a, b):
  return np.dot(a, b)

ref1 = zeros.remote([5, 5])
ref2 = zeros.remote([5, 5])
ref3 = dot.remote(ref1, ref2)
ray.get(ref3)
```

Actors

```
@ray.remote
class Counter(object):
    def __init__(self):
        self.value = 0
    def inc(self):
        self.value += 1
        return self.value
```

The Ray API

Tasks

```python
@ray.remote
def zeros(shape):
    return np.zeros(shape)

@ray.remote
def dot(a, b):
  return np.dot(a, b)

ref1 = zeros.remote([5, 5])
ref2 = zeros.remote([5, 5])
ref3 = dot.remote(ref1, ref2)
ray.get(ref3)
```

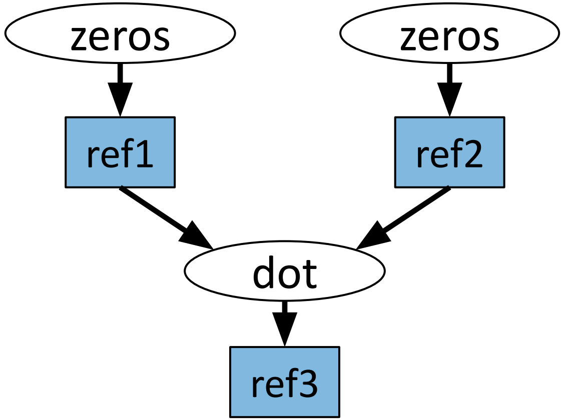
Actors

```python
@ray.remote
class Counter(object):
    def __init__(self):
        self.value = 0
    def inc(self):
        self.value += 1
        return self.value

c = Counter.remote()
ref4 = c.inc.remote()
ref5 = c.inc.remote()
ray.get([ref4, ref5])
```

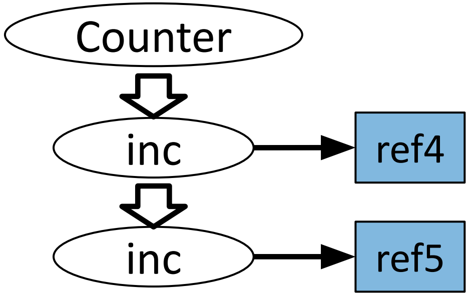
The Ray API

Tasks




```
ref1 = zeros.remote([5, 5])
ref2 = zeros.remote([5, 5])
ref3 = dot.remote(ref1, ref2)
ray.get(ref3)
```

Actors




```
c = Counter.remote()
ref4 = c.inc.remote()
ref5 = c.inc.remote()
ray.get([ref4, ref5])
```

The Ray API: Actor Handles

Invoke actor methods from other tasks/actors.

```python
@ray.remote
class Counter(object):
    def __init__(self):
        self.value = 0
    def inc(self):
        self.value += 1
        return self.value


