ExaWorks

https://exaworks.org

Tutorial: Developing Robust and Scalable Next Generation Workflows Applications and Systems

PEARC 2022

Lawrence Livermore National Laboratory
Argonne National Laboratory
Brookhaven National Laboratory
Oak Ridge National Laboratory
U.S. Department of Energy Office of Science

Swift/T

`http://swift-lang.org/Swift-T`

Swift/T: Enabling high-performance scripted workflows

Write site-independent scripts, translates to MPI

Automatic task parallelization and data movement

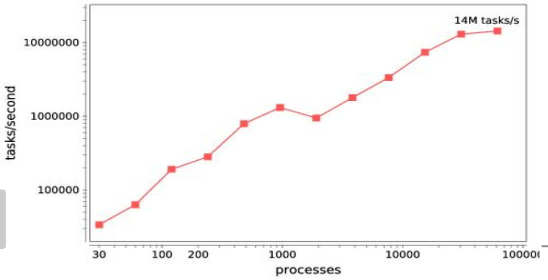
Invoke native code, script fragments in Python and R

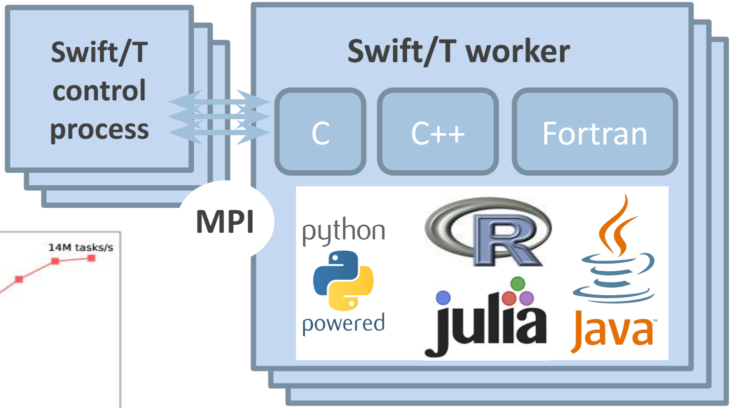
Rapidly subdivide large partitions for MPI jobs in multiple ways (MPI 3.0)

```
$ spack install stc
```

```
$ conda install -c lightsource2-tag swift-t
```

64K cores of Blue Waters
2 billion Python tasks
14 million Pythons/s





Swift/T: Scalable data flow programming for distributed-memory task-parallel applications
Proc. CCGrid 2013.

Goals of the Swift language

Swift was designed to handle many aspects of the computing campaign

- Make it easy to run large batteries of external program or library executions
- Ability to integrate many application components into a new workflow application
- Enable complex tasks based in other scripting languages (e.g., Python) or parallel MPI tasks
- Provide rich programming language at the top level – fully generic
- Data structures for complex data organization
- Portability- separate site-specific configuration from application logic
- Logging, provenance, and plotting features
- Support implicit concurrency and conventional programming constructs

The Swift programming model

All progress driven by concurrent dataflow

```
(int r) myproc (int i, int j)
{
    int x = F(i);
    int y = G(j);
    r = x + y;
}
```

- `F()` and `G()` implemented in native code or external programs
- `F()` and `G()` run in concurrently in different processes
- `r` is computed when they are both done
- This parallelism is *automatic*
- Works recursively throughout the program's call graph

Swift syntax

- Data types

```
int i = 4;
string s = "hello world";
file image<"snapshot.jpg">;
```

- Structured data

```
typedef image file;
image A[];
type protein_run {
  file pdb_in; file sim_out;
}
bag<blob>[] B;
```

- Conventional expressions

```
if (x == 3) {
    y = x+2;
    s = strcat("y: ", y);
}
```

- Parallel loops

```
foreach f,i in A {
    B[i] = convert(A[i]);
}
```

- Data flow

```
merge(analyze(B[0], B[1]),
      analyze(B[2], B[3]));
```


- **Swift: A language for distributed parallel scripting.** J. Parallel Computing, 2011
- **Compiler techniques for massively scalable implicit task parallelism.** Proc. SC, 2014

Swift task invocation

- Shell access

```
app (file o) myapp(file f, int i)
{"./mysim" "-s" i @f @o; }
```

- Or simply invoke a string:

```
output,error = system1("echo HELLO");
```

or

```
output,error = system(["echo", "HELLO"]);
```

- Python and R (etc.)

  - ```
    python("import numpy as np",
           "repr(np.eye(1))");
    ```

  - ```
    R("A=c(4,5,6)", "toString(mean(A))");
    ```

- External MPI programs

```
@par=8 launch("./my-program", args[], envs[]);
```

- In-memory MPI libraries

```
@par (string z)
covid_model_run(string config, string params)
"covid_model" "0.0" "covid_model_tcl";
```

```
@par=procs_per_run covid_model_run(default_model_props, p);
```

