Tutorial: Analytics and AI on Crays
CUG 2018

Kristyn Maschhoff and Michael Ringenburg

Agenda

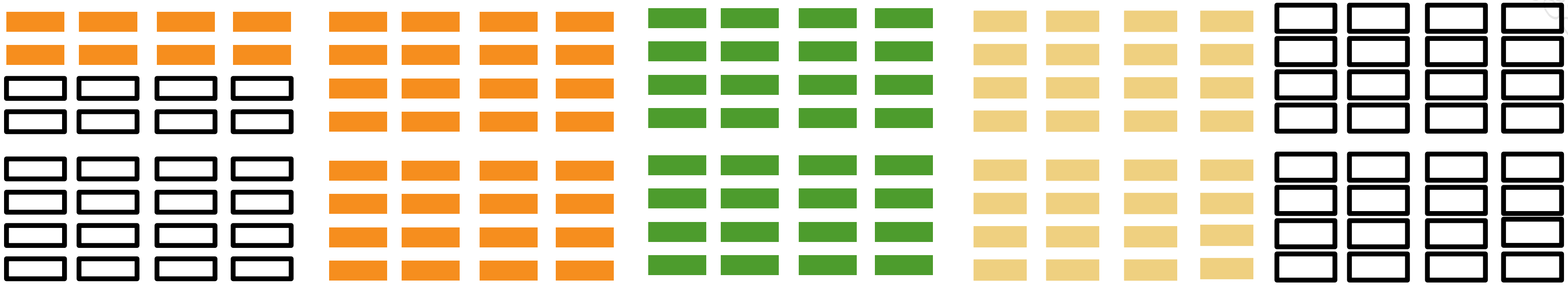
- **Session I**
  - Introduction/Urika-XC/Accounts
  - Python, Anaconda, and Dask
  - Deep Learning with TensorFlow
- **Session II**
  - Scaling Deep Learning with the Cray PE Machine Learning Plugin
  - HPC, AI, and Analytics with R and pbdR
- **Session III**
  - Spark and Alchemist
  - BigDL
- **Session IV**
  - Cray Graph Engine
  - Wrap-up and questions

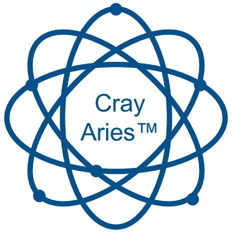
Cray Platforms – Urika-XC

- Urika-XC provides the unified stack to support Analytics, AI and HPC workloads at scale

- Urika-XC enables IT Director to leverage the existing investment in an XC system to support emerging Analytics and AI workloads, along with Simulations:
  - **Data Processing and General-purpose Big Data Analytics** with Spark (Hadoop not supported)
  - **Predictive Analytics (ML)**
  - **Pattern Matching (DL)**
  - **Data discovery with Cray Graph Engine**, with ability to scale to extremely large graphs

At The Core Of Urika-XC: Analytics At Supercomputer Scale





Run a mix of workloads on a shared system, each able to scale to >256 nodes leveraging the power of the Cray Aries network and Cray Linux Environment

🟧 Cray Graph Engine

□ HPC Simulation

🟩 Python distributed Dask

🟨 Apache Spark

Urika-XC includes two core components

CRAY OPEN SOURCE ANALYTICS

- A collection of scalable analytics frameworks and tools, enabling analytics, machine learning, and deep learning at scale

CRAY GRAPH ENGINE

- A unique in-memory Semantic Graph database, implemented using HPC technology, designed to handle the largest and most demanding use cases where graph discovery and pattern matching are required

Urika-XC component packages

CRAY OPEN SOURCE ANALYTICS

CRAY GRAPH ENGINE

- Apache® Spark™
  - Intel BigDL
- TensorFlow
- Cray PE Machine Learning Plugin
- Anaconda Python
- Dask Distributed
- Jupyter Notebook
- Java
- R
- Scala
- Standard build tools

Runs inside a Container

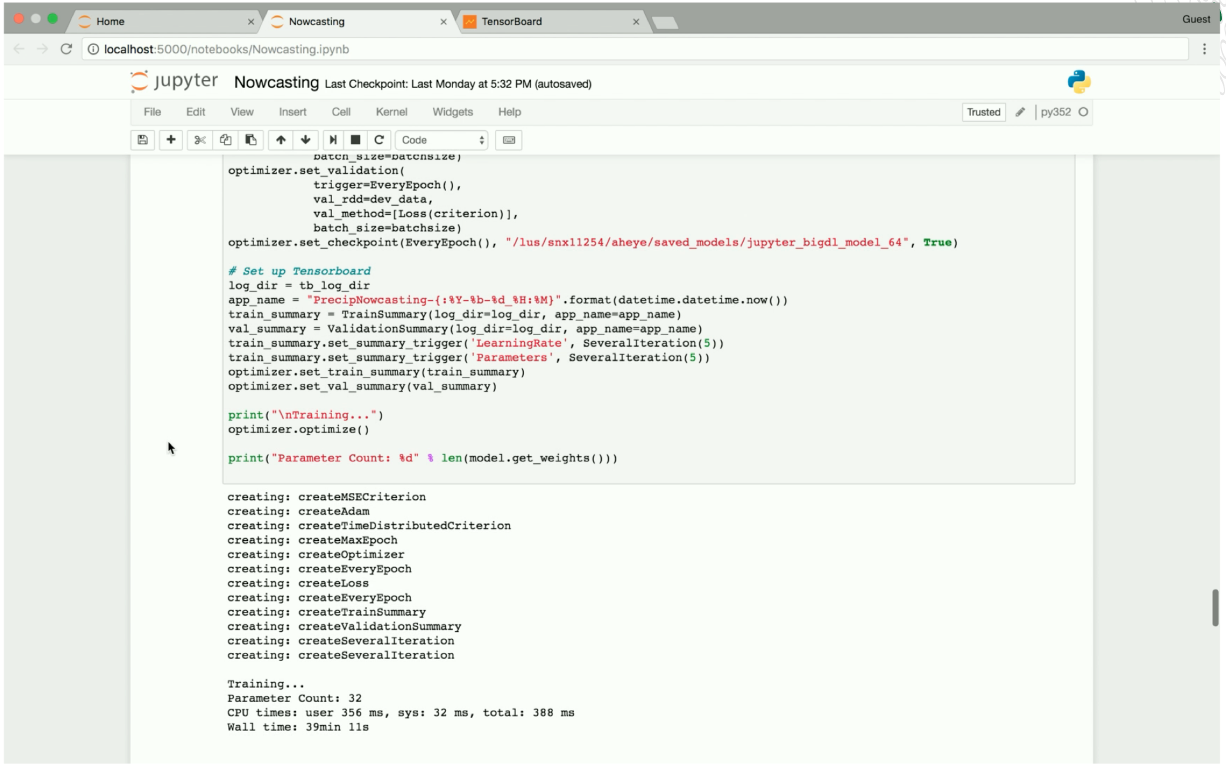
- Cray Graph Engine (CGE)

Bringing Open Source Frameworks to Supercomputing

- Distributed in-memory processing for Big Data
  - Extensive set of libraries and packages for the Spark environment
- Anaconda open data science platform and Dask for Python-based distributed parallel computing
  - Well suited for large-scale data processing, predictive analytics, and scientific computing
  - Jupyter Notebooks for interactive supercomputing
- Integrated Deep Learning
  - TensorFlow popular deep learning framework
  - Native deep learning in Spark

Web UIs for Interactivity and Monitoring

- Jupyter Notebooks
  - Interactive computing and visualization
  - Works with TensorFlow, Dask, PySpark, or other Python-based workflows
- TensorBoard
  - Visualize training
  - Examine DNN layers
  - Run live (monitor training) or after-the-fact


Home
Nowcasting
TensorBoard
Guest
localhost:5000/notebooks/Nowcasting.ipynb
jupyter
Nowcasting Last Checkpoint: Last Monday at 5:32 PM (autosaved)
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Trusted
py352
Code

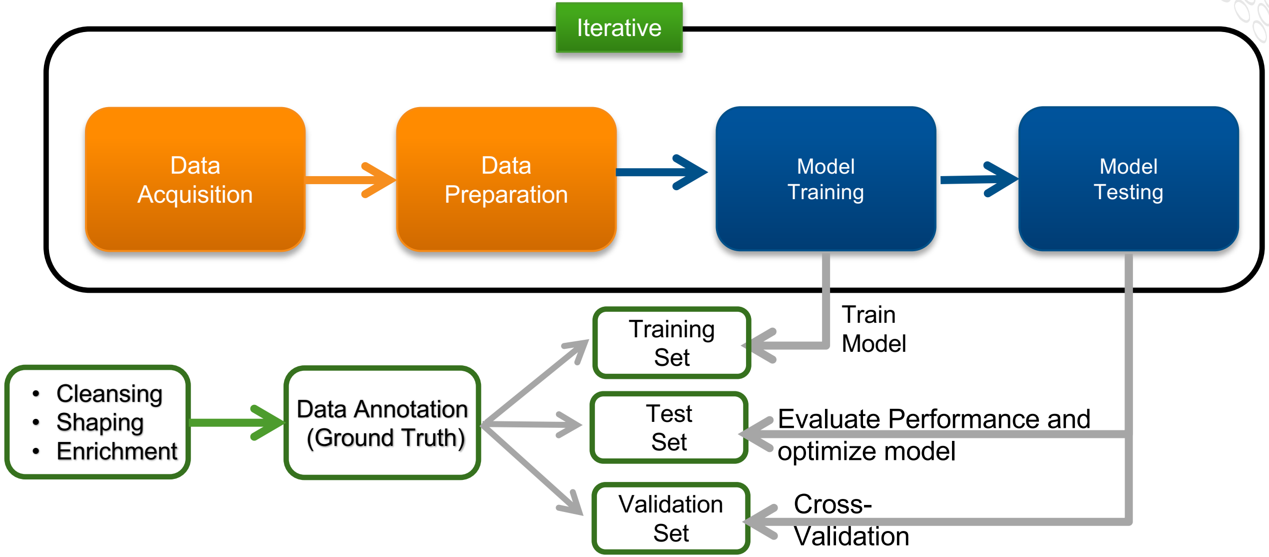
```
              batch_size=batchsize)
optimizer.set_validation(
            trigger=EveryEpoch(),
            val_rdd=dev_data,
            val_method=[Loss(criterion)],
            batch_size=batchsize)
optimizer.set_checkpoint(EveryEpoch(), "/lus/snx11254/aheye/saved_models/jupyter_bigdl_model_64", True)

# Set up Tensorboard
log_dir = tb_log_dir
app_name = "PrecipNowcasting-{:%Y-%b-%d_%H:%M}".format(datetime.datetime.now())
train_summary = TrainSummary(log_dir=log_dir, app_name=app_name)
val_summary = ValidationSummary(log_dir=log_dir, app_name=app_name)
train_summary.set_summary_trigger('LearningRate', SeveralIteration(5))
train_summary.set_summary_trigger('Parameters', SeveralIteration(5))
optimizer.set_train_summary(train_summary)
optimizer.set_val_summary(val_summary)

print("\nTraining...")
optimizer.optimize()

print("Parameter Count: %d" % len(model.get_weights()))
```

```
creating: createMSECriterion
creating: createAdam
creating: createTimeDistributedCriterion
creating: createMaxEpoch
creating: createOptimizer
creating: createEveryEpoch
creating: createLoss
creating: createEveryEpoch
creating: createTrainSummary
creating: createValidationSummary
creating: createSeveralIteration
creating: createSeveralIteration

Training...
Parameter Count: 32
CPU times: user 356 ms, sys: 32 ms, total: 388 ms
Wall time: 39min 11s
```

Urika-XC Focus: Support the Entire AI Workflow

Deep Learning workflows are not limited to training

Similar to other HPC and analytics workloads, significant portions of DL jobs are devoted to data collection, preparation and management.



Anaconda Python and Dask Distributed
CUG 2018

Michael Ringenburg, Cray Inc.
mikeri@cray.com

Training Accounts on Piz Daint

- CSCS has generously provided temporary training accounts on Piz Daint
- Login through the front end, ela.cscs.ch, then can ssh daint
- Setup passwordless ssh. From daint:

```
user@daint102:~> ssh-keygen
user@daint102:~> ssh-copy-id -i .ssh/id_rsa.pub ela1
```

- Add to local .ssh/config to go directly to daint (simplifies notebook setup):

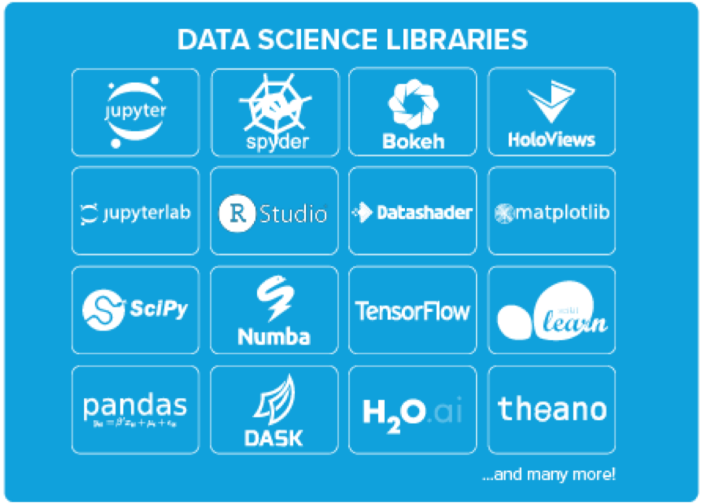
```
Host daint
 HostName daint.cscs.ch
 User your-daint-username
 ProxyCommand ssh -q -Y ela.cscs.ch -W %h:%p
```

- To run Urika-XC (please limit to at most 5 nodes each for now)

```
user@daint104:~> module use /apps/daint/system/cug
user@daint104:~> module load analytics
user@daint104:~> salloc -N 5 -C gpu --reservation=craycug start_analytics
```

Anaconda Distribution of Python

- **Comes with large set of data science packages preinstalled**
  - 250+ Python and R packages preinstalled
  - >1000 available in repositories
  - Many optimized – e.g., work with Intel Python team
- **Conda environment manager**
  - Linked to conda repos for more Python and R packages
  - Ability to create, clone, share custom environments with your own python/package versions
  - Handles all dependencies
  - Allows sharing environments
- **Anaconda and conda built in to Urika-XC**



Using the Conda Environment Manager

- Create a new conda environment with conda create
  - E.g., create an environment with Python 3.5 and biopython:

```
conda create --name bio biopython python=3.5
```

- Activate your environment:

```
source activate bio
(bio) mikeri:~>
```

- Python 3.5 with biopython will now be your default python:

```
(bio) mikeri:~> python
Python 3.5.3 | Anaconda, Inc.
>>> import Bio
>>> from Bio.Seq import Seq
>>> my_seq = Seq('CATGTAGACTAG')
>>> my_seq.translate()
Seq('HVD*', HasStopCodon(ExtendedIUPACProtein(), '*'))
```

More Conda Commands

- Deactivate an environment: **source deactivate**
- Get rid of an environment: **conda remove**
- Clone an environment: **conda clone**
- List environments: **conda info --envs**
- Find available packages: **conda search**
- List packages: **conda list**
- Add package to current environment: **conda install**
  - Can even install pip, and use that to install in a conda environment!
- More in docs: https://conda.io/docs/index.html

Using Conda Environments with PySpark

- Start up Spark cluster (will discuss in Session III)
- Activate your Conda environment

```
source activate bio
```

- Set PYSPARK_PYTHON to point to environment python

```
export PYSPARK_PYTHON=$(which python)
```

- Run pyspark

```
pyspark
>>> import Bio
```

Let’s try this now, in a notebook

- Make sure your local .ssh/config is set up for daint
- Set up a tunnel from your laptop to daint:

```
% ssh -L <local-port>:localhost:<login-port> daint
```

- Launch Urika-XC, pointing to the tunneled port on daint

```
% salloc -N 5 -C mc start_analytics --login-port <login-port> --ui-port <ui-port>
```

- Urika-XC will create a tunnel from *ui-port* on the compute node to *login-port* on daint
- Run your notebook, passing it *ui-port*

```
% export SHELL=$(which bash)
% jupyter notebook --port ui-port
```

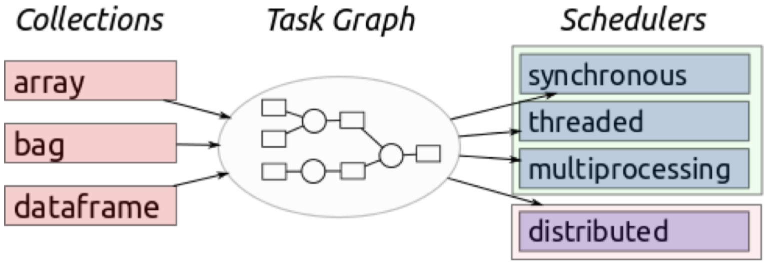
Optional, for terminal access

- Connect browser on laptop to localhost:*local-port*

Anaconda and PySpark on XC Demo

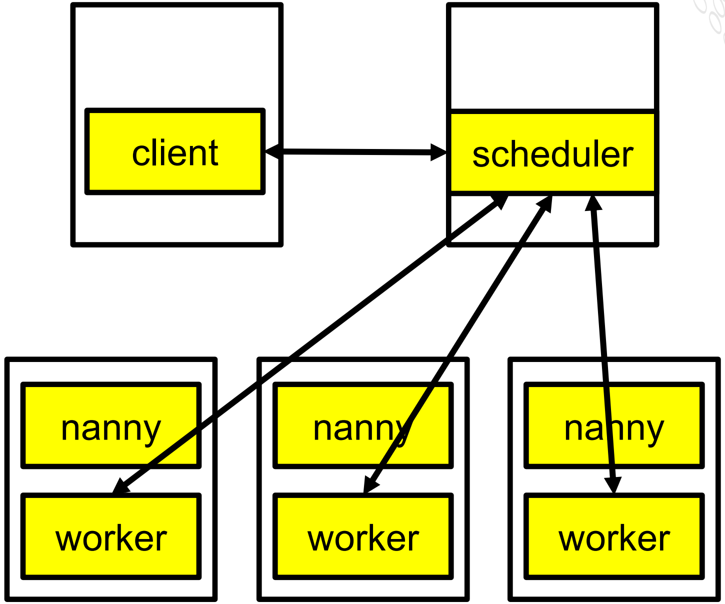
Dask and Dask Distributed

- **Dask**
  - Set of parallel collections and operations for Python
  - Integrated with most common packages, e.g., parallel version of numpy arrays
  - Supports multiple task schedulers
- **Threaded scheduler**
  - Backed by low-overhead thread pool
  - Subject to Python Global Interpreter Lock (GIL)
  - Best if application dominated by non-Python code
- **Multiprocess scheduler**
  - Tasks shipped to separate local processes
  - Not subject to Python GIL – allows true on-node parallelism
  - Low overhead to launch/utilize pool, but overhead of moving data
  - Best for mostly Python code (allows parallelism even with GIL)



Dask and Dask Distributed

- **Distributed scheduler**
  - Dask scheduler for multi-node parallelism
  - Runs a scheduler on one node, workers across allocated nodes
  - Nanny processes for fault tolerance
  - Supports distributed versions of all Dask data structures
  - Allows asynchronous execution (futures)



Setting Up a dask.distributed Cluster

- Set up a dask distributed environment in anaconda

  -
    ```
    conda create --name mydask dask distributed
    ```
- Get allocation

  -
    ```
    salloc -N 4 -C mc
    ```
- Activate dask distributed

  -
    ```
    source activate mydask
    ```
- Start scheduler on one node, start workers on rest
  - Urika-XC can do this automatically:

    -
      ```
      start_analytics --dask-env mydask
      ```
  - Otherwise can use ssh or srun/aprun (details will vary based on your system)

Demo: dask.distributed on XC

Legal Disclaimer

Information in this document is provided in connection with Cray Inc. products. No license, express or implied, to any intellectual property rights is granted by this document.

Cray Inc. may make changes to specifications and product descriptions at any time, without notice.

All products, dates and figures specified are preliminary based on current expectations, and are subject to change without notice.

Cray hardware and software products may contain design defects or errors known as errata, which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Cray uses codenames internally to identify products that are in development and not yet publicly announced for release. Customers and other third parties are not authorized by Cray Inc. to use codenames in advertising, promotion or marketing and any use of Cray Inc. internal codenames is at the sole risk of the user.

Performance tests and ratings are measured using specific systems and/or components and reflect the approximate performance of Cray Inc. products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance.

The following are trademarks of Cray Inc. and are registered in the United States and other countries: CRAY and design, SONEXION, URIKA and YARCDATA. The following are trademarks of Cray Inc.: CHAPEL, CLUSTER CONNECT, CLUSTERSTOR, CRAYDOC, CRAYPAT, CRAYPORT, DATAWARP, ECOPHLEX, LIBSCI, NODEKARE, REVEAL. The following system family marks, and associated model number marks, are trademarks of Cray Inc.: CS, CX, XC, XE, XK, XMT and XT. The registered trademark LINUX is used pursuant to a sublicense from LMI, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis. Other trademarks used on this website are the property of their respective owners.

Q&A

Michael Ringenburg
mikeri@cray.com

Introduction to Deep Learning, TensorFlow, and Keras

CUG 2018

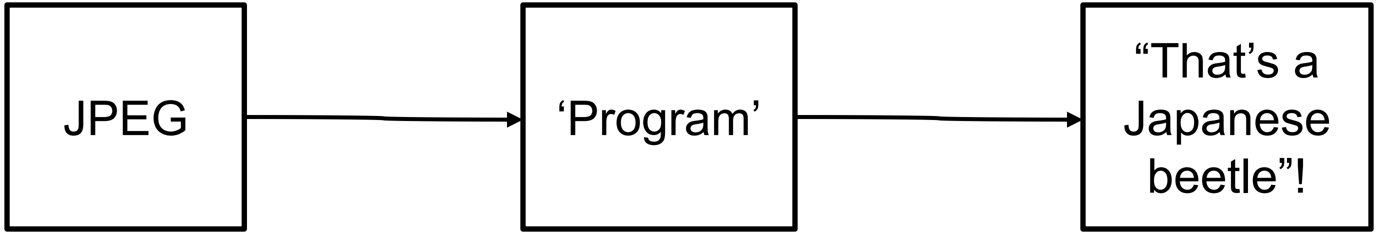
Michael Ringenburg, Cray Inc.

Outline

- Demystifying artificial intelligence / neural networks / deep learning
- What does the computational problem that is deep learning look like?
- Parallel DL – Kristi will cover this in more depth
- What are TensorFlow and Keras?

A Specific Example

- An organic gardener is building a robot in his garage to recognize the 10 insects found in his garden, and decide which ones to kill with a laser
- The robot will have a camera, and will capture JPEG files of the insects
- The robot needs a ‘program’ to classify each JPEG according to which of the 10 kinds of insect was photographed



Inputs & Outputs

- **Our input is a JPEG**
  - 224x224 pixels, 3 colors → a 224x224x3 element vector of the pixel values
- **Our output is a classification**
  - One of 10 categories → a 10 element vector with a “1” in the position representing the category to which the image belongs

How many “IF” statements will we need to figure out that a bunch of pixel values is a Japanese beetle?

This is an Artificial Intelligence Problem

- If you can’t get the output from the input with a bunch of loops and conditionals, it’s AI
- But, if that won’t work, how can we do it?

- Hint #1: Any mapping of inputs to outputs is a function
- Hint #2: A function can be approximated using a (good) approximating function

An Approximating Function

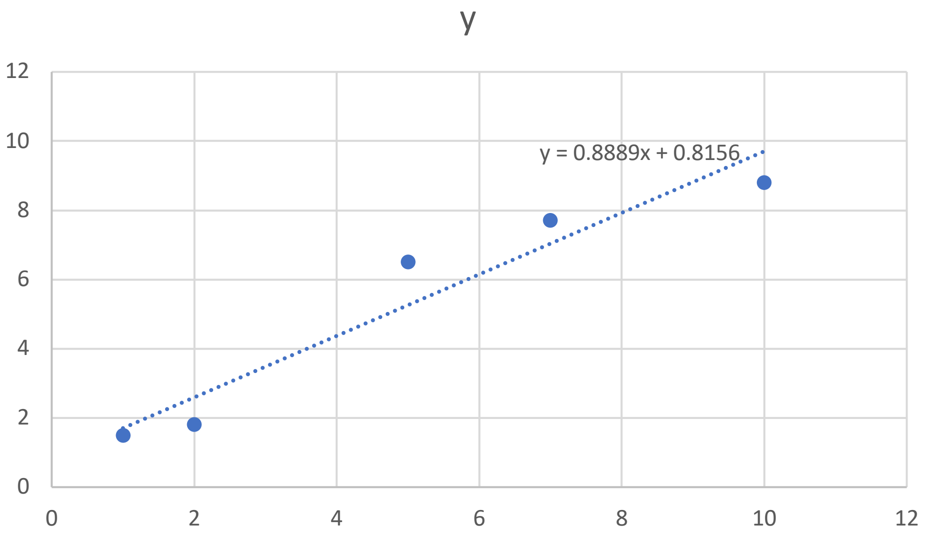
- **How can we determine a good approximating function?**
  - Choose its form (linear, polynomial, …)
  - Minimize the overall error at a finite number of inputs with known outputs - - **fit the curve**
    - We have to find the values of the free parameters of the function that minimize the error – it doesn't matter how we do it

Fitting the curve is a lot like *training* the function to know the answer for arbitrary inputs

Training via Gradient Descent

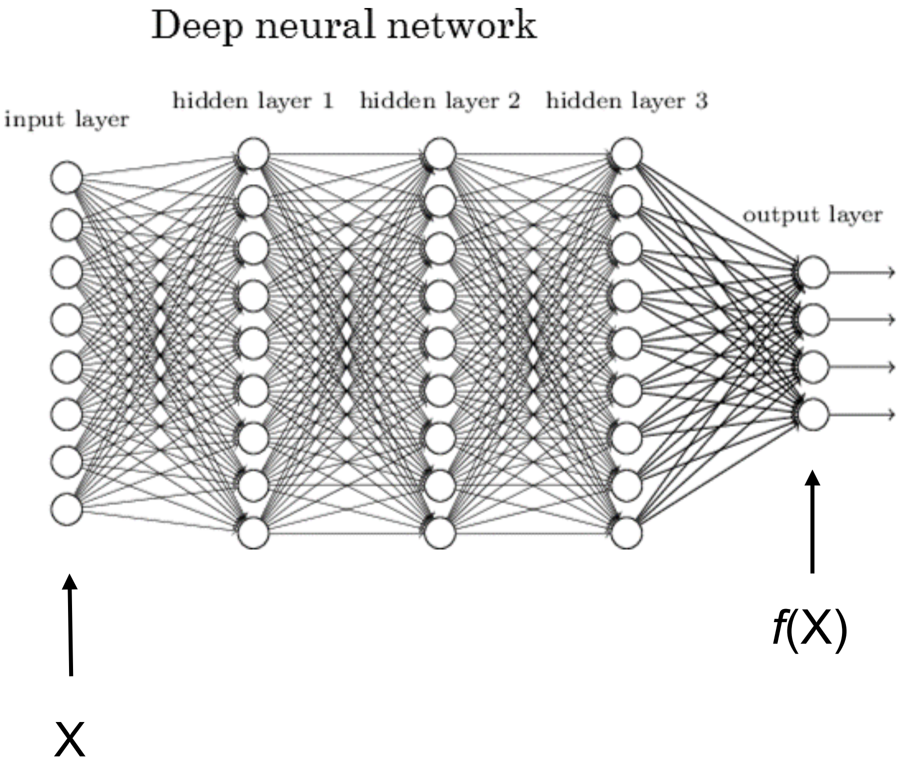
- **We want to approximate y=*f*(x)**
  - Really, we want to find a function that maps a set of inputs to a set of outputs, to some level of accuracy
- **We know $y_i=f(x_i)$, for i=1,N**
- **Iterate:**
  - First iteration only: initialize the free parameters of *f*
  - Calculate error (over our N known points)
  - Calculate gradient of error, as a function of the free parameters of function *f*
  - Adjust the free parameters of function *f* a 'small' distance in the direction of the negative of the error gradient
  - Assess convergence & stop when 'good enough'

Training Error and Validation Error



- Here, we chose the function y=ax+b, with “a” and “b” as the free parameters
- “a” and “b” were chosen to minimize the *training error*, using the 5 points shown
- If we test this function against a distinct set of known data points, we could determine the *validation error*

A Really Useful Kind of Function



- **This image shows a *deep neural network***
  - An approximating function, with free parameters called *weights* and *biases*
  - Deep networks have been found to be especially powerful
  - Neural networks can approximate any continuous function arbitrarily well

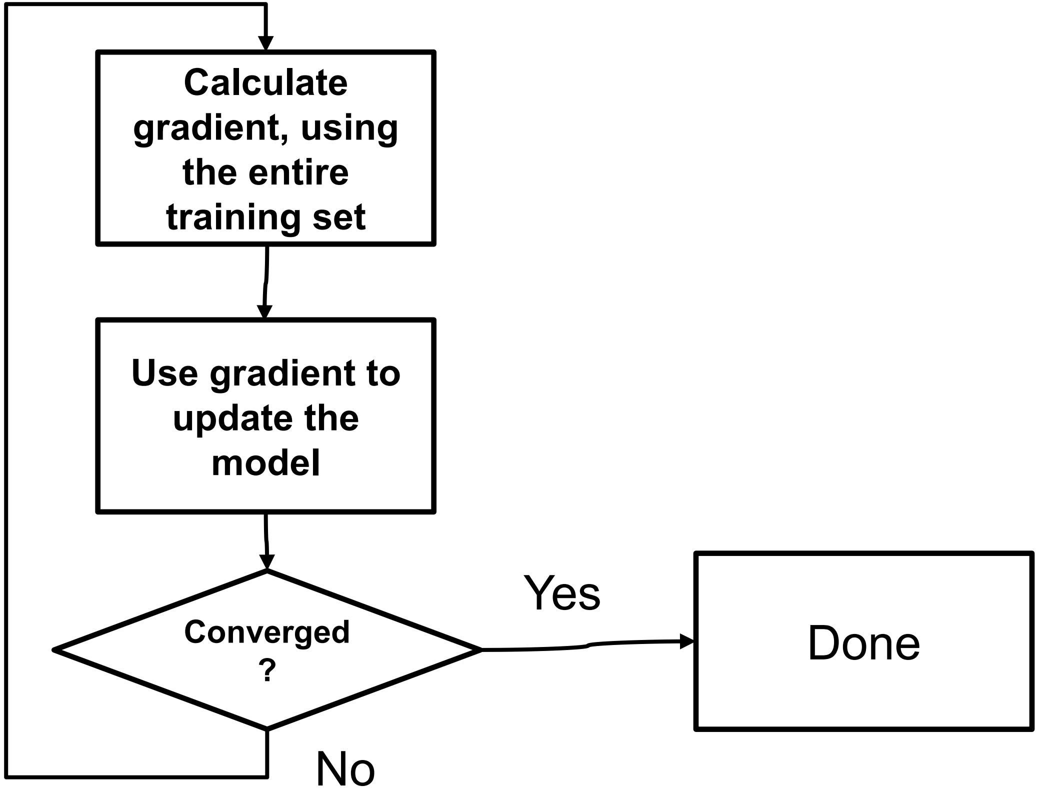
The Big Picture

- **Training a “sufficiently complex” neural network on a “large” and “representative” data set should allow it to “know” about novel data**
  - If we show the neural network 1,000,000 pictures of cats, it should recognize new pictures of cats
  - If we only show the network pictures of black cats, it might not recognize white cats
  - If the network only has 4 “neurons”, it probably can’t learn to recognize cats

Some Terminology

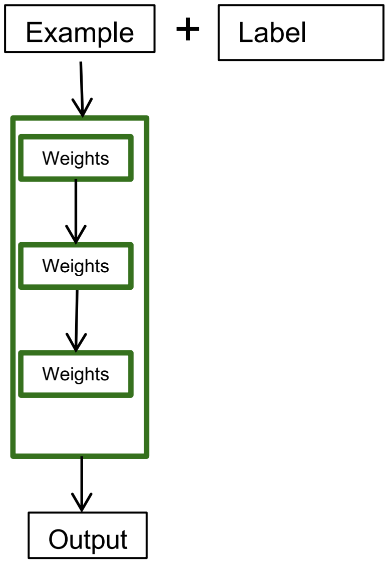
- **The training data consists of training *examples***
  - Each example is an input with a known correct output, called a *label*
  - Having labeled examples is a special but common case, and we won’t go deeper on this topic today
- **A subset of the training data is often called a *minibatch***
- **One ‘trip’ through the whole training set is called an *epoch***
  - Often, bookkeeping, convergence testing, checkpointing, etc. are done after each epoch

Gradient Descent Algorithm


Calculate gradient, using the entire training set
Use gradient to update the model
Converged ?
Yes
Done
No

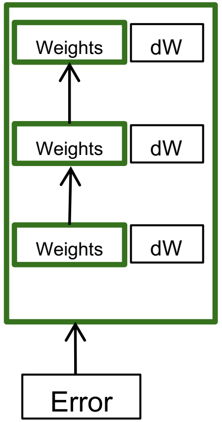
Training Schematic

Feedforward



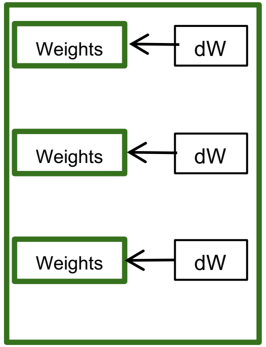
One or more training examples *feedforward* through the layers of *weights*, producing an output

Backpropagate



The error, which is the difference between the label and the output, is *backpropagated* through the layers, producing the *gradients*

Update



The weights are updated by adding the gradients (scaled by a multiplier) to them

Feedforward and backpropagate are much more expensive than update (>100X)

Variations on the Gradient Descent Algorithm

- **Stochastic Gradient Descent**
  - A gradient is calculated, and the model is updated, for each training example
- **Batch Gradient Descent**
  - The training examples are divided into minibatches
  - A gradient is calculated and the model is updated for each minibatch
- **Strict Gradient Descent is seldom if ever used**
- **Strict Stochastic Descent is seldom if ever used**
- **Batch Gradient Descent is almost always used**
  - And, everyone calls it Stochastic Gradient Descent (SGD)

Parallelizing SGD

- **Data parallel methods**
  - “Minibatch Parallel”
  - Every worker independently calculates a “local gradient” using a “local minibatch”
  - All workers participate in an allreduce, or communicate with a parameter server, to average all the gradients and synchronize with other workers
- **Model parallel methods**
  - Break the neural network up – different layers on different nodes
  - Useful if the model is too large for a single node
  - But often more communication that data parallel methods

For More Information…

A good overview:

Efficient Processing of Deep Neural Networks: A Tutorial and Survey

https://arxiv.org/abs/1703.09039

Don’t miss Alex Heye’s talk on Tuesday!

Scaling Deep Learning without Impacting Batchsize

Technical Session 9C

Tuesday, 3:30-4:00PM

TensorFlow

- Developed by Google
- Most popular DL framework
- Large open source community
- APIs for
  - Python
  - C++
  - Go
  - Java
- Optimized for CPU and GPU architectures
- Ships with Urika-XC
- Learn TensorFlow
  - Docs: https://www.tensorflow.org/get_started/
  - Programmer's Guide: https://www.tensorflow.org/programmers_guide/
  - Tutorials: https://www.tensorflow.org/tutorials/

Keras

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3,3),activation='relu’,input_shape=ish))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(), metrics=['accuracy'])
model.fit(x_train, y_train, batch_size=batch_size,
          epochs=epochs, verbose=1, validation_data=(x_test, y_test))
score = model.evaluate(x_test, y_test, verbose=0)
```