c = Counter.remote()
ref = c.inc.remote()
```

```python
# Use the actor from a
# different task

@ray.remote
def use_actor(c):
    ref = c.inc.remote()
    ray.get(ref)
```

NumS

Melih Elibol
Lianmin Zheng
Mohamed Elgharbawy
Sehoon Kim
Devin Petersohn
Alvin Cheung
Michael I. Jordan
Ion Stoica

University of California, Berkeley

NumS

github.com/nums-project

PROBLEM

The Problem

NumS aims to make **terabyte-scale data modeling easier** for the **Python** scientific computing community.

- We have an abundance of very fast compute devices and libraries to manage parallelism among these devices.
- However, existing libraries expect the Python scientific computing community to learn advanced parallel computing concepts and algorithms to make use of these devices, an uncommon skill among Python users.
- What can be done to make numerical computing at these scales accessible to Python programmers?

TRENDS

Trends in Computing

- We can’t expect faster CPU/GPU clock speeds.
- CPU memory is abundant, but shared memory has physical limitations.
- Network speeds in the cloud and GPU interconnects are rapidly increasing. AWS/Azure network bandwidth of 2.5GB/s are common, and latest NVLink/NVSwitch has 75GB/s bandwidth between any two connected GPUs.

SOLUTIONS

Existing Solutions

- NumPy Scales to multiple cores, but only per-operation as determined by the system's BLAS library.

- MPI as a specification provides a programming model for NumPy, Tensorflow, Pytorch, etc. to parallelize numerical algorithms, but MPI (and SPMD in general) is an uncommon programming model for Python programmers.

- Tensorflow and Pytorch have poor CPU support, and good multi-GPU implementations require SPMD-style programming.

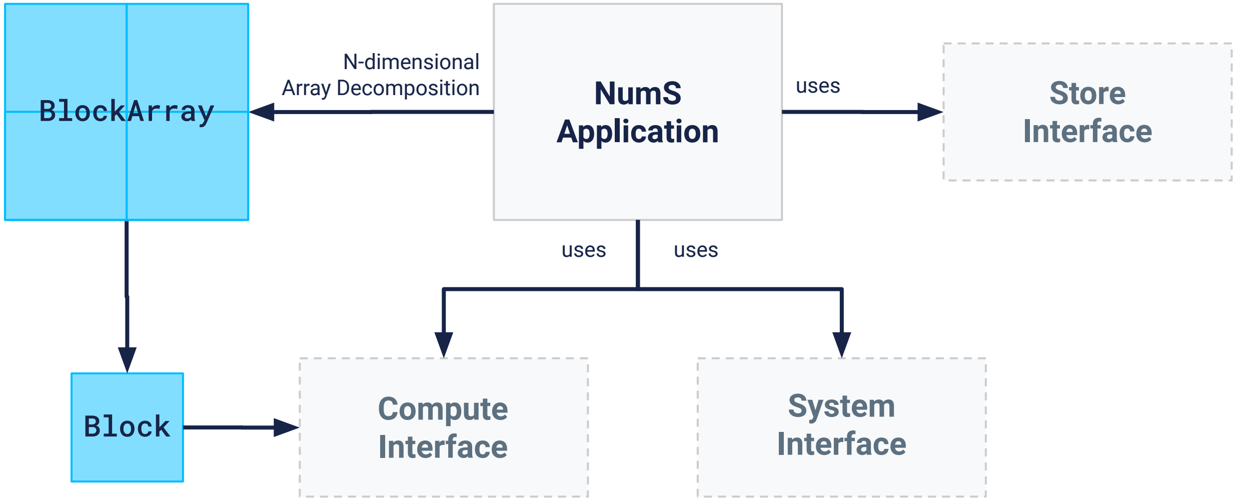
OUR SOLUTION

NumS

A **Num**erical Cloud Computing **S**ystem that automatically translates NumPy to optimized distributed memory code.

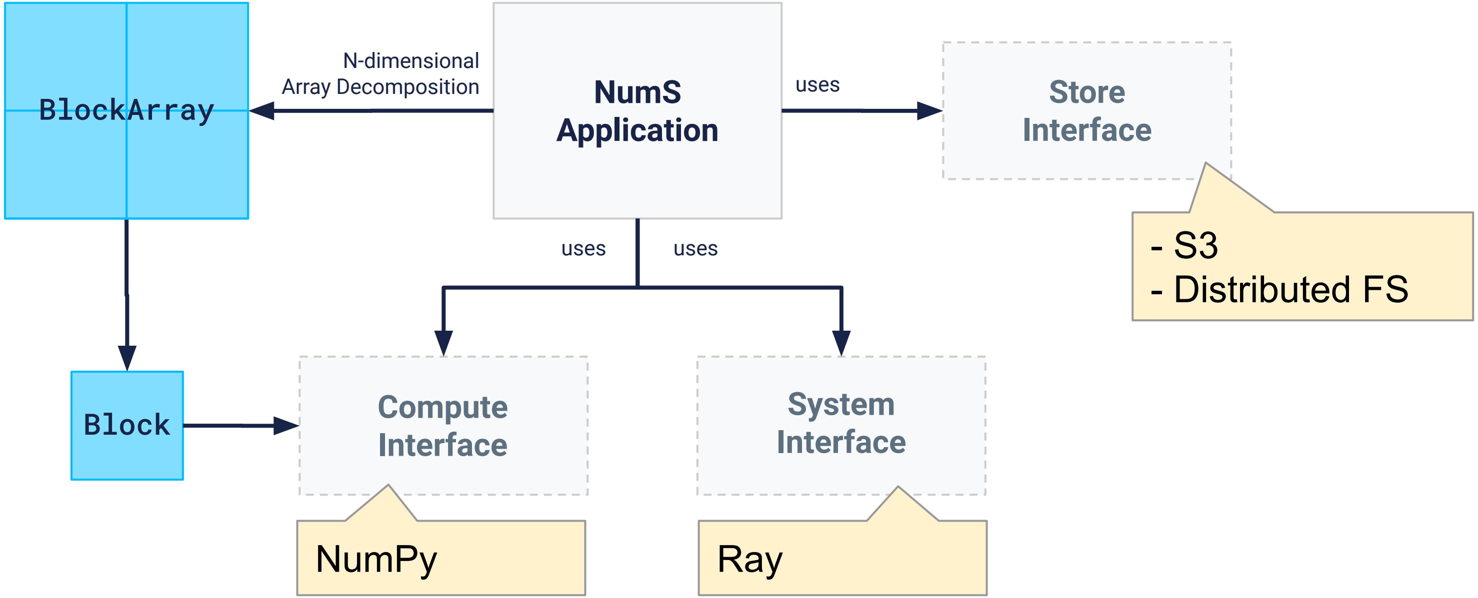
OUR SOLUTION

NumS Design



OUR SOLUTION

NumS Design



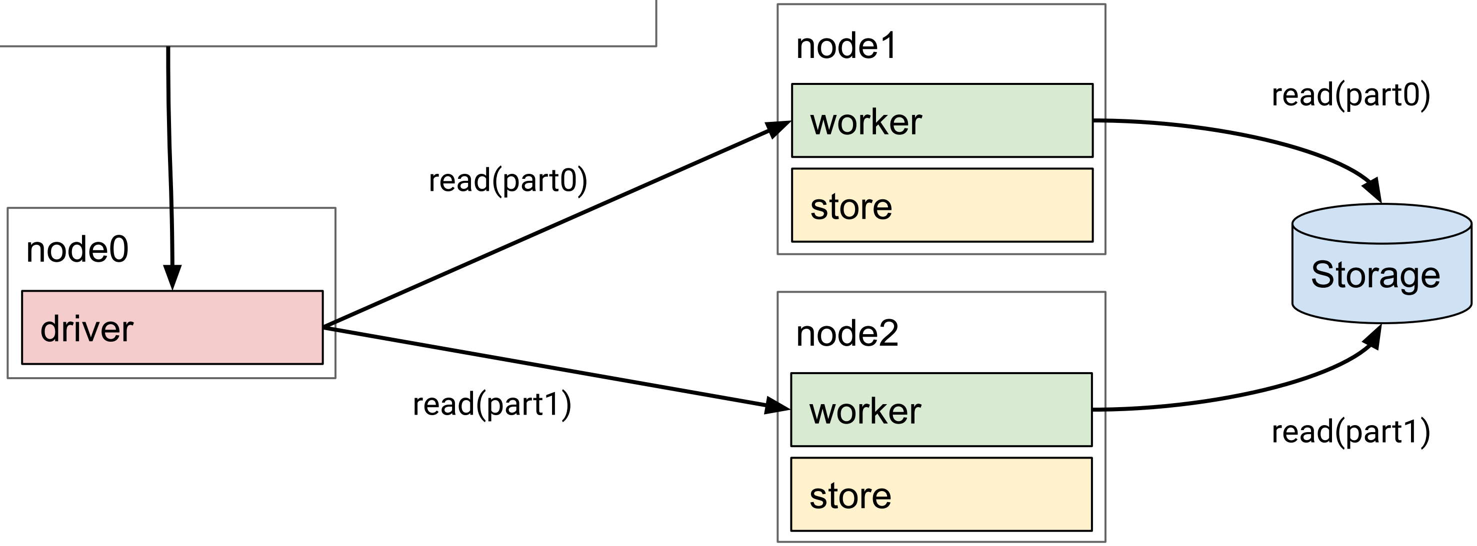
EXAMPLE

Execution on Ray: RPC Calls