- mpi4py

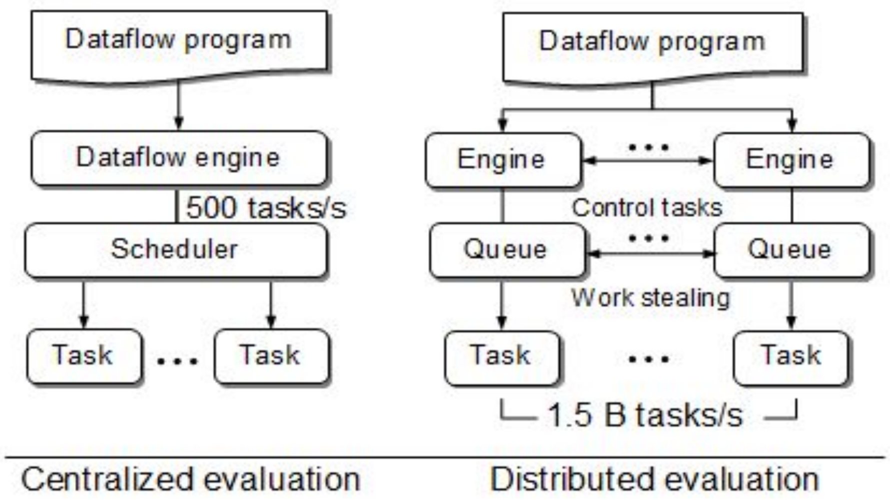
```
@par=6 python_parallel_persist(
      "import test_6;test_6.f('HELLO')");
```

test_6.py:f():

```
comm = turbine_helpers.get_task_comm()

size = comm.Get_size() ; rank = comm.Get_rank()

comm.barrier()

…
```

Centralized evaluation is a bottleneck at extreme scales

Had this (Swift/K): Now have this (Swift/T):



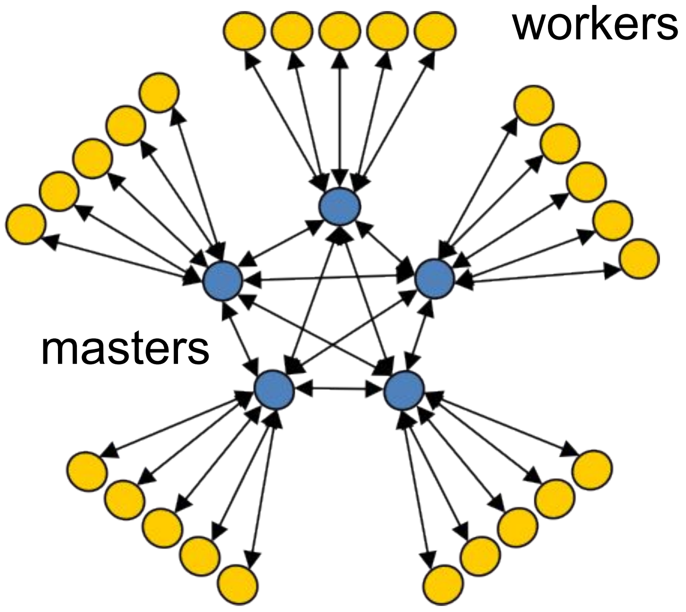
Turbine: A distributed-memory dataflow engine for high performance many-task applications. Fundamenta Informaticae 28(3), 2013

Asynchronous Dynamic Load Balancer

ADLB for short

- An MPI library for master-worker workloads in C
- Uses a variable-size, scalable network of servers
- Servers implement work-stealing
- The work unit is a byte array
- Optional work priorities, targets, types

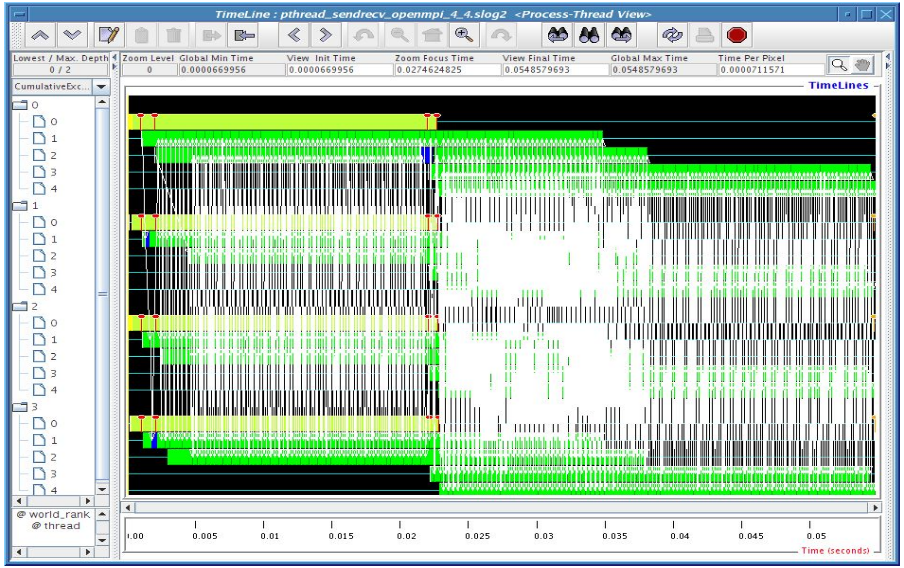
- For Swift/T, we added:
  - Server-stored data
  - Data-dependent execution



