An integrated scripting, execution and collaboration environment for power grid simulation and analysis

Mark Hereld and Michael Wilde
Argonne National Laboratory

June 18, 2014
Computational Environment

- A web-based portal for science infrastructure
  - Reliability, usability increasing; approaching readiness for PGrid group to evaluate
  - Ability to add new applications to portal has been refined
- Swift language and execution environment
  - Ability to gather and aggregate an ad-hoc set of resources is being enhanced
  - Extreme scaling enhanced via Swift/Turbine from ExM project (ASCR X-Stack)
  - Ability to call external library functions well-suited for power grid simulation tools
- Evaluation analysis
  - PIPS-L in development and evaluation – showing promising scalability and usability
  - Example visualization scripts developed for MATPower networks
  - AMPL climate/transmission model parameter sweep example placed under Swift
  - Running on Eureka viz cluster attached to BG/P (climate data analysis)
- Annotation and provenance
  - SQL-based provenance queries are approaching usability
  - Name/value annotation/tagging is being integrated into provenance queries
  - Paper on this work is underway
- Use case: build a new application incrementally
  - Defining requirements: local <-> distributed / parallel, publishing, compatible with standard tools
  - Building example tools and interfaces
Swift: easier high-level programming with extreme-scaling for many-task applications

- Implicitly-parallel functional dataflow programming for upper-level application logic
- Drivers: inverse problems, branch-and-bound, stochastic programming, UQ, ensembles
- Enablers: scalable parallel evaluation, dynamic load balancing, in-RAM datasets
- Benefits: programmability, fault-recovery; possibly, power savings
- Results: new scalable Swift implementation, 25K tasks/sec, 128K-core parallel loop scaling; datastore and MTC publications

Under the ExM ASCR X-Stack project we scaled Swift up to tackle many-task applications from petascale to exascale.
Simulation is typically performed by complex, messy scripts

Computational workflows are expressed as complex, unstructured, ad-hoc scripts which manually perform parallel job execution

Many difficult, distracting details to manage

Researchers manually move and process data over diverse and error-prone compute clusters, networks and security environments.

Collaboration and reuse is hindered

Scripts that explicitly deal with these issues are hard to review, validate, disseminate, and reuse. This makes results very hard to obtain – and even harder to replicate!
0: Develop script
1: Run script(EL1.trj)
2: Lookup file
   name=EL1.trj
   user=Mark
   type=TransLines
3: Transfer inputs
4: Run app
5: Transfer results
6: Update catalogs

Simulation Environment Architecture

Grid Researchers

External collaborators

Collaboration Catalogs

Files & Metadata

Provenance

Script libraries

Swift

Compute Facilities

Storage locations

Simulation Environment Architecture

Architecture

Grid

Researchers

External

collaborators

Collaboration

Catalogs

Provenance

Files & Metadata

Script libraries

Swift

Compute Facilities

Storage locations
Simulation Environment Benefits

- Easier to find, write and run reusable, parallel multiscale scripts
- Automates parallel and distributed workflows
- Enables cataloging, sharing and discovery of results and methods
Simulation Environment Benefits

- Easier to find, write and run reusable multiscale scripts
- Automates parallel and distributed workflows
- Enables cataloging, sharing and discovery of results and methods
Simulation Environment Benefits

- Easier to find, write and run reusable multiscale scripts
- Automates parallel and distributed workflows
- Enables cataloging, sharing and discovery of results and methods
Goal: portal for collaboration and dissemination

• Basing work on GPSI: Generic Portal for Science Infrastructure
• Relevance to the Power Grid simulation collaboration
  – Usability
    • Hides the complexity of running on diverse, complex parallel resources
    • Provides instant documentation on how to run specific tools
    • Interfaces for finding, sharing, analyzing and organizing data, tools and results
  – Collaboration
    • Provides a virtually centralized view of widely distributed resources: datasets, application programs and tools,
  – Dissemination
    • Enables “software as a service” – a powerful way to package a diverse set of applications for use and evaluation by a community
What can the Grid Portal do for researchers?

