CMSC 714
Lecture 10
Scientific Workflows – Pegasus & ADIOS2

Alan Sussman

Notes

- MPI project due Monday, 7PM
  - Questions on project?
- Readings posted up to spring break
  - Don’t forget to send questions when you are assigned

Scientific Workflows

- Generically, a set of scientific applications that work together to solve a problem
  - A set of computations and their data requirements
- Examples include multi-stage simulation and data analysis pipelines (sometimes with input preprocessing), ensembles of simulations varying input parameters
- Many scientific workflow management systems have been built and used
  - Examples include Pegasus, Taverna, Galaxy, Kepler, Swift, Nimrod, MakeFlow, …
  - For a list of actively-developed open source workflow systems, see https://workflows.community/systems
- Can be run on a wide range of computational and storage resources
  - For computation, sets of local machines, campus clusters, national HPC infrastructure, commercial and academic clouds, …
  - For storage, from local filesystems to shared (parallel) filesystems to cloud provided object storage (e.g., Amazon S3), etc.

Pegasus

Pegasus Overview

- A scientific workflow management system (WMS) developed at USC ISI
  - Goal is to manage a workflow executing on potentially distributed data and compute resources
- Separates the workflow description from the description of the execution environment
  - workflows are based on a Directed Acyclic Graph (DAG) representation of a scientific computation, with tasks to be executed represented as nodes, and the data- and control-flow dependencies between them represented as edges
- Basic model
  - the user has access to a machine where the workflow management system resides
  - the input data can be distributed across diverse storage systems connected by wide area or local area networks
  - the workflow computations can also be performed across distributed heterogeneous platforms

Pegasus components

- Mapper
  - Generates an executable workflow based on an abstract workflow provided by the user or workflow composition system
  - Finds the appropriate software, data, and computational resources required for workflow execution
  - Can also restructure the workflow to optimize performance and add transformations for data management and provenance information generation
- Local Execution Engine
  - Submits the jobs defined by the workflow in order of their dependencies
  - Manages the jobs by tracking their state and determining when jobs are ready to run
  - Submits jobs to the local scheduling queue
- Job Scheduler
  - Manages individual jobs and supervises their execution on local and remote resources
- Remote Execution Engine
  - Manages the execution of one or more tasks, possibly structured as a sub-workflow on one or more remote compute nodes
- Monitoring Component – via runtime monitoring daemon
  - Monitors the running workflow, parses the workflow job and tasks logs and populates them into a workflow database.
  - Database stores both performance and provenance information, and sends notifications back to the user notifying them of events such as failures, success, and completion of tasks, jobs, and workflows

DAGs and Mapping

- DAG resource-independent workflow description (DAX) generated through API calls in Python, Java, Perl
  - Also hierarchical description (and execution) for complex DAGs – a node can represent a whole DAG from a lower level
- Mapping is technically the difficult part
  - Map/plan the DAX abstract workflow onto the execution environment
  - Includes computation invocation on the target resources, the necessary data transfers, and data registration
  - To find the necessary information, queries several catalogs:
    - *Replica Catalog* to look up the locations for the logical files referred to in the workflow
    - *Transformation Catalog* to look up where the various user executables are installed or available as stageable binaries
    - *Site Catalog* that describes the candidate computational and storage resources that a user has access to

Workflow Execution

- After mapping the executable workflow is generated in a way that is specific to the target workflow engine – default is *HTCondor DAGMan*
- Then submitted to the engine and its job scheduler on the submit host – default is *HTCondor Schedd*
- Each node in the DAG is a job in the executable workflow and is associated with a job submit file that describes how each job is to be executed
  - Includes executable that needs to be invoked, the arguments with which it has to be launched, the environment that needs to be set before the execution is started, and the mechanism of how the job is to be submitted to local or remote resources for execution
- When jobs are ready to run (their parent jobs have completed) DAGMan releases jobs to a local Condor queue that is managed by the Schedd daemon
  - By default, jobs run in the local HTCondor pool, but can also be submitted with several remote job management protocols, including ssh, SLURM, and other resource managers

