Deployment, Heterogeneity and Dynamic Adaptation

- Overall picture
- Core technologies
  - ALDL Application description
  - GEA–based deployment
  - Support of Heterogeneity
  - Support for Dynamic Adaptive behaviour
    - Reconfiguration at run time
    - Repeated Deployment
  - Extension to component models

Multi-language support and application structure

- ASSIST **`astcc`** is a front-end compiler
  - Employs several other compilers as back-ends
  - Run-time support code, and final linker: C++
- Compiler and sub-compiler configuration
  - The **ast_rc** XML file defines paths, flags, compilers, linker to exploit
- Compiled application is a set of executables
  - Application structure is a directory tree
  - Compact form (**.aar** archive)
  - Structure encoded in a flexible XML format

Application Description

ALDL=Application Level Description Language

- Application-level information
  - Structure and parameters (e.g. degree of parallelism)
  - Application executables
  - Run-time support processes
- Process-level information
  - Architecture, OS
  - HW/SW resources: memory, CPU, libraries…
  - Input and output files
- Run-time parameters
  - E.g. TCP ports, or network configuration

All information gathered by the compiler

(simple) ALDL fragment

```
<?xml version="1.0" ?>
<aldl:application xmlns = "urn:aldl-assist" xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance" xmlns:aldl = "http://
      www.isti.cnr.it/schemas/aldl/" xmlns:ns1 = "http://www.isti.cnr.it/schemas/assist/" xmlns:tns = "urn:aldl-assist"
      targetNamespace = "urn:aldl-assist" xsi:schemaLocation = "http://www.isti.cnr.it/schemas/aldl/ xml/aldl.xsd http://
      www.isti.cnr.it/schemas/assist/ xml/assist.xsd">
  <aldl:requirement name = "libraries">
   <ns1:lib fileName = "libACE.so.5" fileSystemName = "/tmp" arch = "i686" executable = "no">
    <ns1:source url = "file:///home/pascucci/Assist/utils/ACE-5.5/lib/libACE.so.5"/>
   </ns1:lib>
   <ns1:lib fileName = "libm.so.6" fileSystemName = "/tmp" arch = "i686" executable = "no">
    <ns1:source url = "file:///lib/tls/libm.so.6"/>
   </ns1:lib>
   <ns1:lib fileName = "libxml2.so.2" fileSystemName = "/tmp" arch = "i686" executable = "no">
    <ns1:source url = "file:///home/pascucci/Assist/utils/libxml2-2.6.27/lib/libxml2.so.2"/>
   </ns1:lib>
  </aldl:requirement>
  <aldl:requirement name = "ND000__leggiConfiguration">
   <ns1:executable master = "yes" strategy = "no" arch = "i686">/home/pascucci/Assist/compiledAssist/testKMeans//bin//
    i686-pc-linux-gnu/ND000__leggi</ns1:executable>
  </aldl:requirement>
  <aldl:requirement name = "ND001__kmeansConfigurationIsm">
   <ns1:executable master = "no" strategy = "no" arch = "i686">/home/pascucci/Assist/compiledAssist/testKMeans//bin//
    i686-pc-linux-gnu/ND001__kmeans_ism</ns1:executable>
  </aldl:requirement>
  <aldl:requirement name = "ND001__kmeansConfigurationOsm">
   <ns1:executable master = "no" strategy = "no" arch = "i686">/home/pascucci/Assist/compiledAssist/testKMeans//bin//
    i686-pc-linux-gnu/ND001__kmeans_osm</ns1:executable>
```

Generic execution requirements

Process spec.