- Lusk et al. **More scalability, less pain: A simple programming model and its implementation for extreme computing.** SciDAC Review 17, 2010

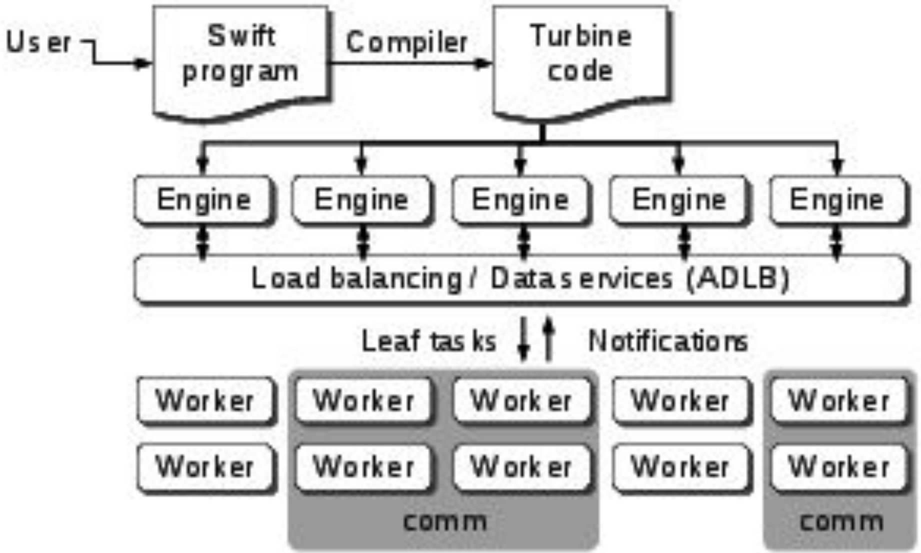
MPI: The message passing interface

- Programming model used on large supercomputers
- Can run on many networks, including sockets, or shared memory
- Standard API for C and Fortran; other languages have working implementations
- Contains communication calls for
  - Point-to-point (send/recv)
  - Collectives (broadcast, reduce, etc.)
- Interesting concepts
  - Communicators: collections of communicating processing and a context
  - Data types: Language-independent data marshaling scheme



Parallel tasks in Swift

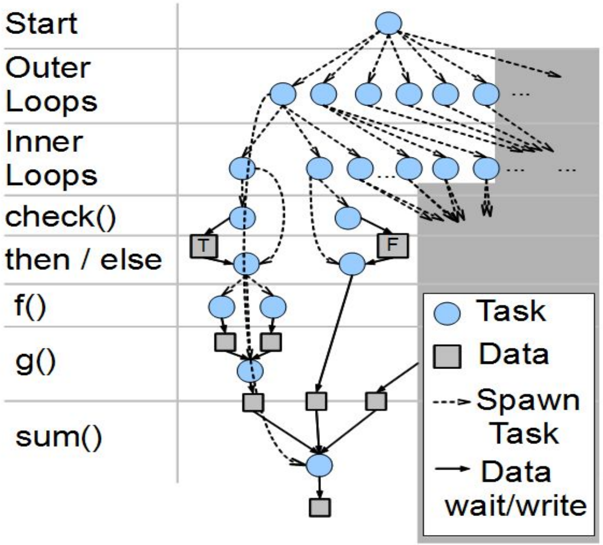
- Swift expression: `z = @par=8 f(x,y);`
- When x, y are stored, Turbine releases task `f` with `parallelism=8`
- Performs `ADLB_Put(f, parallelism=8)`
- Each worker performs `ADLB_Get(&task, &comm)`
- ADLB server finds 8 available workers
- Workers receive ranks from server
    - Perform **`MPI_Comm_create_group()`**
- `ADLB_Get()` returns:
  `task=f, size(comm)=8`
- Workers perform user task
    - communicate on `comm`
- `comm` is released by Turbine
- Can hand the communicator to RepastHPC, LAMMPS, NAMD, DIY, CODES/ROSS, etc.



- Wozniak et al. Dataflow coordination of data-parallel tasks via MPI 3.0. Proc EuroMPI, 2013.

