Integrating HPC into the ATLAS Distributed Computing environment

Doug Benjamin
Duke University

HPC Boundary conditions

- There are many scientific HPC machines across the US and the world.
  - Need to design system that is general enough to work on many different machines
- Each machine is independent of other
  - The "grid" side of the equation must aggregate the information
- There several different machine architectures
  - ATLAS jobs will not run unchanged on many of the machines
  - Need to compile programs for each HPC machine
  - Memory per node (each with multiple cores) varies from machine to machine
- The computational nodes typically do not have connectivity to the Internet
  - connectivity is through login node/edge machine
  - Pilot jobs typically can not run directly on the computational nodes
  - TCP/IP stack missing on computational nodes

Introduction

- My take on comparison between HPC and HTC(grid)

HTC – goes fast and steady

Goes really fast
Similar but different

Additional HPC issues

- Each HPC machine has its own job management system
- Each HPC machine has its own identity management system
- Login/Interactive nodes have mechanisms for fetching information and data files
- HPC computational nodes are typically MPI
- Can get a large number of nodes
- The latency between job submission and completion can be variable. (Many other users)

Work Flow

- Some ATLAS simulation jobs can be broken up into 3 components

  (Tom LeCompte's talked about this in greater detail)

1. Preparatory phase - Make the job ready for HPC
   - For example - generate computational grid for Alpgen
   - Fetch Database files for Simulation
   - Transfer input files to HPC system
2. Computational phase – can be done on HPC
   - Generate events
   - Simulate events
3. Post Computational phase (Cleanup)
   - Collect output files (log files, data files) from HPC jobs
   - Verify output
   - Unweight (if needed) and merge files

HTC->HPC->HTC

- ATLAS job management system (PANDA) need not run on HPC system
  - This represents a simplification
  - Nordu-grid has been running this way for a while
- Panda requires pilot jobs
- Autopy factory is used to submit Panda pilots
- Direct submission of pilots to a condor queue works well.
  - Many cloud sites use this mechanism – straight forward to use
- HPC portion should be coupled but independent of HTC work flow.
  - Use messaging system to send messages between the domains
  - Use grid tools to move files between HTC and HPC

HTC ("Grid side") Infrastructure

- APF Pilot factory to submit pilots
- Panda queue – currently testing an ANALY QUEUE
- Local batch system
- Web server to provide steering XML files to HPC domain
- Message Broker system to exchange information between Grid Domain and HPC domain
- Gridftp server to transfer files between HTC domain and HPC domain.
  - Globus Online might be a good solution here (what are the costs?)
- ATLAS DDM Site - SRM and Gridftp server(s).