Per process run-time support configuration

Deployment : the GEA loader

- Exploits the ALDL description
- Deploys structured parallel programs on top of diverse platforms
- Support adaptivity (partial re-deployment)
- GEA slides included from a separate presentation, some duplication of concepts

ASSIST deployment issues

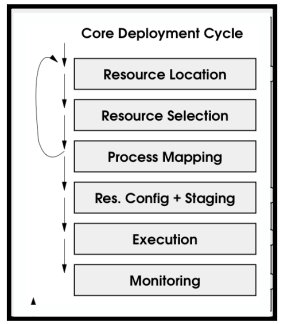
- Parallel modules
  - Reconfigure exploiting QoS models and goals
- Run-time support of the language
  - Multi-architecture (heterogeneous OS, HW)
  - On physical system (no virtual machine)
- Deployment on clusters and Grids
  - SSH, Globus
- Extendable to a component model
  - Grid.it project

GEA, the Grid Execution Agent

- Tool to automatically deploy ASSIST application
  - Implemented in Java for maximum portability
- Provides abstraction of a Grid computing platform (GAM) together with the ASSIST run-time
- Applications = multi-architecture "parallel executable" archives
  - Executable availability directs matchmaking and staging
- Essential use of the ALDL application description language
  - ALDL descriptors are compiler generated
  - do not depend on the source language

GEA Core Deployment Schema

- Deploy ASSIST applications
- Exploit High-level, structured ALDL application description
- Satisfy resource constraints
  - Static and Dynamic
  - HW, SW
  - Aggregate
- Several translation steps
- Finally exploit middleware
  - broker, allocation, staging, network configuration
- GEA provides
  - Filtering, matchmaking, mapping



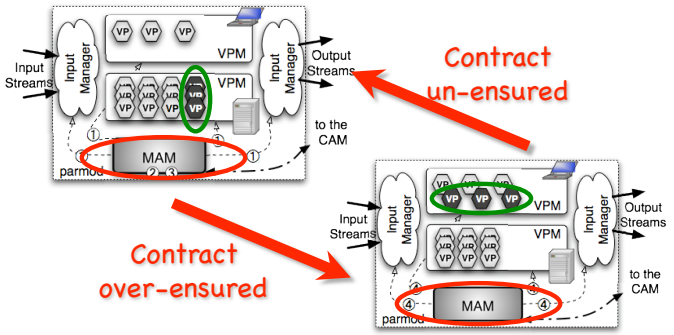
Dynamic Adaptivity

- Change resource configuration
  - Relocate processes and/or computations
  - According to environment changes
- Complex task:
  - Stop/synchronize processes
  - Exchange status / change configuration
  - Restart
- Hard to do with low-level MPI_Send() …
- Easy with ASSIST high-level code
  - Compiler knows relevant program properties (structure!!)
  - All necessary protocols are built into the run-time

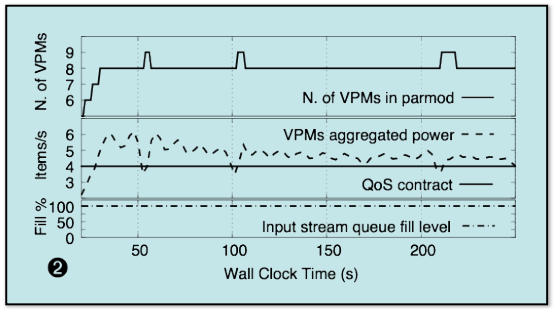
Dynamic Adaptivity

- ASSIST exploits structure-information
  - Avoid unnecessary synchronizations
  - Avoid state propagation when not useful
- Farm skeletons do not almost need to synch
  - Single stream communication can be controlled
  - Overhead is minimal
- Data parallel is reconfigurable too
  - Need to redistribute the computation
  - Same interface to add/reduce resources, and redistribute the load
- Reconfiguration either
  - User-driven
  - Based on **autonomic** control

Autonomic control



Autonomic control (2)



Core GRID

Managed by ERCIM

Execution Support of High Performance Heterogeneous Component-Based Applications on the Grid

Massimo Coppola [1,2], Marco Danelutto [2], Nicola Tonellotto [1,3], Marco Vanneschi [2], Corrado Zoccolo [4]