- High-level neural networks API – just add layers!
- Compatible with TensorFlow, Theano, CNTK (and others)

Keras

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3,3),activation='relu’,input_shape=ish))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(), metrics=['accuracy'])
model.fit(x_train, y_train, batch_size=batch_size,
          epochs=epochs, verbose=1, validation_data=(x_test, y_test))
score = model.evaluate(x_test, y_test, verbose=0)
```

Construct Model

- High-level neural networks API – just add layers!
- Compatible with TensorFlow, Theano, CNTK (and others)

Keras

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3,3),activation='relu’,input_shape=ish))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(), metrics=['accuracy'])
model.fit(x_train, y_train, batch_size=batch_size,
          epochs=epochs, verbose=1, validation_data=(x_test, y_test))
score = model.evaluate(x_test, y_test, verbose=0)
```

Configure Model for Training

- High-level neural networks API – just add layers!
- Compatible with TensorFlow, Theano, CNTK (and others)

Keras

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3,3),activation='relu',input_shape=ish))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(), metrics=['accuracy'])
model.fit(x_train, y_train, batch_size=batch_size,
          epochs=epochs, verbose=1, validation_data=(x_test, y_test))
score = model.evaluate(x_test, y_test, verbose=0)
```

Train Model

- High-level neural networks API – just add layers!
- Compatible with TensorFlow, Theano, CNTK (and others)

Keras

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3,3),activation='relu’,input_shape=ish))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(), metrics=['accuracy'])
model.fit(x_train, y_train, batch_size=batch_size,
          epochs=epochs, verbose=1, validation_data=(x_test, y_test))
score = model.evaluate(x_test, y_test, verbose=0)
```

Evaluate Model

- High-level neural networks API – just add layers!
- Compatible with TensorFlow, Theano, CNTK (and others)

Demo: Keras MNIST

Legal Disclaimer

Information in this document is provided in connection with Cray Inc. products. No license, express or implied, to any intellectual property rights is granted by this document.

Cray Inc. may make changes to specifications and product descriptions at any time, without notice.

All products, dates and figures specified are preliminary based on current expectations, and are subject to change without notice.

Cray hardware and software products may contain design defects or errors known as errata, which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Cray uses codenames internally to identify products that are in development and not yet publicly announced for release. Customers and other third parties are not authorized by Cray Inc. to use codenames in advertising, promotion or marketing and any use of Cray Inc. internal codenames is at the sole risk of the user.

Performance tests and ratings are measured using specific systems and/or components and reflect the approximate performance of Cray Inc. products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance.

The following are trademarks of Cray Inc. and are registered in the United States and other countries: CRAY and design, SONEXION, URIKA and YARCDATA. The following are trademarks of Cray Inc.: CHAPEL, CLUSTER CONNECT, CLUSTERSTOR, CRAYDOC, CRAYPAT, CRAYPORT, DATAWARP, ECOPHLEX, LIBSCI, NODEKARE, REVEAL. The following system family marks, and associated model number marks, are trademarks of Cray Inc.: CS, CX, XC, XE, XK, XMT and XT. The registered trademark LINUX is used pursuant to a sublicense from LMI, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis. Other trademarks used on this website are the property of their respective owners.

Q&A

Michael Ringenburg

mikeri@cray.com

CRAY

Scalable Deep Learning with the Cray PE Machine Learning Plugin

CUG 2018

Kristi Maschhoff, Cray Inc.

Don’t miss Pete Mendygral’s talk on Thursday!

High Performance Scalable Deep Learning with the Cray Programming Environments Machine Learning Plugin

Technical Session 29A

1:00-2:30PM

Peter Mendygral, Nick Hill, Krishna Kandalla, Diana Moise, and Jacob Balma (Cray Inc.) and Marcel Schongens (Swiss National Supercomputing Centre)

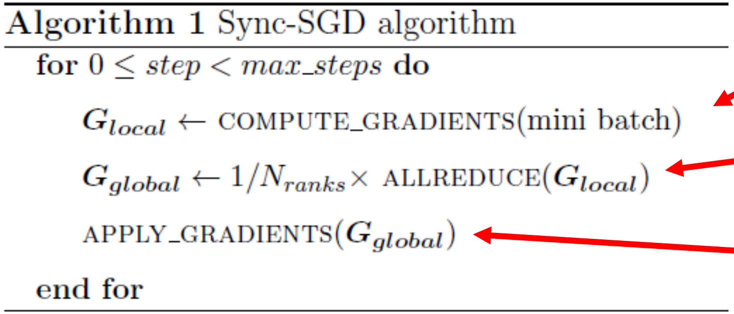
HPC Attributes of Deep Learning

HPC Attributes

- **DL training is a classic high-performance computing problem which demands:**
  - Large compute capacity in terms of FLOPs, memory capacity and bandwidth
  - A performant interconnect for fast communication of gradients and model parameters
  - Parallel I/O and storage with sufficient bandwidth to keep the compute fed at scale

Data Parallelism - Collective-based Synchronous SGD

- Data parallel training divides a global mini-batch of examples across processes
- Each process computes gradients from their local mini-batch
- Average gradients across processes
- All processes update their local model with averaged gradients (all processes have the same model)



Algorithm 1 Sync-SGD algorithm

```
for $0 \leq step < max\_steps$ do
    $G_{local} \leftarrow$ COMPUTE_GRADIENTS(mini batch)
    $G_{global} \leftarrow 1/N_{ranks} \times$ ALLREDUCE($G_{local}$)
    APPLY_GRADIENTS($G_{global}$)
end for
```

Compute intensive

Communication intensive

Typically not much compute

- Not shown is the I/O activity of reading training samples (and possible augmentation)

Why do we want to scale?

- **Deep Network Training**
  - We can strong scale training time-to-accuracy provided
    - Number of workers (e.g., # nodes) << number of training examples
    - Learning rate for particular batch size / scale is known

- **Hyper-Parameter Optimization**
  - For problems and datasets where baseline accuracy is not known
    - learning rate schedule
    - momentum
    - batch size
  - Evolve topologies if good architecture is unknown (common with novel datasets / mappings)
    - Layer types, width, number filters
    - Activation functions, drop-out rates

Parallelization Methods for DL

Parallelization Techniques

- **Data Parallelism**
  - As described earlier, divides global mini-batch among processes
  - Two methods for this:
    - Synchronous: single model (possibly replicated across all processes) updated with globally averaged gradients every iteration
    - Asynchronous: processes provide gradients every iteration but are allowed to fall out of sync from one another. Processes each have their own model that may or may not be the same as any other process

- **Model Parallelism**
  - Single model with layers decomposed across processes
  - Activations communicated between processes

- **Examples will focus on synchronous data parallel approach**

Distributed TensorFlow

- **TensorFlow has a native method for parallelism across nodes**
  - ClusterSpec API
  - Uses gRPC layer in TensorFlow based on sockets

- **Can be difficult to use and optimize**

- **User must specify**
  - hostnames and ports for all worker processes
  - hostnames and ports for all parameter server processes (see next slide)
  - # of workers
  - # of parameter server processes
  - Chief process of workers

Distributed TensorFlow

- **Number of parameter servers (PS) processes to use is not clear**
  - Too few PS results in many-to-few communication pattern (very bad) and stalls delivering updated parameters
  - Too many PS results in many-to-many communication pattern (also bad)
- **Users typically have to pick a scale and experiment for best performance**

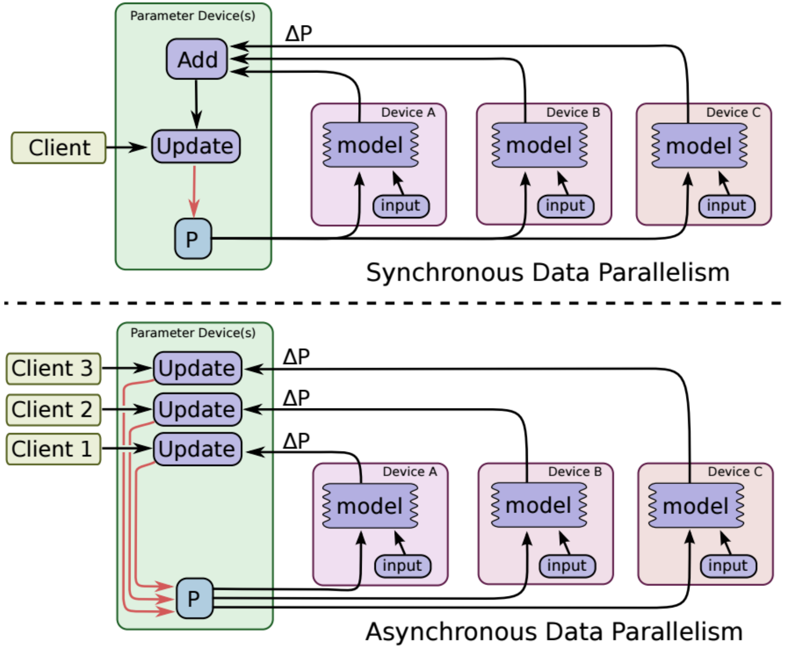
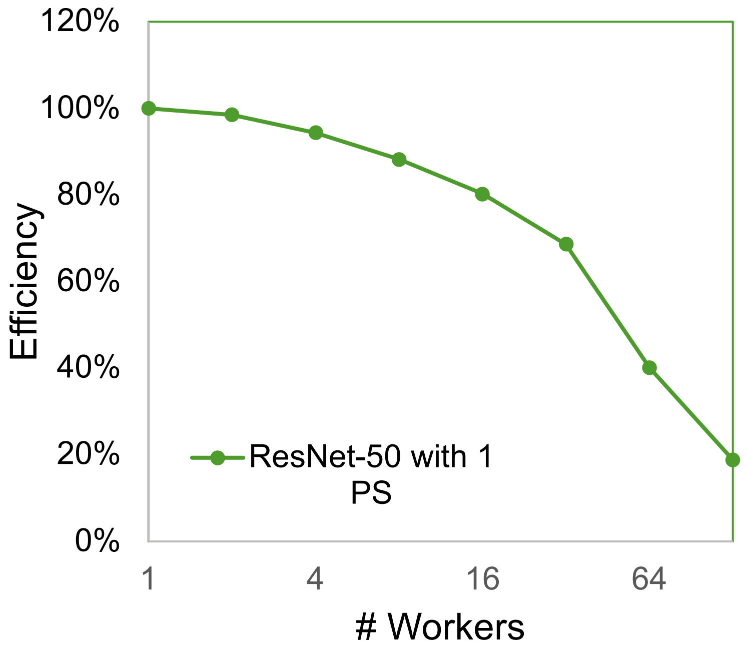
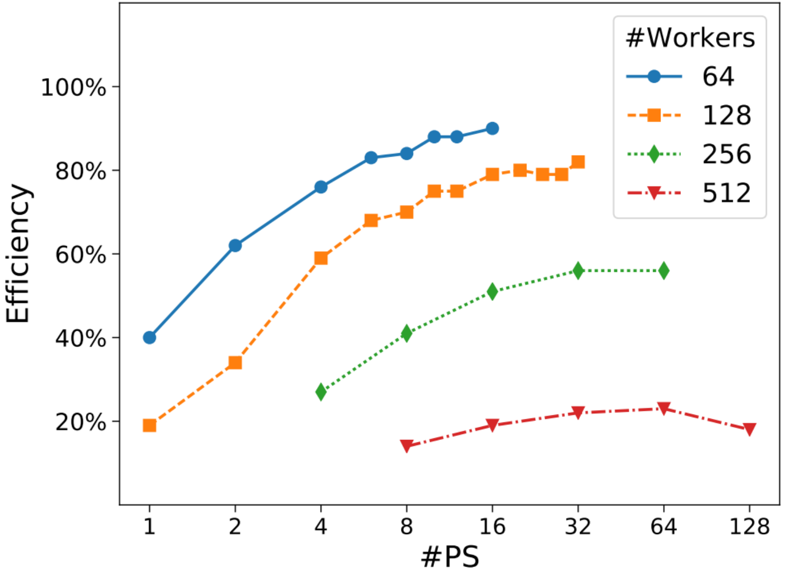


Figure 7: Synchronous and asynchronous data parallel training

Distributed TensorFlow Scaling on Cray XC40 - KNL



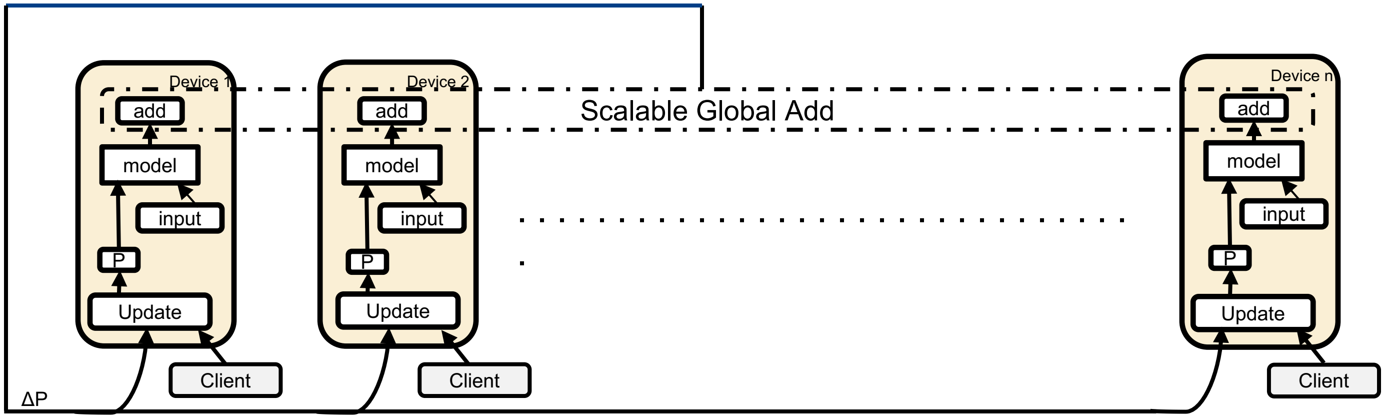


From Mathuriya et al. @ NIPS 2017

MPI-based Data Parallel TensorFlow

- **The performance and usability issues with distributed TensorFlow can be addressed by adopting an MPI communication model**
- **TensorFlow does have an MPI option, but it only replaces point to point operations in gRPC with MPI**
  - Collective algorithm optimization in MPI not used
- **Other frameworks, such as Caffe and CNTK, include MPI collectives**
- **An MPI collective based approach would eliminate the need for PS processes and likely be optimized without intervention from the user**

Scalable Synchronous Data Parallelism



- Note there are no PS processes in this model
- Resources dedicated to gradient calculation

Uber Horovod

- **Uber open source addon for TensorFlow *only* that replaces native optimizer class with a new class**
  - Horovod adds an allreduce between gradient computation and model update in this class

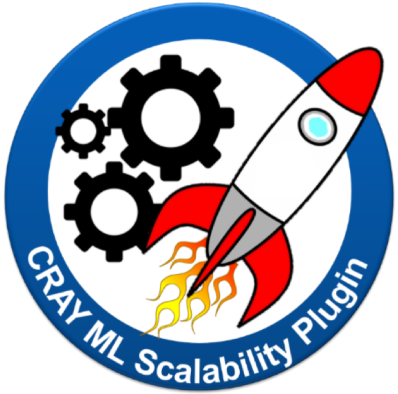
- **New Python class includes NCCL and MPI collective reductions for gradient aggregation**

- **https://github.com/uber/horovod**

- **No modifications to TensorFlow source required**
  - User modifies Python training script instead

Cray Programming Environment Machine Learning Plugin (CPE ML Plugin)

- DL communication plugin with Python and C APIs
- Optimized for TensorFlow but also portable to other frameworks
  - Callable from C/C++ source
  - Called from Python if data stored in NumPy arrays or Tensors
- Like Horovod does not require modification to TensorFlow source
  - User modifies training script
- Uses custom allreduce specifically optimized for DL workloads
  - Optimized for Cray Aries interconnect and IB for Cray clusters
- Tunable through API and environment variables
- Supports multiple gradient aggregations at once with thread teams
  - Useful for Generative Adversarial Networks (GAN), for example



CPE ML Plugin Example

Tensorflow Training Script Modifications

- **Both Horovod and CPE ML Plugin require some modifications to a serial training script**

- **For the CPE ML Plugin the changes are**
  - Importing the Python module (ml_comm)
  - Initialize the module
    - Possibly configure the thread team(s) for specific uses
  - Broadcast initial model parameters
  - Possible modifications to learning rate decay schedules and other mini-batch size dependent parameters to account for the effective mini-batch size across all processes
  - Incorporate gradient aggregation between gradient computation and model update
  - Finalize the Python module

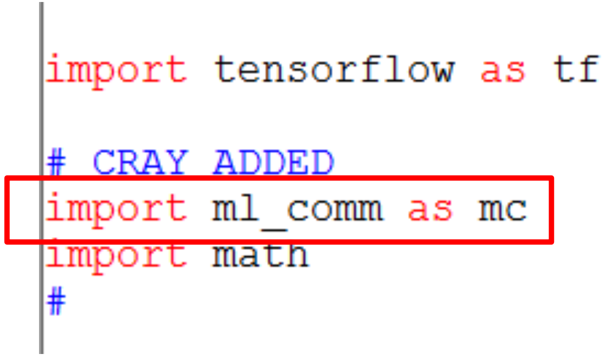
MNIST Example

- Dataset of handwritten digits from 0-9
- Simple CNN can be used to identify handwritten digits

- This example is adapted from the TensorFlow official MNIST example
- https://github.com/tensorflow/models/tree/master/official/mnist
- Modified script included with CPE ML Plugin
  - `module load craype-ml-plugin-py2/1.1.0`
  - `less $CRAYPE_ML_PLUGIN_BASEDIR/examples/tf_mnist/mnist.py`

CPE ML Plugin - Import

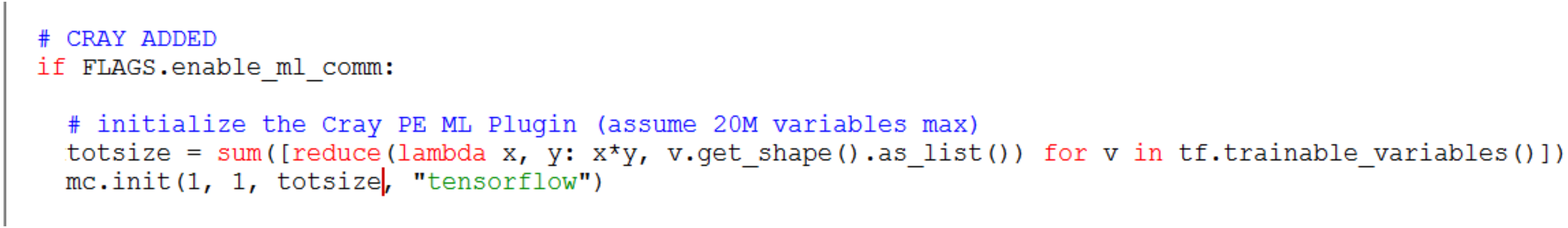
- Access the Python API by importing the module



```
import tensorflow as tf

# CRAY ADDED
import ml_comm as mc
import math
#
```

CPE ML Plugin - Initialization

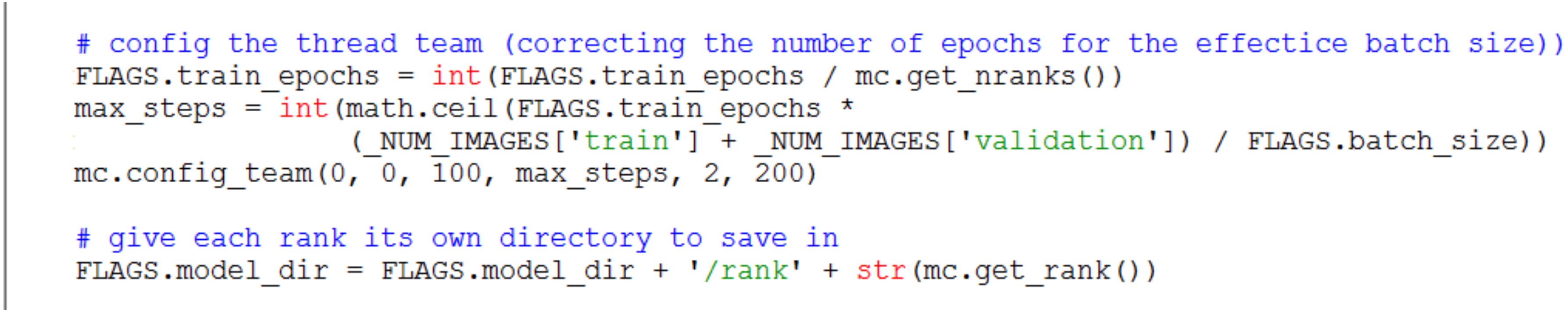
- **Compute the number of trainable variables in the model**
  - Required for the CPE ML Plugin to pre-allocate needed communication buffers
- **Example for init sets up a single thread team with one local thread per team, per MPI rank**
  - For GANs, may want to use 2 teams, to have 2 reductions in flight
  - Number of threads per team, tunable parameter, biggest gain (1 to 2)



```
# CRAY ADDED
if FLAGS.enable_ml_comm:

  # initialize the Cray PE ML Plugin (assume 20M variables max)
  totsize = sum([reduce(lambda x, y: x*y, v.get_shape().as_list()) for v in tf.trainable_variables()])
  mc.init(1, 1, totsize, "tensorflow")
```

CPE ML Plugin – Team Configuration

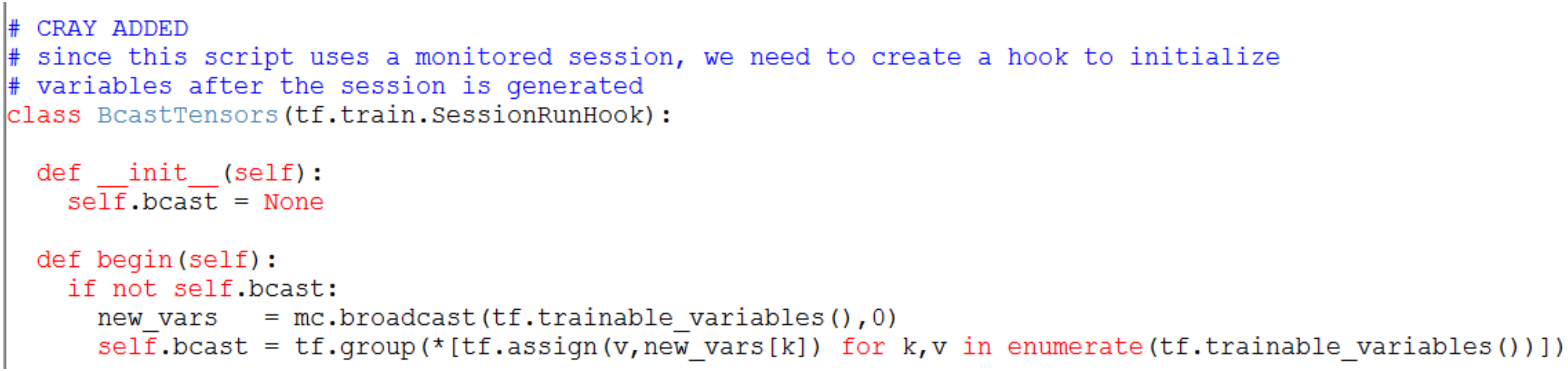
- **Set the maximum number of steps (mini batches) to train for**
  - Verbose output every 200 steps
- **Also set output path to rank-specific location**



```
# config the thread team (correcting the number of epochs for the effectice batch size))
FLAGS.train_epochs = int(FLAGS.train_epochs / mc.get_nranks())
max_steps = int(math.ceil(FLAGS.train_epochs *
                (_NUM_IMAGES['train'] + _NUM_IMAGES['validation']) / FLAGS.batch_size))
mc.config_team(0, 0, 100, max_steps, 2, 200)

# give each rank its own directory to save in
FLAGS.model_dir = FLAGS.model_dir + '/rank' + str(mc.get_rank())
```

