Platform Computing

Perhaad Mistry
(pmistry@ece.neu.edu)
Topics

• Cluster, Grid Computing

• Platform Utilities
  − Platform OCS
  − Platform Symphony & Symphony DE
  − Platform EGO

• Overview of using Platform's Tools

• Infrastructure as of now

• Pointers to useful docs
Open Cluster Stack (OCS)

- Used to build a Beowulf style cluster
- Platform OCS is a integrated, vendor certified, software stack that simplifies building of clusters.
- Applications within academic High Performance Computing domain. Google “Teracluster”.
- Really simple to build and use if you stick to documentation and don't try to improvise.
- Different approach, algorithms to develop applications - MPI, shared memory, synchronization & so on.
Architecture of a Cluster

Platform OCS requires a Front end node with at least two network interfaces for the cluster. Ethernet 1 (Eth1) is connected to the public network and Ethernet 0 (Eth0) is connected to the cluster private network.

All compute nodes are on the same management network.

Compute nodes require at least one ethernet interface connected to the network switch.
Grid v/s Cluster

- Grid Computing
  - Loosely coupled
    (Decentralization)
  - Diversity and Dynamism
  - Distributed Job Management & scheduling
  - Eg: Teragrid

- Cluster computing
  - Tightly coupled systems
  - Single system image
  - Centralized Job management & scheduling
  - Eg: Teracluster
Symphony SOA

• Distributed computing framework – Grid Like
• Embodied the best elements of Web Services
  − Design principles making it open and cross platform,
  − Adaptable to existing applications, low latency
  − Implementation and Language Independent
• Used to implement Client Service Architecture
• Test Utilities and Hello Grid Application available.
• Application logic to be slightly altered – unrolling, so on.
Symphony Characteristics

• Components
  - Service – Application to perform tasks sent by clients
  - Client – Application that sends tasks to service
  - Session – Group of tasks
  - Task – Atomic unit of work
Symphony Architecture

Figure 1: SOAM – An alternative way to build and run high-performance applications
Symphony Architecture

![Diagram showing Symphony Architecture](image)

**Figure 3:** The Platform Symphony Developers Edition
Adapting to Symphony

- Client View
  - Connect to SOAM for a service application
  - Create a session
  - Send tasks and task input data to be processed
  - Retrieve task output data
  - Close session & connection

- Service View
  - Service Container object for the service
  - Different functions launched on events
  - onInvoke() method for the service to perform a task by processing the task input.
  - Returns task output
  - Available Interrupt API
Symphony Client Lifecycle

1. initialize()
2. connect()
4. createSession()
6. sendTaskInput(myMessage)
12. fetchTaskOutput()
15. client iterates
16. uninitialize()

SoamFactory

Connection

Session

OutputStream

myMessage (Message)

EnumItems

Symphony

TaskInputHandle

1. creates
3. creates
5. creates
7. creates
8. onSerialize()
9. writes
10. Sends byte-array representation of myMessage
11. creates
13. sends output message
14. creates
- SoamFactory::initialize()

  - ConnectionPtr conPtr = SoamFactory::connect(appName, &securityCB);

  - SessionCreationAttributes attributes; attributes.setSessionName("mySession");

- Create Message: char hello[] = "Hello Grid !!";

  - MyMessage inMsg(taskCount, true, hello);

- Task attributes

  - TaskSubmissionAttributes attrTask; attrTask.setTaskInput(&inMsg);

- Send & Recieve Task

  - TaskInputHandlePtr input = sesPtr->sendTaskInput(attrTask);

  - EnumItemsPtr enumOutput = sesPtr->fetchTaskOutput(tasksToSend);
Defining a Client

• Group tasks to define a session

• Main thing to take from sampleApp:
  - What is Common(setup etc) and what is Task Specific
  - Task Specific is only message sending

• Task output information using a new mymessage object

• Available catch exception method.