1 ISTI institute, CNR
2 Dipartimento di Informatica, Univ. di Pisa
3 Dipartimento di Ingegneria, Univ. di Pisa
4 IAC Search & Media Italia

CoreGRID Institute on Programming Model
CoreGRID Institute on Resource Management and Scheduling

Core GRID

ASSIST deplyment with GEA : Overview

1. GRID deployment of complex applications
   - What Grid and applications
   - Requirements and methodology
2. Hierarchical deployment of Component-based HPC applications
3. Concrete case : ASSIST/Grid.it
4. Grid Execution Agent architecture
5. Conclusions

Core GRID

Grid scenery

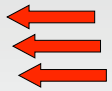
- Geographically distributed
- Heterogeneous:
  - **HW, SW,**
  - **Middleware**
  - **Multiple Virtual Organizations : authorization…**
- Dynamically changing
  - **Above reasons**
  - **Faults and failures**
  - **Load variations**

- No “Standards”: still evolving and variegated
  - Sw components, middleware, brokering …
- Can we make a Grid “Invisible” to the final user?

Core GRID

Grid Applications

- Complex, multi-disciplinary applications, High-performance
  - Distributed and parallel sw components
  - Different component frameworks
  - Heterogeneous Grid resources within a VO
  - Access through heterogeneous middleware



- We don’t want to “program the middleware”
  - Portability and separation of concern
- We want to exploit existing Grid Infrastructure
- We want predictable execution quality!

Conflicting goals?

Composite Adaptive Applications

- Exploit QoS models
  - Performance, throughput, latency, bandwidth, fault tolerance, security…
- Application-wide overall QoS model
- User-agreed application QoS contract
- Hierarchically exploit Component-level models
  - Statically and dinamically (deploy vs. autonomic)
- Deep impact on the deployment process

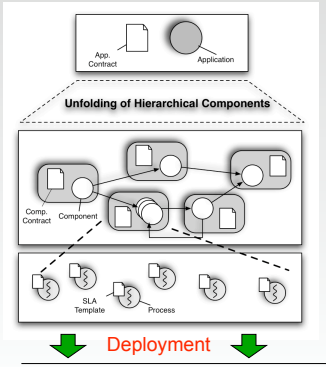
The target

- High-level user view of the deployment process
  - Independent of mentioned issues
  - Exploits application structure
  - Allows resource tuning/re-deployment at run-time
- As opposed to current
  - low level description of deployment
  - flat SPMD bag-of-tasks
  - fixed workflow description

Overview

1. GRID deployment of complex applications
2. Hierarchical deployment of Component-based HPC applications
3. Concrete case : ASSIST/Grid.it
4. Grid Execution Agent architecture
5. Conclusions

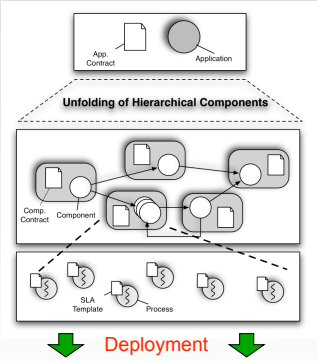
Hierarchical Deployment (1)



Unfolding translation process

- Application
  - QoS global contract
  - Composite hierarchy of components
- Components
  - Hierarchical and Parallel
  - *Parametric* Deployment Units
  - QoS specific contracts
- Component QoS models exploit knowledge from
  - Comm. and computation patterns (skeletons)
  - Compile-time and run-time support implementation

Hierarchical Deployment (2)



- Processes
  - Implement components
  - Single processes, array of processes
  - "Elementary" SLA and resource constraint, supported by middleware
  - Networks: overlay networks, physical network configuration/ reservation
- Middleware
  - Further translation
  - Deploy specs (scripts, TXT/XML)
  - Access protocols