CPE ML Plugin – Broadcast Initial Model

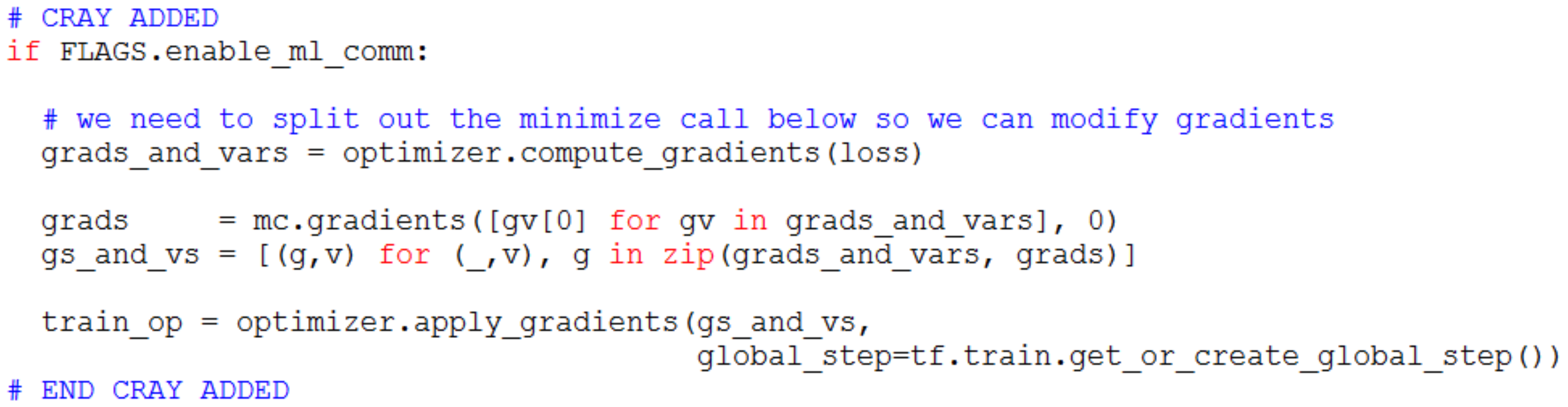
- Broadcast initial model parameter values from rank 0 to all other ranks
- Then assign broadcasted values locally



```
# CRAY ADDED
# since this script uses a monitored session, we need to create a hook to initialize
# variables after the session is generated
class BcastTensors(tf.train.SessionRunHook):

  def __init__(self):
    self.bcast = None

  def begin(self):
    if not self.bcast:
      new_vars    = mc.broadcast(tf.trainable_variables(),0)
      self.bcast = tf.group(*[tf.assign(v,new_vars[k]) for k,v in enumerate(tf.trainable_variables())])
```

CPE ML Plugin – Gradient Aggregation

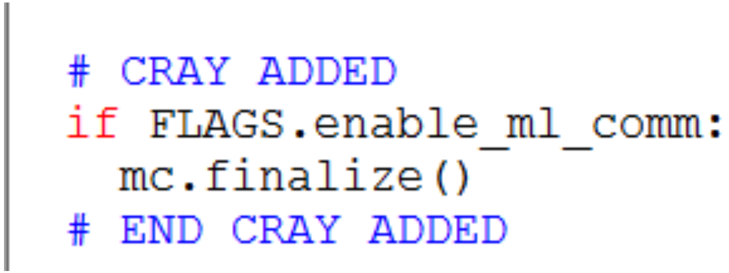
- Perform gradient averaging across all ranks between local gradient calculation and model update



```
# CRAY ADDED
if FLAGS.enable_ml_comm:

  # we need to split out the minimize call below so we can modify gradients
  grads_and_vars = optimizer.compute_gradients(loss)

  grads     = mc.gradients([gv[0] for gv in grads_and_vars], 0)
  gs_and_vs = [(g,v) for (_,v), g in zip(grads_and_vars, grads)]

  train_op = optimizer.apply_gradients(gs_and_vs,
                                       global_step=tf.train.get_or_create_global_step())
# END CRAY ADDED
```

CPE ML Plugin – Finalize

- After all training steps are complete clean up data structures and MPI



```
# CRAY ADDED
if FLAGS.enable_ml_comm:
  mc.finalize()
# END CRAY ADDED
```

CPE ML Plugin – More information

- **module avail craype-ml-plugin**
  - craype-ml-plugin-py2/1.0.1(TF 1.3)
  - craype-ml-plugin-py3/1.0.1(TF 1.3)
- **Load modules in order:**
  - module load cray-python
  - module load craype-ml-plugin-py3

- **Please refer to CPE ML Plugin manpage for more details on usage**
  - `man intro_ml_plugin`
  - `Or from python shell`

```
$ python3
>>> import ml_comm as mc
>>> help(mc)
>>> help(mc.init)
```

- **Examples directory**
  - $CRAYPE_ML_PLUGIN_BASEDIR/examples

CPE ML Plugin – Execution using Urika-XC

Procedure:

1. Load the **analytics** module

```
$ module load analytics
```

2. Allocate interactive nodes via SLURM or PBS

```
$ salloc --nodes=2 --exclusive --gres=gpu -C P100
```

```
$ qsub –I –q p100 –lnodes=2
```

3. I

```
$ run_training –n 2 --ppn 1 --cudnn-libs /path/to/cudnn-8.0-v51/cuda/lib64
  --no-node-list “python mnist.py --enable_ml_comm
  --data_dir=[mnist data pth] --model_dir=[train dir]”
```

CPE ML Plugin Execution – Native XC

Procedure:

1. Locate/Install Tensorflow and set PYTHONPATH
2. Load the module cray-python and the craype-ml-plugin modules

```
$ module load cray-python
$ module load craype-ml-plugin.py3
```

3. Allocate interactive nodes (examples uses SLURM)

```
$ salloc --nodes=2 --exclusive --gres=gpu -C P100
```

4. Execute the modified Tensorflow training script using srun

```
$ srun --ntasks=2 --ntasks-per-node=1 --cpu_bind=none  python3 mnist.py
   --enable_ml_comm --data_dir=[mnist data pth] --model_dir=[train dir]
```

Instructions for Running Tensorflow on Piz Daint

```
% module load daint-gpu
% module avail Tensorflow
    TensorFlow/1.2.1-CrayGNU-17.08-cuda-8.0-python3(default)
    TensorFlow/1.3.0-CrayGNU-17.08-cuda-8.0-python3
    TensorFlow/1.4.1-CrayGNU-17.08-cuda-8.0-python3
    TensorFlow/1.4.1-CrayGNU-17.12-cuda-8.0-python3
    TensorFlow/1.7.0-CrayGNU-17.12-cuda-8.0-python3
% module load TensorFlow/1.3.0-CrayGNU-17.08-cuda-8.0-python3
    Loads cray-python, cuDNN, etc
% module load craype-ml-plugin-py3/1.0.1
```

```
$ salloc --nodes=2 --exclusive --constraint=gpu
```

Note: SLURM option --constraint=gpu on Piz Daint allocates the XC50 Intel Haswell 12-core nodes with GPU devices and automatically sets the SLURM option **--gres=gpu**

Class Exercise: Tensorflow + CPE ML Plugin

- **Run MNIST model using CPE ML Plugin on Piz Daint**
- **Step-by-Step Instructions:**
  - /scratch/snx3000/kristyn/CUG2018/Tensorflow/README
  - Script provided to convert raw MNIST data to TFRecords file format, but can also use
    - --data_dir=/scratch/snx3000/kristyn/datasets/mnist_data

Using CPE ML Plugin with Keras

- **Instructions for installing Keras**
  - Note: need to make sure Tensorflow and Keras versions are compatible
  - For Tensorflow 1.3.0 need to use Keras 2.1.5
    - git clone https://github.com/keras-team/keras.git
    - cd keras
    - git checkout tags/2.1.5
  - Note:
    - Also make sure to use the Keras examples from the 2.1.5 branch

Using CPE ML Plugin with Keras

- **Modifications:**
  - Leverage the existing Tensorflow backend to keras
    - Modify backend file to include CPE ML modifications
      - keras-2.1.5/keras/keras/backend/tensorflow_backend.py
    - Recompile
  - Small modifications to user-level scripts
    - Passing information to the backend for initialization, team configuration, and finalize

- **Thanks to Diana Moise!**

Instructions for installing Keras (cont)

- **Native XC**

```
% module load cray-python
% module load TensorFlow/1.3.0-CrayGNU-17.08-cuda-8.0-python3
% python3 setup.py install --user
```

- **Inside Urika-XC container**
  - Install into root: cd to keras directory, run install

    ```
    % python setup.py install --user
    ```

  - Install into conda environment

    ```
    % conda create --clone py36_tf_cpu --name py36_keras_cray_ml
    % python setup.py install –user
    % export PYTHONPATH=/opt/tensorflow_cpu:$PYTHONPATH
    ```

Keras + CPE ML Plugin: Modifications to backend

- **Added to tensorflow_backend.py**
  - Import ml_comm library and add new functions

```
# CRAY ADDED
import ml_comm as mc

# threads per team, number of teams, algorithm to use, grads len, total steps, ksteps, v, freq
def mc_init(th_team, n_teams, alg, grads_len, total_steps, ksteps, v, freq):
    mc.init(th_team, n_teams, grads_len, "tensorflow")
    for team in range(n_teams):
        mc.config_team(team, alg, ksteps, total_steps, v, freq)

def mc_get_rank():
    return mc.get_rank()

def mc_finalize():
    mc.finalize()
# END CRAY ADDED
```

Keras + CPE ML Plugin: Modifications to backend

- **Broadcast Initial Model**
  - A bit complicated for a slide
    - See source example:
      - /scratch/snx3000/kristyn/CUG2018/keras/ tensorflow_backend_ml_comm.py

- **Gradient Aggregation**

```
# CRAY MOD
    grads =tf.gradients(loss,variables,colocate_gradients_with_ops=True)
    grads_mc = mc.gradients(grads, 0)
    return grads_mc
#     return tf.gradients(loss, variables, colocate_gradients_with_ops=True)
# END CRAY MOD
```

Keras + CPE ML Plugin: Changes to user script

- Starting point was keras/examples/mnist_cnn.py
- Import some extra libraries at the beginning

```
# CRAY ADDED
import numpy as np
import os
import math
nnodes = int(os.environ['SLURM_NNODES'])
# END CRAY ADDED
```

- No other changes to user script until after we compile model

```
model.compile(... )
#CRAY ADDED
# calculate input parameters to pass through to backend
K.mc_init( …)
#END CRAY ADDED
model.fit( … )
```

Keras + CPE ML Plugin: Changes to user script

- Need to determine number of trainable parameters, adjust number of epochs, calculate max_steps, then call backend to perform init

```
# CRAY ADDED

# Determine number of trainable variables
trainable_count = int(np.sum([K.count_params(p) for p in set(model.trainable_weights)]))

# Adjust epochs based on parallel throughput
epochs = int(epochs/nnodes)

# Compute max_steps
ntrain_samples = x_train.shape[0]
ntest_samples = x_test.shape[0]
total_steps = int(math.ceil(epochs * (ntrain_samples + ntest_samples)/batch_size))

# threads per team, number of teams, algorithm to use, grads len, total steps, ksteps, v, freq
K.mc_init(1, 1, 0, trainable_count, total_steps, 2000, 2, 1000)

#  END CRAY ADDED
```

Running Keras + CPE ML Plugin on Piz Daint

Load modules

```
$ module load daint-gpu
$ module load TensorFlow/1.3.0-CrayGNU-17.08-cuda-8.0-python3
$ module load craype-ml-plugin-py3/1.0.1
```

Allocate GPU nodes from SLURM

```
$ salloc --nodes=2 --exclusive --constraint=gpu
```

Run

```
srun -n 2 python3 keras-2.1.5/keras/examples/mnist_cnn_ml.py
```

Class Exercise: Keras + CPE ML Plugin

- Install keras
- Run keras mnist_cnn model (or other examples)
- Modify keras tensorflow_backend.py and mnist_cnn model to use CPE ML Plugin
- Apply CPE ML Plugin to other keras example models
  - acgan_mnist model
- Step-by-Step Instructions:
  - /scratch/snx3000/kristyn/CUG2018/Keras/README

Legal Disclaimer

Information in this document is provided in connection with Cray Inc. products. No license, express or implied, to any intellectual property rights is granted by this document.

Cray Inc. may make changes to specifications and product descriptions at any time, without notice.

All products, dates and figures specified are preliminary based on current expectations, and are subject to change without notice.

Cray hardware and software products may contain design defects or errors known as errata, which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Cray uses codenames internally to identify products that are in development and not yet publicly announced for release. Customers and other third parties are not authorized by Cray Inc. to use codenames in advertising, promotion or marketing and any use of Cray Inc. internal codenames is at the sole risk of the user.

Performance tests and ratings are measured using specific systems and/or components and reflect the approximate performance of Cray Inc. products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance.

The following are trademarks of Cray Inc. and are registered in the United States and other countries: CRAY and design, SONEXION, URIKA and YARCDATA. The following are trademarks of Cray Inc.: CHAPEL, CLUSTER CONNECT, CLUSTERSTOR, CRAYDOC, CRAYPAT, CRAYPORT, DATAWARP, ECOPHLEX, LIBSCI, NODEKARE, REVEAL. The following system family marks, and associated model number marks, are trademarks of Cray Inc.: CS, CX, XC, XE, XK, XMT and XT. The registered trademark LINUX is used pursuant to a sublicense from LMI, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis. Other trademarks used on this website are the property of their respective owners.

Q&A

Kristi Maschhoff

kristyn@cray.com

Analytics and AI on XC Tutorial
Setting up an R Environment
CUG 2018

Kristi Maschhoff, Cray Inc.

What is R?

- **R project for Statistical Computing**
  - https://www.r-project.org
  - Environment for statistical computing and graphics
  - "GNU S"
  - Freely available – but note most R packages have licenses
    - (GPL-2, GPL-3, MIT, Apache, etc.)
  - Latest Version R 3.5.0 (Joy in Playing)
    - R version 3.5.0 (2018-04-23) -- "Joy in Playing"
- **CRAN - The Comprehensive R Archive Network**
  - https://cran.r-project.org
  - Network of ftp and web servers that store identical, up-to-date, versions of code and documentation for R
  - R manuals
    - https://cran.r-project.org/doc/manuals/

Interactivity and R

- **R community was developed with the goal of interactive exploration of data**
  - Basic R interactive console – provided with standard distribution
  - Many R users work use R using an IDE
- **RStudio is by far the most popular IDE for R**
  - R Markdown files and R Notebooks
  - Files have extension .Rmd
- **R can also be run using Jupyter Notebooks**
  - Install IRKernel

What we plan to cover in the tutorial

- **Setting up an R environment on XC**
- **R provided with Urika-XC**
  - Urika-1.1 release pulls R from the EPEL repository
  - For Urika-1.2 release, the plan is to provide optimized R using Cray libsci inside the container
- **Update on R support on XC built with Cray libsci**
  - module load cray-R
  - Currently support R-3.3.3
- **Build Instructions - for those needing additional customization**
  - Build R using gcc/gfortran
  - Build R using gcc/gfortran + Cray libsci
  - Build R using gcc/gfortran + Cray libsci_acc (GPUs)
  - Build R using Intel C++ and Fortran Compilers + MKL
  - Build R using gcc/gfortran + MKL
- **Installing R packages**
- **Using R from Jupyter Notebooks**
- **Using Anaconda to manage R packages and multiple R versions (environments)**
- **Setting up a R cluster using "parallel" package**
- **Setting up a pdbR environment (pdbMPI)**

Setting up an R environment on XC

- Base R Install
  - Why build from source?
    - Allows one to build optimized versions which use optimized math libraries (Cray libsci, Intel MKL)
    - Download most recent version from CRAN
      - R-3.5.0.tar.gz
      - wget https://cran.r-project.org/src/base/R-3/R-3.0.0.tar.gz
    - Enable additional non-default capabilities
      - Memory profiling
  - CRAN repository also provides precompiled binaries
    - Linux, OS X, Windows

Be prepared to fight with the configure script!

- **R build uses a configure script to build Makefile**
  - Be prepared to fight!
- **This is especially true when trying to customize R**
  - Linking to vendor BLAS/LAPACK
  - Enabling additional capabilities
  - Carefully review config.log

Simple R build from source using gcc/gfortran

- module swap PrgEnv-cray PrgEnv-gnu
- wget https://cran.r-project.org/src/base/R-3/R-3.5.0.tar.gz
- (Note: other mirror sites work as well, for example
- wget http://cran.rstudio.com/src/base/R-3/R-3.5.0.tar.gz
- tar -xzf R-3.5.0.tar.gz
- cd R-3.5.0/

- ./configure --prefix=/tmp/CUG2018/R/sandbox/install/R-3.5.0
- make
- make check; make install
- cd /tmp/CUG2018/R/sandbox/install/R-3.5.0/bin

- ./R
- file R
  - (R: POSIX shell script, ASCII text executable)
- file exec/R
  - exec/R: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked (uses shared libs), for GNU/Linux 3.0.0, not stripped

Example summary output from configure script

```
R is now configured for x86_64-suse-linux-gnu

  Source directory:          .
  Installation directory:
   /tmp/CUG2018/R/sandbox/install/cray-R-3.5.0

  C compiler:                gcc  -g -O2
  Fortran 77 compiler:       gfortran  -g -O2

  Default C++ compiler:      g++   -g -O2
  C++98 compiler:            g++ -std=gnu++98 -g -O2
  C++11 compiler:            g++ -std=gnu++11 -g -O2
  C++14 compiler:            g++ -std=gnu++14 -g -O2
  C++17 compiler:            g++ -std=gnu++17 -g -O2

  Fortran 90/95 compiler:    gfortran -g -O2
  Obj-C compiler:

  Interfaces supported:      X11, tcltk
  External libraries:        readline, BLAS(generic),
                             LAPACK(generic), curl
  Additional capabilities:   PNG, JPEG, TIFF, NLS, cairo, ICU
  Options enabled:           R profiling

  Capabilities skipped:
  Options not enabled:       shared BLAS, memory profiling

  Recommended packages:      yes
```

Installed Packages – Base Install
> installed.packages()[,c("Version","License")]

```
           Version   License
base       "3.5.0"   "Part of R 3.5.0"
boot       "1.3-20"  "Unlimited"
class      "7.3-14"  "GPL-2 | GPL-3"
cluster    "2.0.7-1" "GPL (>= 2)"
codetools  "0.2-15"  "GPL"
compiler   "3.5.0"   "Part of R 3.5.0"
datasets   "3.5.0"   "Part of R 3.5.0"
foreign    "0.8-70"  "GPL (>= 2)"
graphics   "3.5.0"   "Part of R 3.5.0"
grDevices  "3.5.0"   "Part of R 3.5.0"
grid       "3.5.0"   "Part of R 3.5.0"
KernSmooth "2.23-15" "Unlimited"
lattice    "0.20-35" "GPL (>= 2)"
MASS       "7.3-49"  "GPL-2 | GPL-3"
Matrix     "1.2-14"  "GPL (>= 2) | file LICENCE"
```

```
           Version   License
methods    "3.5.0"   "Part of R 3.5.0"
mgcv       "1.8-23"  "GPL (>= 2)"
nlme       "3.1-137" "GPL (>= 2) | file LICENCE"
nnet       "7.3-12"  "GPL-2 | GPL-3"
parallel   "3.5.0"   "Part of R 3.5.0"
rpart      "4.1-13"  "GPL-2 | GPL-3"
spatial    "7.3-11"  "GPL-2 | GPL-3"
splines    "3.5.0"   "Part of R 3.5.0"
stats      "3.5.0"   "Part of R 3.5.0"
stats4     "3.5.0"   "Part of R 3.5.0"
survival   "2.41-3"  "LGPL (>= 2)"
tcltk      "3.5.0"   "Part of R 3.5.0"
tools      "3.5.0"   "Part of R 3.5.0"
utils      "3.5.0"   "Part of R 3.5.0"
```

>capabilities()

- **Description**
  - Report on the optional features which have been compiled into this build of R

Prebuilt versions of R provided on XC

- **Cray PE provides R prebuilt with Cray libsci using the GNU compiler**
  - module load cray-R
  - Currently supported version is 3.3.3
- **R is also provided with Urika-XC**
  - Urika-XC 1.1 release
    - Uses prebuilt R from EPEL repository (R version 3.4.2)
      - Installed in /usr/lib64/R in the Urika-XC container
      - Provides R built as a shared/dynamic library
        - Can use R helper routines such a ”littler”
        - Flag --enable-R-shlib causes the make process to build R as a dynamic (shared) library, typically called libR.so, and link the main R executable against that library
        - Possible performance penalty (10%) mentioned in R install notes – have not verified on XC
    - Primarily included to support sparkR
  - Planned for Urika-XC 1.2 release
    - R version 3.5.0 (or most recent release)
    - Plan to provide R prebuilt with Cray libsci + GNU compiler
    - R built as a shared/dynamic library
    - pbdR Ecosytem pre-installed using Cray MPI (initial base set of packages)
    - Support for using Jupyter notebooks via IRKernel

Build of R using gcc/gfortran + Cray libsci

Example build recipe: (for those needing additional customization)

```
module swap PrgEnv-cray PrgEnv-gnu
wget https://cran.r-project.org/src/base/R-3/R-3.5.0.tar.gz
tar -xzf R-3.5.0.tar.gz
cd R-3.5.0/
```

# Cray provides an environment variable with the path to the libsci directory

```
echo $CRAY_LIBSCI_PREFIX_DIR
```

```
./configure --build=x86_64-suse-linux --prefix=${install_dir} --with-blas="-fopenmp -L${CRAY_LIBSCI_PREFIX_DIR}/lib -lsci_gnu_61_mp"  --with-lapack"-Wl,-ydgetrf”
```

# Note: additional arguments provided with the --with-lapack option to configure is really a workaround
# to a bug in the configure script (Thanks to Faisal Hadi - Cray libsci manager for the fix). This allows the
# configure script to find the LAPACK routine in libsci

```
make
make install
```

Build R using Intel C++ and Fortran Compilers + MKL

```
module swap PrgEnv-cray PrgEnv-intel
wget https://cran.r-project.org/src/base/R-3/R-3.4.0.tar.gz
tar -xzf R-3.4.0.tar.gz

setenv CC icc
setenv CXX icpc
setenv AR xiar
setenv LD xild

setenv CFLAGS “-03 –ipo –qopenmp –xHost”
setenv CXXFLAGS “-03 –ipo –qopenmp –xHost”
setenv MKL “-lmkl_gf_lp64 -lmkl_intel_thread -lmkl_core -liomp5 -lpthread”

./configure --prefix=/lus/scratch/kristyn/R/R-3.4.0/R-3.4.0 CC="icc -mkl" CXX="icpc -mkl" FC="ifort -mkl" F77="ifort -mkl" FPICFLAGS="-fPIC" AR=xiar LD=xild --with-x=no --with-blas=-lmkl --with-lapack=-lmkl

make
make install
cd /lus/scratch/kristyn/R/R-3.4.0/bin
```

https://software.intel.com/en-us/articles/build-r-301-with-intel-c-compiler-and-intel-mkl-on-linux

Simple build using gcc/gfortran + MKL

```
> module load PrgEnv-intel
> module load gcc

> setenv CC gcc
> setenv F77 gfortran
> setenv AR xiar
> setenv LD xild
> setenv MKL “-lmkl_gf_lp64 -lmkl_intel_thread -lmkl_core -liomp5 -lpthread”

> ./configure --prefix=/lus/scratch/kristyn/R/R-3.4.0/R-3.4.0 --with-blas="$MKL" --with-lapack
> make
> make install
> cd /lus/scratch/kristyn/R/R-3.4.0/bin
> ./R
```

https://cran.r-project.org/doc/manuals/r-release/R-admin.html#MKL

Build R + MKL build notes

Default is to build shared libraries:
Useful to print out shared library dependencies to verify MKL is being used

```
> ldd exec/R
linux-vdso.so.1 (0x00007ffc247ed000)
libmkl_gf_lp64.so =>
/opt/intel/compilers_and_libraries_2017.1.132/linux/mkl/lib/intel64_lin/libmkl_gf_lp64.so
(0x00007f47f09da000)
libmkl_intel_thread.so =>
/opt/intel/compilers_and_libraries_2017.1.132/linux/mkl/lib/intel64_lin/libmkl_intel_thread.so
(0x00007f47eefcc000)
libmkl_core.so => /opt/intel/compilers_and_libraries_2017.1.132/linux/mkl/lib/intel64_lin/libmkl_core.so
(0x00007f47ed525000)
libiomp5.so => /opt/intel/compilers_and_libraries_2017.1.132/linux/compiler/lib/intel64/libiomp5.so
(0x00007f47ed182000)
```

Can also specify to build static binary by using --enable-static when running ./configure

Set up simple modulefile

- Create a modulefiles directory
  - /lus/scratch/kristyn/modulefiles/R
- module use /lus/scratch/kristyn/modulefiles
- module load R/R-3.4.0

where the file R-3.4.0 contains

```
#%Module2.0
##
module load java
module load gcc

set R_VERSION R-3.4.0
set R_PATH /lus/scratch/kristyn/R/$R_VERSION/

prepend-path PATH $R_PATH/bin
prepend-path LD_LIBRARY_PATH $R_PATH/lib64/R/library
prepend-path MANPATH $R_PATH/share/man
```

Installing R Packages from CRAN

- **Bring up R on login node and install needed packages**
  - Need external access to download packages
  - In general, most tested, and most reliable compiler for R packages are the GNU compilers (gcc, gfortran)
  - Note, if using a site-installed version, any additional installed packages will be saved to a location in your home directory
    - ~/R/x86_64-suse-linux-gnu-library/3.3

> **R packages we will be using for the tutorial**
> - install.packages(“foreach”)
> - install.packages(“doParallel”)
> - install.packages(“rlecuyer”)
> - install.packages(“randomForest”)
> - install.packages(“SPARQL”)

Installing R packages within Urika-XC

- **Use start_analytics**
  - Specify interactive node to run on login node
  - Better connectivity than from XC compute node
- **For Urika-XC 1.1**
  - R version 3.4.2 (2017-09-28) -- "Short Summer"
  - User packages installed to
    - ~/R/x86_64-redhat-linux-gnu-library/3.4

Using R on Piz Daint using Urika-XC

```
kristyn@daint101:~> module load analytics
kristyn@daint101:~> start_analytics –d
```

Once inside the container, bring up R interactive shell to install packages

```
bash-4.2$ R
```

Inside R interactive shell

```
> install.packages("foreach")
```

```
Installing package into ‘/usr/lib64/R/library’
(as ‘lib’ is unspecified)
Warning in install.packages("foreach") :
  'lib = "/usr/lib64/R/library"' is not writable
Would you like to use a personal library instead?  (y/n) y
Would you like to create a personal library
~/R/x86_64-redhat-linux-gnu-library/3.4
to install packages into?  (y/n) y
```

Running R in Jupyter With The R Kernel

- **Install R packages**

```
> install.packages(c('repr','IRdisplay','evaluate','crayon','pbdZMQ','devtools’, 'uuid', 'digest’))
> devtools::install_github('IRkernel/Irkernel')
```

- **Need to make R kernel visable for Jupyter**
  - Install IRKernel for the current user

```
> IRkernel::installspec()
```

- **Start up Jupyter Notebook**
  - R should now be one of the kernels in the upper right corner in the “New” drop-down menu

Running R in Jupyter using Urika-XC

Since Piz Daint is only accessible via ssh from the front end ela.cscs.ch, we need to create a tunnel through the front end to daint to use Web Uis (Jupyter, also for CGE)

Example:

Create tunnel from laptop to internal daint node through ela.cscs.ch

```
$ ssh -L localhost:8022:daint:22 ela.cscs.ch
```

Then in a second terminal, log directly into the daint node

```
$ ssh -p 8022 -L localhost:15000:localhost:15000 localhost
```

Bring up Urika-XC

```
$ start_analytics -d --login-port 8022 --ui-port 15000
```

Then inside container, start Jupyter Notebook

```
bash-4.2$ jupyter notebook --port 15000
```

Managing R using Anaconda

- **Anaconda R**
  - Quite useful for managing R packages and multiple R environments on XC
  - List of R language packages available for install from conda is located at http://repo.continuum.io/pkgs/r/
  - R Essentials bundle includes about 100 of the most popular packages for R

> **conda create --name myR -c r r-essentials**
> **source activate myR**

- **Also can specify specific versions of R**

> **conda create --name myR_3.2.2 -c r r=3.2.2**

- **When using an older version of R I found it works better to create the conda environment first, activate this, then install the allowing packages, allowing conda to manage the package version dependencies**

> **source activate myR_3.2.2**
> **conda install -c r r-essentails r-xml**

Running R using CCM

- salloc -N 4 --partition=ccm_queue

- # Determine nid allocations
- echo “$SLURM_NODELIST” or env | grep SLURM
  - SLURM_NODELIST=nid0000[4-7]

- # load R module
- module use /lus/scratch/R/modulefiles
- module load R/R-3.4.0

- # Log into head node and propagate environment
- module load ccm
- ccmlogin –V

- # Start up R on head node
- R

Note: CCM may not be available on all XC systems. This is a site-configuration.
Piz Daint does not have CCM running, but is set up so one can use ssh between nodes within a job.

See /scratch/snx3000/kristyn/CUG2018/R/README for additional details.

R using “parallel” package using CCM mode
Setting up a simple parallel socket cluster

“parallel” package

https://stat.ethz.ch/R-manual/R-devel/library/parallel/doc/parallel.pdf

```
>   library(parallel)
>   machineVec <- c(rep("nid00004",4), rep("nid00005",4), rep("nid00006",4), rep("nid00007",4))

>   machineVec
>    [1] "nid00004" "nid00004" "nid00004" "nid00004" "nid00005" "nid00005"
>    [7] "nid00005" "nid00005" "nid00006" "nid00006" "nid00006" "nid00006"
>   [13] "nid00007" "nid00007" "nid00007" "nid00007"
>   cl <- makeCluster(machineVec)
>   cl
>   socket cluster with 16 nodes on hosts 'nid00004', 'nid00005', 'nid00006', 'nid00007’

>   help(makeCluster)

>   stopCluster(cl)
```

Running parallel R package on Piz Daint

- Note: CCM may not be available on all XC systems. This is a site-configuration.
- Piz Daint does not have CCM running, but is set up so one can use ssh between nodes within a job.