Swift/T: Fully parallel evaluation of complex scripts

```
int X = 100, Y = 100;
int A[][];
int B[];
foreach x in [0:X-1] {
  foreach y in [0:Y-1] {
    if (check(x, y)) {
      A[x][y] = g(f(x), f(y));
    } else {
      A[x][y] = 0;
    }
  }
  B[x] = sum(A[x]);
}
```

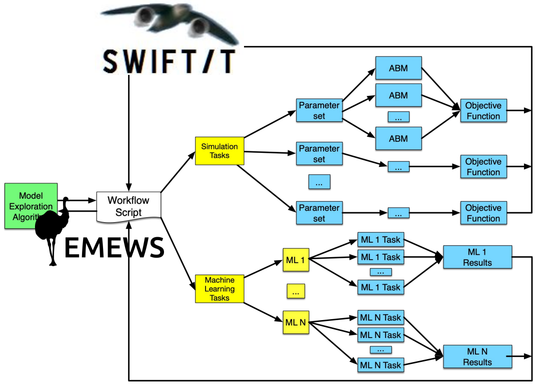


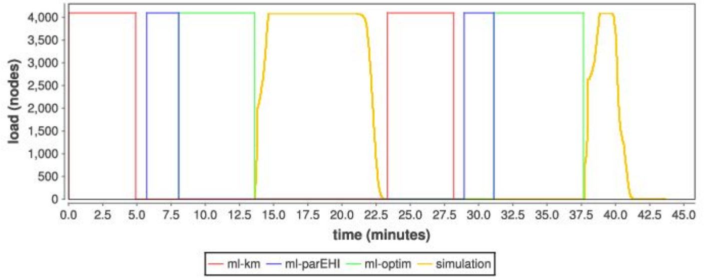
Compiler techniques for massively scalable implicit task parallelism.
SC 2014.

City-scale COVID-19 epidemic modeling

ACM Gordon Bell Special Prize
for High Performance Computing-Based
COVID-19 Research
Finalist
2020

- **Observed city data**
  - Hospitalizations, cumulative deaths from Chicago Department of Public Health
  - Detailed line list data from Illinois Department of Public Health
- **ML Phases:**
  - Distribution of R code snippets via R `foreach %dopar%` syntax; work distributed by EMEWS
- **Simulation:**
  - Distribution of MPI tasks via Swift/T `@par` syntax.
    MPI communicators are dynamically allocated over in-order cores by Swift/T using `MPI_Comm_create_group()`
- **Key performance metric:** Scalability and Time to Solution
  - Keep the cores busy in presence of changing task types and workflow dynamics
- **Simulation phase** starts 1024 MPI tasks, each on a 256-rank MPI communicator
  - Assigns tasks at rate of 4,695 core-tasks per second, 73 communicators per second
- **The ML phases** are single-node, vendor-optimized R calls

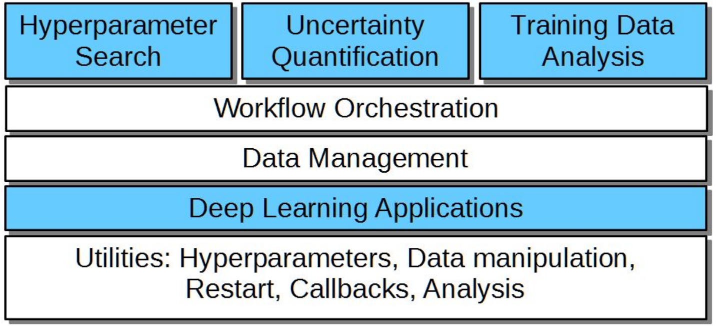


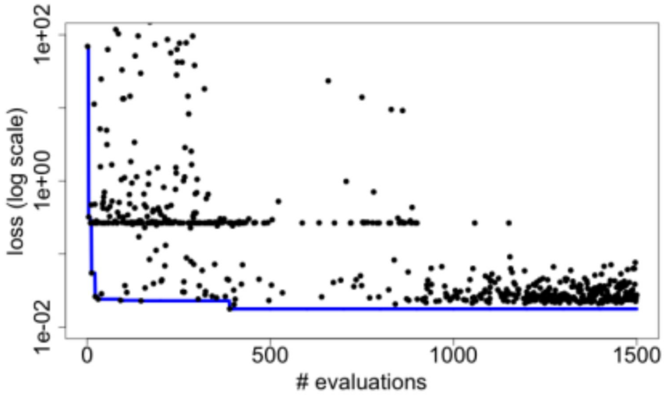


A population data-driven workflow for COVID-19 modeling and learning. IJHPCA 2021.

ECP CANDLE Hyperparameter optimization

- Goal: Develop an exascale deep learning environment for cancer, enabling the most challenging deep learning problems in cancer research to run on the most capable supercomputers
- Neural networks have a large number of possible configuration parameters, called hyperparameters
- CANDLE/Supervisor consists of several high-level workflows
- Capable of modifying/controlling application parameters dynamically as the workflow progresses and training runs complete
- Distribute work across large computing infrastructure, manage progress
- Underlying applications are Python programs that use Keras/TensorFlow