```
import nums

x: BlockArray = nums.read("data/x")
```

read returns immediately, executing tasks required to construct x asynchronously.

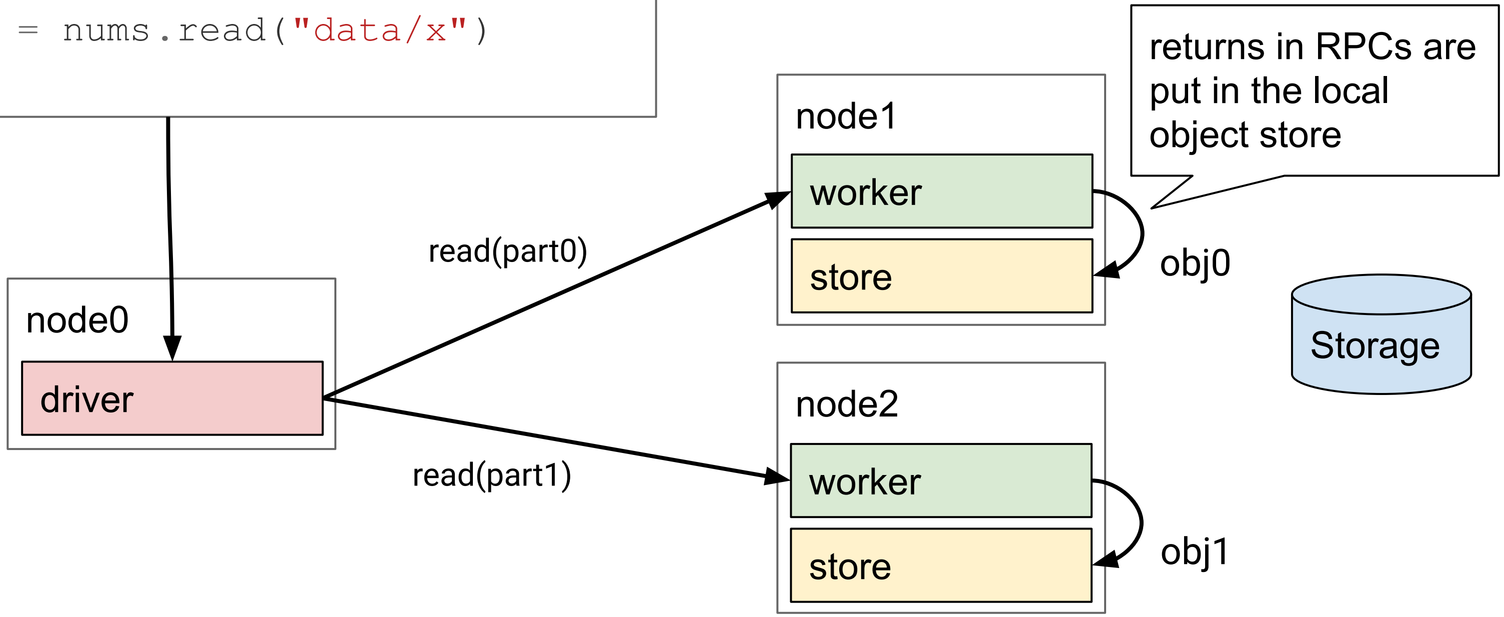


EXAMPLE

Execution on Ray: RPC Returns