- Allocate nodes
  - salloc -N 4 -C mc

  - # Determine nid allocations
  - echo "$SLURM_NODELIST" or env | grep SLURM
    - SLURM_NODELIST=nid0000[4-7]

  - # load R module
  - module load cray-R

  - # Start up R on login node
  - R

  - # Set up socket-based cluster (follow same instructions as previous slide)

See /scratch/snx3000/kristyn/CUG2018/R/README for additional details.

Enabling ssh between nodes using start_analytics

- **On SLURM-based systems**
  - salloc -N 4 -C mc --image=custom:analytics-1.01.0000.201712122205_0082-latest start_analytics –ssh
- **On interactive node**
  - > # Determine nid allocations
  - > echo “$SLURM_NODELIST” or env | grep SLURM
  - > SLURM_NODELIST=nid0000[4-7]

  - > # Start up R
  - > R

Simple Parallel Socket Cluster

- **Basic functionality**
  - Runs 'Rscript' on the specified host(s) to set up a worker process which listens on a socket for expressions to evaluate, and returns the results (as serialized objects).
- **Commonly used R packages which then build upon the “parallel” package**
  - “foreach” package
    - Provides looping construct
  - “doParallel” package
    - Provides mechanism needed to execute **foreach** loops in parallel
  - https://cran.r-project.org/web/packages/doParallel/vignettes/gettingstartedParallel.pdf

Example datasets

```
>   # Base install of R already includes several datasets
>   # To look at the datasets available in loaded packages
>   data()

>   # load the iris dataset
>   data(iris)
>   head(iris)

>   # Many R packages also contain additional datasets
>   install.package(‘rattle’)
>   data(wine, package='rattle')

>   # Also can import data directly
>   # Here read.table reads a file in table format and creates a dataframe from it
>   url <- 'https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-white.csv'
>   whitewine <- read.table(url,header=TRUE,sep=“;”)
>   head(whitewine)
```

Example Code: using foreach and doParallel

```
>  library(parallel)
>  library(foreach)
>  library(doParallel)
>  machineVec <- c(rep("nid00004",4), rep("nid00005",4), rep("nid00006",4), rep("nid00007",4))

>  cl <- makeCluster(machineVec)
>  # To use the "foreach", we need to register the cluster with
>  registerDoParallel(cl)
>  getDoParWorkers()

>  # sequential execution
>  system.time(foreach(i=1:100000) %do% sum(tanh(1:i)))
>  # parallel execution
>  system.time(foreach(i=1:10000) %dopar% sum(tanh(1:i)))

>  mcoptions <- list(preschedule=FALSE, set.seed=FALSE, cores=4)
>  system.time(foreach(i=1:100000,.options.multicore=mcoptions) %dopar% sum(tanh(1:i)))
```

Example Code: randomForest

```
> # Parallel execution of randomForest
> x <- matrix(runif(500), 100)
> y <- gl(2, 50)
>
> library(randomForest)
>
> rf <- foreach(ntree=rep(25000, 6), .combine=combine,
  .multicombine=TRUE, .packages='randomForest')
  %dopar% { randomForest(x, y, ntree=ntree)}
```

R profiling

- Standard approach – use Rprof
  - Profile R code is to use the Rprof function to profile and the summaryRprof function to summarize the result

```
> help(Rprof)
```

```
> Rprof(tmp <- tempfile())
> example(glm)
> Rprof()
> summaryRprof(tmp)
```

Programming with Big Data in R (pbdR)

- Set of highly scalable R packages for distributed computing in data science
  - http://r-pbd.org/
- George Ostrouchov, Wei-Chen Chen, Drew Schmidt, Pragneshkumar Patel
- Winner of the Oak Ridge National Laboratory 2016 Significant Event Award for "Harnessing HPC Capability at OLCF with the R Language for Deep Data Science

Installing pdbMPI package

- **If not already installed, install rlecuyer package**
  - wget https://cran.r-project.org/src/contrib/rlecuyer_0.3-4.tar.gz
  - R CMD INSTALL --no-test-load rlecuyer_0.3-4.tar.gz
- **Install pdbMPI package**
  - wget https://cran.r-project.org/src/contrib/pbdMPI_0.3-3.tar.gz
  - R CMD INSTALL pbdMPI_0.3-3.tar.gz --configure-args="--with-mpi=/opt/cray/pe/mpt/default/gni/mpich-gnu/51/ --disable-opa --with-mpi-type=MPICH2" --no-test-load
- **https://cran.r-project.org/web/packages/pbdMPI/pbdMPI.pdf**

pdbMPI: run “Hello World”

Create file mpi_hello_world.r

```
 # load the package
 suppressMessages(library(pbdMPI, quietly = TRUE))

 # initialize the MPI communicators
 init()

 # Hello world
 message <- paste("Hello from rank", comm.rank(), "of", comm.size())
comm.print(message, all.rank=TRUE, quiet=TRUE)

# shut down the communicators and exit
finalize()
```

```
> srun -N 4 Rscript mpi_hello_world.r
```

pbdMPi – beyond “Hello World”

- HPSC Cookbook – Wei-Chen Chen
- https://snoweye.github.io/hpsc/cookbook.html

- In addition there are several tutorials available with source code available for download
- Tutorials 1 and 2 both use the Iris dataset already available with base R install

Parallel (SPMD) pi Example (from HPSC)

```
# File name: ex_pi_spmd.r
# Run: srun -N 2 Rscript --vanilla ex_pi_spmd.r

### Load pbdMPI and initial the communicator.
library(pbdMPI, quiet = TRUE)
init()
.comm.size <- comm.size()
.comm.rank <- comm.rank()

### Compute pi.
n <- 1000
totalcpu <- .comm.size
id <- .comm.rank + 1
mypi <- 4*sum(1/(1+((seq(id,n,totalcpu)-.5)/n)^2))/n    # The example from Rmpi.
mypi <- reduce(mypi, op = "sum")

### Output from RANK 0 since mpi.reduce(...) will dump only to 0 by default.
comm.print(mypi)
finalize()
```

Legal Disclaimer

Information in this document is provided in connection with Cray Inc. products. No license, express or implied, to any intellectual property rights is granted by this document.

Cray Inc. may make changes to specifications and product descriptions at any time, without notice.

All products, dates and figures specified are preliminary based on current expectations, and are subject to change without notice.

Cray hardware and software products may contain design defects or errors known as errata, which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Cray uses codenames internally to identify products that are in development and not yet publicly announced for release. Customers and other third parties are not authorized by Cray Inc. to use codenames in advertising, promotion or marketing and any use of Cray Inc. internal codenames is at the sole risk of the user.

Performance tests and ratings are measured using specific systems and/or components and reflect the approximate performance of Cray Inc. products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance.

The following are trademarks of Cray Inc. and are registered in the United States and other countries: CRAY and design, SONEXION, URIKA and YARCDATA. The following are trademarks of Cray Inc.: CHAPEL, CLUSTER CONNECT, CLUSTERSTOR, CRAYDOC, CRAYPAT, CRAYPORT, DATAWARP, ECOPHLEX, LIBSCI, NODEKARE, REVEAL. The following system family marks, and associated model number marks, are trademarks of Cray Inc.: CS, CX, XC, XE, XK, XMT and XT. The registered trademark LINUX is used pursuant to a sublicense from LMI, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis. Other trademarks used on this website are the property of their respective owners.

Q&A

Kristi Maschhoff

kristyn@cray.com

Spark on Cray Systems
CUG 2018

Michael Ringenburg, Cray Inc
mikeri@cray.com

Agenda

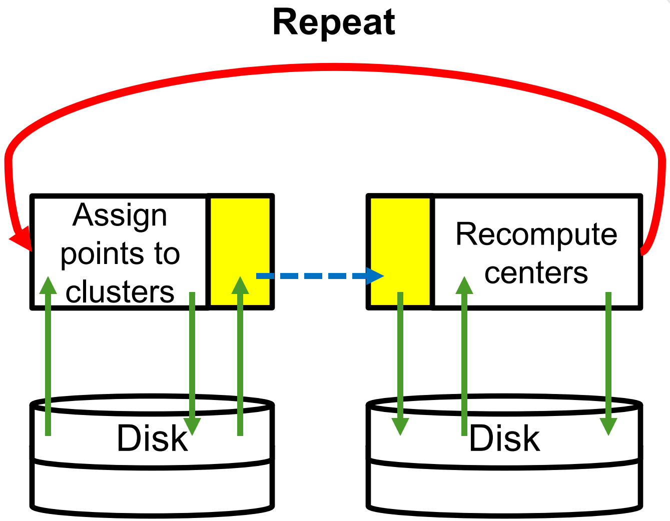
- **Introduction to Spark**
  - History and Background
  - Computation and Communication Model
- **Spark on the XC40**
  - Installation and Configuration
  - Local storage
- **Alchemist: MPI and Spark**
- **BigDL: Deep Learning in Spark**

In the beginning, there was Hadoop MapReduce…

- **MapReduce: simplified parallel programming model**
  - All computations broken into two parts
    - **Embarassingly parallel map phase**: apply single operation to every key,value-pair, produce new set of key,value-pairs
    - **Combining reduce phase**: Group all values with identical key, performing combining operation to get final value for key
  - Can perform multiple iterations for computations that require
  - I/O intensive
    - Map writes to local storage. Data shuffled to reducer's local storage, reduce reads.
    - Additional I/O between iterations in multi-iteration algorithms (map reads from HDFS, reduce writes to HDFS)
  - Effective model for many data analytics tasks
- **HDFS distributed file system (locality aware – move compute to data)**
- **YARN cluster resource manager**

Example: K-Means Clustering with MapReduce

- **Initially: Write out random cluster centers**
- **Map:**
  - Read in cluster centers
  - For each data point, compute nearest cluster center and write <key: nearest cluster, value: data point>
- **Reduce:**
  - For each cluster center (key) compute average of datapoints
  - Write out this value as new cluster center
- **Repeat until convergence (clusters don’t change)**



MapReduce Problems

- **Gated on IO bandwidth, possibly interconnect as well**
  - Must write and read between map and reduce phases
  - Multiple iterations must write results in next time (e.g., new cluster centers)
- **No ability to persist reused data**
- **Must re-factor all computations as map then reduce (rinse and repeat?)**

What is Spark?

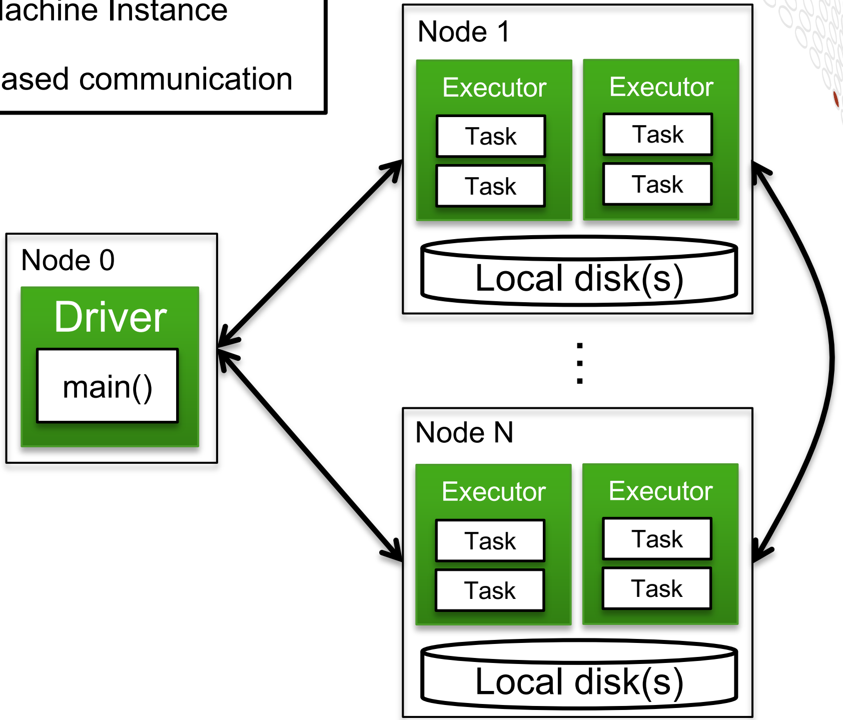
- **Newer (2014) analytics framework**
  - Originally from Berkeley AMPLab/BDAS stack, now Apache project
  - Native APIs in Scala. Java, Python, and R APIs available as well.
  - Many view as successor to Hadoop MapReduce. Compatible with much of Hadoop Ecosystem.
- **Aims to address some shortcomings of Hadoop MapReduce**
  - More programming flexibility – not constrained to one map, one reduce, write, repeat.
  - Many operations can be pipelined into a single in-memory task
  - Can "persist" intermediate data rather than regenerating every stage

Spark Execution Model

- **Master-slave parallelism**
- **Driver (master)**
  - Executes main
  - Distributes work to executors
- **Executors (slaves)**
  - Lazily execute tasks (local operations on partitions of the RDD)
  - Rely on local disks for spilling data that's too large, and storing shuffle data
- **Resilient Distributed Dataset (RDD)**
  - Spark's original data abstraction
  - Partitioned amongst executors
  - Fault-tolerant via lineage
  - Dataframes/Datasets extend this abstraction

🟩 = Java Virtual Machine Instance

↔ = TCP Socket-based communication



RDDs (and DataFrames/DataSets)

- **RDDs are original data abstraction of Spark**
  - DataFrames add structure to RDDs: named columns
  - DataSets add strong typing to columns of DataFrames (Scala and Java only)
  - Both build on the basic idea of RDDs
    - DataFrames were originally called SchemaRDDs
- **RDD data structure contains a description of the data, partitioning, and computation, but not the actual data … why?**
  - Lazy evaluation

Lazy Evaluation and DAGs

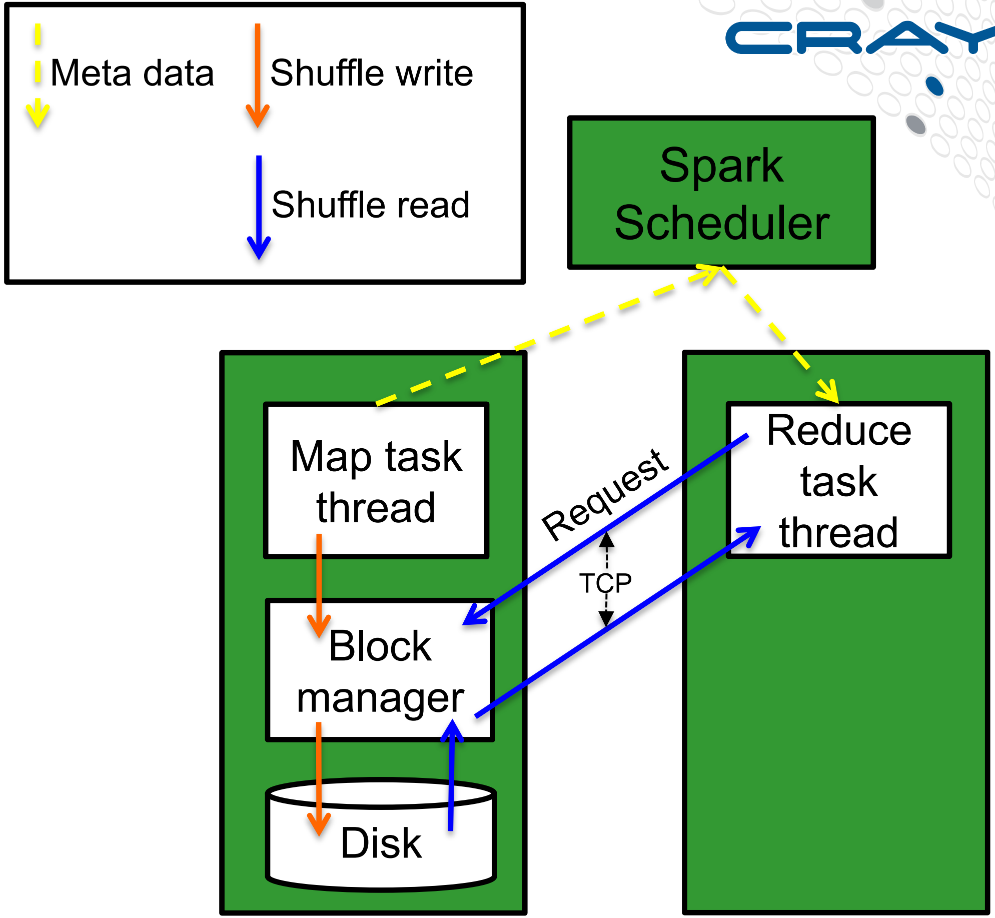
- **Spark is lazily evaluated**
  - Spark operations are only executed when and if needed
  - Needed operations: produce a result for driver, or produce a parent of needed operation (recursive)
- **Spark DAG (Directed Acyclic Graph)**
  - Calls to transformation APIs (operations that produce a new RDD/DataFrame from one or more parents) just add a new node to the DAG, indicating data dependencies (parents) and transformation operation
  - Action APIs (operations that return data) trigger execution of necessary DAG elements
- **Example shortly…**

Pipelining in Spark

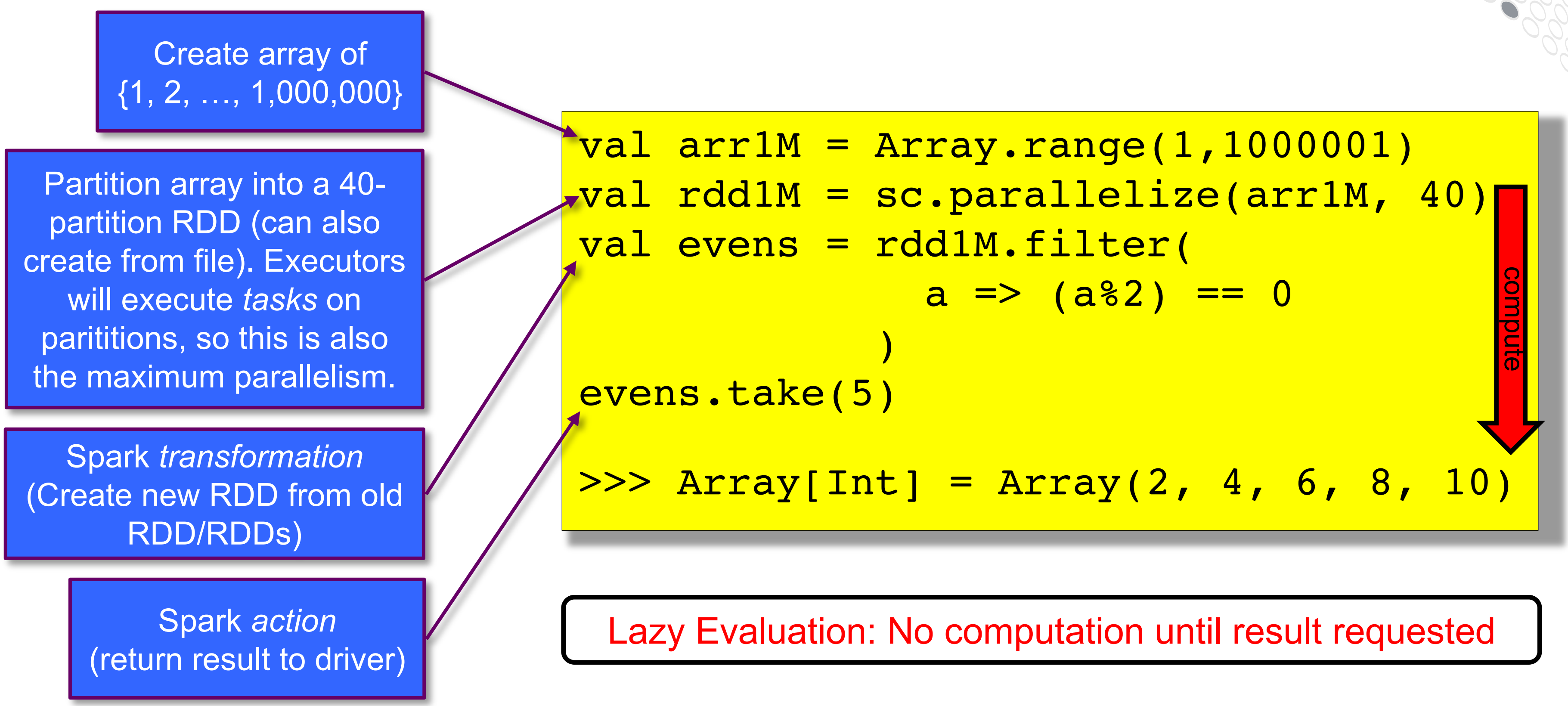
- **If an RDD partition's dependencies are on a single other RDD partition (or on *co-partitioned* data), the operations can be *pipelined* into a single *task***
- **Spark stage: Execution of same task on all partitions**
  - Every stage ends with a *shuffle* (all-to-all communication), an output, or returning data back to the driver.
  - Global barrier between stages. All senders complete shuffle write before receivers request data (shuffle read)

Spark Communication Model (Shuffles)

- **All data exchanges between executors implemented via *shuffle***
  - Senders (“mappers”) send data to block managers; block managers write to disks, tell scheduler *how much* destined for each reducer
  - Barrier until all mappers complete shuffle writes
  - Receivers (“reducers”) request data from block managers *that have data for them*; block managers read and send



Spark Programming Model: Example



Create array of {1, 2, …, 1,000,000}

Partition array into a 40-partition RDD (can also create from file). Executors will execute *tasks* on parititions, so this is also the maximum parallelism.

Spark *transformation* (Create new RDD from old RDD/RDDs)

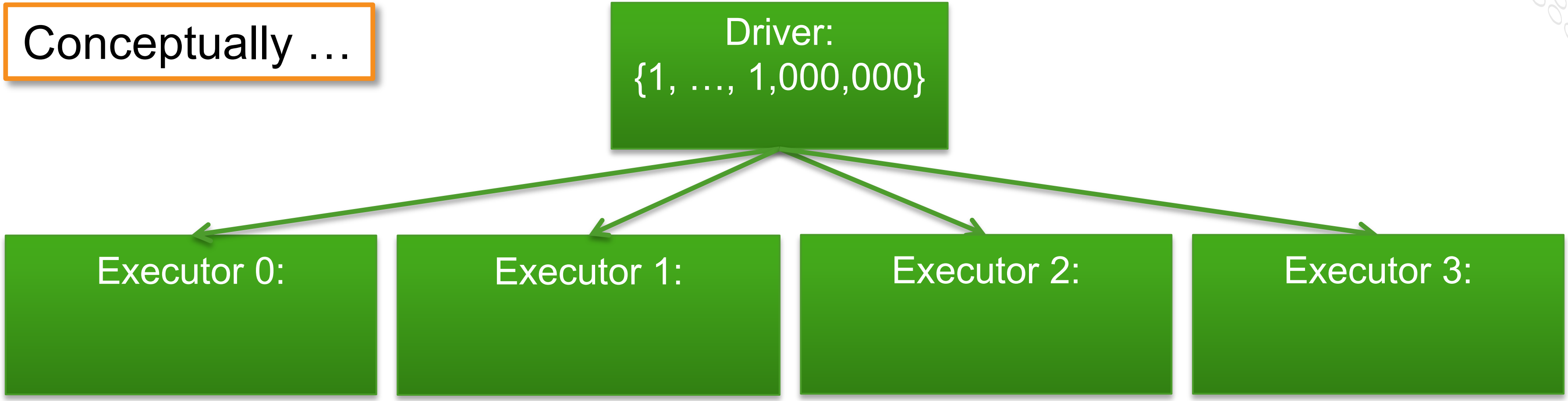
Spark *action* (return result to driver)

```
val arr1M = Array.range(1,1000001)
val rdd1M = sc.parallelize(arr1M, 40)
val evens = rdd1M.filter(
              a => (a%2) == 0
            )
evens.take(5)

>>> Array[Int] = Array(2, 4, 6, 8, 10)
```

Lazy Evaluation: No computation until result requested

Example: Line-by-line

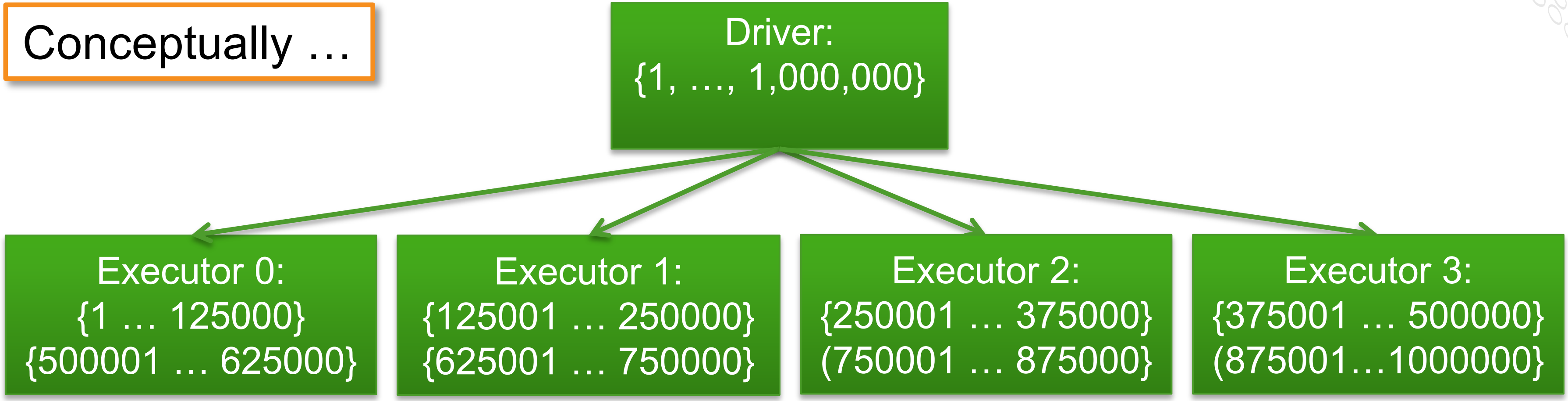
Conceptually …



```
val arr1M = Array.range(1,1000001)
```

Example: Line-by-line

Conceptually …



Driver:
{1, …, 1,000,000}

Executor 0:
{1 … 125000}
{500001 … 625000}

Executor 1:
{125001 … 250000}
{625001 … 750000}

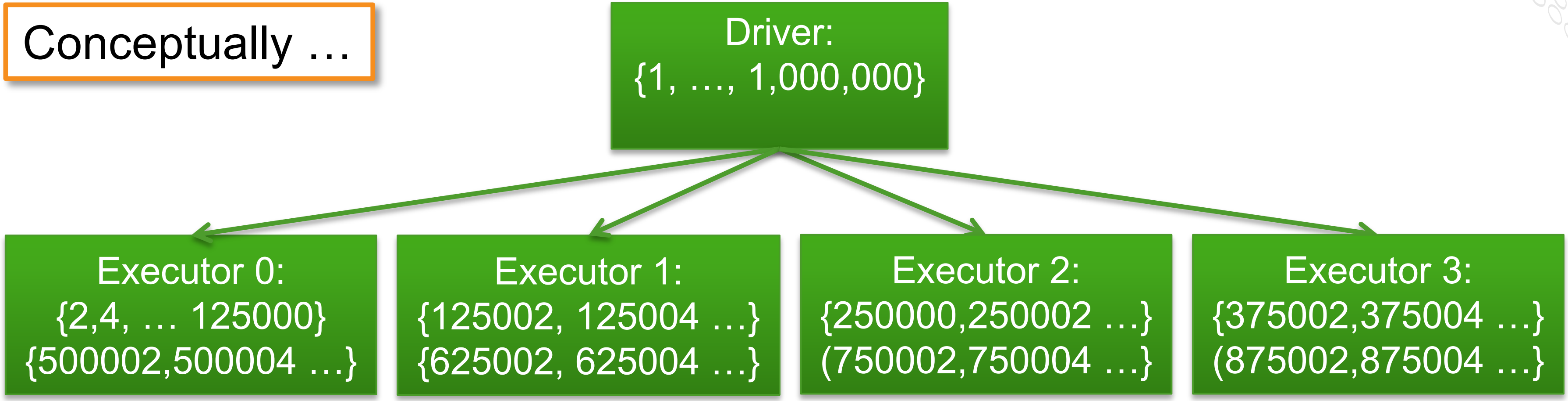
Executor 2:
{250001 … 375000}
(750001 … 875000}

Executor 3:
{375001 … 500000}
(875001…1000000}

```
val rdd1M = sc.parallelize(arr1M, 8)
```

Example: Line-by-line

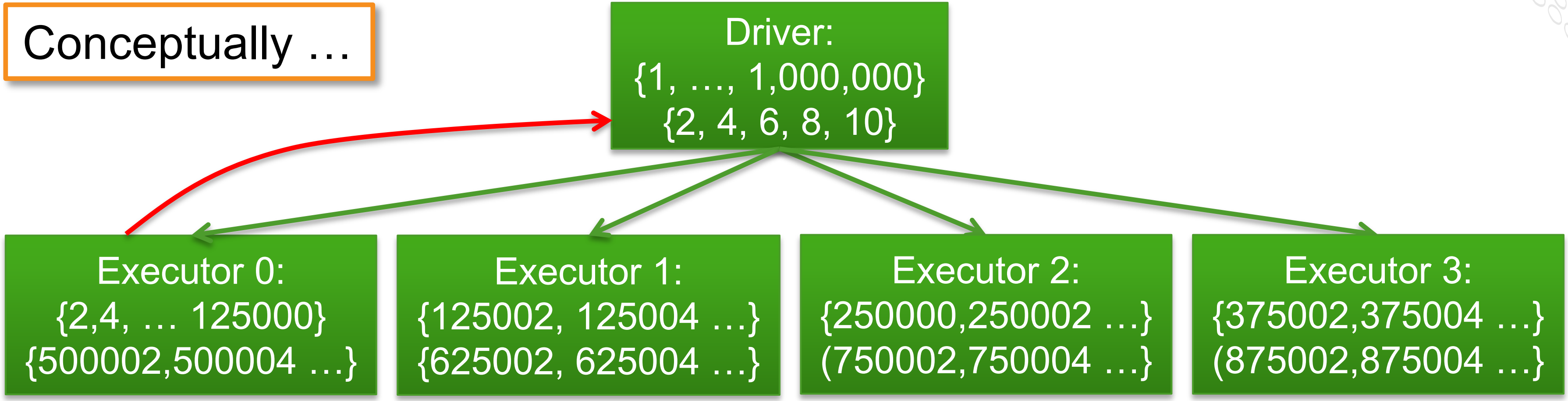
Conceptually …



```
val evens = rdd1M.filter(a => a%2==0)
```