• Uninitialize SoamFactory ALWAYS.
Defining Services

- Any executable accessible to Symphony can be a service.
- As per doc: Services are virtualized. So, a service should not read from stdin or write to stdout.
- Services can, read /write files accessible to compute hosts.
- Reconnecting to a client is possible – See example.
- Need to look for a task identifier function call within API, like a task id.
- Service Container invoked once using “run”, in main().
Service Life-cycle
Defining Services - Code

- Service Instance – Inherit from ServiceContainer class.
- Functions defined by granularity
  - void onCreateService(ServiceContextPtr& serviceContext)
  - void onSessionEnter(SessionContextPtr& sessionContext)
  - onInvoke(TaskContextPtr& taskContext)
- Obtain data as sent by client is similar fashion
  - taskContext->populateTaskInput(inMsg);
  - outMsg.setString(str.c_str());
  - taskContext->setTaskOutput(outMsg);
- Service (the executable) invoked using servicecontainer.
Symphony & Web Services

- Symphony's Web Service Documentation
- SOAP used for communication between the Web Service and the client application.
- WSDL: Syntax associated with invocation of web services.
- WSDL: Defines web service operations.
- Use of UserName-Token based authentication between the Web Service client and the Symphony Web Service
Steps for Web Services

- Client serializes the arguments of the method call into the XML payload of the SOAP message
  - Done as a RPC Model – Parameters wrapped in SOAP body
  - Document style - Parameters to Web Service in an XML doc.

- Send message to the Web Service – WSDL.
  - Wait for a response (or timeout)
  - Deserialize the XML payload in the response message to a local type/structure method call.
Using Apache AXIS

• Apache Axis: implementation of SOAP, it abstracts dealing with SOAP and WSDL

• Creates a proxy (or stub) for your clients to abstract away SOAP.

• Make method calls on the Web Service proxy as if it were a local object.
Enterprise Grid Orchestrator

- Grid infrastructure that provides resource management.
- Allows us to specify policies that can vary by time.
- Integrate 3rd party middleware.
- Provides fail-over at a service level, but recovery has to be written by you.
EGO & Web Services

- A framework for “stateful” web service.
- Needs a module to help a client to use services exclusively, without interference from other clients.
- 3rd party applications run as EGO services.
- Running web service as EGO service means that the web services is integrated into an EGO cluster.
- Life cycle of the web service will be managed and monitored by EGO.
EGO - Contd

- Open, Standards-based SDK and Architecture
  - Platform EGO supports a broad set of standards-based APIs that conform to the Web Services Interoperability (WS-I) and WSDL specification.
How To

- Module named Web Service Locater (WSL for short):
  - To call EGO API to query the information (host’s load information) of the web service instances.
  - Get notifications (such as: an instance is not available) about the web service instances.

- Three methods (WSL) in form of web service interface: FindService(), LockService() and UnlockService()

- Load balance: FindService() Find optimum instances.
- Starting new web service instance on demand, stopping useless instances.
Present Infrastructure

- OCS Implemented on cluster, 7 node-10 CPUs
- Sadly, benchmarks only LINPACK 3GFLOPS.
- Distributed version of Symphony DE is all set
- OCS monitoring tools co-existing with Symphony
- Performance tune symphony – Symping is Brilliant.
- Installing EGO along with Open Cluster Stack being present has been a nightmare.
Open Questions & Pending

• Getting MYSQL involved – Need more information on projects and tools that will be needed, since UNIX administration isn't my forte.

• Need more study of EGO environment and so on, Will try and set-up a separate EGO framework.

• Need to set-up JAVA on system since I don't know how popular is C here.
Conclusions

- We have seen how we can implement a service using symphony, Can do a LOT more if we work with EGO and it's resource allocation capabilities.
- Many bells and whistles like security, multiple clusters etc.
- Advantages of using Symphony are:
  - Low Latency – not really
  - Easy Development
  - Scales well, from a host to a cluster.
What You can do

• Install SymphonyDE on local machine
  - Get Symping and other examples going locally – same as a cluster
  - Will get out information on remote access after I get FQDN.
  - Can demo locally now.
  - Suspect you can make a cluster on wireless, if you have time

• API Documentation – LOTS of reading

• Other tools like MYSQL, AXIS and so on needed?
References

• Platform's Documentation
  - A Developer’s Guide to Building High Performance Service-Oriented Applications

• Useful Docs – From Symphony DE Library
  - Simple Introduction
  - Complete example of financial calculations (needs registration)
  - Platform Symphony – Overview (Includes EGO).
  - Web Services with EGO - Haven't yet understood it.

• Using SOAP with JAVA – RPC Conventions
QUESTIONS???

&

COMMENTS???