Adaptivity and Deployment

- Deployment is a composition of workflows: Application level × component level × process level
- We exploit this recursive definition
- Adaptivity needs progressive deployment
  - We can re-deploy some sub-structures
  - We accept a good mapping at deployment
  - Improve during execution
- Avoid too large problems in matchmaking/mapping
- Structure, model description provided by compilers

Overview

1. GRID deployment of complex applications
2. Hierarchical deployment of Component-based HPC applications
3. Concrete case : ASSIST/Grid.it
   - ASSIST programming environment
   - GEA tool for deployment
   - Grid.it component model and new requirements
4. Grid Execution Agent architecture
5. Conclusions

ASSIST

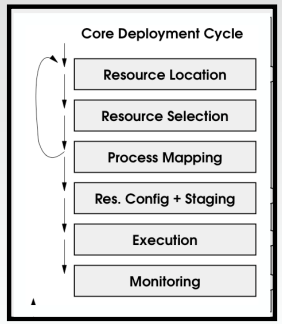
- Structured parallel language for HPC
- Modular design of applications
- Parallel modules
  - Reconfigure exploiting QoS models and goals
  - Implement well-known parallel skeletons
- Run-time support of the language
  - Multi-architecture (heterogeneous OS, HW)
  - On physical system (no virtual machine)
- Deployment on clusters and Grids
  - SSH, Globus
- Extended to a component model (Grid.it project)

GEA, the Grid Execution Agent

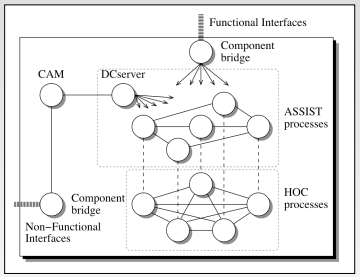
- Tool to automatically deploy ASSIST application
- Together with the language run-time provides the abstraction of a Grid computing platform (GAM)
- Deals with multi-architecture “parallel executable” archives
  - Executable availability directs matchmaking and staging
- Implemented in Java for maximum portability
- Essential use of the ALDL application description language
- ALDL descriptors are compiler generated
- ALDL does not depend on the source language

GEA Core Deployment Schema

- Previous work (Dagstuhl Seminar ‘04, FGG ‘06)
- Deploy ASSIST applications
- High-level, structured ALDL application description
- Resource constraints
  - Static and Dynamic
  - HW, SW
  - Aggregate
- Translation steps
- Exploit middleware
  - broker, allocation, staging, network configuration
- GEA provides
  - Filtering, matchmaking, mapping



Grid.it components



- Host a self-adapting structured parallel app.
- Host QoS model
- Application support proc.
  - AdHOC sh.mem
  - Reconfig. Managers
  - Framework proxies
- Can wrap / interface to CCM, WS

- ASSIST executable reacts to contracts on Non-functional ports
- Can free/ask resources to deploy tool
- Different process sets require proper synchronization protocols and deploy priorities for components’ safe startup and termination

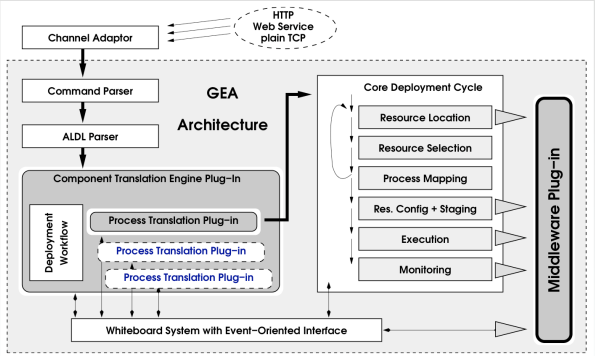
GEA New Requirements

- Multiple access protocols
  - HTTP, TCP, WS; GUI as well as system-oriented
- Separate deployments of components
- Persistent server for progressive/dynamic deployment of components and additional processes
  - support self-adaptation at run-time
- Flexibility w.r.t. component models
  - to broaden support (CCM, WS …) and ease experimentation
- Higher scalability
- Crossing of Domain Boundaries (Network, VO …)