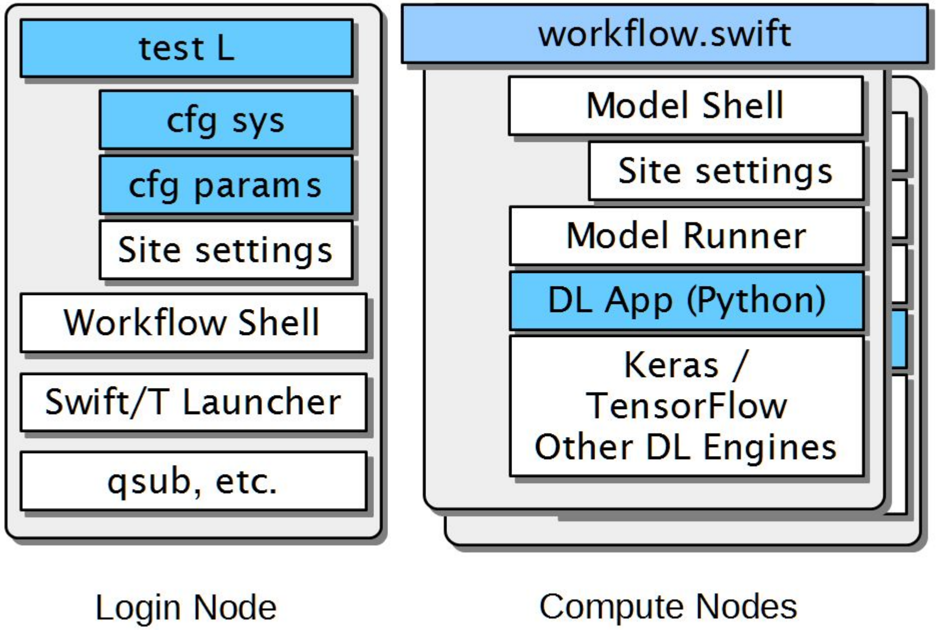
- Hyperparameter search plot:
- Search trajectory of mlrMBO (R model-based optimization) algorithm
- Each iteration does 300 evaluations (batch size)
- Minimum and average performance on validation data set decreases as the ME algorithm learns

CANDLE/Supervisor: A workflow framework for machine learning applied to cancer research. BMC Bioinform. 2018.

CANDLE/Supervisor Implementation

Script schematic

- Runs start with a test script
- CFG scripts contain settings for a system or parameters for a
  given study (e.g., search space)
- Reusable site settings
- The workflow shell script sets up the run
- Swift/T launches and manages the workflow
- Reusable Model scripts set up each app run
- The DL app uses Keras/TF plus CANDLE Python utilities



Exercises

https://tinyurl.com/exaworks

How to run the tutorial exercises

- In the provided instance,

1. `source ~/tutorial/2-workflow-dl-swift.env`
2. `cd tutorial/2-workflow-dl-swift/2-workflow-dl-swift`
3. `git pull`

-

Installation

`http://swift-lang.github.io/swift-t/guide.html#install_source`

New MPI_LAUNCH feature

- Allows Swift/T to run external parallel programs on subcommunicators inside a large allocation on a big machine

- Swift/T syntax:

```
@par=8 launch("./my-program", args[], envs[]);
```

- Provides:
  - Scalable, in-place job launch
  - Handles cases where called program crashes
  - Can pack many such variably-sized programs within a large workflow

Newer MPI_LAUNCH_MULTI feature

- Allows Swift/T to run external parallel program groups on subcommunicators inside a large allocation on a big machine
  - Call these groups Functional Online Bundles (FOBs)
- Swift/T syntax:

```
@par=8 launch_multi(procs[], programs[], args[][], envs[][],
                                 <colors>);
```

- Provides:
  - Scalable, in-place simultaneous job launch
  - The programs are able to find each other and communicate with ADIOS (or other techniques)
  - Job layout can be controlled with the optional colors argument
  - A variety of other controls are available via special environment variables

Swift/T example

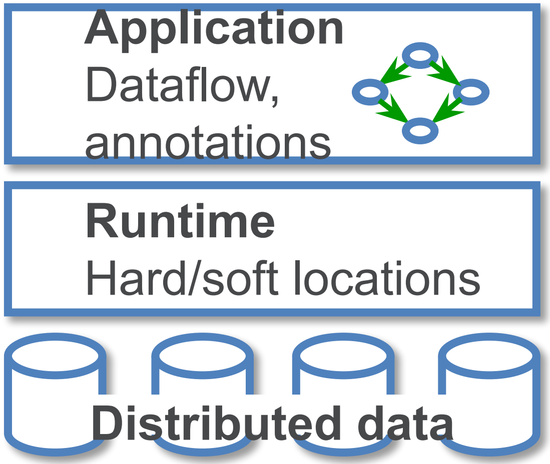
When file `A` is created, launch `N` sub jobs of varying size

```
file B[]; // Define an array of file variables
A => {
  foreach i in [0:N-1] {
    file B_i<"B-%i.txt"%i>;
    string args_B[] = [ int2string(i),
                        filename(A), filename(B_i) ];
    @par=i launch("./child.x", args_B) => B_i = touch();
    B[i] = B_i;
  }}
```

