5/30/2014

- SAGA
  - standardized API for developing distributed applications that can run on grid and cloud infrastructure.
  - A light-weight access layer for distributed computing infrastructure.
  - Emphasis on job handling, monitoring, file transfer/management
  - Goals:
    - Uniform access to distributed computing infrastructures and middleware
    - Stable programming interface for distributed applications/frameworks/tool development
    - Ease of use
    - Simple user-space deployment in heterogeneous distributed computing environments
  - Implements flexible adaptor architecture.
  - Adaptor: dynamically loadable modules that interface the API with different middleware systems and services.
- Job Submission Systems: SSH and GSISSH, PBS and Torque, Sun Grid Engine
- File/Data Management: HTTP/HTTPS SFTP/GSIFTP
- Resource Management/Clouds: EC2(libcloud)
- PanDA Project / Brookhaven National Laboratory
  - BNL uses RADICAL-SAGA to extend PanDA (the workload management system for the ATLAS project) to US HPC resources.
  - It is an advanced scheduling and analysis tool
  - Short for Production and Distributed Analysis.
  - Manages ATLAS's data tasks from CERN server.
  - Workflow is 1.8 million computing jobs/day distributed among about 100 computing centers worldwide.

  - Manages 150 petabytes of data (about 75 million hours of HD video).
  - Scheduling is an issue. Not working at full capacity if you're not using every node in your super-computing system.
  - Grid System
    * Uses a Tier system. Data is passed from tier to tier for analysis
    * Tier-0: LHC @ CERN. Gets the raw data, then passes to Tier-1
    * Tier-1: ten locations which recieve the data from Tier-0. BNL is in Tier-1. Connected directly to Tier-0 using a dedicated high-performance optical network path.
    * Tier-2: Computing facilities which provide data storage and processing capacities for in-depth user analysis and simulation. Ex.: other facilities in the US. I think RADICAL is here.
  - Grid Infrastructure
    * Consists of three key components: "fabric", "applications", "middleware"
    * Fabric: Hardware. Processor farms with thousands of computing nodes, disk and tape storage, networking
    * Applications: Software that scientists use to interpret the data
    * Middleware: links the Fabric and the Applications. I believe SAGA fits here.

- Physics
  - Field associated with Higgs necessary for other particles to have mass
  - Higgs boson is very massive and decays almost instantly.
  - LHC creates 800 million collisison between protons per second, yet it only creates a Higgs boson only once every 1-2 hours.

6/22/14

- Distributed Computing
  - Distributed system: network of autonomous computers that communicate with each other in order to achieve a goal.
  - Computers in the distributed system are independent and do not physically share memory or processes
  - Communite through messages.
- Client/Server Systems
  - Single server that provides a service, and multiple clients that communicate with the server to consumer its products.
  - valing=t [width=.8]img/clientServer.jpg
  - Client does not need to know how the service is provided or how the data is calculated.
  - Server does not need to know how the data is going to be used.
  - **Drawback:** If the server goes down, the entire system stops working.
  - **Drawback:** Resources become scarce if there are too many clients. Cliients increase the demand on the system without contributing any computing resources
  - **Drawback:** Client-server systems cannot shrink and grow with changing demand.
- Peer-to-Peer Systems
  - Labor is divided among all components of the system
  - All the computers send and receive data, and they all contribute some processing power and memory.
  - Peers need to communicate with each other reliably. Need organized network structure.
  - Data transfer and storage are most common applications.
  - Data transfer: each computer contributes to send data over the network. If the destination computer is in a particular computer's neighborhood, that computer helps send data along.

  - Data storage: data set may be too large to store on one computer. Portions are stored on each computer in the system.

- Modularity
  - Components of a system should be black boxes with respect to each other.
  - Easy to understand, change, and expand.
  - Defective components can be easily swapped out
  - Bugs/malfunctions are easy to localize

- Message Passing
  - Message has three parts: **sender**, **recipient**, **content**.
  - Sender and receiver must be explicitly encoded in the message.
  - Message content can be complex data structures, but they are all sent as 1s and 0s.
  - Message protocol: set of rules for encoding and decoding messages.
  - All components in distributed system must understand the protocol in order to communicate with each other.

- Correctness in Parallel Computation
  - Two criteria: Outcome should always be the same. Outcome should be the same as if the code was executed sequentially.
  - Critical section: sections of code that need to be executed as if they were a single instruction, but are actually made up of smaller statements.
  - Atomicity: quality that describes instructions that cannot be broken into smaller units or interrupted because of the design of the processor.
  - Serialization: processes temporarily act as if they were being executed in serial.
  - Synchronization: uses mutual exclusion and conditional synchronization
    * Mutual exclusion: processes taking turns to access a variable

    - Conditional synchronization: processes wait until a condition is satisfied before continuing.

- Protecting Shared State: Locks and Semaphores
  - Locks/Mutexes: shared objects that are commonly used to signal that shared state is being read or modified.
  - Python uses **acquire()** and **release()**
  - When a lock is acquired by a process, any other process that tries to perform the **acquire()** action will automatically be made to wait until the lock becomes free.
  - Semaphores: signals used to protect access to limited resources. Only a certain number of processes are allowed to access the shared data.
  - Condition variables:
    - Objects that act as signals that a condition has been satisfied.
    - Processes that need a condition to be satisfied can make themselves wait on a condition variable until some other process modifies it to tell them to proceed.
    - Python uses **condition.wait()** to wait on a CV.
    - **condition.notify()** wakes up one process, and **condition.notifyAll()** wakes up all waiting processes.

- Deadlock
  - Two or more processes are stuck, waiting for each other to finish.
  - **Circular Wait:** No process can continue because it is waiting for other processes that are waiting for it to complete.
  - **No preemption:** One process cannot just yank a shared variable from another process that is using it
  - **Hold and wait:**
  - **Mutual Exclusion:**
  - How to prevent deadlock: lock the mutexes in the same order.

References

[1] Williams, Leo. "World's Most Powerful Accelerator Comes to Titan with a High-Tech Scheduler." *World's Most Powerful Accelerator Comes to Titan with a High-Tech Scheduler*. Brookhaven National Laboratory, 7 May 2014. Web. 31 May 2014.

[2] "Computing Support for ATLAS." *Computing*. Brookhaven National Laboratory, n.d. Web. 31 May 2014.

[3] "Chapter 4: Distributed and Parallel Computing." *Chapter 4: Distributed and Prallel Computing.". University of California, Berkeley, n.d. Web. 22 June 2014.*