```
import nums

x: BlockArray = nums.read("data/x")
```

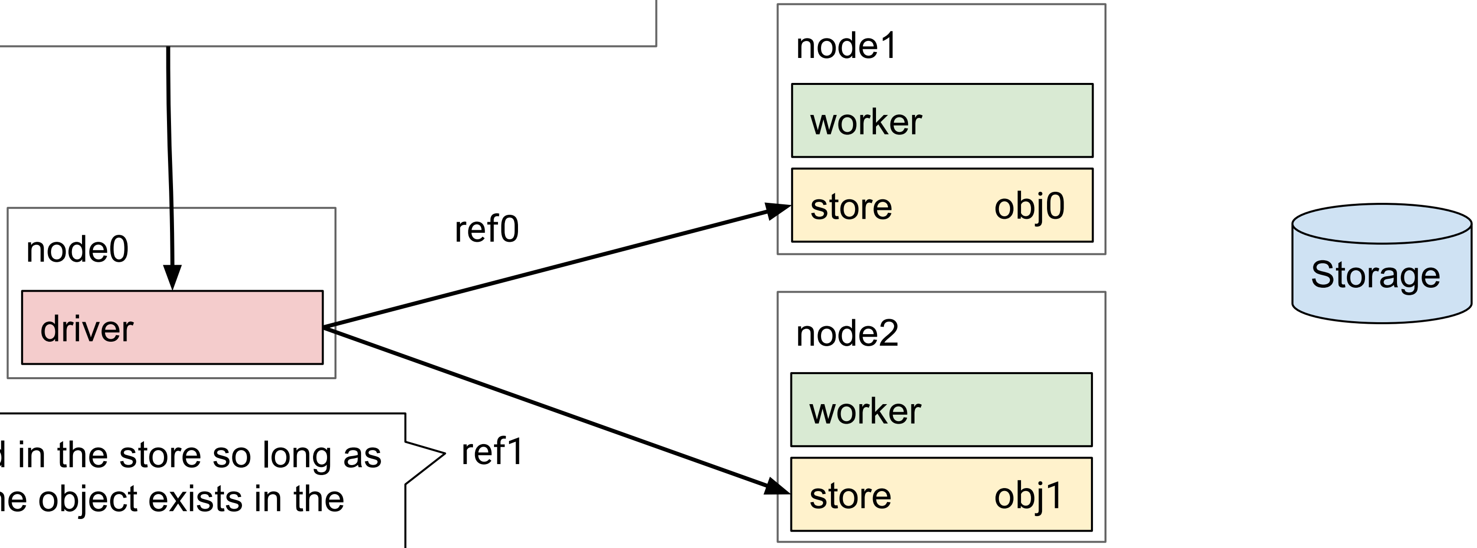


EXAMPLE

Execution on Ray: References

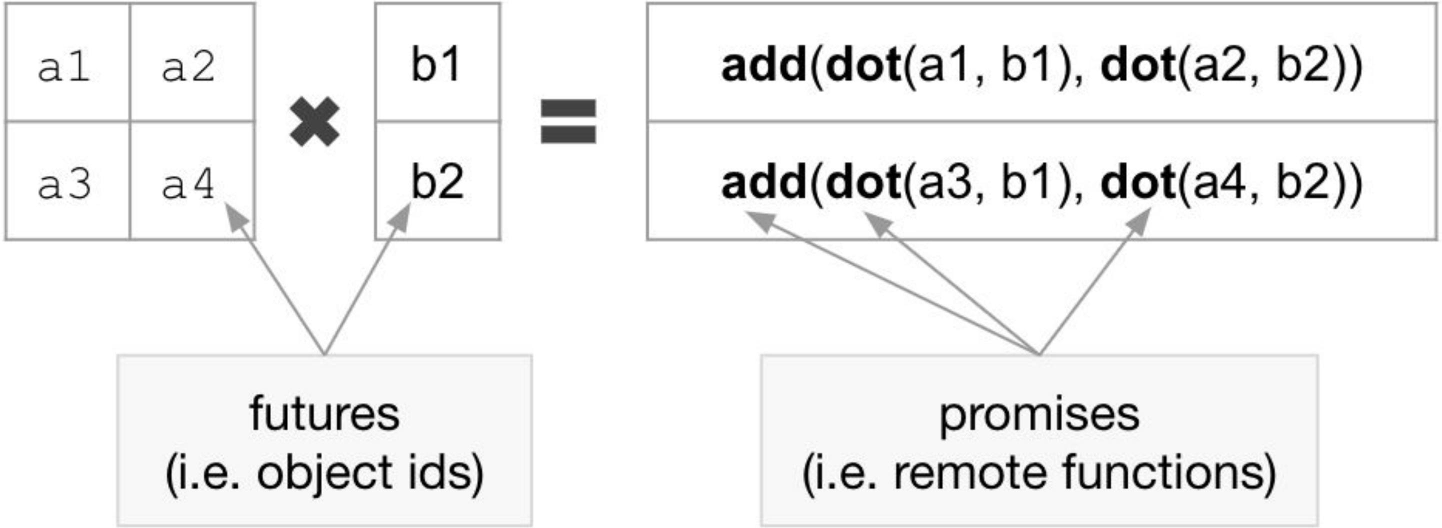
```
import nums