Example: Line-by-line

Conceptually …



Driver:
{1, …, 1,000,000}
{2, 4, 6, 8, 10}

Executor 0:
{2,4, … 125000}
{500002,500004 …}

Executor 1:
{125002, 125004 …}
{625002, 625004 …}

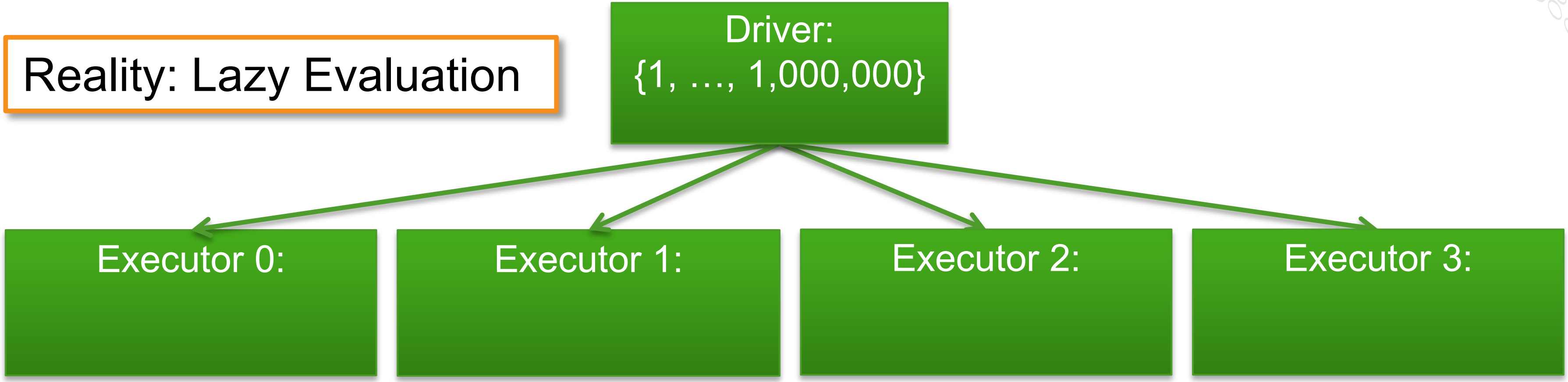
Executor 2:
{250000,250002 …}
(750002,750004 …}

Executor 3:
{375002,375004 …}
(875002,875004 …}

```
evens.take(5)
```

Example: Line-by-line

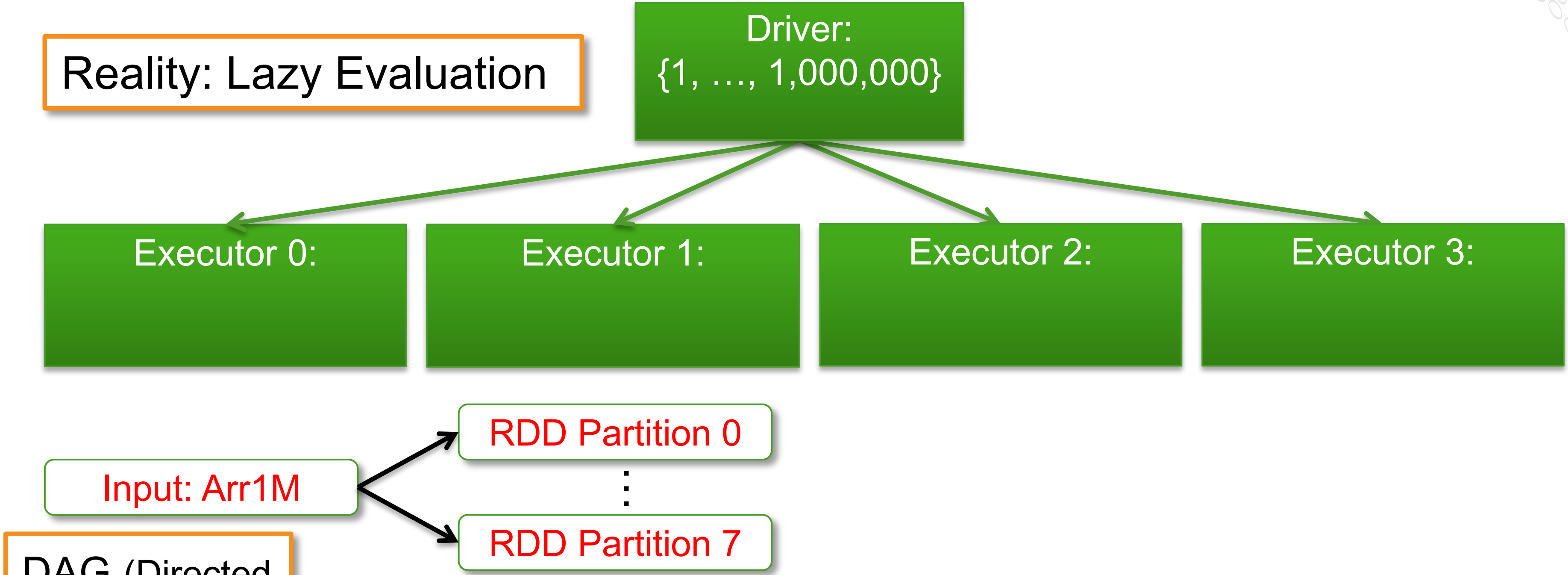
Reality: Lazy Evaluation



```
val arr1M = Array.range(1,1000001)
```

Example: Line-by-line

Reality: Lazy Evaluation

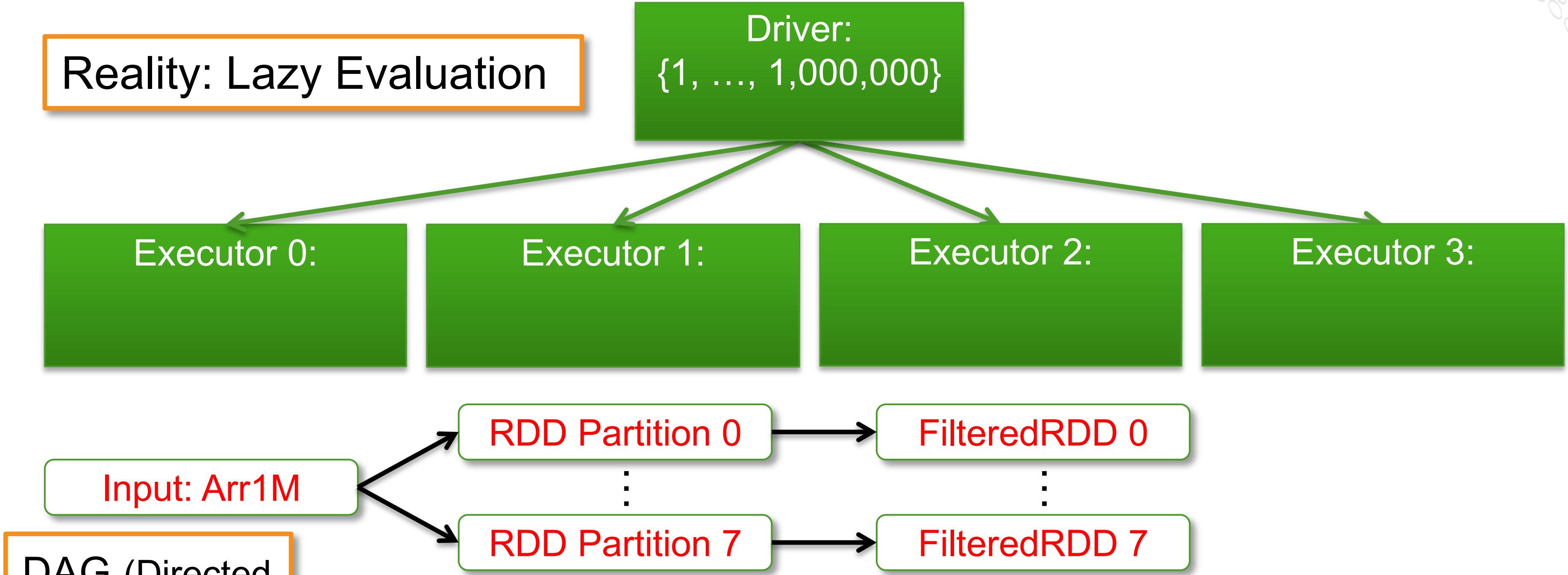


DAG (Directed Acyclic Graph) schedule

```
val rdd1M = sc.parallelize(arr1M, 8)
```

Example: Line-by-line

Reality: Lazy Evaluation

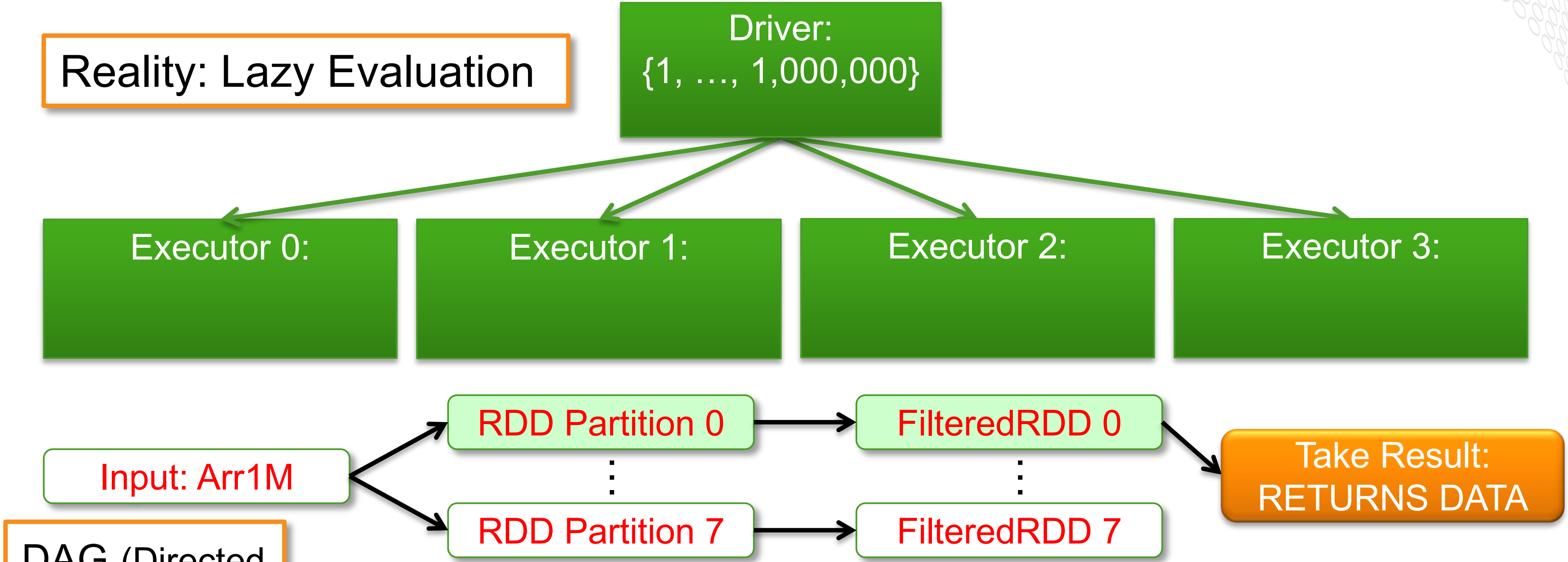


DAG (Directed Acyclic Graph) schedule

```
val evens = rdd1M.filter(a => a%2==0)
```

Example: Line-by-line

Reality: Lazy Evaluation



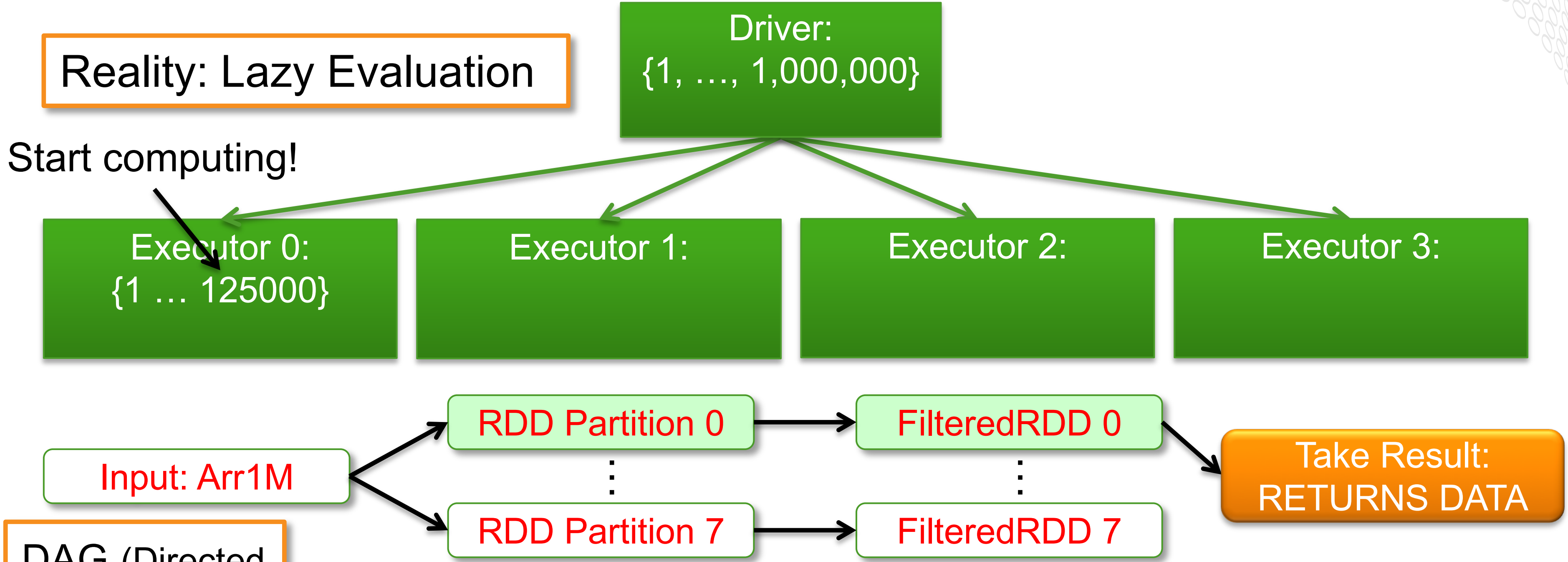
DAG (Directed Acyclic Graph) schedule

```
evens.take(5)
```

Example: Line-by-line

Reality: Lazy Evaluation

Start computing!

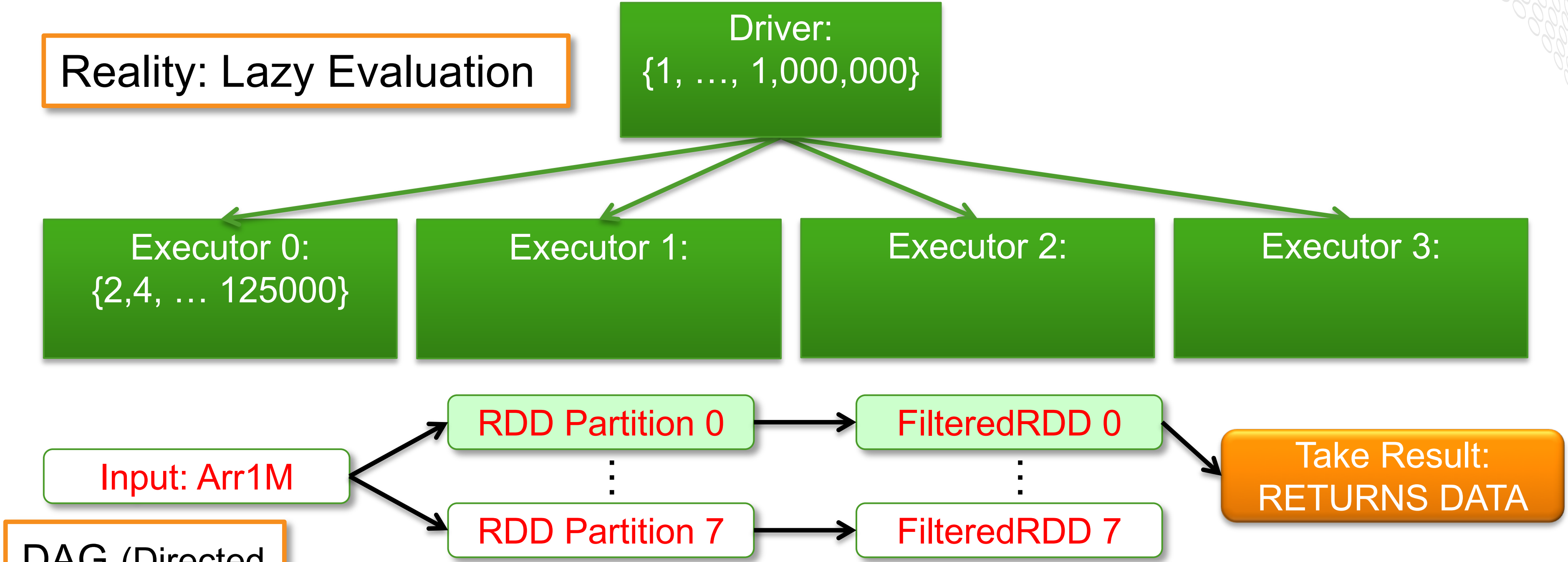


DAG (Directed Acyclic Graph) schedule

```
evens.take(5)
```

Example: Line-by-line

Reality: Lazy Evaluation

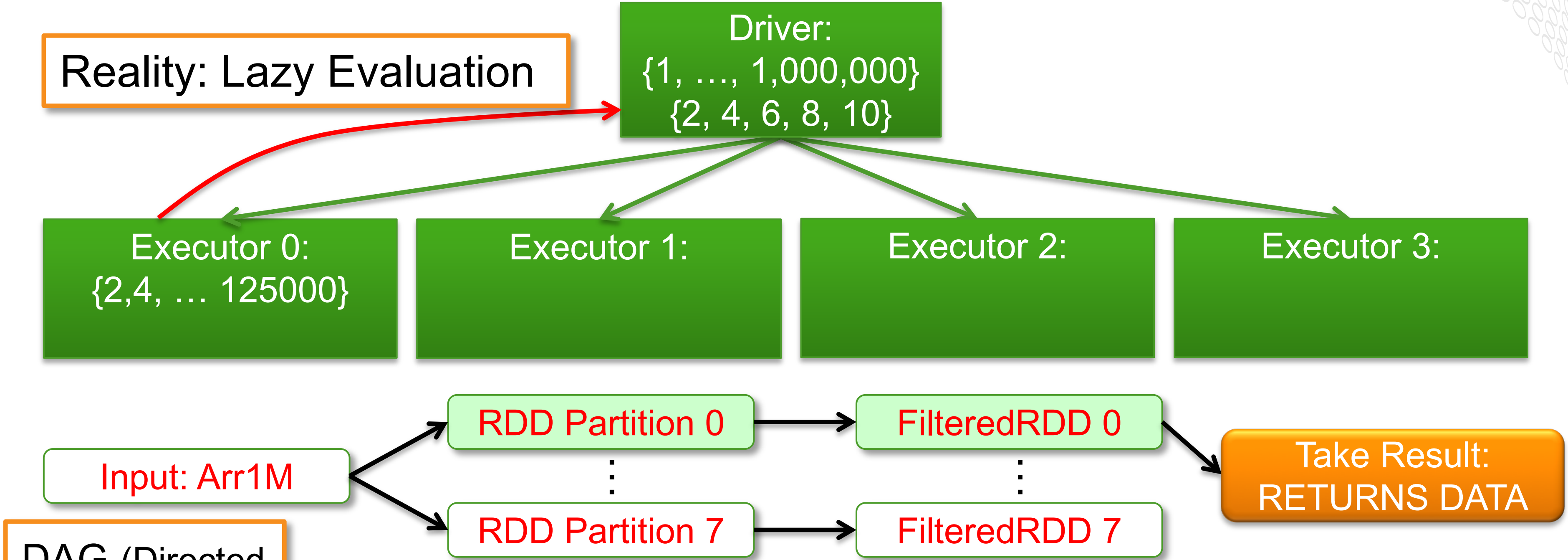


DAG (Directed Acyclic Graph) schedule

```
evens.take(5)
```

Example: Line-by-line

Reality: Lazy Evaluation



DAG (Directed Acyclic Graph) schedule

```
evens.take(5)
```

Wait a second …

- **How did Spark know that take() would only require data from one partition?**
  - What if filter() left fewer than 5 elements in the first partition?

Wait a second …

- **How did Spark know that take() would only require data from one partition?**
  - What if filter() left fewer than 5 elements in the first partition?
- **Answer … It didn't.**
  - Take is typically used to fetch a small initial piece of the data
  - Spark guesses that it will all be available in the first partition
  - If not, tries the first four partitions …
  - Then the first 16 …
  - Etc…

Modified example

Count returns the total size of an RDD

Reduce performs a reduction over the dataset, combining elements with the argument function.

```
val arr1M = Array.range(1,1000001)
val rdd1M = sc.parallelize(arr1M, 8)
val evens = rdd1M.filter(a => (a%2) == 0)
val firstFiveEvens = evens.take(5)
// How many evens?
val totalEvens = evens.count()
// Sum of evens
val evenSum = evens.reduce((a,b) => a+b)
```

- Imagine we want to perform a number of actions on (i.e., return different data about) our filtered RDD.
- For each action, Spark computes all the DAG steps…

Modified example

Count returns the total size of an RDD

Reduce performs a reduction over the dataset, combining elements with the argument function.

```
val arr1M = Array.range(1,1000001)
val rdd1M = sc.parallelize(arr1M, 8)
val evens = rdd1M.filter(a => (a%2) == 0)
val firstFiveEvens = evens.take(5)
// How many evens?
val totalEvens = evens.count()
// Sum of evens
val evenSum = evens.reduce((a,b) => a+b)
```

- **Problem: This means recomputing the filtered "evens" RDD three times – inefficient.**

Modified example

Persist tells Spark to keep the data in memory even after it is done with the action. Allows future actions to reuse without recomputing. Cache is synonym for default storage level (memory). Can also persist on disk, etc.

```
val arr1M = Array.range(1,1000001)
val rdd1M = sc.parallelize(arr1M, 8)
val evens = rdd1M.filter(a => (a%2) == 0)
evens.persist()  // or cache()
val firstFiveEvens = evens.take(5)
// How many evens?
val totalEvens = evens.count()
// Sum of evens
val evenSum = evens.reduce((a,b) => a+b)
```

- **Problem: This means recomputing the filtered "evens" RDD three times – inefficient.**
- **Solution: Persist the RDD!***

*Relies on immutability of val

Multi-stage Spark Example: Word Count

Load file

flatMap maps one value to (possibly) many, instead of one-to-one like map

groupByKey combines all key-value pairs with the same key (k, v1), …, (k,vn) into a single key-value pair (k, (v1, …, vn)).

Collect returns all elements to the driver

More efficient: replace group and sum with reduceByKey

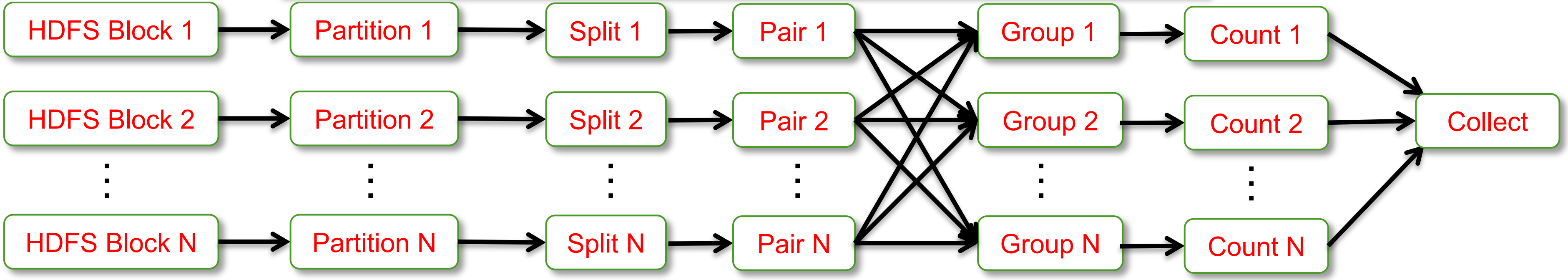
```
val lines = sc.textFile("mytext")
val words = lines.flatMap (
              line => line.split(" ")
            )
val wordKV = words.map(s => (s, 1))
val groupedWords = wordKV.groupByKey()
val wordCounts = groupedWords.map(
                      t => (t._1, t._2.sum)
                 )
val counts = wordCounts.collect()
```

- Let's like at a simple example: computing the number of times each word occurs
  - Load a text file
  - Split it into words
  - Group same words together (all-to-all communication)
  - Count each word

The Spark DAG

```
val lines = sc.textFile("mytext")
val words = lines.flatMap (
              line => line.split(" ")
            )
val wordKV = words.map(s => (s, 1))
val groupedWords = wordKV.groupByKey()
val wordCounts = groupedWords.map(
                   t => (t._1, t._2.sum)
                 )
val counts = wordCounts.collect()
```

Execute!