- Simplify access to compute resources
- Organize analysis and simulation data
- Capture project workflow for reuse
- Simplify development of complex workflows
- Tame repetitive work
Portal Design Goals

• Smooth scaling
  – from prototype codes, scripts, and experiments to large scale data and computational applications

• Familiar development tools
  – promote rapid prototyping

• Modularity, reuse, and sharing
  – productivity

• Enable broad community
  – developers, users (lab AND industry), and decision-makers
Data and compute resources at labs, agencies, LCFs, grid industry stakeholders, operators, and commercial cloud resource providers (Data servers managed and federated by Globus Transport, Catalog, and Collaborate)
Metadata and annotation management

• We have explored 2 metadata databases: TagFiler from ISI and ElasticSearch, an open source project
• Developing a database structure that integrates metadata tags, dataset location, and provenance tracking of derived results.
• Example queries:
  – Find all datasets matching:
    • type=transmission.log and state=(MD or VA) and cdate between 1-Jan-2011 and 31-Dec-2012
  – Find all datasets matching:
    • type=PIPS-L.output and version=2.1 and user=cpetra and state=MD and (cdate between 1-Jul-2012 and 1-Aug-2012) and (efficiency > 0.92)
• Queries can be used both interactively and from Swift scripts
• Derived datasets can be tagged as part of the executable workflow
• Metadata tags can be applied to datasets, applications, scripts, and runs.
frames 31 to 99, also sequester ampl data in pseudo-repo
Data Browsing
Data Browsing

Your Applications

ID | Application | Desc
---|-------------|------
70 | sft10APPLICANT | past
69 | bitSwiftWorkers | past
68 | helloWorld | my

Files

- ball.tar
- ed.mod
- imp2mp
- pgp.out.swift
- print_results.inc
- run_alpha864
- set_data.swift
- shift_horizon.inc

Parameters

- Start-Hour (default: )
- EndHour (default: )

Your Jobs

ID | Application | Owner
---|-------------|------
324 | helloWorld | hereid
320 | helloWorld | hereid
321 | helloWorld | hereid

Your Files

ID | Filename | Output
---|---------|------
40256 | il666.mp4 | output
40256 | ch666.mp4 | output
40256 | foo.png | output
7356 | Imp.mp4 | output

CODE

```swift
app ( file result ) foo ( string args )
{ 
  ls -lt args stdout=result;
}
```

GENERAL OUTPUT

```
Jobs version 3.10, running with linear solver.
```

IMAGES

```
Output
```

MOVIES

```
mp4
```
Swift programming model: implicitly parallel functional dataflow

(int r) myproc (int i, int j)
{
    int f = F(i);
    int g = G(j);
    r = f + g;
}

• F() and G() implemented in native code
• 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/T Goal: programmability for extreme scale

- Focus is “many-task” computing: higher-level applications composed of many run-to-completion tasks (input-process-output vs. message passing)
- Why is it relevant to DOE extreme-scale computing?
  - Programmability
    - Increasing number of applications have this natural structure: material by design, inverse problems, stochastic programming, branch-and-bound problems, UQ.
    - Coupling extreme-scale applications to preprocessing, analysis, and visualization
  - Resilience
    - The functional programming model provides a modular hierarchy for re-execution of failing units of an application
  - Power management
    - Graph structure of application upper levels may enable functional units to be quiesced and data transfer “distance” to be reduced
- Challenges
  - Data locality and load balancing!
Centralized evaluation can be a bottleneck

Have this:

Want this:
Solved by fully parallel evaluation of complex scripts

```c
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]);
}
```

(To simplify diagram, array references are not shown for the loops above)
PIPS-L Swift/T Application test

- Swift/T was used in the PIPS-L C++ framework for the scalable solution of stochastic integer problems
- The second phase of PIPS-L was coded in Swift/T to perform the computationally intensive search for feasible and near-optimal decisions in the stochastic unit commitment problem with wind power integration (work by Cosmin Petra and Justin Wozniak)
  - From Cosmin: “One major advantage of Swift is that its high-level programming language considerably shorten coding times, hence it allows more efforts to be dedicated to algorithmic developments. Remarkably, using a scripting language has virtually no impact on the solution times and scaling efficiency, as recent Swift/PIPS-L runs shows.”
PIPS-L Swift/T Application structure

- Swift/T is used with PIPS-L, the solver for stochastic integer problems (such as stochastic unit commitment)
- Phase 1: PIPS-L solves a so-called relaxation problem of the stochastic integer problem.
  - The solution found in this phase is sub-optimal: solution does not give the lowest operating cost of the generators
  - Possibly non-integer: may have a fractional value like 0.46 for a on/off (1/0) decision: a non-implementable policy
- Phase 2: round the solution (with different cutoffs) to obtain candidate integer solutions. Ensure candidates are feasible for each scenario.
  - *rounding-simple.swift* checks the feasibility of a rounded solution, for a given cutoff. It loops over the scenarios (*and to do*: over the cutoffs).
- Phase 3: a branch-and-bound search may be needed for some problems (feasible integers solutions may still be sub-optimal). This is not yet implemented, but hopefully can be done in Swift.
  - have prototyped branch-and-bound algorithms in Swift (PPoPP submission).
The Swift phase has a scale of 400K tasks x 75sec/task
Goal: <5min on full-scale Intrepid BG/P (163K cores)
PIPS-L Swift/T Performance