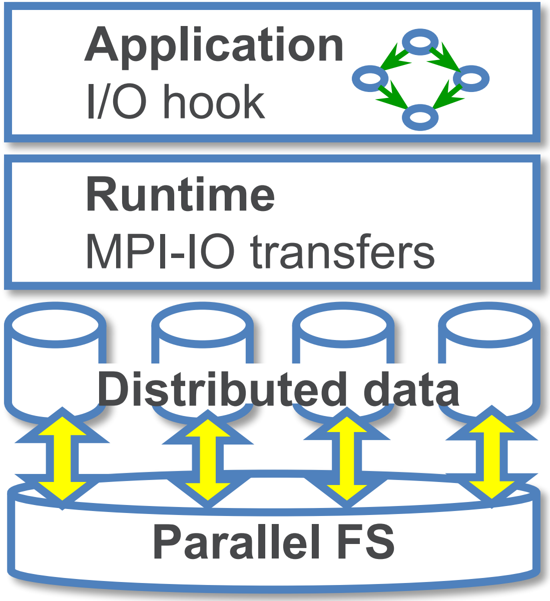
- Child tasks are load-balanced, `MPI_Comm_create_group()` is done automatically!

Features for Big Data Analysis

- **Location-aware scheduling**
  User and runtime coordinate data/task locations



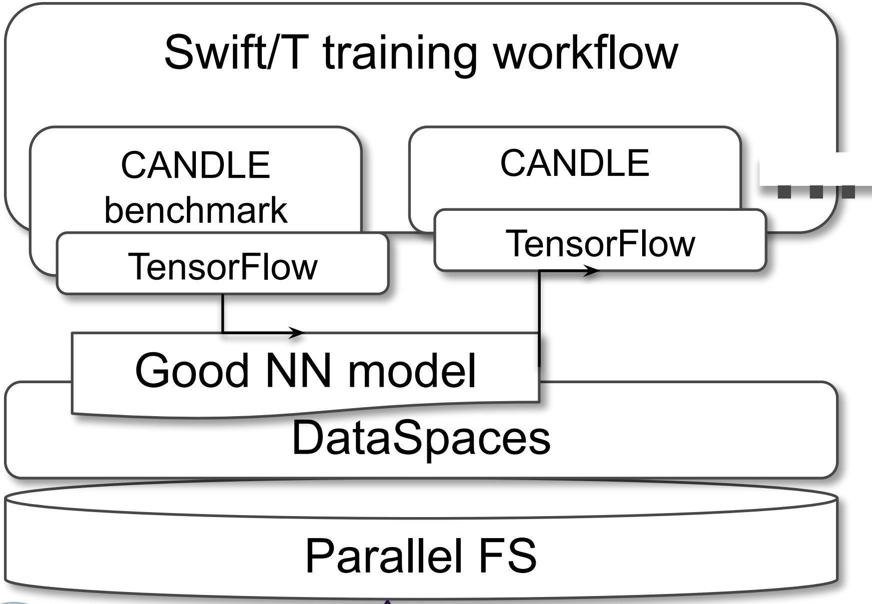
- **Collective I/O**
  User and runtime coordinate data/task locations



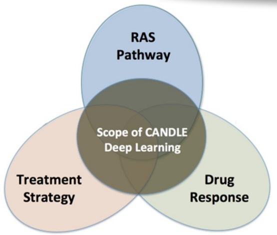
- **Big data staging with MPI-IO for interactive X-ray science.** Wozniak et al. Proc. Big Data Computing 2014.
- **Experimental evaluation of a flexible I/O architecture for accelerating workflow engines in ultrascale environments.** F. Duro, Wozniak, et al. Parallel Computing 61, 2017.

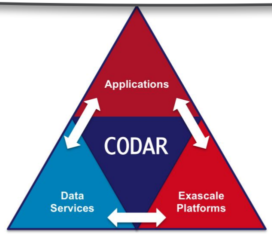
ECP INTERACTION: CODAR, CANDLE

- **CANDLE** workflows produce a great number of medium-sized ML models
- **Goal:** Cache these on compute node storage for *possible* later use. Need to flush to global FS before end of run, but many models will be discarded
- **Approach:** Integrated Swift/T workflow system used in CANDLE with DataSpaces client
- Provide an opportunity for workflow-based data analysis and I/O reduction
- Demonstrate the utility of node-local storage for complex workflows