x: BlockArray = nums.read("data/x")
```



Objects are held in the store so long as a reference to the object exists in the application.

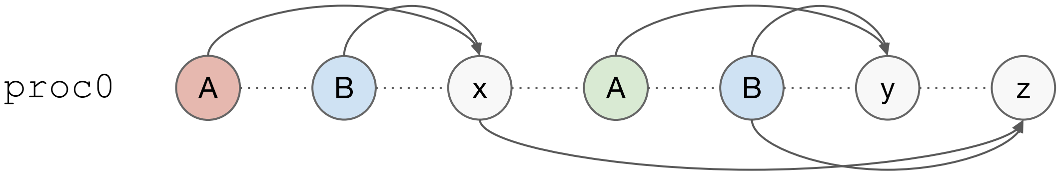
Futures and Promises



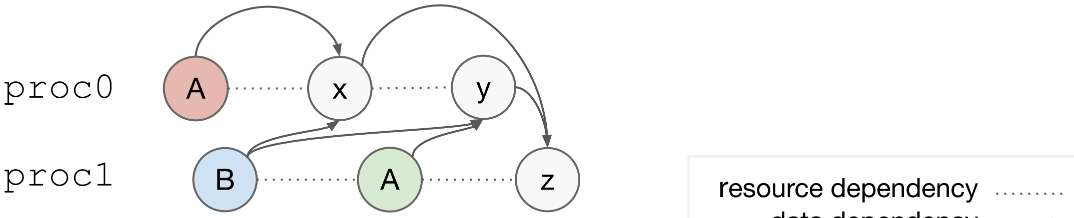
Array Access Dependency Resolution