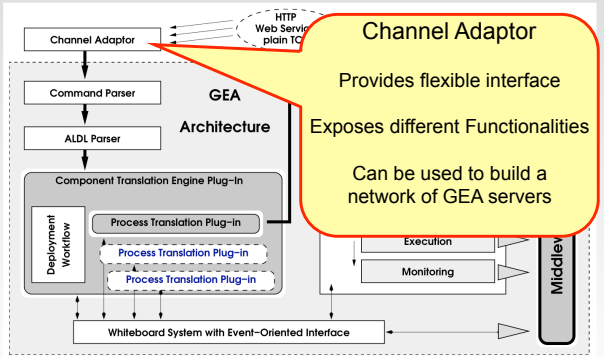
Overview

1. GRID deployment of complex applications
2. Hierarchical deployment of Component-based HPC applications
3. Concrete case : ASSIST/Grid.it
4. Grid Execution Agent architecture
   - Overall architecture
   - Component, Process Translation Plug-ins
   - Whiteboard and Middleware plug-in
5. Conclusions

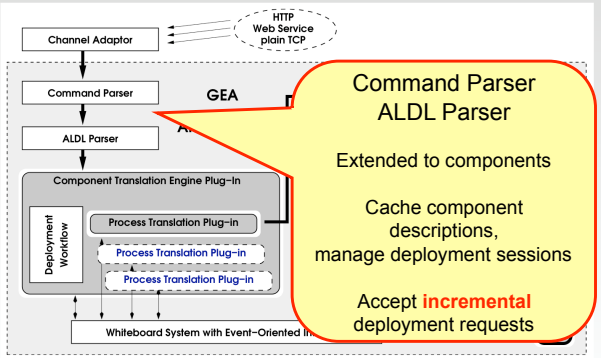
GEA Architecture Overview



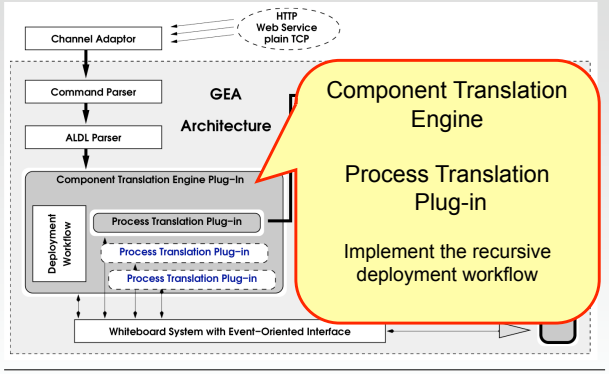
Channel Adaptor



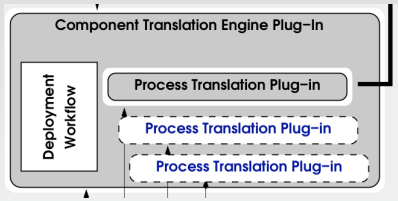
Parsers



The Plug-in based Deployment Workflow



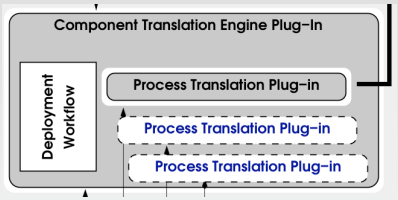
Component Translation Plug-in



- Component translation implements dependencies among support processes of a component framework
- Component plug-in is chosen at run-time

- Extract groups of homogeneous processes from component
- Check dependecies
- Pass all groups to the right plug-ins
- Execute some deploy steps through plug-ins:
  - Query & Aggregate
  - Stage & Execute