More Pegasus Features

- Mapper performs transformations to improve the overall workflow performance, both at “compile” and ”runtime”
  - Reuse data (files) if they already have been computed and are available
  - *Site Selector* maps the jobs in the optimized workflow to the candidate execution sites specified by the user
  - Tasks can be clustered to deal with large numbers of short running tasks, to minimize overhead
  - Mapper accesses input data (files) through Replica Catalog to find and stage data (add nodes in DAG), and adds data stageout nodes to stage intermediate and output datasets to user specified storage locations
- For reliability (dealing with failures) jobs can be retried, file transfers also retried, only parts of the workflow that have not completed are retried if the entire workflow fails, and the retried workflow can be replanned (new mapping)

CyberShake Application

- Large scale earthquake modeling application from Southern California Earthquake Center (SCEC)
- Over 6 studies in 5 years, Pegasus executed
  - hundreds of millions of tasks on 6 different HPC resources using millions of core hours, taking thousands of hours to execute (makespan)
  - producing hundreds of millions of files, totaling tens of TB of data
- For more info on Pegasus, see https://pegasus.isi.edu/

ADIOS2
(with thanks to N. Podhorszki at ORNL)

ADIOS2 Overview

ADIOS brings a programming interface and a framework of many solutions to the generic problem of producing and consuming data in scientific applications

- The interface frees scientists from the limited scope of file-based data processing
  - Still fully applicable to file-based data processing
- Scalable IO: number of processes, variables and steps; and amount of data
- Offers a bridge from scientific workflows that work now to the future, where scientists will extend their workflows with
  - More efficient data processing
  - Interactive visualization
  - Code coupling
  - On-the-fly AI training
  - Combining experimental data with simulation data

A high-performance publisher/subscriber I/O framework

Vision

- Create an easy-to-use, high performance I/O abstraction to allow for on-line/off-line memory/file data subscription service
- Create a sustainable solution to work with multi-tier storage and memory systems for self-describing data-streams

Details

- Declarative, publish/subscribe API separated from the I/O strategy
- Multiple implementations (engines) provide functionality and performance
- Rigorous testing ensures portability
- GPU-aware to reduce data movement

ADIOS Basic Concepts

- Step
  - Producer outputs a set of variables and attributes at once
    - This is an ADIOS Step
  - Producer iterates over computation and output steps
- Producer outputs multiple steps of data
  - e.g., into multiple, separate files, or into a single file
  - e.g., steps are transferred over network
- Consumer processes step(s) of data
- e.g., one by one, as they arrive
- e.g., all at once, reading everything from a file
  - post-processing only, not able to process in situ this way

Step is a Transaction between producer and its consumers

Data Staging Options in ADIOS

Transfer mechanisms

- File based (BP4, BP5)
- Network based on the same resource (SST, SSC)
- RDMA (libfabric, UCX)
- MPI (one sided, two sided)
- TCP/ RUDP
- Memory references
- WAN data transfer (DataMan,SST)
  - Streams – TCP, RUDP, RoCE

Placement options

- Same core (inline code)
- Different cores/same node
- Different nodes
- Different resource (LAN)
- Different resource (WAN)
- Hybrid (mixture of options)

Scheduling options

- Fully synchronous
- Fully asynchronous
- Hybrid

Refactoring options

- Prioritize which data gets
- moved first

Storage options

- ADIOS-BP5
- HDF5

ADIOS Engines

Table 2
ADIOS 2 Engines and areas of application

Application	Engine	Application Areas
File	BPFile	Checkpoint/restart, analysis data, zero-copy buffer, file-based streaming, code coupling
	HDF5	HDF5 compliant files
Data staging	SST	Interprocess communication
	SSC	Interprocess communication
	InSituMPI	On-node interprocess communication
	DataMan	Peer-to-peer TCP/IP Wide-Area-Networks (WAN)

ADIOS Useful Information and Common tools

- ADIOS tutorial: https://tinyurl.com/adios-sc2023
- ADIOS documentation: https://adios2.readthedocs.io/en/latest/index.html
- ADIOS source code: https://github.com/ornladios/ADIOS2
  - Written in C++, wrappers for Fortran, Python, Matlab, C
  - Contains command-line utilities (bpls, adios2_reorganize ..)
  - Examples in C++, Fortran and Python
- Online help:
  - ADIOS2 GitHub Issues: https://github.com/ornladios/ADIOS2/issues