```
x = A[:, i].T @ B[:, i]
y = A[:, j].T @ B[:, i]
z = x * y
```

Serial Execution



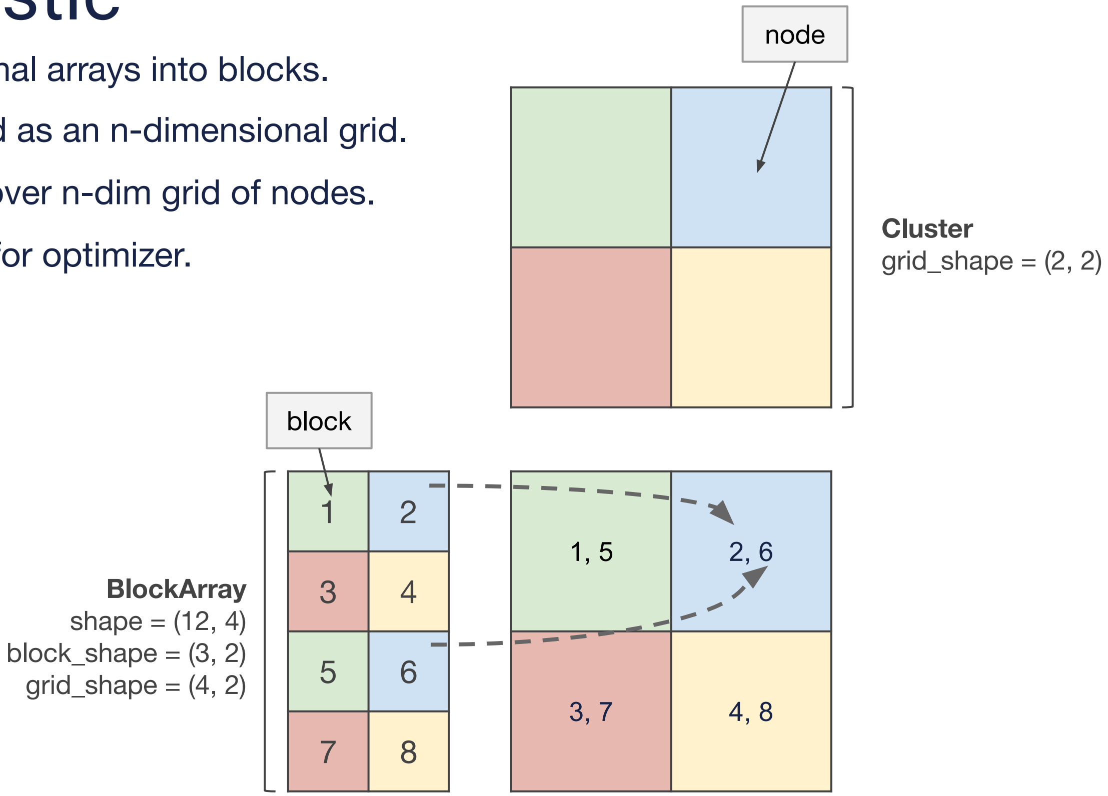
Futures with Concurrency



STRUCTURES

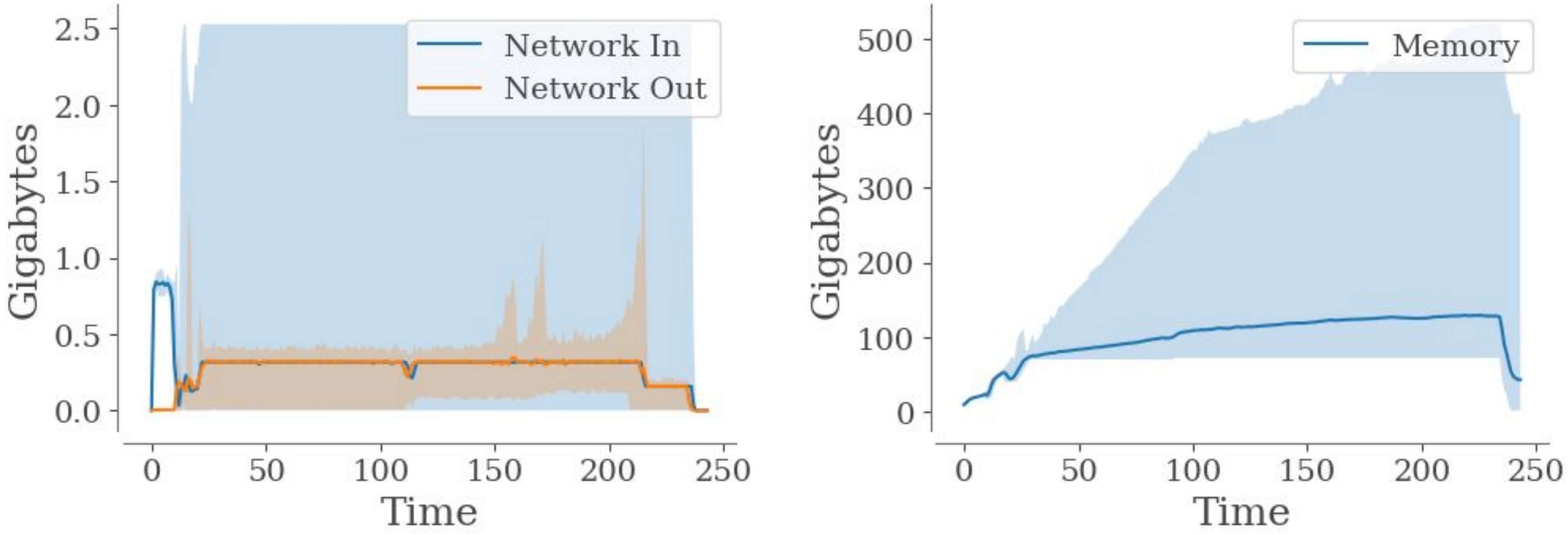
Block-Cyclic Heuristic

- **NumS** decomposes n-dimensional arrays into blocks.
- A cluster of nodes is represented as an n-dimensional grid.
- Persistent arrays are dispersed over n-dim grid of nodes.
- Balances data load and locality for optimizer.



STRUCTURES

Newton Opt. Logistic Regression With Block-Cyclic



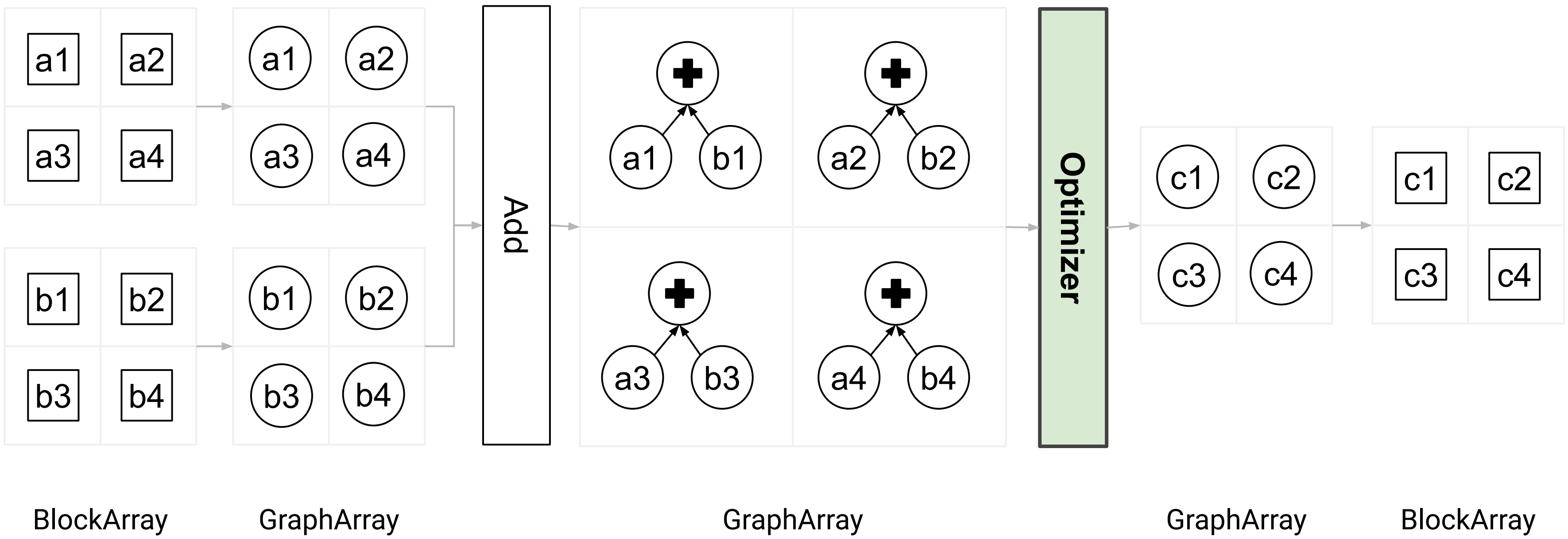
* Mean statistic with min/max spread across 16 nodes on 128GB dataset.

Optimizer

- **Cluster State**: Estimate memory and network load on each node using array size.
- **Objective**: Place operations so that maximum memory and network load over all nodes is minimized.
- **Computation State**: An array-of-trees data structure on which we perform computations.
- **Tree Search**: An iterative algorithm that places a single operation per iteration according to the objective, and updates both the cluster state and computation state.

OPTIMIZATION

Execution of Element-wise Addition



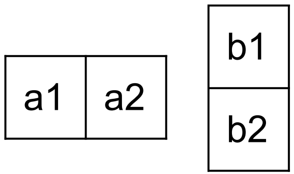
OPTIMIZATION

Representations of Tensor Dot