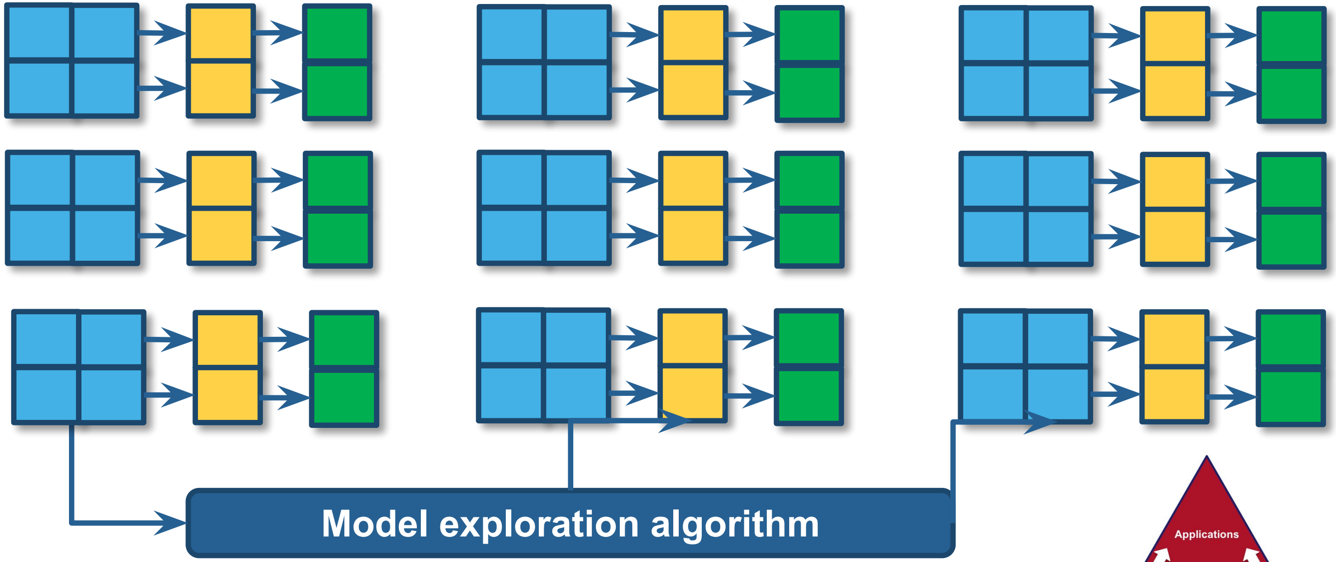
- **Scaling deep learning for cancer with advanced workflow storage integration.** Proc. MLHPC @ SC 2018.





ECP CODAR: Workflows of ADIOS transfers

- Enable Swift to dynamically lay out tasks
- Control large simulation/redistribute/analysis ensembles
- Highly flexible, programmable use of MPI subjobs



ADIOS analysis pipelines

Swift workflow control and load balancing

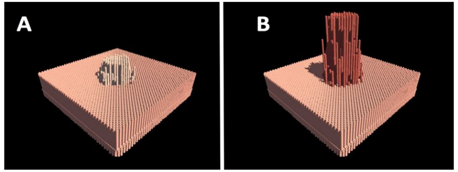
U. Chicago Hospital: Cancer ensembles

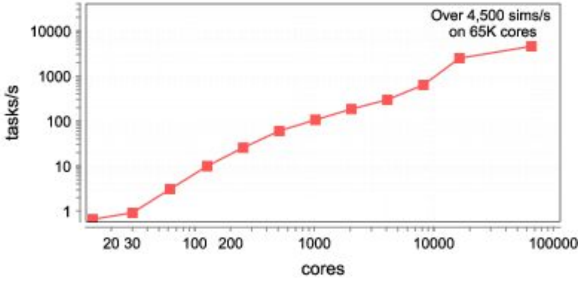
Best paper at SC Cancer Workshop 2016

- Parameter fitting for biological phenomenon (DNA repair rate) via massive scale evolutionary algorithm in Swift/T framework

GIOABM – Integration into SEGMEnT

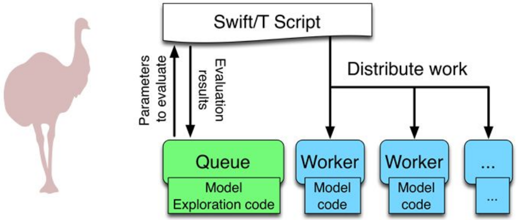
- A cancerous cell has three features: immortality, invasiveness, and ability to proliferate unnaturally
- GIOABM call functionality overlaps with SEGMEnT at four locations:
  - B-catenin: proliferation
  - PI3K: Proliferation/Apoptosis
  - TGF-B/SMAD: Proliferation/Apoptosis
  - P53: Gene repair/Apoptosis
- Added E-cadherin protein mutation to SEGMEnT representing invasiveness