HDFS Block 1

HDFS Block 2

⋮

HDFS Block N

Partition 1

Partition 2

⋮

Partition N

Split 1

Split 2

⋮

Split N

Pair 1

Pair 2

⋮

Pair N

Group 1

Group 2

⋮

Group N

Count 1

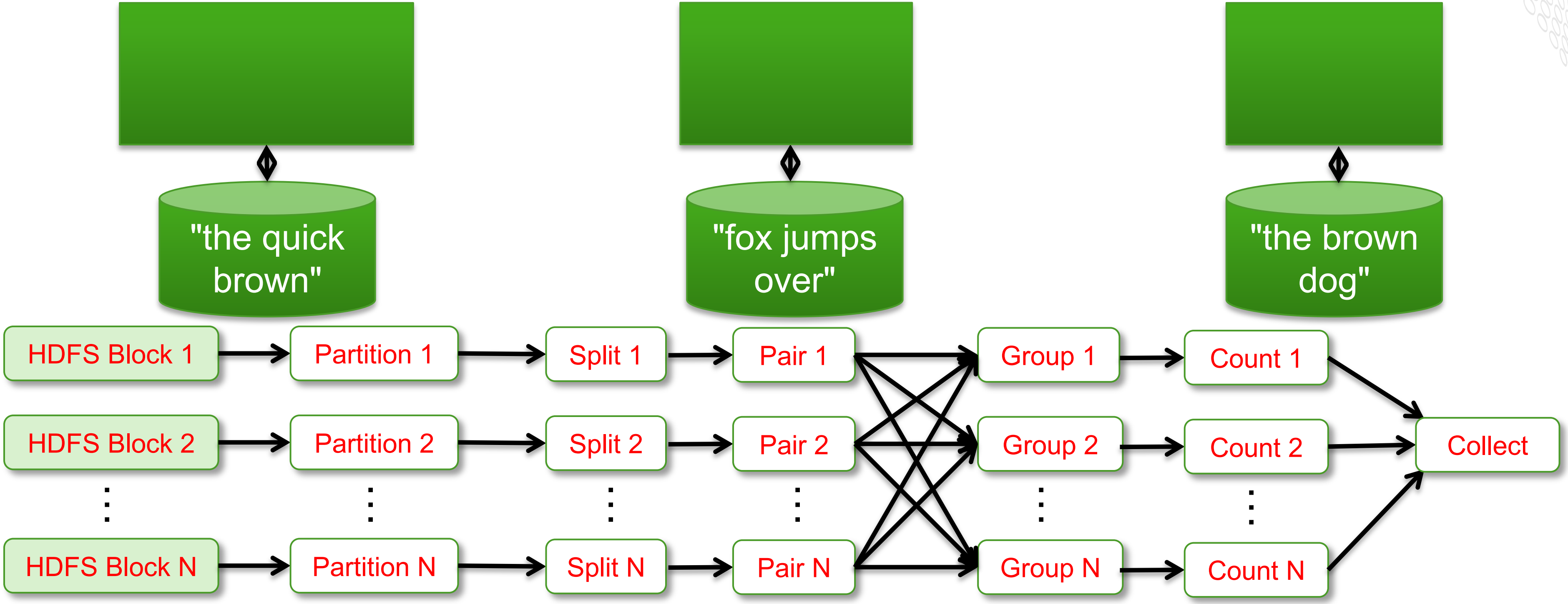
Count 2

⋮

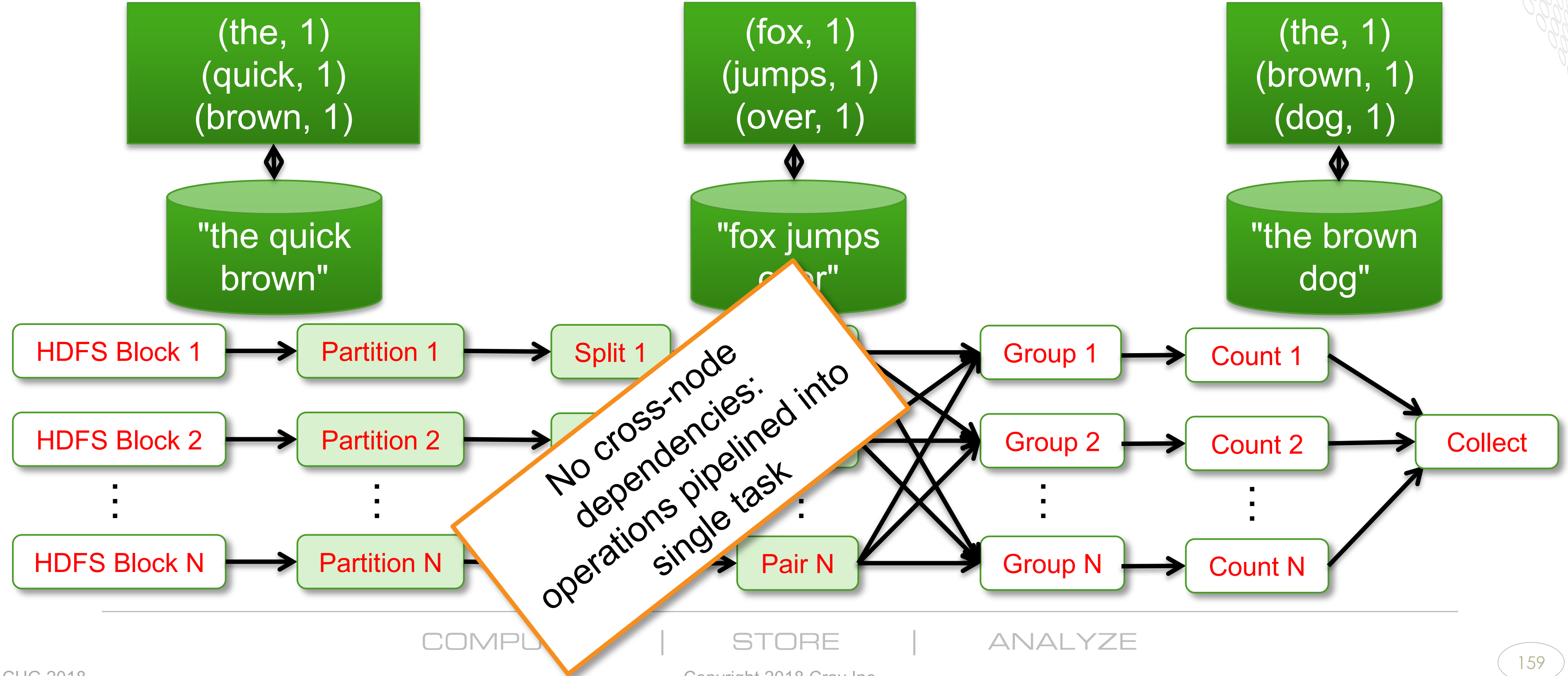
Count N

Collect

Execution

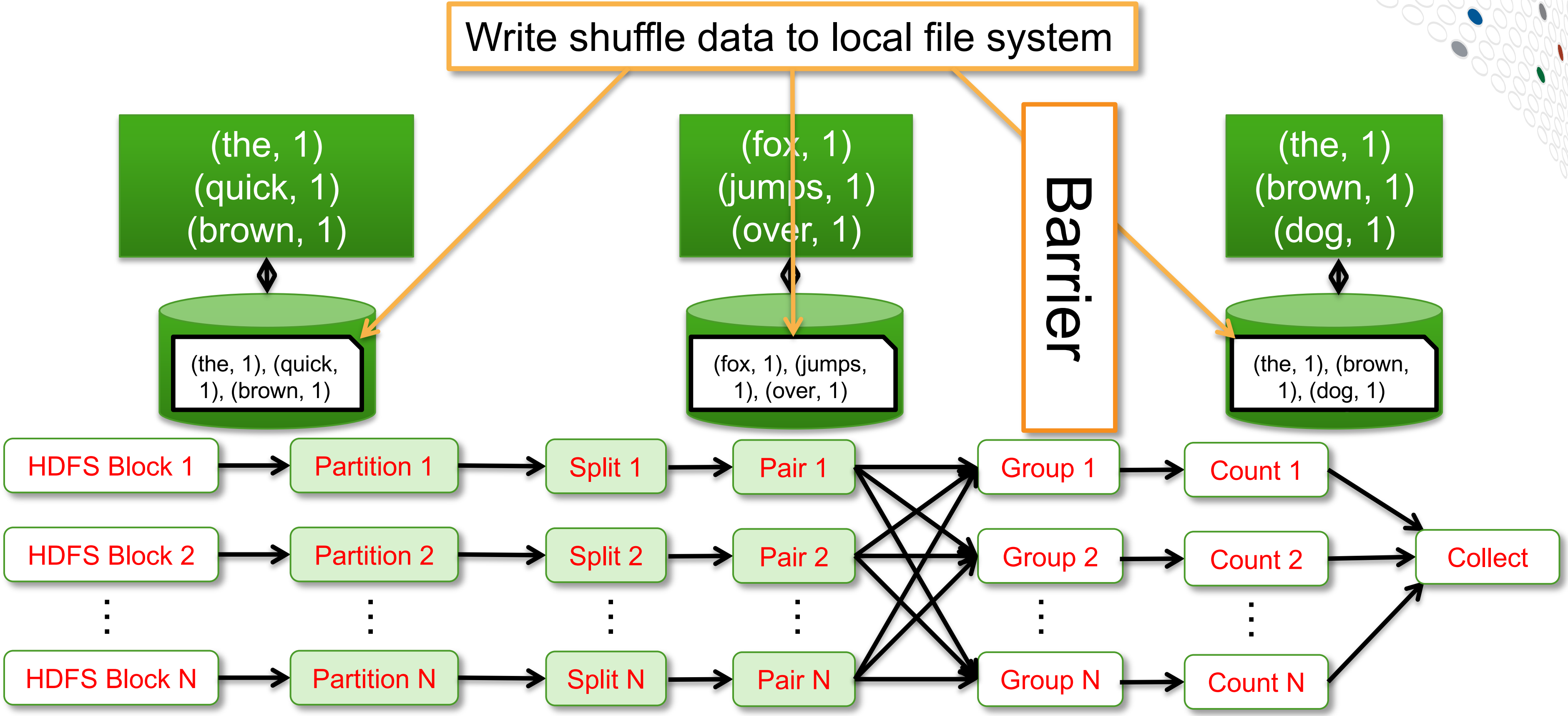

"the quick brown"
"fox jumps over"
"the brown dog"
HDFS Block 1
HDFS Block 2
HDFS Block N
Partition 1
Partition 2
Partition N
Split 1
Split 2
Split N
Pair 1
Pair 2
Pair N
Group 1
Group 2
Group N
Count 1
Count 2
Count N
Collect

Execution



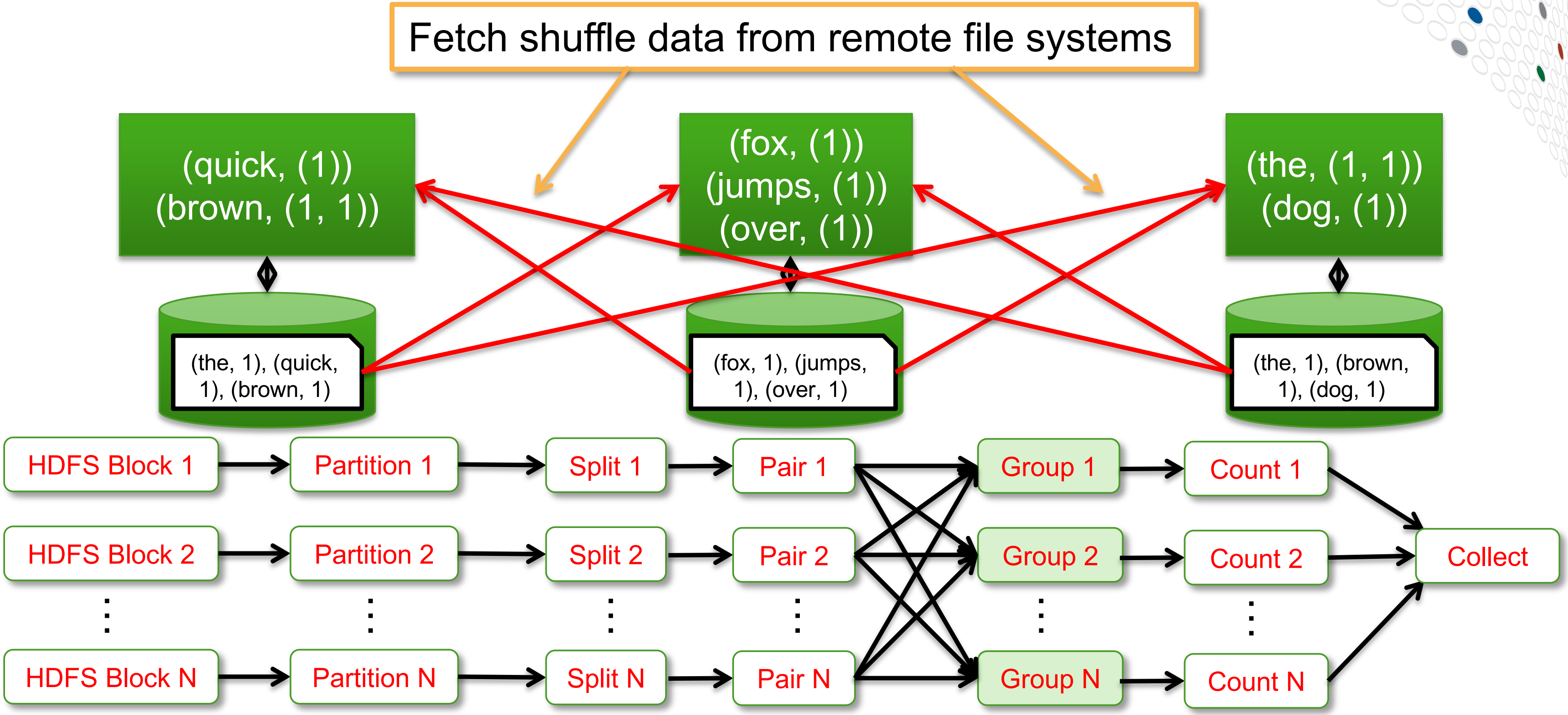
No cross-node dependencies: operations pipelined into single task

Execution

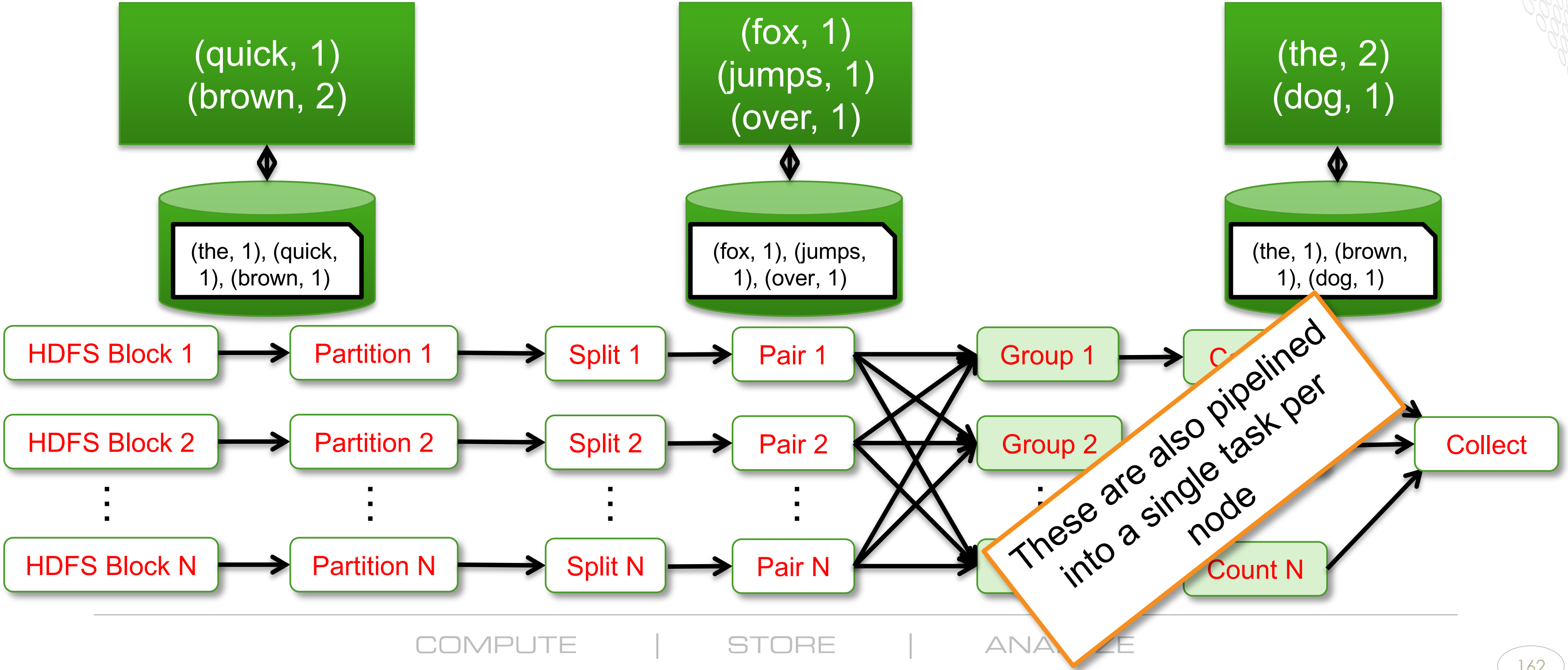


Write shuffle data to local file system
(the, 1)
(quick, 1)
(brown, 1)
(the, 1), (quick, 1), (brown, 1)
(fox, 1)
(jumps, 1)
(over, 1)
(fox, 1), (jumps, 1), (over, 1)
Barrier
(the, 1)
(brown, 1)
(dog, 1)
(the, 1), (brown, 1), (dog, 1)
HDFS Block 1
HDFS Block 2
HDFS Block N
Partition 1
Partition 2
Partition N
Split 1
Split 2
Split N
Pair 1
Pair 2
Pair N
Group 1
Group 2
Group N
Count 1
Count 2
Count N
Collect

Execution

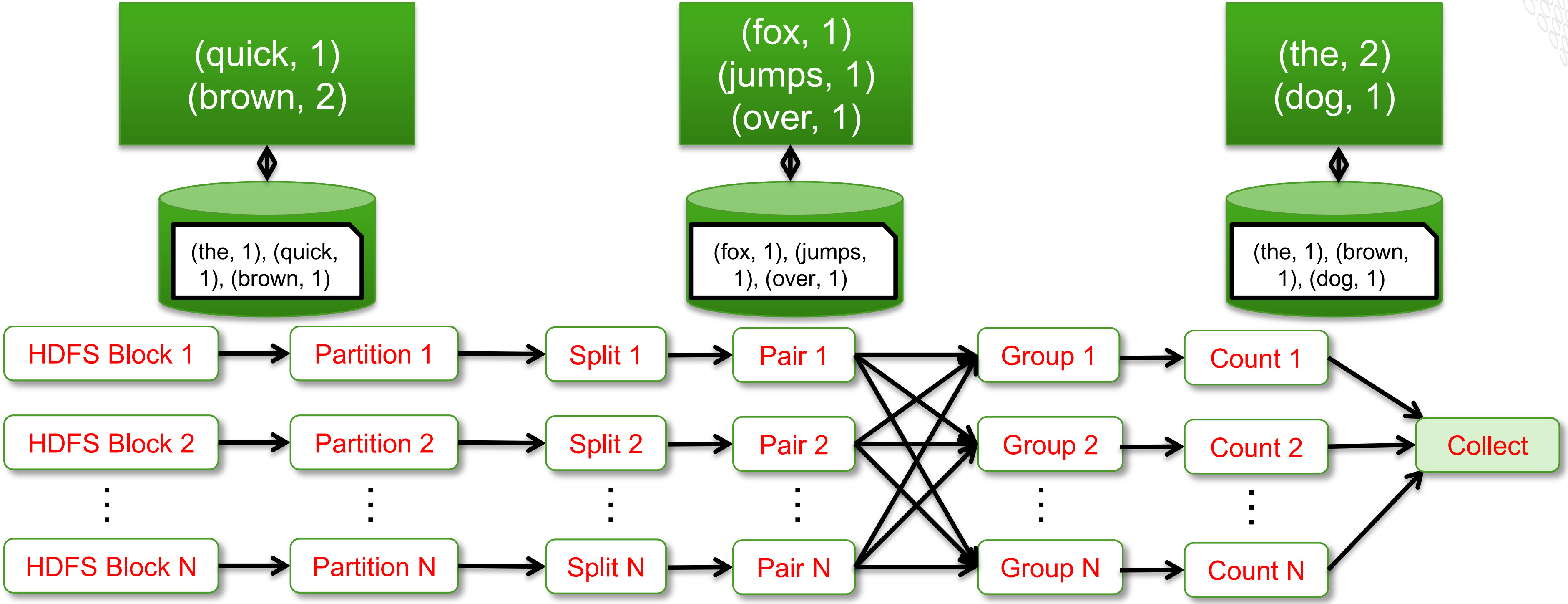

Fetch shuffle data from remote file systems
(quick, (1))
(brown, (1, 1))
(fox, (1))
(jumps, (1))
(over, (1))
(the, (1, 1))
(dog, (1))
(the, 1), (quick, 1), (brown, 1)
(fox, 1), (jumps, 1), (over, 1)
(the, 1), (brown, 1), (dog, 1)
HDFS Block 1
HDFS Block 2
HDFS Block N
Partition 1
Partition 2
Partition N
Split 1
Split 2
Split N
Pair 1
Pair 2
Pair N
Group 1
Group 2
Group N
Count 1
Count 2
Count N
Collect

Execution

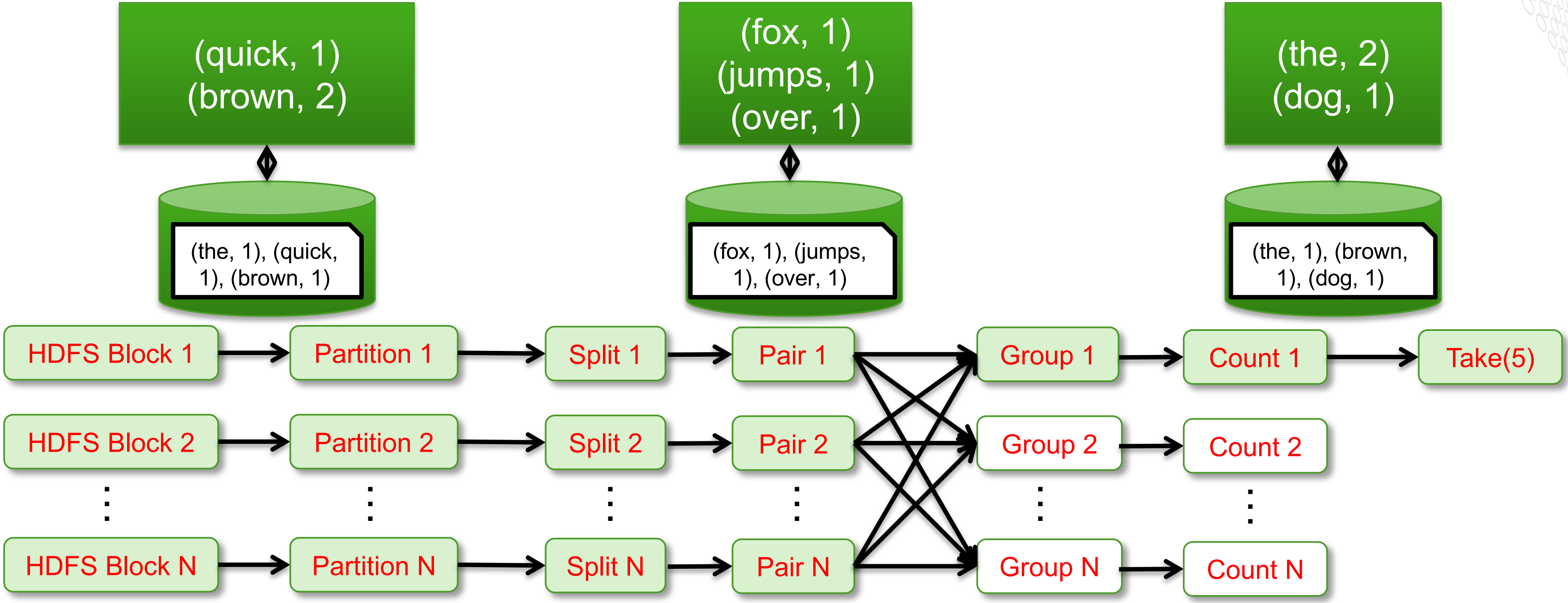


These are also pipelined into a single task per node

Execution


(quick, 1)
(brown, 2)
(fox, 1)
(jumps, 1)
(over, 1)
(the, 2)
(dog, 1)
(the, 1), (quick, 1), (brown, 1)
(fox, 1), (jumps, 1), (over, 1)
(the, 1), (brown, 1), (dog, 1)
HDFS Block 1
HDFS Block 2
HDFS Block N
Partition 1
Partition 2
Partition N
Split 1
Split 2
Split N
Pair 1
Pair 2
Pair N
Group 1
Group 2
Group N
Count 1
Count 2
Count N
Collect

Execution


(quick, 1)
(brown, 2)
(fox, 1)
(jumps, 1)
(over, 1)
(the, 2)
(dog, 1)
(the, 1), (quick, 1), (brown, 1)
(fox, 1), (jumps, 1), (over, 1)
(the, 1), (brown, 1), (dog, 1)
HDFS Block 1
Partition 1
Split 1
Pair 1
Group 1
Count 1
Take(5)
HDFS Block 2
Partition 2
Split 2
Pair 2
Group 2
Count 2
HDFS Block N
Partition N
Split N
Pair N
Group N
Count N

Spark on Cray XC

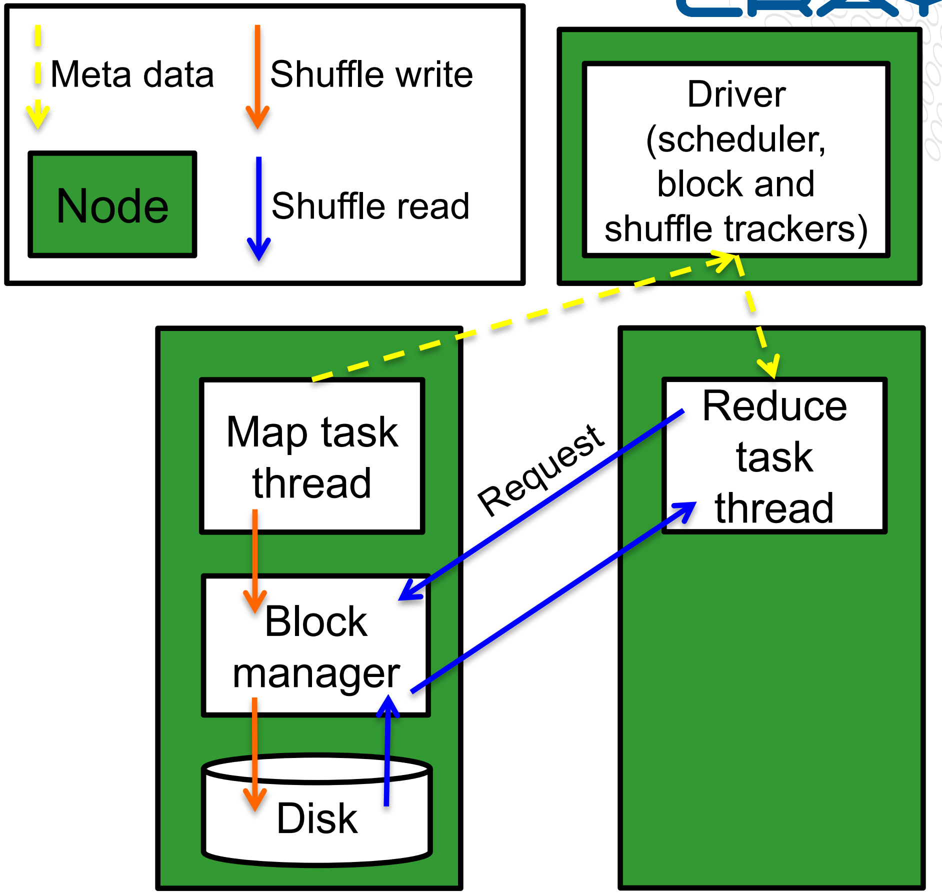
Spark on XC: Typical Setup Options

- **Cluster Compatibility Mode (CCM) option**
  - Set up and launch standalone Spark cluster in CCM mode; run interactively from Mom node or submit batch script
  - An example recipe can be found in:

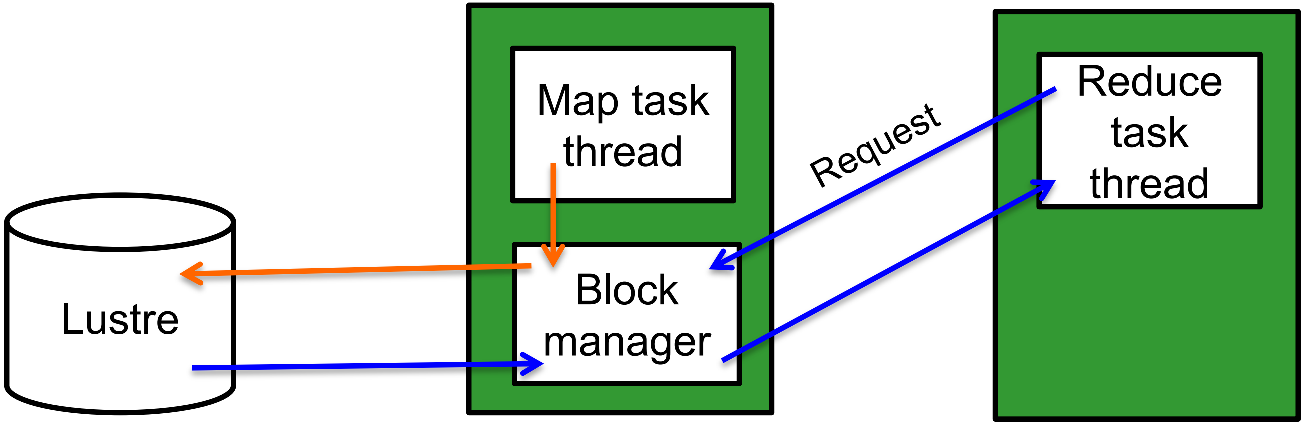
    “Experiences Running and Optimizing the Berkeley Data Analytics Stack on Cray Platforms”, Maschhoff and Ringenburg, CUG 2015
- **Container option**
  - Shifter container runtime (think “Docker for XC”) developed at NERSC
  - Acquire node allocation: run master image on one node, interactive image on another, worker images on rest
  - Cray’s Urika-XC analytics suite uses this approach
- **Challenge: Lack of local storage for Spark shuffles and spills.**

Reminder: Spark Shuffle – Standard Implementation

- **Senders ("mappers") send data to block managers; block managers write to local disks, tell driver how much destined for each reducer**
- **Barrier until all mappers complete shuffle writes**
- **Receivers ("reducers") request data from block managers that have data for them; block managers read from local disk and send**
- **Key assumption: large, fast local block storage device(s) available on executor nodes**

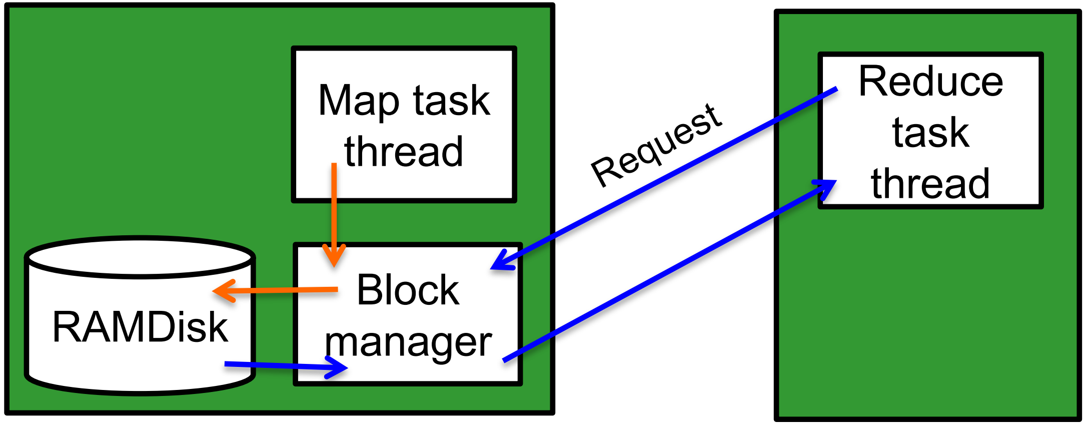


Shuffle on XC – Version 1



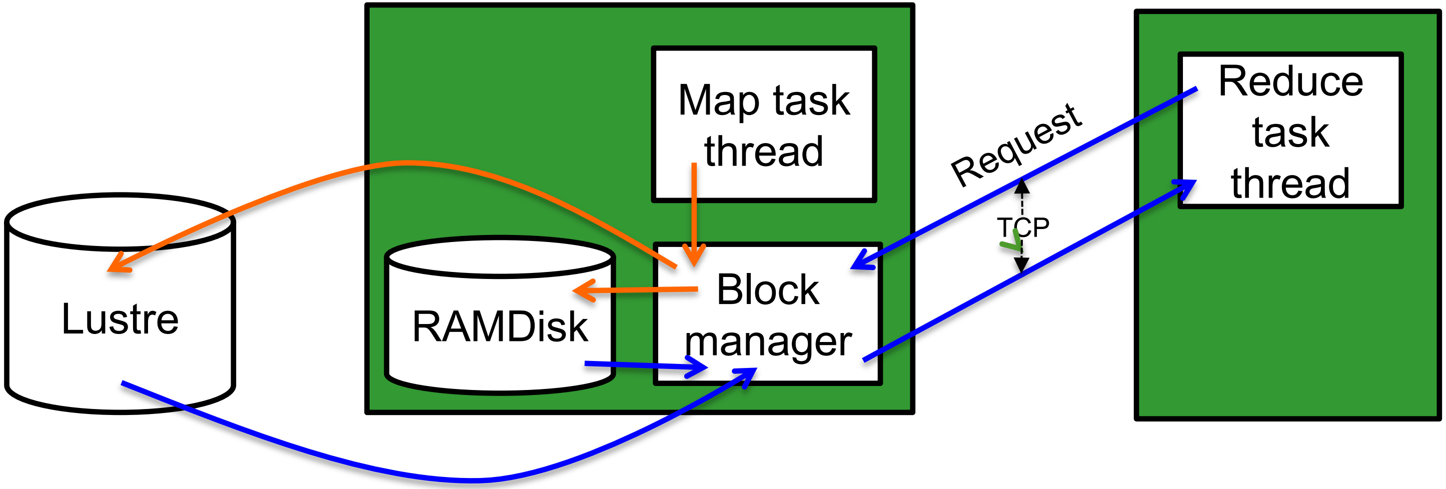
- **Problems: No local disk on standard XC40**
- **First try: Write to Lustre instead**
  - Biggest Issue: Poor file access pattern for lustre (lots of small files, constant opens/closes). Creates a major bottleneck on Lustre Metadata Server (MDS).
  - Issue 2: Unnecessary extra traffic through network