A @ B

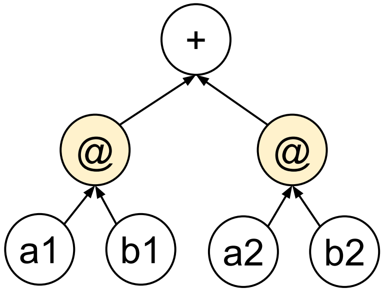
Syntactic Representation

- A is 4 by 8
- B is 8 by 4



BlockArray Representation

- a_i is 4 by 4
- b_i is 4 by 4

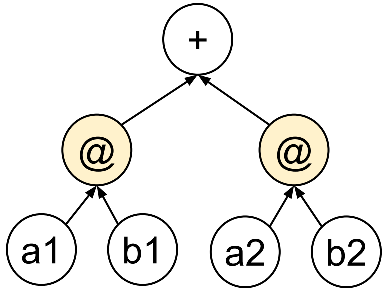


GraphArray Representation

- frontier nodes

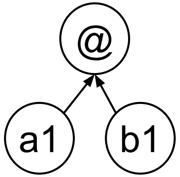
OPTIMIZATION

Optimization of Tensor Dot

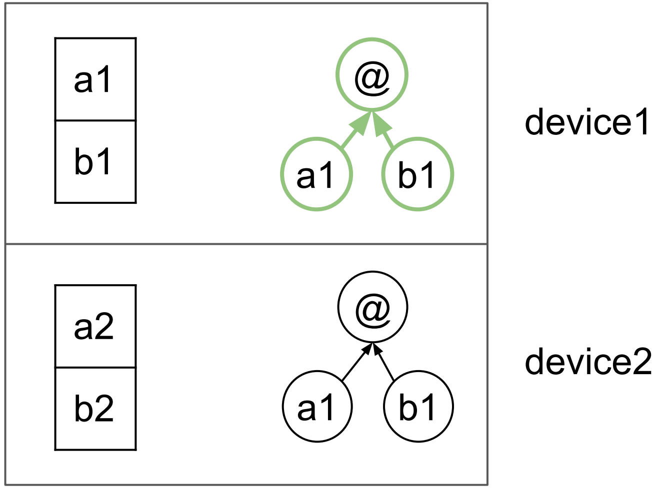


GraphArray Representation

- frontier nodes



Randomly sample a frontier node.



Schedule the operation based on cluster simulation.

Optimization Objective

Schedule op on device1

	memory	net_in	net_out
device1	48	0	0
device2	32	0	0

Schedule op on device2

	memory	net_in	net_out
device1	32	0	32
device2	48	32	0

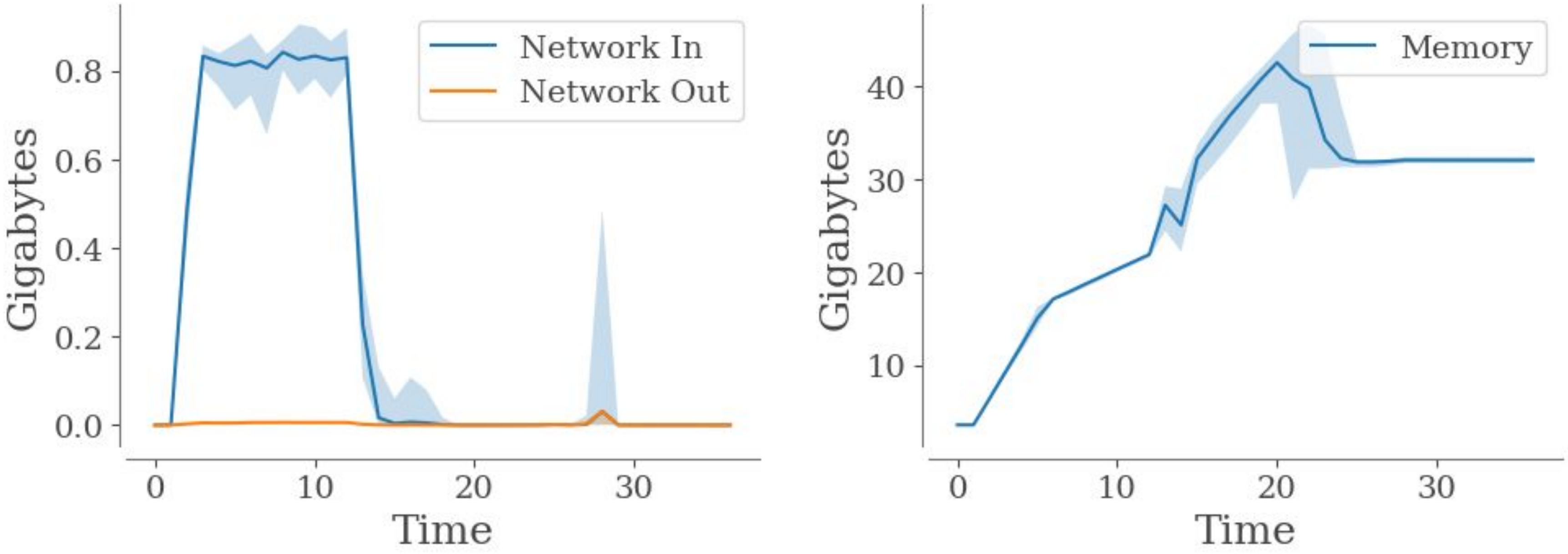
Capture desired scheduling behavior in a simple objective:

$$f(s_i) = \|M_i\|_\infty + \|I_i\|_\infty + \|O_i\|_\infty$$

Where s_i corresponds to scheduling option i from state s. We minimize this objective over i.

OPTIMIZATION

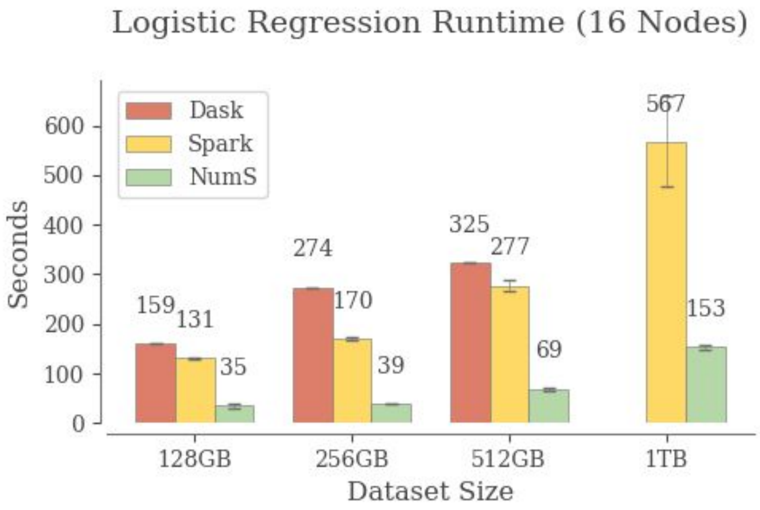
Newton Opt. Logistic Regression With NumS

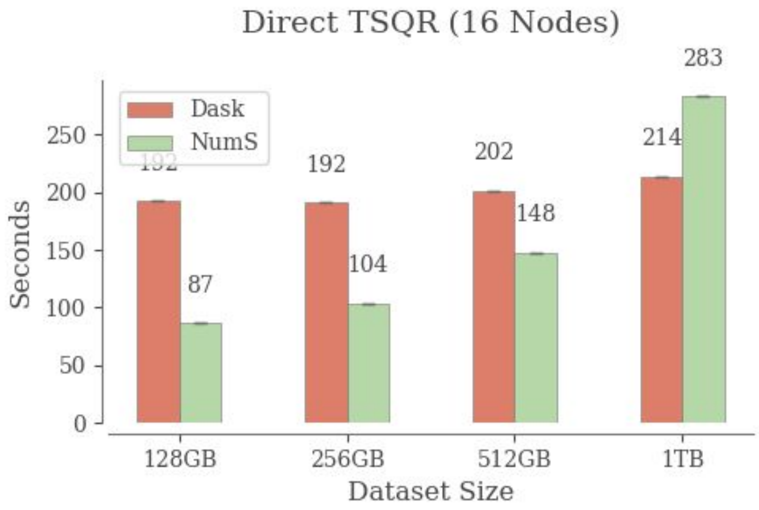


* Mean statistic with min/max spread across 16 nodes on 128GB dataset.

RESULTS

Benchmarks





Setup

- 16 nodes with 64 cores, 512 RAM, 2.5GB/s network.
- Tests loading, executing, and evaluating each task.
- Missing bars == OOM.

Results

- **3-6x** speedup on LR over Dask and Spark.
- Competitive performance on QR Decomposition.
- **10-20x** speedup on basic linear algebra over Dask.

OPEN SOURCE RELEASE

NumS 0.1.3

```
pip install nums
```

- Top-level API coverage **40%**, random module coverage **> 90%**.
- Full support for array assignment, broadcasting, and basic operations.
- I/O support for distributed file systems, S3, and CSV files.
- Prepackaged support for GLMs.
- Experimental integration with Modin (DataFrames) and XGBoost (Tree-based models).

FUTURE WORK

CS 267 Project Ideas

- Compile-time optimization of tensordot.
- Add support for sparse arrays via sparse probabilistic matrix factorization.
- Implement various notions of matrix inversion, ideally with guarantees on numerical stability.
    - Could also implement any linear algebra operation not currently supported.
- Implement statistical machine learning models, such as Gaussian processes, k nearest neighbors, naive Bayes, etc.
- Empirically, we see that our optimizer achieves near-optimal solutions. Can we prove a bound on the NumS optimization algorithm using L2 norms?