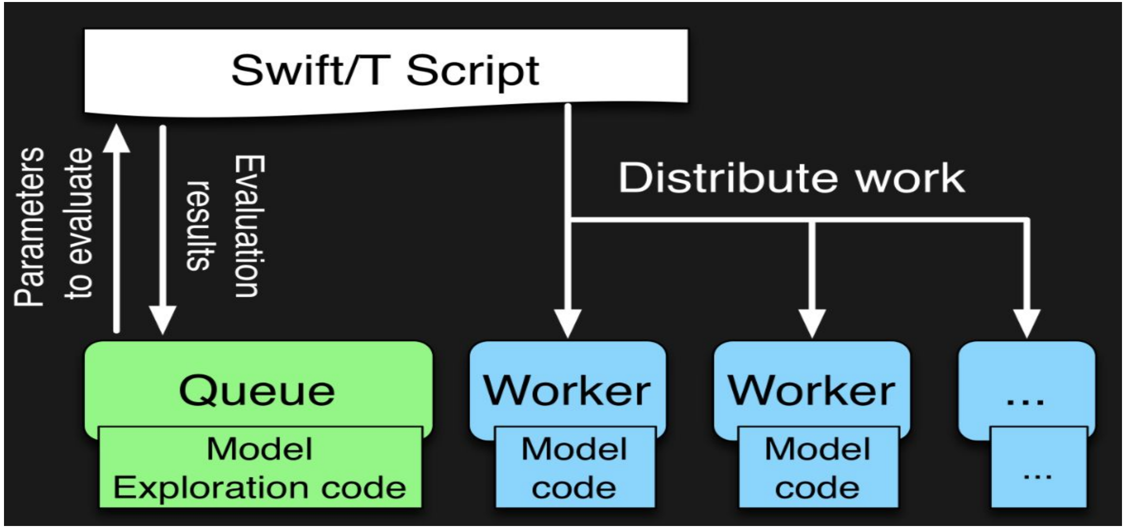
Extreme-scale Model Exploration with Swift (EMEWS)

- EMEWS offers:
  – the capability to run very large, highly concurrent ensembles of simulations of varying types
  – supports a wide class of ME algorithms, including those increasingly available to the community via Python and R libraries
- EMEWS design goal: to ease software integration while providing scalability to the largest scale (petascale plus) supercomputers, running millions of models

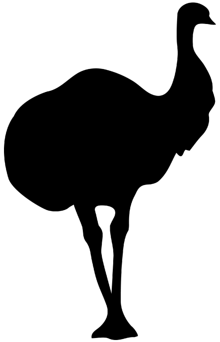


- **Anatomic-scale cancer modeling using the Extreme-scale Model Exploration with Swift (EMEWS) framework.** Proc. Cancer Workshop @ SC, 2016. (Best paper)

EMEWS workflow structure

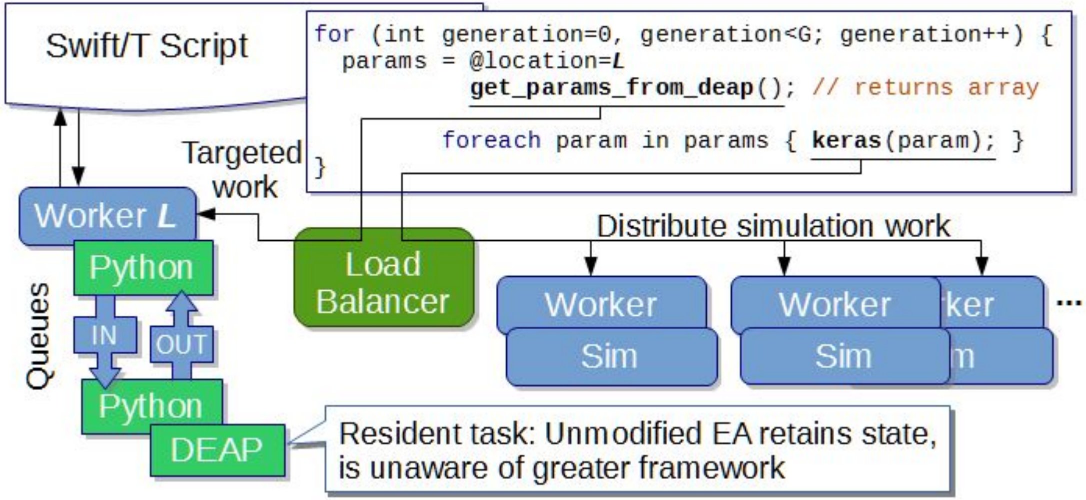


- The core novel contributions of EMEWS are shown in green, these allow the Swift script to access a running **Model Exploration (ME)** algorithm, and create an **inversion of control (IoC)** workflow
- Both green and blue boxes accept **existing multi-language code**
- **http://emews.org**



EMEWS: Extreme-scale model exploration workflows in Swift/T

- How do we couple complex model exploration algorithms with workflows? Optimization, active learning, uncertainty quantification…



- **From desktop to large-scale model exploration with Swift/T**
  Proc. Winter Simulation Conference 2016

Links

- Swift/T Home: http://swift-lang.org/Swift-T
- Swift/T Guide: http://swift-lang.github.io/swift-t/guide.html
- Swift/T Sites Guide: http://swift-lang.github.io/swift-t/sites.html
- Swift/T GitHub: https://github.com/swift-lang/swift-t
- Support: https://groups.google.com/forum/#!forum/swift-t-user
- Book chapter (easiest introduction): http://www.mcs.anl.gov/~wozniak/papers/ProgrammingModels_Swift_2015.pdf
- Other papers: http://swift-lang.github.io/swift-t/pubs.html