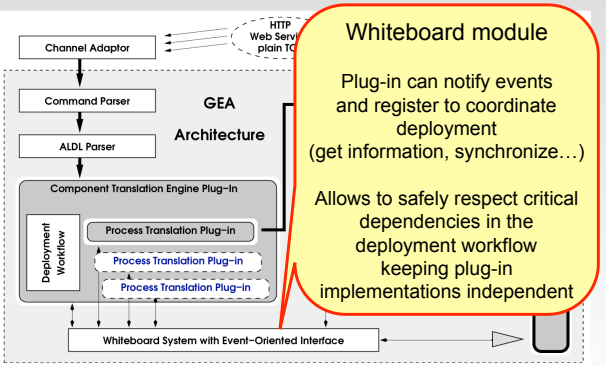
Process Translation Plug-in



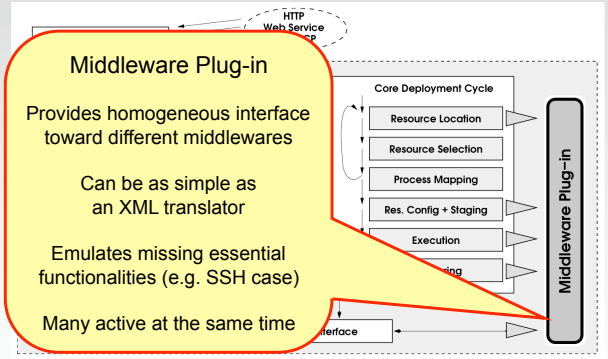
- Process plug-ins know how a kind of process is implemented
- They accordingly modify resource constraint from application

- Alter constraints before query step
  - need public IP
- Gather information before stage/exec
- Check configuration dependencies before exec.
- Synchronize exec.
- Provide config. Information to other plug-ins

Whiteboard communication



Middleware Plug-in



Work done so far (2006)

- The original GEA tools has been available within the ASSIST 1.3 distribution, supporting GT3, SSH
- Experiments were done with Network reservation middleware (PDCN '06)
- Adapted to deploy Grid.it components
- Experiments in deploying POP-C++ programs
- Comparison, possible integration with other approaches (Adage)
- Design of the GEA plug-in based architecture
- Implementation of the new architecture is ongoing

Conclusions

- We presented an extension of the deployment process which makes it customizable and exploits application structure for adaptivity
- GEA is a second-generation architecture for deployment of
  - Component-based applications
  - Autonomic self-adapting
- GEA architecture is highly configurable and extendable
- GEA acts as meta-broker
  - Heterogeneous middleware systems

References

- Web site : http://www.di.unipi.it/Assist.html
- Some related previous works
  - *HPC Application Execution on Grids*
    Dagstuhl seminar 04451, 2004; in Future Generation Grids, 2006
  - *Design and Implementation of a Grid Network-Aware Resource Broker*
    PDCN 2006, Insbruck
  - *Self-Configuring and Self-Optimising Grid Components in the GCM model and their ASSIST Implementation*
    HPC-GECO CoreGRID workshop 2006, Paris, currently available as TR-06-13, Dipartimento di Informatica (UNIPI)

Heterogeneity and Adaptivity

switch to Marco Danelutto's slides on Heterogeneity and Adaptivity in ASSIST

(file ASSIST part III B on the course wiki)

Summing up on ASSIST

- Parallel program made up by modules
  - Declared, data-flow stream interfaces
  - Unconstrained graph
- Seq modules to encapsulate seq code
  - Multi-language, code reuse
- Parmod to express parallel activities
  - High level powerful syntax
  - Skeleton oriented, shared memory available
- Run-time exploits structure information
  - Low-level details hidden to programmers
  - Automatic mapping over platforms
  - Dynamic reconfiguration
  - Portability, performance, efficiency, load balancing…

Tool References for ASSIST

- Web site:
  www.di.unipi.it://Assist.html
- Informations:
  - Massimo.coppola@isti.cnr.it
  - pascucci@di.unipi.it
  - vannesch@di.unipi.it