Shuffle on XC – Version 2



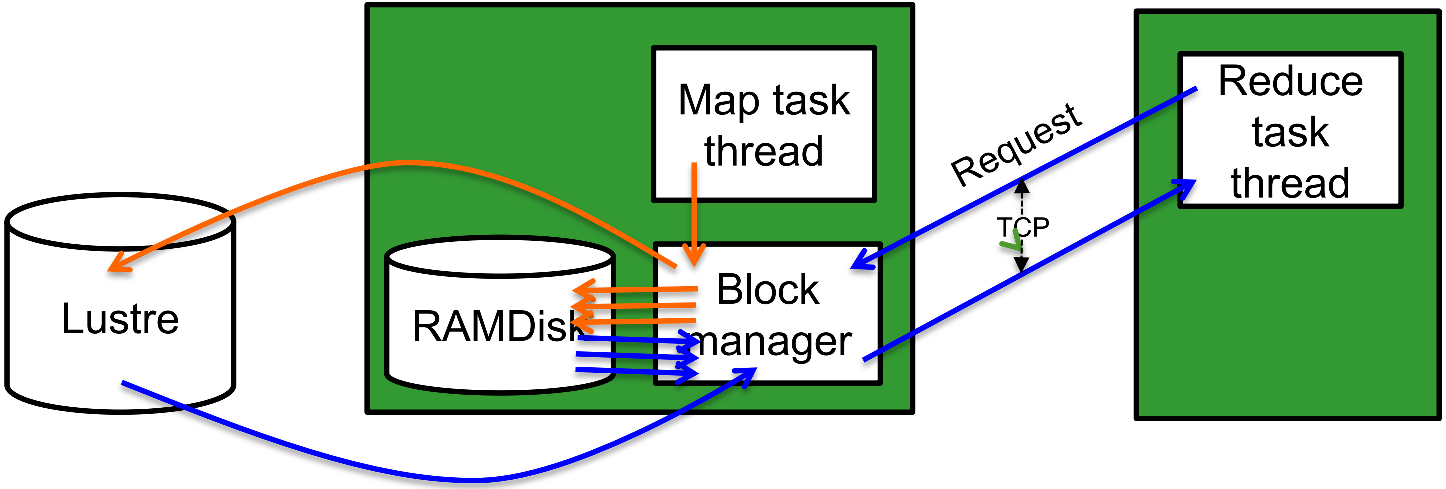
- **Second try: Write to RAMDisk**
  - Much faster, but …
  - Issues: Limited to lessor of: 50% of node DRAM or unused DRAM; Fills up quickly; takes away memory that could otherwise be allocated to Spark
  - Spark behaves unpredictably when it's local scratch space fills up (failures not always simple to diagnose)

Shuffle on XC – Version 3



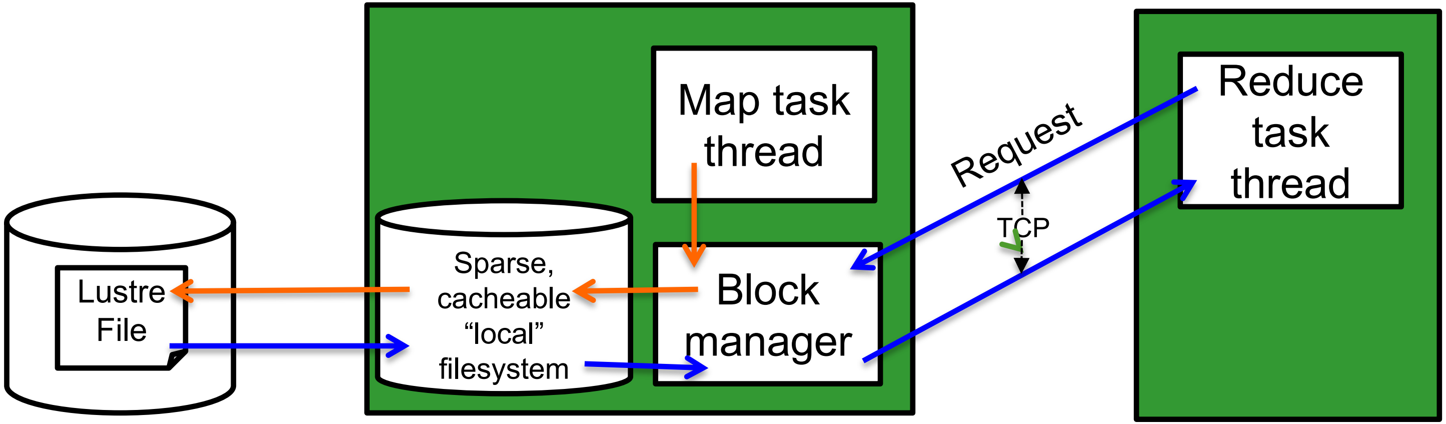
- **Third try: Write to RAMDisk and Lustre**
  - Set local directories to RAMdisk and lustre (can be list)
  - Initially fast and keeps working when RAMDisk full
  - Issues: Slow once RAMDisk fills; Round robin between directories (no bias towards faster RAM)

Shuffle on XC – Version 3



- **Third try: Write to RAMDisk and Lustre**
  - Set local directories to RAMdisk and lustre (can be list)
  - Initially fast and keeps working when RAMDisk full
  - Issues: Slow once RAMDisk fills; Round robin between directories (no bias towards faster RAM), *but can specify multiple RAM directories*

Shuffle on XC – with Shifter PerNodeCache



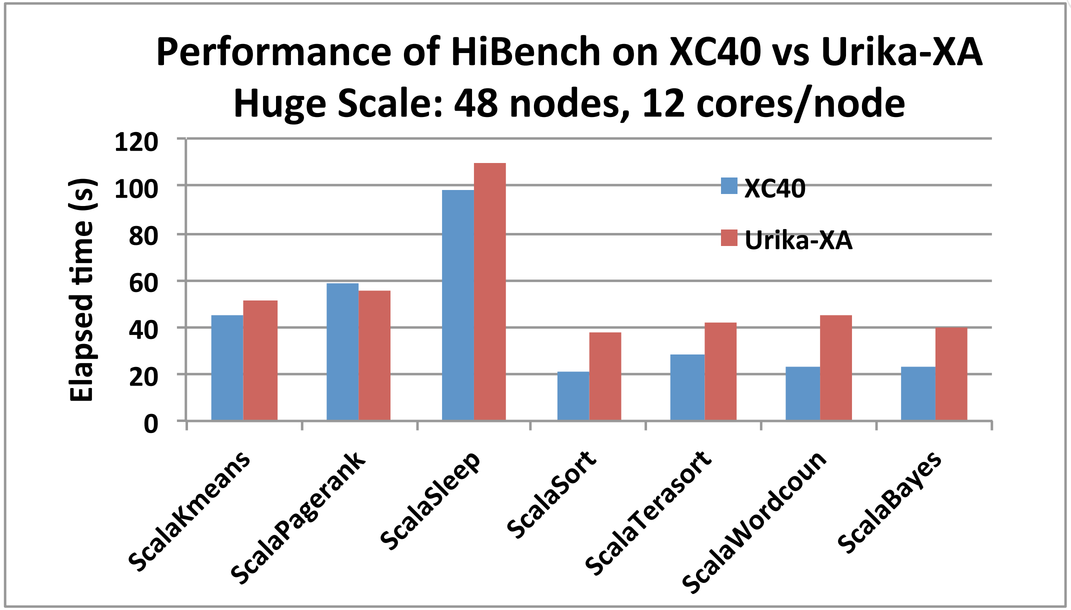
- **Shifter implementation: Per-node loopback file system**
  - NERSC's Shifter containerization (in Cray CLE6) provides optional loopback-mounted per-node temporary filesystem
  - Local to each node – fully cacheable
  - Backed by a single sparse file on Lustre – greatly reduced MDS load, plenty of capacity, doesn't waste space
  - Performance comparable to RAMDisk, without capacity constraints (Chaimov et al, CUG '16)
- **Urika-XC ships as a Shifter image and uses this approach**

Other Spark Configurations

- **Many config parameters … some of the more relevant:**
    - **spark.shuffle.compress**: Defaults to true. Controls whether shuffle data is compressed. In many cases with fast interconnect, compression and decompression overhead can cost more than the transmission time savings. However, can still be helpful if limited shuffle scratch space.
    - **spark.locality.wait**: Defaults to 3 (seconds). How long to wait for available resources on a node with data locality before trying to execute tasks on another node. Worth playing around with - decrease if seeing a lot of idle executors. Increase if seeing poor locality. (Can check both in history server.) Do not set to 0!

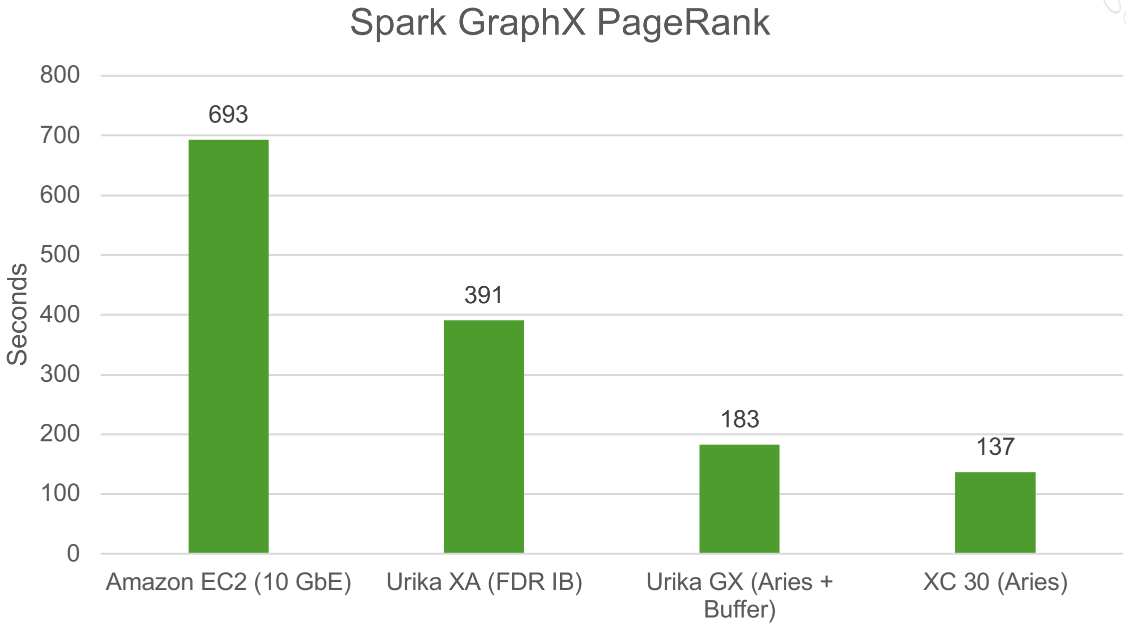
Spark Performance on XC: HiBench

- **Intel HiBench**
  - Originally MapReduce, Spark added in version 4
- **Compared performance with Urika XA system**
  - XA: FDR Infiniband, XC40: Aries
  - Both: 32 core Haswell nodes
  - XA: 128 GB/node, XC40: 256 GB/node (problems fit in memory on both)
- **Similar performace on Kmeans, PageRank, Sleep**
- **XC40 faster for Sort, TeraSort, Wordcount, Bayes**



Spark Performance on XC: GraphX

- **GraphX PageRank**
  - 20 iterations on Twitter dataset
  - Interconnect sensitive
- **GX has slightly higher latency and lower peak TCP bandwidth than XC due to buffer chip**



Demo: PySpark in Jupyter

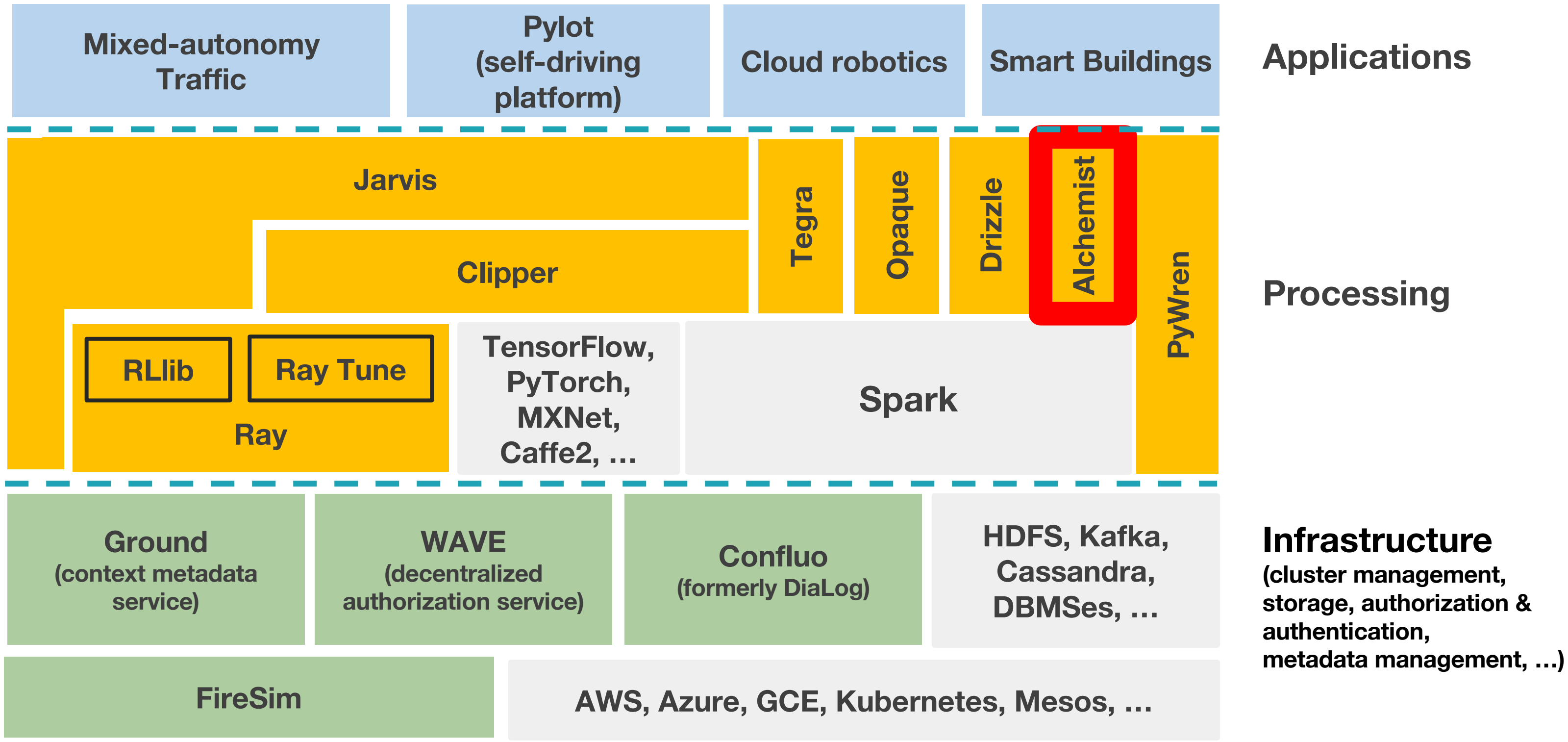
Alchemist

An Apache Spark ⇔ MPI Interface

A Collaboration of Cray and the UC Berkeley RiseLab (Alex Gittens, Kai Rothauge, Michael W. Mahoney, Shusen Wang, Jey Kottalam)

Slides courtesy Kai Rothauge

RISE Stack



Applications

Mixed-autonomy Traffic

Pylot (self-driving platform)

Cloud robotics

Smart Buildings

Processing

Jarvis

Clipper

Tegra

Opaque

Drizzle

Alchemist

PyWren

RLlib

Ray Tune

Ray

TensorFlow, PyTorch, MXNet, Caffe2, …

Spark

Infrastructure
(cluster management, storage, authorization & authentication, metadata management, …)

Ground (context metadata service)

WAVE (decentralized authorization service)

Confluo (formerly DiaLog)

HDFS, Kafka, Cassandra, DBMSes, …

FireSim

AWS, Azure, GCE, Kubernetes, Mesos, …

Slides courtesy Kai Rothauge, UC Berkeley

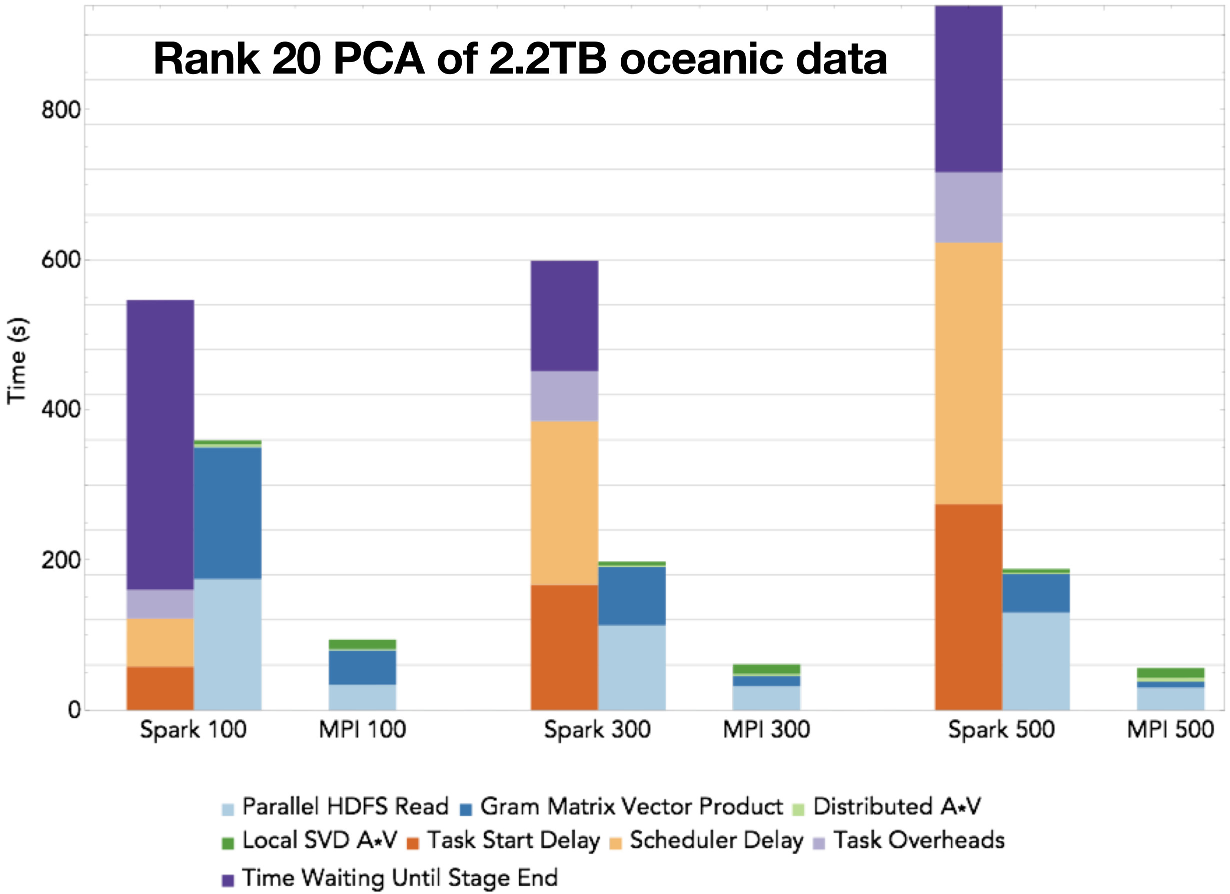
MPI vs Spark

- Cray and AMPLab performed case study for numerical linear algebra on Spark vs. MPI
- Why do linear algebra in Spark?
    - **Pros**:
        - Faster development, easier reuse
        - One abstract uniform interface (RDD)
        - An entire ecosystem that can be used before and after the NLA computations
        - Spark can take advantage of available local linear algebra codes
        - Automatic fault-tolerance, out-of-core support
    - **Con**:
        - Classical MPI-based linear algebra implementations will be faster and more efficient

Slides courtesy Kai Rothauge, UC Berkeley

MPI vs Spark

- Performed a case study for numerical linear algebra on Spark vs. MPI:
  - Matrix factorizations considered include Principal Component Analysis (PCA)
  - Data sets include
    - Oceanic data: 2.2 TB
    - Atmospheric data: 16 TB



Rank 20 PCA of 2.2TB oceanic data

A. Gittens et al. “Matrix factorizations at scale: A comparison of scientific data analytics in Spark and C+MPI using three case studies”, 2016 IEEE International Conference on Big Data (Big Data), pages 204–213, Dec 2016.

Slides courtesy Kai Rothauge, UC Berkeley

MPI vs Spark: Lessons learned

- With favorable data (tall and skinny) and well-adapted algorithms, linear algebra in Spark is 2x-26x slower than MPI when I/O is included
- Spark's overheads are orders of magnitude higher than the actual computations
  - Overheads include time until stage end, scheduler delay, task start delay, executor deserialize time, inefficiencies related to running code via JVM
- **The gaps in performance suggest it may be better to interface with MPI-based codes from Spark**

Slides courtesy Kai Rothauge, UC Berkeley

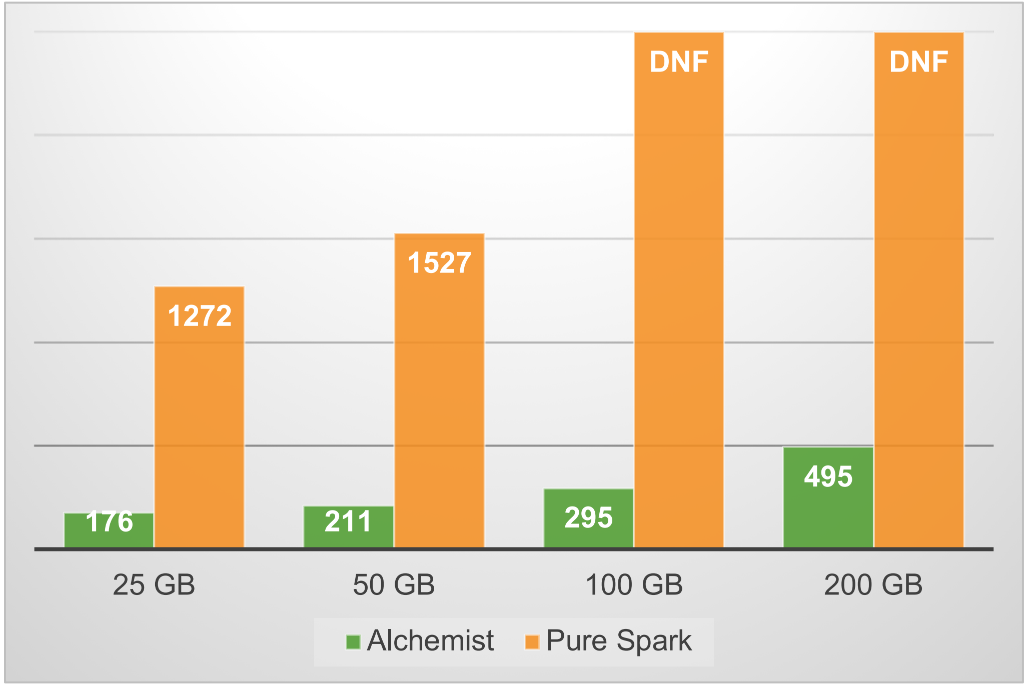
Alchemist

- Interface between Apache Spark and ***existing*** MPI-based libraries for NLA, ML, etc.
- Design goals include making the system *easy to use*, *efficient*, and *scalable*
- Two main tasks:
  - Send distributed **input** matrices from Spark to MPI-based libraries (**Spark => MPI**)
  - Send distributed **output** matrices back to Spark (**Spark <= MPI**)
- Want as little overhead as possible when transferring data between Spark and a library
- Three possible approaches:
  - File I/O (e.g. HDFS) **too slow!** ✗
  - Use shared memory buffers, Apache Ignite, Alluxio, etc. **extra copy in memory** ✗
  - Use in-memory transfer, send data between processes using sockets ✓

Slides courtesy Kai Rothauge, UC Berkeley

Truncated SVD Alchemist vs Pure Spark

- Use Alchemist and MLlib to get rank 20 truncated SVD
- Setup:
  - 30 KNL nodes, 96GB DDR4, 16GB MCDRAM
  - Spark: 22 nodes; Alchemist: 8 nodes
  - A: m-by-10K, where m = 5M, 2.5M, 1.25M, 625K, 312.5K
  - Ran jobs for at most 60 minutes (3600 s)
- Alchemist times include data transfer



Slides courtesy Kai Rothauge, UC Berkeley

Don’t miss Alex Gitten’s talk on Tuesday!

