Neptune

A Domain Specific Language for Deploying HPC Software on Cloud Platforms

Chris Bunch  Navraj Chohan  Chandra Krintz  Khawaja Shams

ScienceCloud 2011 @ San Jose, CA  June 8, 2011
Cloud Computing

- Three tiers of abstraction:
  - Infrastructure: Scalable hardware
  - Platform: Scalable APIs
  - Software: Scalable application
HPC in the Cloud

- Hard to automatically configure and deploy libraries
- Especially for the average scientist
- Even harder for those w/o grid experience
- Hard to get high performance on opaque cloud
- Wide range of APIs for similar services (e.g., compute, storage)
Introducing Neptune

- A Domain Specific Language that facilitates HPC configuration and deployment
- Serves High Throughput Computing (HTC) as well as Many Task Computing (MTC)
- Part language component, part runtime component
Design

• Language - Consists of metaprogramming extensions to Ruby
• Runtime - Operates at cloud platform layer
• Can control VMs as well as applications
• Can act standalone or as a library to existing Ruby programs
neptune :type => “service-type”,
:option1 => “setting-1”,
:option2 => “setting-2”
Semantics

• Options are unique per job
• Not all jobs are equal
• Uses Ruby’s dynamic typing so option and setting types can be dynamic
• Extensible - add your own job types / options / settings
Semantics

• Neptune jobs return a hashmap
• :result is either :success or :failure
• Additional parameters are job-specific
• Customizable per job
Design

• Need something more than XML
• Need to control the execution
• Allows for integration with Rails
• Code is reusable across job types
  • e.g., job input / output / ACLs
To Run a Job:

- Minimum set of requirements for jobs:
  - Acquire resources
  - Run job
  - Get output
- Leverage AppScale cloud platform
AppScale

• Open source implementation of the Google App Engine APIs

• Can deploy to Amazon EC2 or Eucalyptus

• Standard three tier deployment model:
  • Load balancer ↔ app server ↔ database
AppScale Tools

- Modified the command line tools
- Added support for placement strategies
  - e.g., hot spares for HPC computation
- Added hybrid cloud support
  - Can run in Amazon EC2 and Eucalyptus simultaneously
AppController

• A special daemon on all systems that configures necessary services
• Now acts as part of Neptune’s runtime component
• Can receive Neptune jobs
• Acquires nodes from infrastructure, configures, and deploys as needed
Virtual Machine Reuse

• Machines are charged by the hour - so keep them for an hour in case users need them later

• Scheduling policies - hill climbing algorithm used based on execution time or total cost

• Relies on user to specify how many nodes the code can run over
• Many implementations exist - we use a commonly used version (C/C++)

• Data is shared via NFS

• User specifies how many nodes are needed, and processors are to be used, and where their code is located
An Example

neptune :type => :mpi,
:nodes_to_use => 64,
:code => “/code/NQueens”,
:output => “/results/nqueens.txt”,
:storage => “s3”
• IBM’s effort at simplifying parallel computing
• Java-like syntax, no pointers needed
• Can use Java backend or MPI backend via C++
• Thus, same parameters as MPI jobs
• Popularized by Google in 2004

• Neptune supports both ‘flavors’:
  
  • Regular: Write Java Mapper and Reducer, specify a single JAR for execution and main class
  
  • Streaming: Write code in any language, specify location of Map and Reduce files
Stochastic Simulation Algorithms

- Computational scientists do large numbers of Monte Carlo simulations
- Embarassingly parallel
- Two types of simulations (DFSP and dwSSA) supported by Neptune
- User specifies how many nodes and how many simulations to run
An Example

path = "'/racelab/dfsp-run-#{rand}""

neptune :type => "dfsp",
    :output => path,
    :simulations => 1_000_000

result = neptune :type => "output",
    :output => path

puts "DFSP result is #{result[:stdout]}"
Storage Backends

• Can store to any database AppScale uses

- Cassandra
- HBase
- Hypertable

• Can also store to Amazon S3

• Or anything that uses the same APIs

• e.g., Eucalyptus Walrus, Google Storage
Not Just for HPC

• Can be used to scale AppScale itself

• Specify which component to add and how many

• Can specify resources across clouds

• e.g., ten database nodes, five in each of two clouds
Limitations

- Not amenable to codes that require hard-coded IP addresses or other identifiers
- Not amenable for codes that require closed source software to run
- Is ok if the software is only needed for compilation or linking
Evaluation

• Physical hardware:
  • Intel Xeon, 8 cores, 16GB of memory

• Virtual machines:
  • 1 virtual core, 1GB of memory

• MapReduce Java WordCount: 2.5GB input file containing Shakespeare 500 times
Java WordCount

Running Time (seconds)

Number of Nodes
<table>
<thead>
<tr>
<th>Type of Job</th>
<th>Cost with VM Reuse</th>
<th>Cost without VM Reuse</th>
</tr>
</thead>
<tbody>
<tr>
<td>NQueens(MPI)</td>
<td>$12.92</td>
<td>$64.60</td>
</tr>
<tr>
<td>NQueens(X10)</td>
<td>$13.01</td>
<td>$64.60</td>
</tr>
<tr>
<td>MapReduce</td>
<td>$13.01</td>
<td>$64.18</td>
</tr>
<tr>
<td>DFSP</td>
<td>$35.70</td>
<td>$78.63</td>
</tr>
<tr>
<td>dwSSA</td>
<td>$12.84</td>
<td>$64.18</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>$87.48</strong></td>
<td><strong>$336.19</strong></td>
</tr>
</tbody>
</table>
Wrapping it Up

• Thanks to the AppScale team, especially co-lead Navraj Chohan and advisor Chandra Krintz

• Currently Neptune is at version 0.1.1, with added support for UPC, Erlang, Go and R

• gem install neptune

• Visit us at http://neptune-lang.org