- 4 tasks/CPU
- ~ 75 secs/task
- Linear scaling measured up to 4K tasks, 1K cores
- Tail dropoff removed from scaling tests (becomes insignificant as scale increases)

Scaling to 4K tasks, 1K cores (4 tasks/core)

Distribution of task durations

Load balancing on 128 CPUS
The number of datasets is realistic from the application perspective. Increasing it over ~4K datasets is unnecessary.

Need to enhance the Swift code to complete the algorithmic side, in particular two nested parallel loops, which will generate a greater amount of parallelism.

The large runs with the new script needs a better candidate solution, for which we need to run the PIPS-L stage-1 code on BG/P (need to generate data for the full model and for large number of scenarios and run PIPS-L on BG/P to obtain the candidate solution).

Need to add some more realistic algorithmic features. With larger subproblems we can show similar scaling up to a larger number of cores.

Need warm-start for the large scale runs (crucial if we want to solve the problem in realtime).

In larger problems we will loop over the cutoffs (now being tested).
PIPS-L Swift Code (initial version)

```swift
1 main () {
2   string data = "/path/to/data";
3   int nScenarios = 4096;
4   blob s = _B_readConvSolution_C_(dataPath,solutionPath);
5   float cutoff = 0.8;
6   blob r = _B_roundSolution_C_(s,cutoff);
7   float v[];
8   foreach i in [0 : nScenarios-1] {
9       v[i] = _B_evaluateRecourseLP_C_(data,nScenarios,i,r);
10   }
11   float result = sum(v);
12   printf("result: %f
", result);
13 }
```
PIPS-L Swift code with cutoffs

```swift
(\text{float result[]}[]) \text{cutoffs} (\text{float step})
{
    \text{for (int i = 0; i < 10; i = i + 1)}
    {
        \text{result}[i] = step*itof(i);
    }
}

\text{main}
{
    \text{string data} = "/home/wozniak/PIPS-data-2/";
    \text{string dataPath} = data + "4h_dump/uc_4h";
    \text{string solutionPath} = data + "primalsol_conv8";
    \text{int nScenarios} = \text{toint} (\text{argv} ("N"));
    \text{blob s} = \text{readConvSolution} (\text{dataPath}, \text{solutionPath});

    \text{foreach cutoff in cutoffs} (0.1)
    {
        \text{blob r} = \text{roundSolution} (s, \text{cutoff});
        \text{float v[]};
        \text{foreach i in} [0 : \text{nScenarios}-1]
        {
            \text{v}[i] = \text{evaluateRecourseLP} (\text{dataPath}, \text{nScenarios}, i, r);
        }
        \text{float result} = \text{sum_float} (v);
    }
}
```
PIPS-L Swift/T branch-and-bound structure

Minimize some function via recursive search, allow only for integer solutions

Creates task parallelism in Swift
Swift Branch-and-bound code

```swift
1 double search(Problem p,
2         double upperBound)
3 {
4     double maximum = 0;
5     Solution s = optimize(p);
6     if (s.feasible) {
7         if (s.solution == Integer) {
8             if (s.objective < upperBound) {
9                 // bound
10                 upperBound = s.objective;
11             }
12         }
13     } else {
14         // branch
15         search(s.left, upperBound);
16         search(s.right, upperBound);
17     }
18 }
19 }
```
Future Plans

• Swift application codes
  – Complete the PIPS-L phase-2 stage in Swift/T; scale up, measure scaling performance.
  – Develop Minotaur branch and bound code for doing in parallel with a Swift main loop.
  – Identify and port additional tool frameworks to Swift

• GPSI portal for dissemination and collaboration of codes and frameworks
  – Continued usability improvement
  – Cataloging of multiple tools and workflows
  – Evaluation and use by team members and program management
  – Ideally, engagement with, and feedback from industry
Summary

• **Swift parallel language** enables integration of simulation frameworks with implicit parallelization and portability between parallel platforms
  – Reduces development work compared to MPI and other explicitly parallel programming models
  – Handles fault tolerance, load balancing, scheduling

• **Grid research portal** enables collaboration through dissemination and evaluation of codes and frameworks
  – Discover and share application programs, frameworks, scripts
  – Catalog and annotate raw and derived data, metadata, provenance