Alchemist: An Apache Spark <=> MPI Interface

Technical Session 9C

Tuesday, 4:00-4:30PM

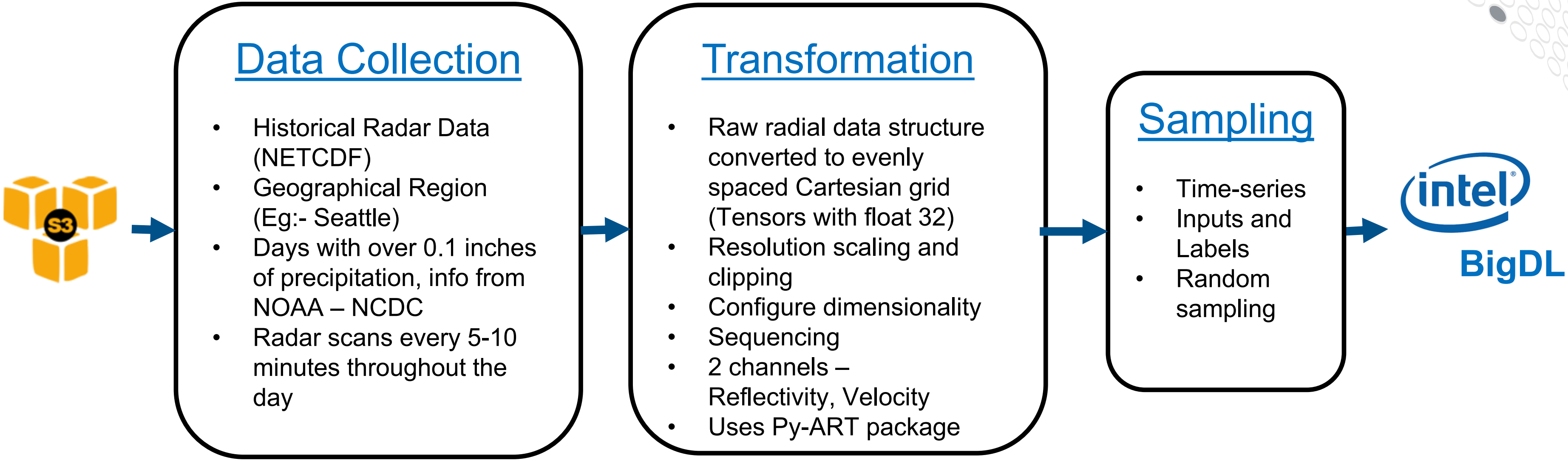
Right after Alex Heye’s talk

Deep Learning in Spark with BigDL

Motivating Example: Precipitation Nowcasting

- **Problem: Predict precipitation locations and rates at a regional level over a short timeframe**
  - Neighborhood level predictions
  - T+0 – T+6 hours
- **Standard Approach: Numerical Weather Prediction**
  - Physics based simulations
  - High computational cost limits performance and accessibility
- **Cutting edge approach: Deep Learning**
  - Predict rainfall by learning from historical data
  - Heavy computation occurs ahead of time
  - Pre-Trained models can be deployed as soon as data is available

Data Processing Pipeline



Data Collection

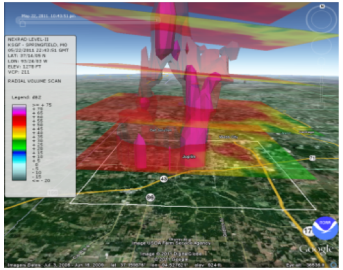
- Historical Radar Data (NETCDF)
- Geographical Region (Eg:- Seattle)
- Days with over 0.1 inches of precipitation, info from NOAA – NCDC
- Radar scans every 5-10 minutes throughout the day

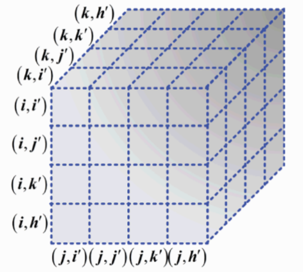
Transformation

- Raw radial data structure converted to evenly spaced Cartesian grid (Tensors with float 32)
- Resolution scaling and clipping
- Configure dimensionality
- Sequencing
- 2 channels – Reflectivity, Velocity
- Uses Py-ART package

Sampling

- Time-series
- Inputs and Labels
- Random sampling

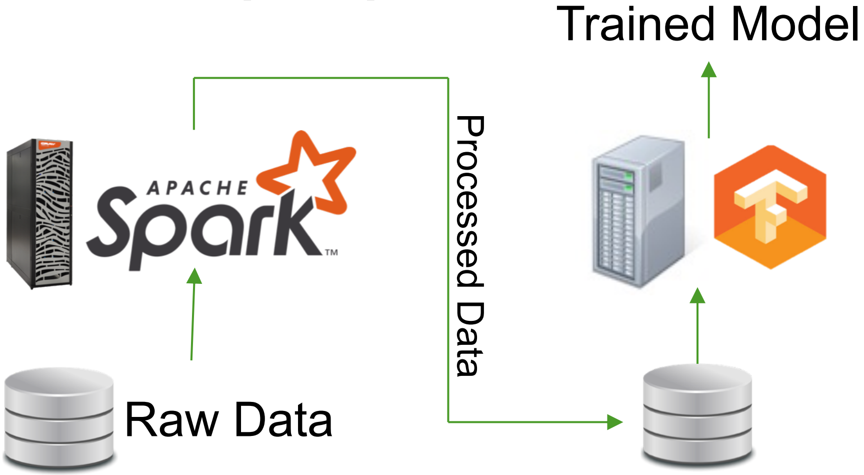




Initial Implementation: Tensorflow + Spark

- **Separate workflows – no integration**
  - Forced overhead – data movement
  - Distinct data pipelines
- **Data processing – highly distributed analytics platform**
- **DL Training implementation – dense compute platform**

- **Pro:**
  - Specialized hardware
  - good individual performance
- **Con:**
  - Productivity loss
  - Fragmented workflow

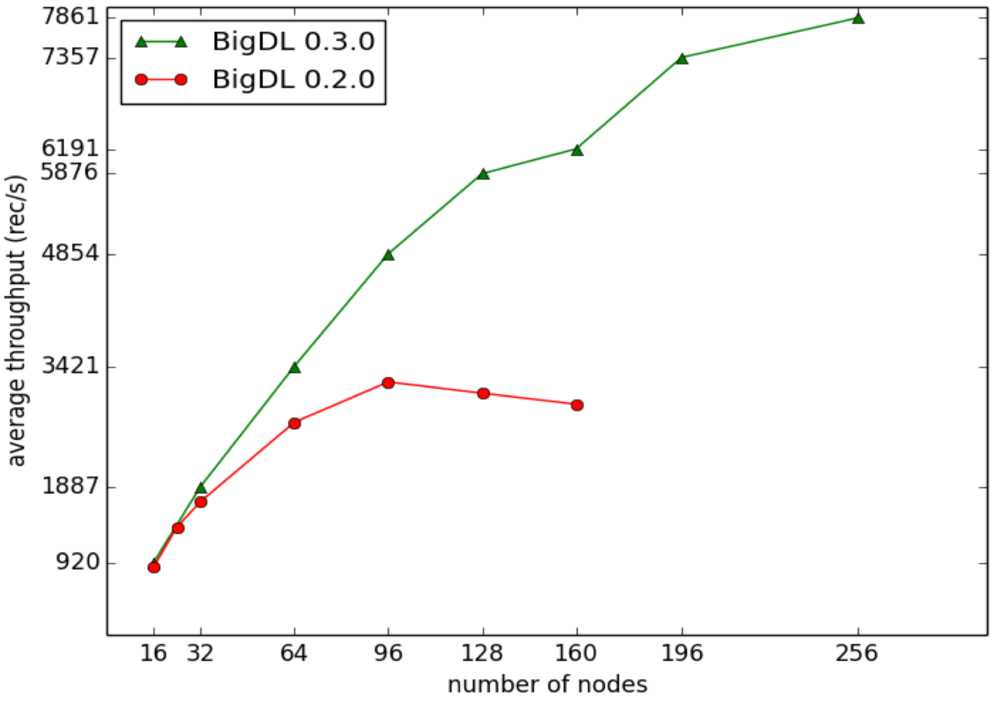


New Implementation: Intel BigDL

- **Distributed Deep Learning Library**
- **Natively integrated with Spark**
  - Single Spark Context
  - Dataset stays in memory
  - Effortless distributed training
- **Optimized with MKL-DNN libraries**
- **Interface similar to Torch**
  - Stacked NN layers
  - Define a very complex model in very few lines
- **Quickly integrate Deep Learning and Machine Learning into Spark-based data analytics workloads**

BigDL

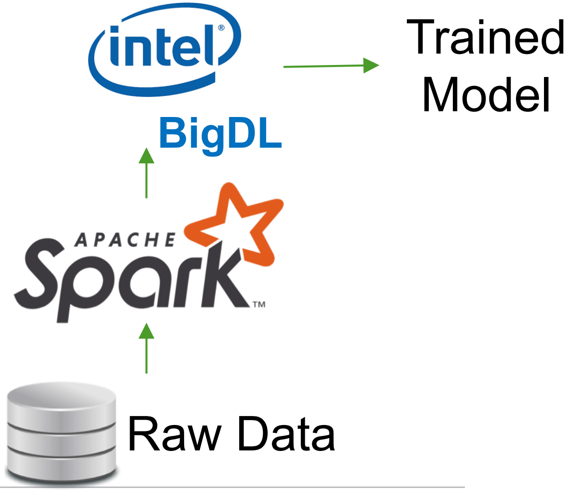
BigDL Training Scaling



COMPUTE | STORE | ANALYZE

Implementation: BigDL on Spark

- **Singular workflow**
  - Data processing on spark flows directly into the training process with BigDL
- **HPC scale with Urika-XC**
  - High performance compute nodes excel at data analytics
  - MKL, MKL-DNN provide suitable optimization for DL workloads
  - Suite of analytics tools to aid in development
- **Pros:**
  - Single platform
  - Highly productive development environment
  - Effortless distribution
- **Cons:**
  - Less flexible expressive Deep Learning tools
  - Less flexible compute environment



Demo: BigDL MNIST

Legal Disclaimer

Information in this document is provided in connection with Cray Inc. products. No license, express or implied, to any intellectual property rights is granted by this document.

Cray Inc. may make changes to specifications and product descriptions at any time, without notice.

All products, dates and figures specified are preliminary based on current expectations, and are subject to change without notice.

Cray hardware and software products may contain design defects or errors known as errata, which may cause the product to deviate from published specifications. Current characterized errata are available on request.

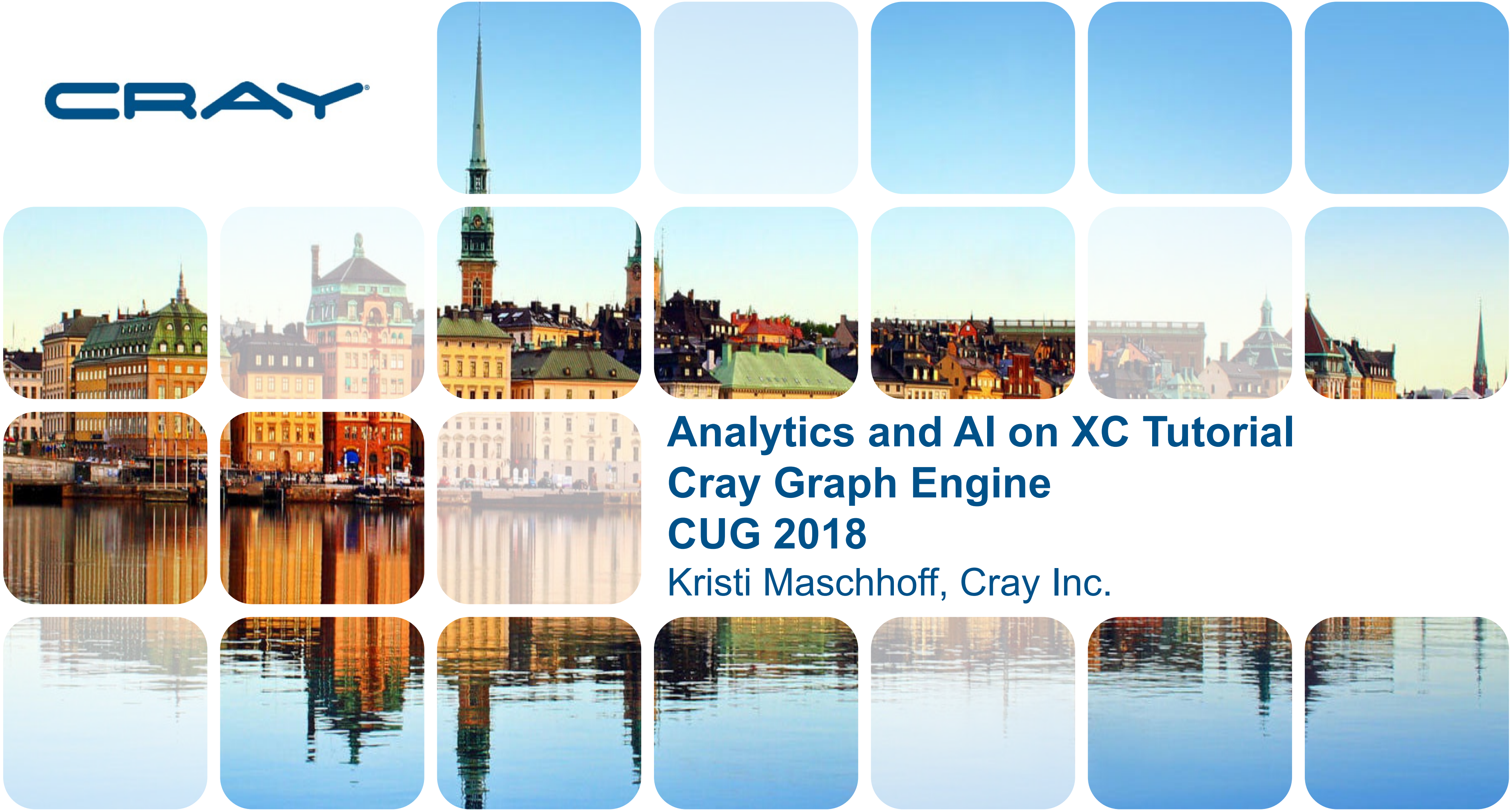
Cray uses codenames internally to identify products that are in development and not yet publicly announced for release. Customers and other third parties are not authorized by Cray Inc. to use codenames in advertising, promotion or marketing and any use of Cray Inc. internal codenames is at the sole risk of the user.

Performance tests and ratings are measured using specific systems and/or components and reflect the approximate performance of Cray Inc. products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance.

The following are trademarks of Cray Inc. and are registered in the United States and other countries: CRAY and design, SONEXION, URIKA and YARCDATA. The following are trademarks of Cray Inc.: CHAPEL, CLUSTER CONNECT, CLUSTERSTOR, CRAYDOC, CRAYPAT, CRAYPORT, DATAWARP, ECOPHLEX, LIBSCI, NODEKARE, REVEAL. The following system family marks, and associated model number marks, are trademarks of Cray Inc.: CS, CX, XC, XE, XK, XMT and XT. The registered trademark LINUX is used pursuant to a sublicense from LMI, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis. Other trademarks used on this website are the property of their respective owners.

Q&A

Mike Ringenburg
mikeri@cray.com



Analytics and AI on XC Tutorial
Cray Graph Engine
CUG 2018

Kristi Maschhoff, Cray Inc.

Cray Graph Engine (CGE)

- **Scalable parallel graph analytics framework**
  - Semantic in-memory graph database
    - Basic graph pattern search
    - Graph-theoretic algorithms (whole graph algorithms)
  - W3C Standards Based
    - Uses RDF Data representation
    - Uses SPARQL as query language
  - Built for “vertical scaling” based on parallel and distributed computing principles — competitors are all horizontally scaled
- **Brings interactivity to graph-based discovery**
  - Scaling and performance enables interactive analysis of very large datasets

Cray Graph Engine: Updates and Features

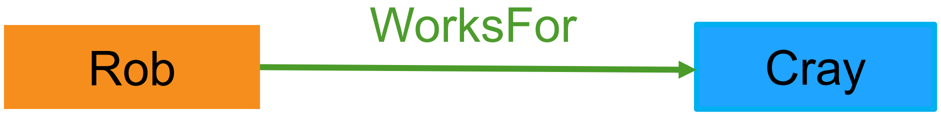
- **Multi-Architecture Support**
  - CGE is available on the Urika-GX and the XC platforms.
  - Strong scaling becomes a key differentiator
    - Bigger datasets => more nodes => better performance
- **Integration with Spark**
  - Interface to data sources - support for end-end analytic workflow realization
- **Integration with Python/Jupyter Notebooks**
  - Connect to SPARQL endpoint using sparqlwrapper or sparql-client packages
  - CGE Python API – utilizes the CGE Java API
    - Start up server, run queries, updates, checkpoint, shut down
- **Integration with R**
  - SPARQL package – connect to SPARQL endpoint, run queries, updates

- **Don’t miss the talk on Thursday!**
  - Thursday, Technical Session 24C
    - “Loading and Querying a Trillion RDF triples with Cray Graph Engine on the Cray XC”
- **And you may also be interested in …**
  - BOF “Tools and Utilities for Data Science Workloads and Workflows,”

What is RDF?

- Resource Description Framework (RDF)
- A standardized abstract data model centered around the notion of Triples
- A Triple expresses a directed relationship between two entities e.g.

http://schema.org/worksFor



WorksFor

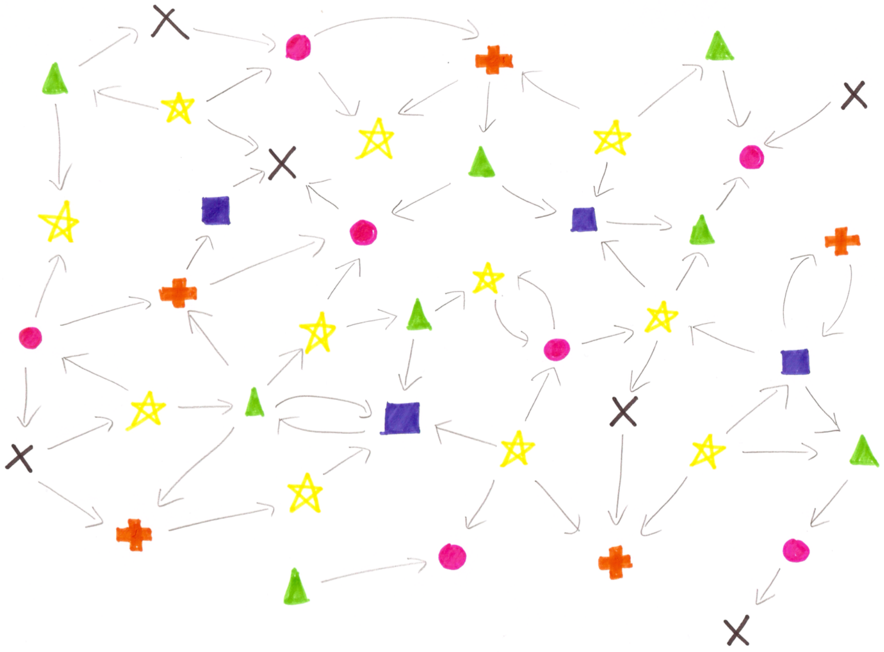
Rob

Cray

http://www.dotnetrdf.org/people/RobVesse http://www.cray.com (URIs)

- **Components of a Triple are commonly known as Subject, Predicate and Object**
  - Subject – The thing I am making a statement about
  - Predicate – The relationship being stated
  - Object – The thing which is related

Graph analysis workloads



Two main workloads

- Pattern matching
- Whole graph analysis

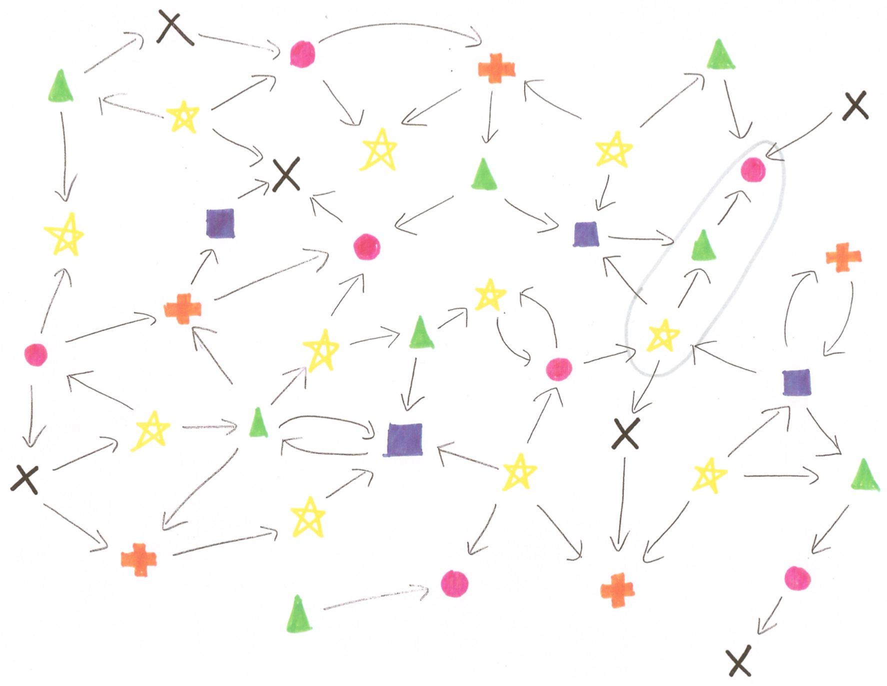
Typical systems only good at one

CGE excels at both

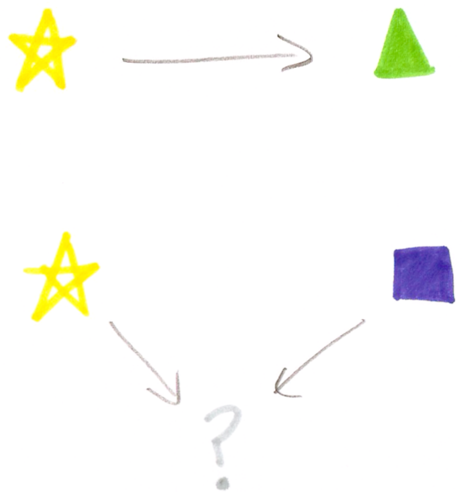
What we plan to cover in the tutorial

- **Background on CGE**
  - Pattern matching, whole-graph analysis
- **Hands-on exercises**
  - Build and start up a database (cge-launcher)
  - Run queries
    - Using the cge-cli command line
    - Using the CGE Web UI
  - Integration with R and Python
    - Connecting to the CGE SPARQL endpoint
      - Using R SPARQL package
      - Using Python SPARQLwrapper package

A Graph-pattern matching workload



Given a pattern of interest find all instances thereof…

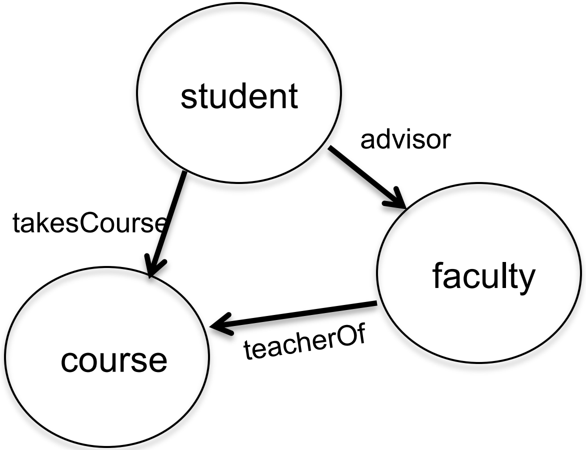


What SPARQL Can Do

- Subgraph isomorphism on specific, fixed patterns

"LUBM Query 9"

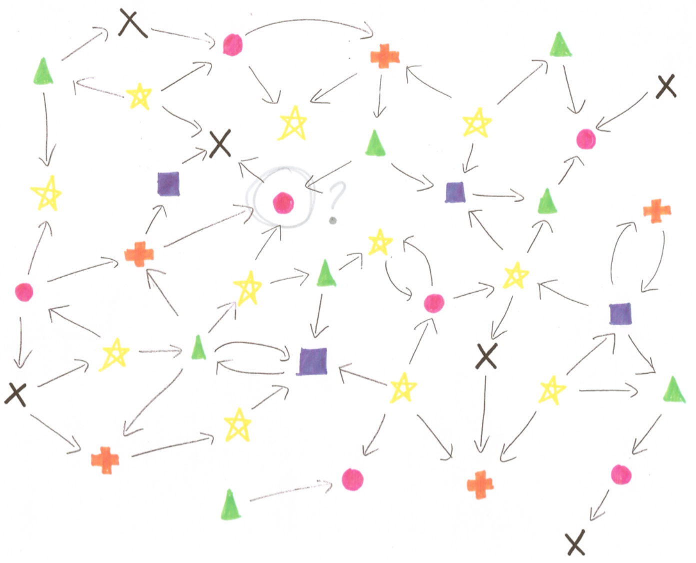
```
SELECT ?X, ?Y, ?Z
WHERE
{ ?X rdf:type ub:Student .
  ?Y rdf:type ub:Faculty .
  ?Z rdf:type ub:Course .
  ?X ub:advisor ?Y .
  ?Y ub:teacherOf ?Z .
  ?X ub:takesCourse ?Z}
```

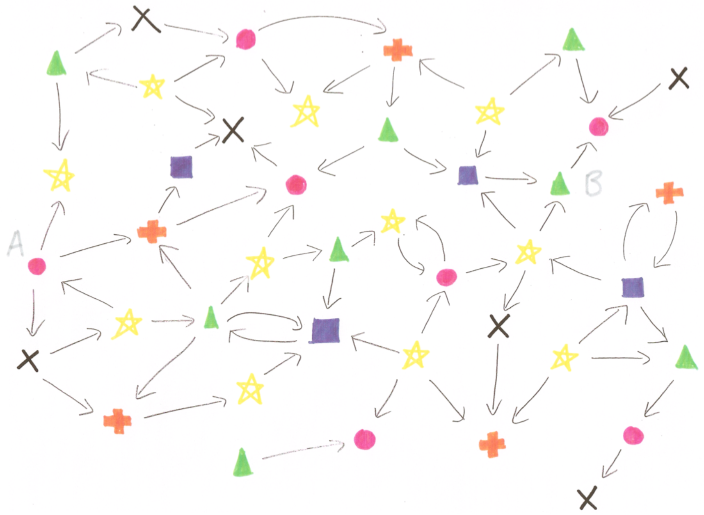


- `Plus lots of useful database features: filter, group, update…`

A Graph-theoretic Workload

What's the shortest route from A to B?





What is the ranking of the targeted vertex?

Built-in Graph Functions (BGFs)

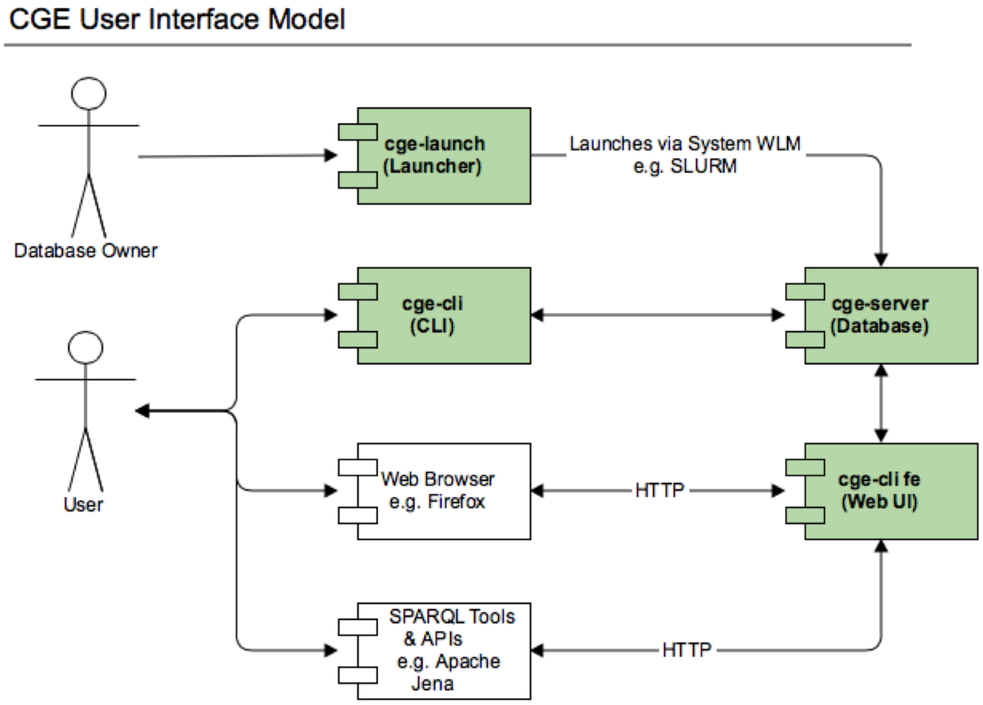
- RDF and SPARQL are graph-oriented, but SPARQL is limited in its ability to express graph processing

- We augmented SPARQL with a capability of calling library graph algorithms

- You can go from SPARQL to a graph algorithm and back to SPARQL for further refinement

- The whole is greater than the sum of its parts

CGE User Interface Model



- **Database owner launches the database server**
- **Users interact via their preferred interface**
  - Commands Line
  - Web Browser
  - SPARQL Tools & APIs
  - CLI may be used for scripted workflows

Building and launching

- cge-launch is used to build databases:

```
cge-launch –N 8 –I 16 –o /mnt/lustre/myresults –d
/mnt/lustre/mydata –l logfile
```

- `cge-launch` is a script that takes care of resource allocation for the user!

- After a successful build, the database directory will contain:

  dataset.nt
  rules.txt
  dbQuads
  string_table_chars
  string_table_chars.index
  graph.info

The database port

- A TCP port used for communication with this server instance:

```
cge-launch –N 8 –I 16 –p 3750 …
```

- The default is 3750
- Changing this port allows multiple versions

The database directory

- The database directory, typically:

```
/mnt/lustre/user/datasets/lubm0
```

- Is the start of a directory tree containing all checkpoints, and potentially authorized_keys
- It can be moved, archived and returned (!)
- Multiple users can access it, with permissions

The Command Line Interface (CLI)

```
cge-cli —db-port 3750 query myquery.rq
```

- The CLI is used for most interactions with the server, and has many options…
- `cge-cli help` (or `cge-cli help checkpoint`) will give verbose information on options
- Designed for scripted control, querying and updates with database server
- Communications are secure SSH

CLI — most common options

`query` – submits SPARQL queries

`update` – submits SPARUL updates

`sparql` – submits both queries and updates

`checkpoint` – creates a database checkpoint

`echo` – check status of server

Customization using NVPs and cge.properties file

- **Retrieve the Default NVP Configurations**

```
$ cge-cli nvp-info
```

- **For Piz Daint, need to modify internal memory allocator defaults settings due to accommodate smaller 64GB nodes**
  - CGE uses a internal memory allocator to avoid issues with observed memory fragmentation on XC systems
  - cge.server.BuddyMemPercent 20 (current default 35)
  - cge. server.PersistBuddyMemPercent 20 (current default 25)

- **More information**
  - https://pubs.cray.com/content/S-3014/3.2.UP01/cray-graph-engine-user-guide

Hands on Exercises: Running CGE on Piz Daint

- **See README for instructions and exercises**
  - /scratch/snx3000/kristyn/CUG2018/CGE/README
- **Load CGE module**
  - module use /scratch/snx3000/kristyn/CUG2018/modulefiles
  - module load cge
- **To use CGE Web UI, need to set up ssh tunneling**
  - Piz Daint is only accessible via ssh from the front end ela.cscs.ch,need to create a tunnel through the front end to Daint
    - Use a random port number (8022) to connect to ssh port 22
      - ssh -L localhost:8022:daint:22 ela.cscs.ch
    - Then ssh directly into daint node, choosing another random port number (15000) for CGE fe
      - ssh –p 8022 –L localhost:15000:localhost:15000 localhost

Hands on Exercises: Running CGE on Piz Daint (2)

- **Create cge.properties file in ~/.cge/cge.properties**
- **Set up database directory on Lustre**
  - Make sure Lustre striping is set
    - lfs setstripe –c 16 --stripe-size 16m .
  - Needed files: dataset.nt, graph.info, rules.txt
- **Set up query_results directory on Lustre**
  - Make sure Lustre striping is set
- **Be sure to set passwordless ssh**
  - ssh-keygen
  - cat id_rsa.pub >> authorized_keys

Legal Disclaimer

Information in this document is provided in connection with Cray Inc. products. No license, express or implied, to any intellectual property rights is granted by this document.

Cray Inc. may make changes to specifications and product descriptions at any time, without notice.

All products, dates and figures specified are preliminary based on current expectations, and are subject to change without notice.

Cray hardware and software products may contain design defects or errors known as errata, which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Cray uses codenames internally to identify products that are in development and not yet publicly announced for release. Customers and other third parties are not authorized by Cray Inc. to use codenames in advertising, promotion or marketing and any use of Cray Inc. internal codenames is at the sole risk of the user.

Performance tests and ratings are measured using specific systems and/or components and reflect the approximate performance of Cray Inc. products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance.

The following are trademarks of Cray Inc. and are registered in the United States and other countries: CRAY and design, SONEXION, URIKA and YARCDATA. The following are trademarks of Cray Inc.: CHAPEL, CLUSTER CONNECT, CLUSTERSTOR, CRAYDOC, CRAYPAT, CRAYPORT, DATAWARP, ECOPHLEX, LIBSCI, NODEKARE, REVEAL. The following system family marks, and associated model number marks, are trademarks of Cray Inc.: CS, CX, XC, XE, XK, XMT and XT. The registered trademark LINUX is used pursuant to a sublicense from LMI, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis. Other trademarks used on this website are the property of their respective owners.

Q&A

Kristi Maschhoff
kristyn@cray.com