HPC code stack

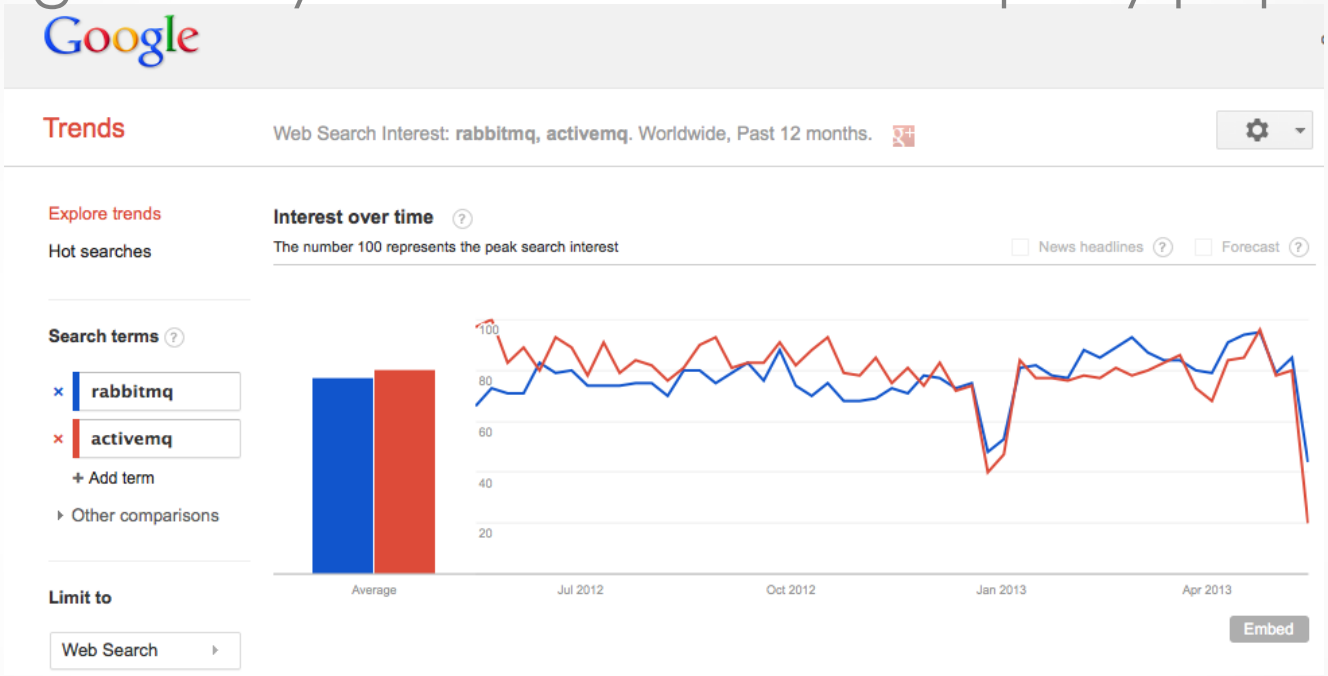
- Work done by Tom Uram - ANL
- Work on HPC side is performed by two components
  - **Service**: Interacts with message broker to retrieve job descriptions, saves jobs in local database, notifies message broker of job state changes
  - **Daemon**: Stages input data from HTC GridFTP server, submits job to queue, monitors progress of job, and stages output data to HTC GridFTP server
- Service and Daemon are built in Python, using the Django Object Relational Mapper (ORM) to communicate with the shared underlying database
  - Django is a stable, open-source project with an active community
  - Django supports several database backends
- Current implementation relies on GridFTP for data transfer and the ALCF Cobalt scheduler
- Modular design enables future extension to alternative data transfer mechanisms and schedulers

Message Broker system

- System must have large community support beyond just HEP
- Solution must be open source (Keeps Costs manageable)
- Message Broker system must have good documentation
- Scalable
- Robust
- Secure
- Easy to use
- Must use a standard protocol (AMQP 0-9-1 for example)
- Clients in multiple languages (like JAVA/Python)

RabbitMQ message broker

- ActiveMQ and RabbitMQ evaluated.
- Google analytics shows both are equally popular



- Bench mark measurements show that RabbitMQ server out performs ActiveMQ
- Found it easier to handle message routing and our work flow
- Pika python client easy to use.

Basic Message Broker design

- Each HPC has multiple permanent durable queues.
  - One queue per activity on HPC
  - Grid jobs send messages to HPC machines through these queues
  - Each HPC will consume messages from these queues
  - Routing string is used to direct message to the proper place
- Each Grid Job will have multiple durable queues
  - One queue per activity (Step in process)
  - Grid job creates the queues before sending any message to HPC queues
  - On completion of grid job job queues are removed
  - Each HPC cluster publishes message to these queues through an exchange
  - Routing string is used to direct message to the proper place
  - Grid jobs will consume messages the messages only on its queues.
- Grid domains and HPC domains have independent polling loops
- Message producer and Client code needs to be tweaked for additional robustness

Open issues for a production system

- Need a federated Identity management
  - Grid identify system is not used in HPC domain
  - Need to strictly regulate who can run on HPC machines
- Security-Security (need I say more)
- What is the proper scale for the Front-End grid cluster?
  - Now many nodes are needed?
  - How much data needs to be merged?
- Panda system must be able to handle large latencies.
  - Could expect jobs to wait a week before running
  - Could be flooded with output once the jobs run.
- Production task system should let HTC-HPC system have flexibility to decide how to arrange the task.
  - HPC scheduling decisions might require different Task geometry to get the work through in an expedient manner

Conclusions

- Many ATLAS MC jobs can be divided into a Grid (HTC) component and a HPC component
- Have demonstrated that using existing ATLAS tools that we can design and build a system to send jobs from grid to HPC and back to Grid
- Modular design of all components makes it easier to add new HPC sites and clone the HTC side if needed for scaling reasons.
- Lessons learned from Nordugrid Panda integration will be helpful
- A lightweight yet powerful system is being developed.